# NORMALIZING FLOWS FOR CALIBRATION AND RECALIBRATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

In machine learning, due to model misspecification and overfitting, estimates of the aleatoric uncertainty are often inaccurate. One approach to fix this is isotonic regression, in which a monotonic function is fit on a validation set to map the model's CDF to an optimally calibrated CDF. However, this makes it infeasible to compute additional statistics of interest on the model distribution (such as the mean). In this paper, we replace isotonic regression with normalizing flows. This allows us to retain the ability to compute the statistical properties of the model and provides an opportunity for additional capacity at the cost of possible overfitting. This approach also gives closed-form likelihoods as long as the original model distribution is also closed-form. Most importantly, the fundamental properties of normalizing flows allow us to recalibrate correlations. To aid in detecting miscalibration and measuring our success at fixing it, we introduce a new CDF Performance Plot, allowing a practitioner to diagnose calibration issues at a glance.

## 1 INTRODUCTION

Recent advances in deep learning have led to models with significantly higher overall accuracy on both classification and regression tasks compared to what was achievable in the past. However, an important component in conjunction with accuracy is a model's ability to accurately assess the uncertainty in its prediction. Most taxonomies classify uncertainty into three sources: approximation, aleatoric, and epistemic uncertainty (Der Kiureghian & Ditlevsen, 2009). Approximation uncertainty quantifies the error from fitting a simple model to complex data. Aleatoric uncertainty quantifies the uncertainty of the conditional distribution of the target variable given features. This uncertainty arises from hidden variables or measurement errors and cannot be reduced through collecting more data under the same experimental conditions. Epistemic uncertainty quantifies the uncertainty arising from fitting a model utilizing finite data, i.e. it is inversely proportional to the density of the training examples and can be reduced by collecting data in the low density regions.

These different sources of uncertainty have different techniques for handling them. Using high capacity models such as neural networks removes a large part of the approximation uncertainty. By fitting a full distribution on the target conditional on features, we can model the aleatoric uncertainty from observations. Inaccurate estimates of aleatoric uncertainty can be explained by underfitting (insufficient complexity in the conditional distributions) or overfitting (models with sufficient capacity can memorize the data, leading to the distributions collapsing to deltas). Though epistemic uncertainty is important for the model to answer what it does not know, the focus of this paper is on improving estimates of the aleatoric uncertainty.

Our approach in this paper is to handle both model fit and calibration using normalizing flows. Normalizing flows can be used in conjunction with amortized inference to improve the flexibility of the output distribution, and normalizing flows can be used in recalibration to handle any miscalibration found on a validation set. Further, we introduce a new plot to help with the analysis of the calibration of a model as well as give the statistical significance of the errors in the fit.

## 2 RELATED WORK

One method for handling aleatoric uncertainty is amortized inference with Gaussians (Lakshminarayanan et al., 2017; Nix & Weigend, 1994; Kendall & Gal, 2017) where a model, such as a

neural network, maps from features to the parameters of a Gaussian. This approach models aleatoric uncertainty directly but suffers from approximation uncertainty as a Gaussian cannot model complex targets.

Another approach is Bayesian methods such as Bayesian Ridge Regression (Tipping, 2001) and MC Dropout (Gal & Ghahramani, 2016). Similar to amortized inference with Gaussians, the output distribution limits the capacity of the model. Full Bayesian techniques with neural networks are often too computationally expensive in practice, and approximate methods often fail to capture the full complexity of the uncertainty (Lakshminarayanan et al., 2017).

Another family of methods uses quantile regression with non-linear techniques such as decision trees or neural networks. Some of these methods require a predefined set of quantiles (Takeuchi et al., 2006; Wen et al., 2017; Rodrigues & Pereira, 2018; Taylor, 2000). Simultaneous Quantile Regression (SQR, Tagasovska & Lopez-Paz (2019)) trains one model on all quantiles and is able to learn complex shaped distributions. However the training procedure requires the model to learn to be monotonic instead of being constrained to be so and is not trivial to extend to multidimensional outputs. Pearce et al. (2018) learns a finite set of quantiles by using quality metrics for predictive intervals.

Normalizing flows have been used in both the context of variational inference as well as generative modeling. The approaches to normalizing flows can be categorized into autoregressive methods (Kingma et al., 2016; Papamakarios et al., 2017; Huang et al., 2018; Cao et al., 2019), coupling layers (Dinh et al., 2014; 2016; Kingma & Dhariwal, 2018; Ho et al., 2019), residual networks (Rezende & Mohamed, 2015; van den Berg et al., 2018; Gopal, 2020) and continuous flows (Grathwohl et al., 2018).

## 3 Background

### 3.1 Recalibration

An important goal in modeling is to have well-calibrated distributions as this allows for users of the model to understand the confidence the model places on its prediction. In other words, we can better ascertain the uncertainty in the model's predictions and respond differently to those predictions depending on the uncertainty surrounding them.

Guo et al. (2017) showed that, unlike techniques used decades ago such as Bayesian Ridge Regression, modern neural network-based classifiers are very poorly calibrated; a simple variant of Platt scaling and other histogram based techniques applied to a validation set were shown to help alleviate the calibration problem where perfect calibration is defined as

$$\mathbb{P}(\hat{Y} = Y | \hat{P} = p) = p, \quad \forall p \in [0, 1]$$

where $\hat{Y}$ is a class prediction and $\hat{P}$ is its predicted probability of correctness.

Kuleshov et al. (2018) extended the analysis in Guo et al. (2017) to neural network-based regressors; isotonic regression, a method for learning monotonic univariate functions, was applied to map from $\mathcal{F}_{x_j}(y_j) = \hat{p}_j$ to $|\{y_n | \mathcal{F}_{x_n}(y_n) < \hat{p}_j, n = 1, \ldots, N\}|/N$ (the fraction of the data where the model CDF is less than $\hat{p}$) to improve calibration where perfect calibration is defined as

$$\mathbb{P}(Y < \mathcal{F}_X^{-1}(p)) = p, \quad \forall p \in [0, 1]$$

where $\mathcal{F}_X$ is the predicted CDF function.

Kuleshov et al. (2018) further introduced calibration error as a metric to quantitatively measure how well the quantiles are aligned:

$$\begin{aligned} \hat{p}_j &= |\{y_n | \mathcal{F}_{x_n}(y_n) < p_j, n = 1, \ldots, N\}| \ / \ N \\ \text{cal}(y_1, \ldots, y_N) &= \sum_{j=1}^{M} (p_j - \hat{p}_j)^2 \end{aligned} \tag{1}$$

where $M$ is the number of quantiles that are evaluated. In this paper, we set this to 100 evenly spaced quantiles.

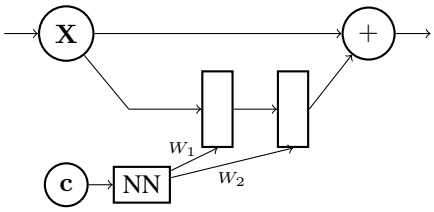



Figure 1: Hypernetwork QuAR

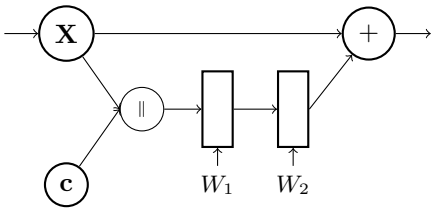



Figure 2: Conditional QuAR

### 3.2 Normalizing Flows

A crucial component in improving uncertainty estimates in this paper is to fit normalizing flows, a generative model for density estimation using invertible functions.

Suppose that we wish to formulate a joint distribution on an $n$-dimensional real vector $x$. A flow-based approach treats $x$ as the result of a transformation $g$ applied to an underlying vector $z$ sampled from a base distribution $p_z(z)$. The generative process for flows is defined as:

$$z \sim p_z(z)$$
$$x = g(z)$$

where $p_z$ is often a Normal distribution and $g$ is an invertible function. Notationally, we will use $f = g^{-1}$. Using change of variables, the log likelihood of $x$ is

$$\log p_x(x) = \log p_z\left(f(x)\right) + \log \left|\det\left(\frac{\partial f(x)}{\partial x}\right)\right|$$

To train flows (i.e. maximize the log likelihood of data points), we need to be able to compute the logarithm of the absolute value of the determinant of the Jacobian of $f$, also called the *log-determinant*. To construct large normalizing flows, we can compose smaller ones as this is still invertible and the log-determinant of this composition is the sum of the individual log-determinants.

An important observation is that every univariate distribution can be viewed as a flow in which the base distribution $p_z(z)$ is a Uniform distribution over $[0, 1]$ and $g$ is the inverse CDF of the univariate distribution. This can be generalized to multivariate distributions using the chain rule.

### 3.3 Conditional QuAR Flows

In this paper, the specific normalizing flow used is Quasi-Autoregressive Residual Flows (QuAR Flows, Gopal (2020)). QuAR Flows were shown to handle modeling complex distributions while having nice optimization properties and computationally efficient log-likelihoods. Other commonly used normalizing flows have drawbacks. Coupling layers do not work for one-dimensional inputs; autoregressive flows are often conditional Gaussians which means in the one-dimensional case, the flow is simply a Gaussian. Residual Flows (Chen et al., 2019) are expressive but, though equivalent to QuAR Flows in the one-dimensional case, are computationally expensive for high-dimensional distributions.

If we would like to condition a QuAR Flow given some features $c$, we could use a hypernetwork that maps $c$ to the parameters of the QuAR Flow (Figure 1). However, this can be computationally expensive if the QuAR Flow is parameterized with thousands of parameters. Instead, conditional information can be incorporated by concatenating the conditions $c$ before applying the first layer in the residual connection within a QuAR Flow (Figure 2).

## 4 Regression Using Normalizing Flows

The goal of regression is to model the conditional probability distribution $p(y|x)$. Frequently, this task is reduced to obtaining point estimates of the mean or median by minimizing the mean squared error or mean absolute error respectively. In order to access uncertainties, we must model the full conditional distribution. In this section, we discuss how normalizing flows can be used to this end.

### 4.1 CALIBRATION WITH NORMALIZING FLOWS

One approach to modeling aleatoric uncertainty is to use amortized inference with a Normal distribution as its output distribution, conditioned on features (Lakshminarayanan et al., 2017). However, if a Gaussian is not appropriate to fit the target, then the error in miscalibration can be attributed to approximation error, i.e. insufficient capacity to learn the distribution. A simple example of this is where we have no features to condition on (or all the features we have contain no relevant information) with a one-dimensional target. In Figure 3, we can see that since the target is bimodal, the Gaussian fit does not well describe the data.

To improve this model, we replace the Gaussian with a normalizing flow conditioned on our data, similar to Figure 2. In this way, the distribution fit to the targets has sufficient capacity to learn complex distributions, including multimodal distributions (Figure 3).

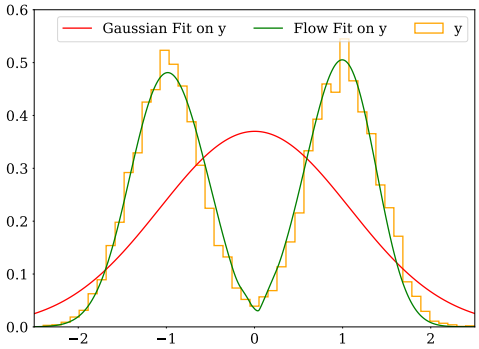



Figure 3: One-dimensional example of Gaussian versus Flow

### 4.2 RECALIBRATION WITH NORMALIZING FLOWS

To handle overfitting of the uncertainty, we recalibrate the results of the original model where we assume the original model outputs a full distribution that has closed-form likelihoods. Since the only requirement for using normalizing flows is a full distribution with a closed-form likelihood, the original model that we recalibrate can be as simple as a Bayesian Linear Regression model or as complicated as a conditional normalizing flow. We recalibrate the original model by fitting a normalizing flow on top of it. In this way, the training process is now the following two-step procedure:

1. $f_\theta = \arg\max_{f_\theta} \sum_{i=1}^{N_{tr}} \log p_{f_\theta}(y_i^{tr}|x_i^{tr})$
2. $g_\phi = \arg\max_{g_\phi} \sum_{i=1}^{N_{val}} \log p_{g_\phi}(F_\theta(y_i^{val}|x_i^{val}))$

where $f_\theta(y|x)$ is the predicted pdf function given $x$, $F_\theta(y|x)$ is the CDF of $f_\theta(y|x)$, $\{x^{tr}, y^{tr}\}_{i=1}^{N_{tr}}$ is the training set and $\{x^{val}, y^{val}\}_{i=1}^{N_{val}}$ is the validation set.

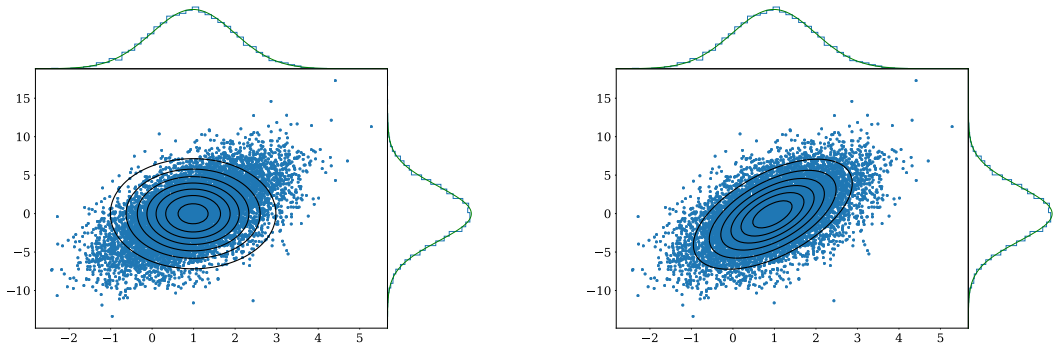
After the two-step training, the new generative process is:

$$z \sim \text{Uniform}(0, 1)$$
$$u = G_\phi^{-1}(z)$$
$$y = F_\theta^{-1}(u|x)$$

where $G_\phi$ is the CDF of $g_\phi$. The density function of the above generative process is simply the product of the individual density functions: $f_\theta$ and $g_\phi$. Given this view of the generative process, one interpretation of the two-step process is to first fit $\theta$ on the training set where $\phi$ is initialized so that $G_\phi^{-1}$ is the identity function, and then to fit $\phi$ on the validation set while keeping $\theta$ frozen. Though $\theta$ and $\phi$ could be trained end-to-end, similar to the approach in Section 4.1, we show in our experiments (Section 6.2) that there are benefits to this two-step approach.

Given this log-likelihood viewpoint of recalibration, we can trivially incorporate conditioning of the normalizing flow on additional information. For example, say $f_\theta$ outputs $\mu$ and $\sigma$ that parameterize a Normal distribution, we could then condition $g_\phi$ on these two outputs. If we were to use many features to condition on, we would return to the first situation where we could overfit to the finite number of observations we have. We can think of the first model $f_\theta$ as compressing/embedding the data into two dimensions, and the second model utilizing the embedding to create a meaningful and well-calibrated distribution on the target. Compared to isotonic regression, we can easily incorporate conditional information while retaining monotonicity as normalizing flows are monotonic by construction.

Recalibration is interesting in that it is useful for both overfitting as well as underfitting. From Guo et al. (2017), we see that multinomial classification models have sufficient capacity to overfit




(a) Results where the marginals are recalibrated via isotonic regression

(b) Results where the $p(x)$ and $p(y|x)$ are recalibrated via normalizing flows

Figure 4: Comparison plots of Isotonic Regression and Flow Recalibration on multiple outputs. The two axes are different targets being modeled, and the scatter plot shows that the targets are correlated. The top and right sides show the marginal histogram as well as the fit from the recalibrated distribution. The contour plot shows the two-dimensional contours of the recalibrated distribution which can be seen to not align well for isotonic regression.

uncertainties and benefit from recalibration. On the other hand, the simple example in Figure 3 suffers from miscalibrated uncertainties due to approximation uncertainty (underfitting), and recalibration reduces the approximation error.

One solution to handling overfitting is to early-stop on a validation set. Whether early stopping occurs or not, our solution incorporates the information of the validation set to correct potential overfitting. However, if we try to correct for multiple sources of overfitting, then we could introduce a new overfitting problem. In essence, there is a tradeoff between the validation size (which in turn controls training size) and how many sources of overfitting one can try to correct after training.

### 4.3 Multivariate Calibration

Whereas Platt scaling (Guo et al., 2017) and isotonic regression (Kuleshov et al., 2018) have been shown to improve the calibration of distributions when we have one-dimensional outputs, scaling these solutions to high-dimensional targets is non-trivial.

Say we are modeling two-dimensional real-valued targets, and we use isotonic regression to calibrate the targets individually. For simplicity, say we use amortized inference with independent output distributions. If the targets are not conditionally independent, the distribution will not be well-calibrated, even after recalibration with isotonic regression (Figure 4a). However, using normalizing flows, we can compute the CDF of the observed targets per output distribution and recalibrate this two-dimensional distribution (Figure 4b).

The general procedure of recalibrating $N$-dimensional distributions is to fit a normalizing flow on the $N$-dimensional vector $x_i = F(y_i|y_{<i})$.

## 5 CDF Performance Plot

In multiclass classification, we often focus on individual metrics such as accuracy, precision, recall, F1, etc. However, exploring the accuracy of the separate classes using a confusion matrix can show if the model is biased towards or against a specific class. Similar to this view for classification, we introduce a new plot to help diagnose if a regression model is biased in terms of scale of prediction or features. Thus, we will be able to answer questions such as "is our model systematically biased for small predictions?"

An example of the CDF (calibration) performance plot we introduce is Figure 5. The key observation of this plot is that if we bucket the model predicted CDFs based off feature values, the CDFs should be uniformly distributed.

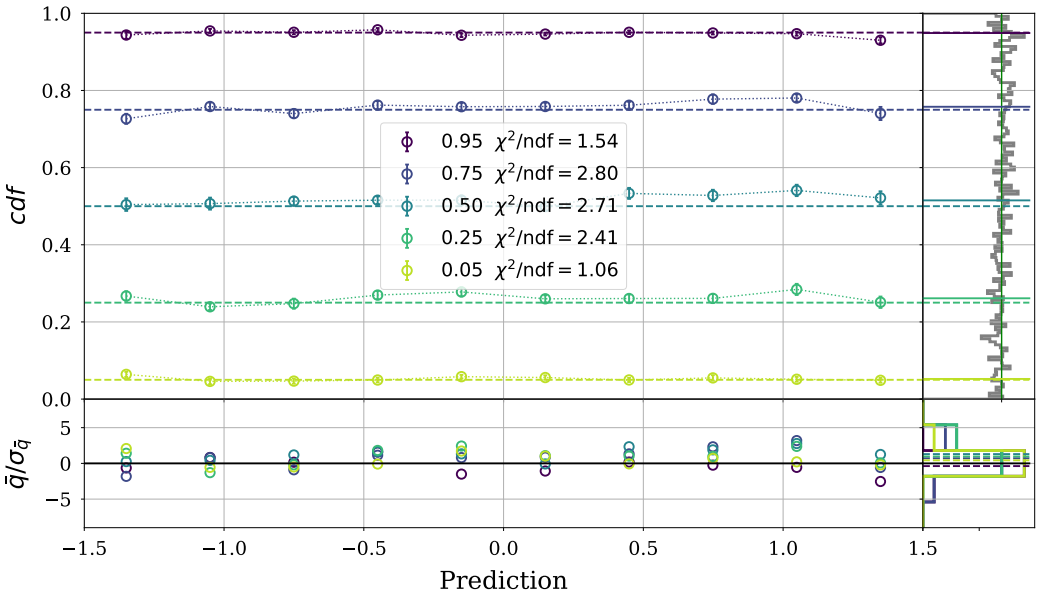



Figure 5: Example of CDF Performance Plot. The plot was generated from 20 random seeds (generating 20 random test sets) on "kin8nm" trained with a Conditional Flow

Implicitly, in the upper left panel, there is a two-dimensional histogram of model CDF versus mean prediction. Instead of showing the histogram, we show empirical quantiles within each vertical bucket. The dashed lines represent the optimal solution for the corresponding quantile. For a quantile $q$, the optimal solution would be $q$. Along with empirical quantiles, the error bars show the standard error for the quantile in the vertical bucket where the standard error is estimated using the Maritz-Jarrett estimator (Maritz & Jarrett, 1978). The upper right panel shows the marginal of the "hidden" two-dimensional histogram, i.e. a histogram of the model CDFs across the full dataset. The optimal solutions and the empirical quantiles are represented by the dashed lines and the solid lines, respectively.

The lower left panel gives, quantitatively, how close our empirical quantiles $\hat{q}$ are to the true quantiles $q$, by normalizing by the standard errors $\hat{\sigma}$ which we call the *pulls* ($\frac{\hat{q}-q}{\hat{\sigma}}$). Ideally, what we want to see here is the pulls be randomly distributed around zero. If for example, the pulls are consistently positive for small values along the $x$-axis, this suggests that our model is biased towards underestimating for small $x$-values, where $x$-values could either be the mean prediction or some feature. The lower right panel shows the marginal of the lower left panel.

Within the legend, we show the $\chi^2/\text{ndf}$ of the pulls per plotted quantile, i.e. $\sum_i(\text{pull}_i^2)/(|\text{pull}| - 1)$. The optimal value for this statistic is one. For multivariate tasks, the x-axis can be changed to one target while plotting against another target's CDFs, giving a visual of how well the model has learned the correlations.

Due to the size of the full plot, we will show just the lower left and lower right panels for our experimental results, as those panels contain the quantitative information of the fit. The full versions of the plots and more detailed descriptions are in the Appendix, along with an analogous mean performance plot.

# 6 Experiments

## 6.1 Calibration

To test the effect of increased capacity in distributions, we tested how well normalizing flows improve performance in distributional estimates compared to a few other popular approaches for estimating

Table 1: Results before and after recalibration on the UCI dataset "naval-propulsion-plant". Details on the model methodology as well as performance on other UCI datasets are in Appendix C.

| | Before Recalib. | | Isotonic Recalib. | | Flow Recalib. | |
|---|---|---|---|---|---|---|
| | MSE | Calib | MSE | Calib | MSE | Calib |
| ConditionalFlow | $0.009 \pm 0.004$ | 0.45 | N.A. | $< 0.01$ | $0.016 \pm 0.012$ | $< 0.01$ |
| ConditionalGaussian | $0.012 \pm 0.006$ | 2.35 | N.A. | $< 0.01$ | $0.012 \pm 0.006$ | $< 0.01$ |
| BayesianRidgeRegression | $0.055 \pm 0.004$ | 8.03 | N.A. | 0.01 | $0.48 \pm 0.02$ | $< 0.01$ |
| MC Dropout | $0.005 \pm 0.002$ | 0.51 | N.A. | N.A. | N.A. | N.A. |
| SQR | $0.0052 \pm 0.0012$ | 1.44 | N.A. | N.A. | N.A. | N.A. |

uncertainties. All results were evaluated on a held out test set that was not used for calibration or recalibration. Due to limited space, the full extent of the results and details of the models are left for Appendix B and Appendix C. To show the usefulness of the plot we introduced in Section 5, we focus on the results on the UCI dataset "naval-propulsion-plant" (Dheeru & Karra Taniskidou, 2017) in Table 1 and Section 6.2.

The models with consistent performance across UCI datasets in terms of calibration are Bayesian Ridge Regression, Conditional Gaussian, and Conditional Flow, with Conditional Flow often performing well in cases where Conditional Gaussian is unable to. Whereas these three models are well calibrated, the mean performance is much weaker for Bayesian Ridge Regression due to the limitations of linear functions.

### 6.2 Recalibration

Of the models we tested, we recalibrated the Bayesian Ridge Regression, Conditional Gaussian, and Conditional Flow models. MC Dropout and SQR could also be recalibrated, though we chose not to as they lack closed form CDFs, meaning the technique would be approximate. In Table 1, we see the calibration errors after recalibrating with normalizing flows are comparable to those from isotonic regression. However, after isotonic regression is applied, only the calibration error can be computed. In contrast, we can still compute ordinary distribution statistics such as the mean after applying normalizing flows. Due to this property, we can even detect cases in which an improvement in calibration error coincides with a degradation of mean performance. For example, in Table 1 this was the case for Bayesian Ridge Regression.

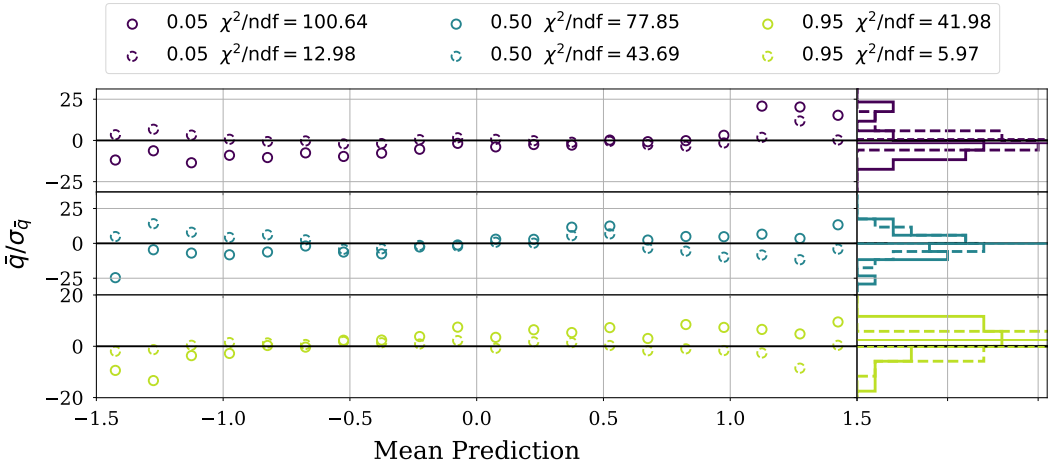



Figure 6: Comparison of CDF performance between recalibration with an unconditional flow and recalibration with a conditional flow on the dataset "naval-propulsion-plant". The solid lines are from using an unconditional flow, and the dashed lines are from using a flow conditioned on the mean from the original model.

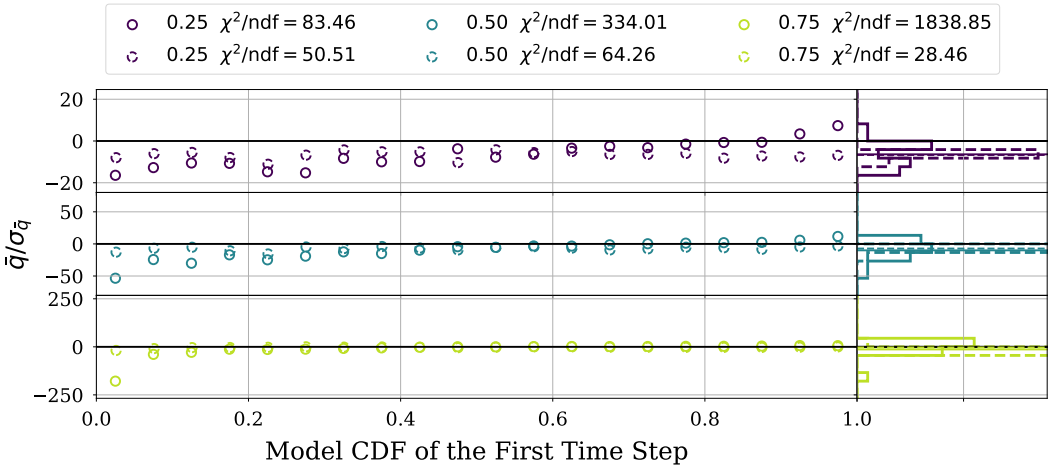



Figure 7: Comparison of CDF performance between recalibration with isotonic regression and recalibration with normalizing flows on the second time step on the dataset "electricity". The solid lines are from using isotonic regression, and the dashed lines are from using a flow trained jointly on the CDFs of the two time steps. The calibration error from isotonic regression is 0.82; the calibration error from the flow is 0.77.

Whereas the calibration error metric shows the flow recalibrated model is well-calibrated averaged across the full dataset, in Figure 6 we see (in the solid lines) that there is a systematic bias with respect to the mean prediction as well as statistically significant errors.

By systematic bias, we mean that for negative predictions, the model consistently overestimates the quantiles as the pulls are consistently before the zero line. For positive predictions, the model consistently underestimates the quantiles as the pulls are consistently above the zero line.

To improve this, we recalibrate with a normalizing flow conditioned on the mean predicted by the original model. In this way, we see (in the dashed lines) in Figure 6 that the systematic bias is reduced as the pulls are more randomly distributed about the zero line, and the errors are not as statistically significant with respect to the mean prediction.

Full version of Figure 6 is in Appendix D.1.

### 6.3 Time Series

For our time series example, we trained a model on the UCI Dataset "electricity". We predict two time steps and show that even though the model could learn correlated distributions, there is still some correlation not captured that recalibration can help with. Figure 7 shows (in the solid lines) that recalibrating independently with isotonic regression has very statistically significant errors as well as a very strong trend as a function of the CDF of the first time step. From the dashed lines, we can see that a flow trained jointly on the CDFs significantly reduced the errors. Furthermore, although it was still systematically biased, the trend was less pronounced than when using isotonic regression. This is visually clearer in the full versions of the plot in Appendix D.2. Finally, although the calibration error of the marginal distribution for the second step is comparable between the two approaches, the conditional performance favors normalizing flows.

## 7 Conclusion and Future Work

In this paper, we addressed the calibration issue in modern neural networks through using higher capacity distributions via normalizing flows. We showed that normalizing flows can not only help during calibration, but also in recalibrating miscalibrated distributions, even improving the fit in the underfitting case. Further, we introduce a new plot to help researchers detect calibration issues concisely.

## REFERENCES

Nicola De Cao, Wilker Aziz, and Ivan Titov. Block neural autoregressive flow. In *Conference on Uncertainty in Artificial Intelligence*, volume 3, pp. 1723–1735. UAI, 2019. URL http://auai.org/uai2019/proceedings/papers/511.pdf.

Ricky T. Q. Chen, Jens Behrmann, David K Duvenaud, and Joern-Henrik Jacobsen. Residual flows for invertible generative modeling. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dÁlché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 9916–9926. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9183-residual-flows-for-invertible-generative-modeling.pdf.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *arXiv e-prints*, art. arXiv:1511.07289, November 2015.

Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural safety*, 31(2):105–112, 2009.

Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear Independent Components Estimation. *arXiv e-prints*, art. arXiv:1410.8516, October 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *arXiv e-prints*, art. arXiv:1605.08803, May 2016.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/gal16.html.

Achintya Gopal. Quasi-Autoregressive Residual (QuAR) Flows. *arXiv e-prints*, art. arXiv:2009.07419, September 2020.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *arXiv e-prints*, art. arXiv:1810.01367, October 2018.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1321–1330, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/guo17a.html.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2722–2730, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/ho19a.html.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2078–2087, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/huang18d.html.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In

I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 3146–3154. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5574–5584. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7141-what-uncertainties-do-we-need-in-bayesian-deep-learning-for-computer-vision.pdf.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 10215–10224. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4743–4751. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6581-improved-variational-inference-with-inverse-autoregressive-flow.pdf.

Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2796–2804, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/kuleshov18a.html.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6402–6413. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7219-simple-and-scalable-predictive-uncertainty-estimation-using-deep-ensembles.pdf.

JS Maritz and RG Jarrett. A note on estimating the variance of the sample median. *Journal of the American Statistical Association*, 73(361):194–196, 1978.

David A Nix and Andreas S Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*, volume 1, pp. 55–60. IEEE, 1994.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 2338–2347. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6828-masked-autoregressive-flow-for-density-estimation.pdf.

Tim Pearce, Alexandra Brintrup, Mohamed Zaki, and Andy Neely. High-quality prediction intervals for deep learning: A distribution-free, ensembled approach. volume 80 of *Proceedings of Machine Learning Research*, pp. 4075–4084, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/pearce18a.html.

Boris Polyak and Anatoli Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30:838–855, 07 1992. doi: 10.1137/0330046.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/rezende15.html.

Filipe Rodrigues and Francisco C. Pereira. Beyond expectation: Deep joint mean and quantile regression for spatio-temporal problems. *arXiv e-prints*, art. arXiv:1808.08798, August 2018.

David Salinas, Valentin Flunkert, and Jan Gasthaus. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *arXiv e-prints*, art. arXiv:1704.04110, April 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Natasa Tagasovska and David Lopez-Paz. Single-model uncertainties for deep learning. In *Advances in Neural Information Processing Systems 32*, pp. 6417–6428. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/8870-single-model-uncertainties-for-deep-learning.pdf.

Ichiro Takeuchi, Quoc V Le, Timothy D Sears, and Alexander J Smola. Nonparametric quantile estimation. *Journal of Machine Learning Research*, 7:1231–1264, 2006.

James W Taylor. A quantile regression neural network approach to estimating the conditional density of multiperiod returns. *Journal of Forecasting*, 19:299–311, 2000.

Michael E Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1(Jun):211–244, 2001.

Rianne van den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling. Sylvester Normalizing Flows for Variational Inference. *arXiv e-prints*, art. arXiv:1803.05649, March 2018.

Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A Multi-Horizon Quantile Recurrent Forecaster. *arXiv e-prints*, art. arXiv:1711.11053, November 2017.

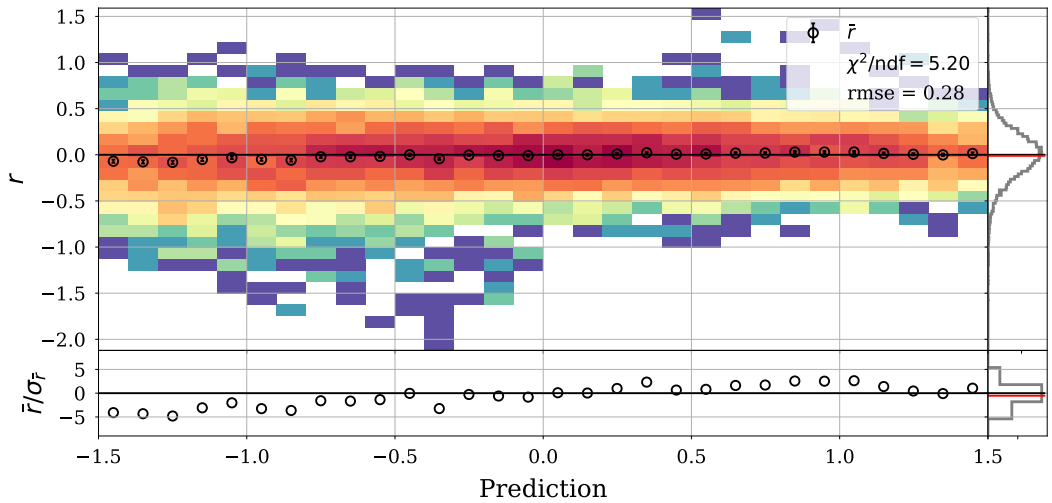



Figure 8: Example of Mean Performance Plot. Plot generated from 20 seeds (generating 20 random validation sets) on kin8nm trained with a Conditional Flow

## A MEAN PERFORMANCE PLOT

Analogous to the CDF performance plot we introduced in Section 5, we also created a mean performance plot. An example of the mean performance plot we introduce is Figure 8. We will describe the four components of the plot: a) upper left b) upper right c) lower left d) lower right.

a) Often to analyze the performance of the mean, the plot generated is observations versus predictions where Appendix E.1 shows why this is the correct direction. However, even if we were to plot the $y = x$ line, it's not visually clear if the two are aligned. Instead, we plot the residuals (observations - predictions) versus the predictions; in this case, we want the mean of each vertical bucket to be close to zero.

   Taking this approach, we can extend the plot by changing the $x$-axis from predictions to any feature we have. Appendix E.1 shows that the mean residual should still be zero. If we were to plot observations versus feature values, we would not intuitively know what the mean in each vertical bucket should be; if we plot the residuals on the y-axis, we know that the mean should be zero.

   Within each vertical bucket, we plot the mean residual as well as the standard error within the bucket, $\frac{\sigma}{\sqrt{n}}$.

   Visually, we would like to see the variance in the $y$ direction be low; however, this is going to be limited by the true aleatoric uncertainty.

b) The upper right plot shows the marginal of the residuals. The red line shows the mean of the histogram and thus similar to the upper left plot, we want the mean of this histogram to be zero.

c) Though the upper left plot shows qualitatively how well aligned the mean of each vertical bucket is with respect to zero, we would like to know quantitatively how close the mean is to zero. Instead of showing just the difference between the mean and zero, we normalize the difference by the standard error, which we call the pull. Without this component of the plot, we would not have a quantitative metric of how well our model is fit. Ideally, what we want to see here is the pulls be randomly distributed around zero.

   If for example, the pulls are consistently positive for small values along the $x$-axis, this suggests that our model is biased towards underpredicting for small $x$-values, where $x$-values could either be the mean prediction or some feature.

d) Similar to how the upper right plot is the marginal of the upper left plot, the lower right plot is the marginal of the lower left plot. Ideally, we want to see this normally distributed around zero.

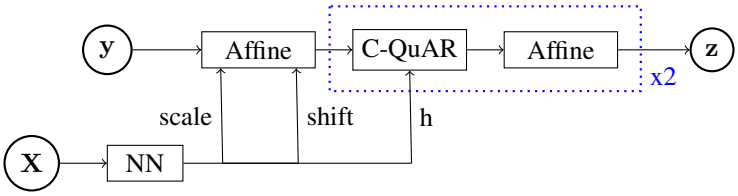



Figure 9: In this figure, we show our conditional flow model. A neural network takes $X$ and outputs the scale and shift of an affine flow and a condition vector for QuAR Flows. The condition used in the QuAR Flow is the same for both QuAR Flows. After the first conditional affine flow, there are two QuAR Flows with unconditional affine flows placed in between these two and at the end. The Conditional Flow model is equivalent to a Conditional Gaussian if we removed the QuAR Flows.

# B Model Training Details

## B.1 Calibration

We compare the performance of our model and multiple other popular approaches on 8 UCI datasets (Dheeru & Karra Taniskidou, 2017): boston-housing, concrete, energy, kin8nm, naval-propulsion-plant, power-plant, wine-quality-red and yacht. The models we compared Conditional Flows to were: Conditional Gaussian (amortized inference with Gaussians, Lakshminarayanan et al. (2017)), Bayesian Ridge Regression (Tipping, 2001), MC Dropout (Gal & Ghahramani, 2016), Simultaneous Quantile Regression (SQR) (Tagasovska & Lopez-Paz, 2019), Conditional Quantile (SQR where we only learn prespecified quantiles), Quality Driven (Pearce et al., 2018), and a Gradient Boosted Decision Tree with LightGBM (Ke et al., 2017).

Figure 9 shows a representation of the Conditional Flow network we used. The QuAR Flows are conditioned on a 64-dimensional representation from the condition network. Each Conditional QuAR Flow has two hidden layers in the residual connection with a hidden size 64 with ELU (Clevert et al., 2015) as our activation. The condition network has a single hidden layer with size 64 and ReLU as our activation. The QuAR Flows are conditioned on the 64-dimensional representation from the condition network.

For the other neural network approaches, we used networks with two hidden layers with size 128 and used ReLU as our nonlinearity. For hyperparameter tuning on all our neural networks, we searched the following grid:

- Learning rate: 1e-2, 1e-3, 1e-4
- Weight decay: 0, 1e-3, 1e-2, 1e-1
- Dropout Rates: 0.1 0.25, 0.5 0.75

The hyperparameters we used were taken from Tagasovska & Lopez-Paz (2019). We trained each of our neural networks for 2000 epochs with batch size 128.

For our LightGBM tree model (Ke et al., 2017), we used mean squared error as our loss to learn the mean of the output distribution. The grid of hyperparameters we searched was:

- Number of Trees: 100, 1000
- Maximum Depths: 3, 5, 7, 9
- Learning Rates: 1e-1, 1e-2, 1e-3, 1e-4
- Minimum Child Samples: 1, 20
- Dropout Rates: 0.1 0.25, 0.5 0.75

To make the tree model distributional, assuming homoscedasticity, we computed the standard deviation of the predicted normal distribution by computing the standard deviation of the residuals (observations minus predictions).

For SQR and MC Dropout, we computed the estimated mean and CDF from 1000 samples. For Conditional Flows, we computed the estimated mean by using the Gauss-Legendre quadrature with 100 points to approximate the integral.

### B.2 RECALIBRATION

For recalibration, we trained the recalibration model on the validation set of each of the 20 seeds. For the flow architecture, we stacked 8 QuAR Flows with affine flows in between each one as well as at the beginning and end. Since the input to the flow is a CDF $\in [0, 1]$, we also transform the data with a Logit Function as the first layer of the flow and account for the change of variables in the loss. In addition to this, we used a dropout rate of 0.25, two hidden layers with size 64 in each residual connection, and used Polyak averaging (Polyak & Juditsky, 1992) with a decay of 0.999 for evaluation.

### B.3 TIME SERIES

For our time series experiments, we tested the performance of flow recalibration on the UCI dataset "electricity". We filtered the data down to observations after 2014-01-01. We split the data by date, where the first 72% of dates are in the train set, the next 18% are in validation set, and the last 10% are in the test set.

The data is given in 15 minute increments, and so similarly to Deep AR (Salinas et al., 2017), we averaged the values per hour to have hourly data. Since we were trying to analyze the effect of correlated targets, we predicted two time steps conditioned on 48 hours of data using an autoregressive model.

We included three numeric covariates: the time step within the sampled time series, the hour of the day, and the day of the week. Each of the covariates were normalized to have zero mean and one standard deviation. In addition to this, since there are 370 different types of time series, we also added an embedding layer from 370 to an embedding dimension of 20, same as that used in Deep AR (Salinas et al., 2017). We concatenated this vector with the observations and covariates, leading to a 27-dimensional vector.

Since all the observations are positive, we model the target with a conditional Gamma distribution (i.e. amortized inference with Gamma distribution instead of a Gaussian). Similar to Deep AR (Salinas et al., 2017), we normalized each time series $\{z_{i,t}\}_{t=0}^{T-1}$ by $v_i = 1 + 1/T \sum_{t=0}^{T-1} z_{i,t}$. We account for the scaling in the output distribution, i.e. Gamma$(k, \theta * v_i)$.

The model we used was a 1 layer LSTM, and then two linear layers with a ReLU and Dropout (Srivastava et al., 2014) in between were applied to the hidden representation. We set the dropout rate to 0.5 and used a hidden size of 48 everywhere. For evaluation, we used Polyak averaging (Polyak & Juditsky, 1992) with a decay of 0.999.

For recalibration with flows, we trained on the CDFs for both time steps with a flow with the same architecture as that used in Appendix B.2.

## C EXPERIMENTAL RESULTS

### C.1 CALIBRATION

In Table 2, we compare Conditional Flows against multiple other approaches for estimating uncertainty, and we measure performance using Prediction Interval Coverage Probability (PICP), Mean Squared Error (MSE), and Calibration Error.

For each model, we trained the model on 20 different seeds where each seed randomly splits the data in 72% train, 18% validation, and 10% test. We picked the best hyperparameter by choosing the hyperparameters that gave the best average validation loss across the 20 seeds where the loss used is the loss defined by the training procedure for the model, e.g. for Conditional Gaussian, we used log likelihood; for LightGBM, we used MSE; for QualityDriven, we used the Quality Driven Loss, etc.

Table 2: Results on percentage of data captured from 0.025 to 0.975 quantiles (PICP), MSE, and calibration error with 100 buckets. The results are from minimizing the average validation loss across 20 seeds. The optimal value of PICP is 0.95, and the optimal value for MSE and Calib is 0.00.

| | bostonHousing | | | concrete | | |
|---|---|---|---|---|---|---|
| | PICP | MSE | Calib | PICP | MSE | Calib |
| ConditionalFlow | $0.93 \pm 0.04$ | $0.13 \pm 0.06$ | 0.06 | $0.92 \pm 0.03$ | $0.09 \pm 0.03$ | 0.01 |
| ConditionalGaussian | $0.93 \pm 0.03$ | $0.14 \pm 0.07$ | 0.13 | $0.91 \pm 0.02$ | $0.09 \pm 0.03$ | 0.06 |
| BayesianRidgeRegression | $0.95 \pm 0.03$ | $0.28 \pm 0.08$ | 0.34 | $0.95 \pm 0.03$ | $0.40 \pm 0.07$ | 0.01 |
| MC Dropout | $0.50 \pm 0.08$ | $0.11 \pm 0.05$ | 2.99 | $0.69 \pm 0.05$ | $0.07 \pm 0.03$ | 0.89 |
| SQR | $0.73 \pm 0.08$ | $0.11 \pm 0.05$ | 0.48 | $0.74 \pm 0.04$ | $0.07 \pm 0.03$ | 0.47 |
| ConditionalQuantile | $0.92 \pm 0.05$ | N.A. | N.A. | $0.93 \pm 0.03$ | N.A. | N.A. |
| QualityDriven | $0.83 \pm 0.06$ | N.A. | N.A. | $0.83 \pm 0.04$ | N.A. | N.A. |
| LightGBM | $0.04 \pm 0.03$ | $0.12 \pm 0.06$ | 7.74 | $0.63 \pm 0.07$ | $0.07 \pm 0.02$ | 1.25 |

| | energy | | | kin8nm | | |
|---|---|---|---|---|---|---|
| | PICP | MSE | Calib | PICP | MSE | Calib |
| ConditionalFlow | $0.96 \pm 0.02$ | $0.0022 \pm 0.0006$ | 0.12 | $0.952 \pm 0.006$ | $0.081 \pm 0.013$ | 0.01 |
| ConditionalGaussian | $0.97 \pm 0.03$ | $0.0030 \pm 0.0009$ | 0.26 | $0.947 \pm 0.007$ | $0.071 \pm 0.005$ | $< 0.01$ |
| BayesianRidgeRegression | $0.89 \pm 0.04$ | $0.09 \pm 0.02$ | 0.24 | $0.951 \pm 0.007$ | $0.59 \pm 0.02$ | 0.09 |
| MC Dropout | $0.97 \pm 0.02$ | $0.0026 \pm 0.0010$ | 0.17 | $0.69 \pm 0.02$ | $0.069 \pm 0.005$ | 1.14 |
| SQR | $0.97 \pm 0.02$ | $0.0028 \pm 0.0012$ | 0.37 | $0.892 \pm 0.010$ | $0.072 \pm 0.005$ | 0.01 |
| ConditionalQuantile | $0.98 \pm 0.02$ | N.A. | N.A. | $0.943 \pm 0.011$ | N.A. | N.A. |
| QualityDriven | $0.96 \pm 0.04$ | N.A. | N.A. | $0.88 \pm 0.02$ | N.A. | N.A. |
| LightGBM | $0.24 \pm 0.05$ | $0.0013 \pm 0.0006$ | 5.18 | $0.205 \pm 0.010$ | $0.190 \pm 0.011$ | 5.59 |

| | naval-propulsion-plant | | | power-plant | | |
|---|---|---|---|---|---|---|
| | PICP | MSE | Calib | PICP | MSE | Calib |
| ConditionalFlow | $0.98 \pm 0.03$ | $0.009 \pm 0.004$ | 0.45 | $0.952 \pm 0.007$ | $0.048 \pm 0.007$ | 0.01 |
| ConditionalGaussian | $0.9995 \pm 0.0007$ | $0.012 \pm 0.006$ | 2.35 | $0.947 \pm 0.010$ | $0.048 \pm 0.006$ | $< 0.01$ |
| BayesianRidgeRegression | $1.0 \pm 0.0$ | $0.055 \pm 0.004$ | 8.03 | $0.964 \pm 0.007$ | $0.072 \pm 0.007$ | 0.01 |
| MC Dropout | $0.98 \pm 0.02$ | $0.005 \pm 0.002$ | 0.51 | $0.59 \pm 0.02$ | $0.046 \pm 0.007$ | 1.88 |
| SQR | $0.998 \pm 0.002$ | $0.0052 \pm 0.0012$ | 1.44 | $0.93 \pm 0.02$ | $0.048 \pm 0.007$ | 0.02 |
| ConditionalQuantile | $0.97 \pm 0.02$ | N.A. | N.A. | $0.937 \pm 0.014$ | N.A. | N.A. |
| QualityDriven | $0.949 \pm 0.010$ | N.A. | N.A. | $0.89 \pm 0.02$ | N.A. | N.A. |
| LightGBM | $0.714 \pm 0.011$ | $0.0056 \pm 0.0009$ | 0.79 | $0.17 \pm 0.02$ | $0.035 \pm 0.007$ | 5.95 |

| | wine-quality-red | | | yacht | | |
|---|---|---|---|---|---|---|
| | PICP | MSE | Calib | PICP | MSE | Calib |
| ConditionalFlow | $0.90 \pm 0.02$ | $0.6 \pm 0.2$ | 0.04 | $0.95 \pm 0.04$ | $0.004 \pm 0.003$ | 0.27 |
| ConditionalGaussian | $0.95 \pm 0.02$ | $0.59 \pm 0.09$ | 0.22 | $0.99 \pm 0.02$ | $0.004 \pm 0.002$ | 0.62 |
| BayesianRidgeRegression | $0.95 \pm 0.02$ | $0.63 \pm 0.11$ | 0.05 | $0.94 \pm 0.03$ | $0.35 \pm 0.10$ | 0.25 |
| MC Dropout | $0.35 \pm 0.04$ | $0.57 \pm 0.09$ | 4.01 | $0.97 \pm 0.04$ | $0.004 \pm 0.002$ | 0.21 |
| SQR | $0.81 \pm 0.09$ | $0.59 \pm 0.10$ | 0.15 | $0.96 \pm 0.04$ | $0.004 \pm 0.004$ | 0.49 |
| ConditionalQuantile | $0.95 \pm 0.02$ | N.A. | N.A. | $0.99 \pm 0.02$ | N.A. | N.A. |
| QualityDriven | $0.84 \pm 0.03$ | N.A. | N.A. | $0.97 \pm 0.04$ | N.A. | N.A. |
| LightGBM | $0.40 \pm 0.05$ | $0.55 \pm 0.10$ | 2.93 | $0.45 \pm 0.08$ | $0.003 \pm 0.003$ | 3.04 |

Table 3: Calibration error (with 100 buckets) before and after recalibration.

| | BayesianRidgeRegression | | | ConditionalGaussian | | | ConditionalFlow | | |
|---|---|---|---|---|---|---|---|---|---|
| | Before | Flow | Iso | Before | Flow | Iso | Before | Flow | Iso |
| bostonHousing | 0.34 | 0.02 | 0.02 | 0.13 | 0.04 | 0.02 | 0.06 | 0.02 | 0.01 |
| concrete | 0.01 | 0.01 | 0.01 | 0.06 | 0.02 | 0.02 | 0.01 | 0.02 | 0.03 |
| energy | 0.24 | 0.02 | 0.01 | 0.26 | $< 0.01$ | 0.01 | 0.12 | 0.01 | 0.01 |
| kin8nm | 0.09 | 0.01 | $< 0.01$ | $< 0.01$ | $< 0.01$ | $< 0.01$ | 0.01 | $< 0.01$ | $< 0.01$ |
| naval-propulsion-plant | 8.03 | 0.01 | $< 0.01$ | 2.35 | $< 0.01$ | $< 0.01$ | 0.45 | $< 0.01$ | $< 0.01$ |
| power-plant | 0.01 | $< 0.01$ | $< 0.01$ | $< 0.01$ | $< 0.01$ | $< 0.01$ | 0.01 | $< 0.01$ | $< 0.01$ |
| wine-quality-red | 0.05 | 0.02 | $< 0.01$ | 0.22 | 0.04 | 0.02 | 0.04 | 0.01 | 0.01 |
| yacht | 0.25 | 0.05 | 0.07 | 0.62 | 0.03 | 0.03 | 0.27 | 0.02 | 0.03 |

Table 4: MSE performance before and after recalibration. We do not show the MSE for isotonic regression as the mean cannot be computed after recalibrating with isotonic regression.

| | BayesianRidgeRegression | | ConditionalGaussian | |
|---|---|---|---|---|
| | Before | Flow | Before | Flow |
| bostonHousing | $0.28 \pm 0.08$ | $0.28 \pm 0.08$ | $0.14 \pm 0.07$ | $0.14 \pm 0.07$ |
| concrete | $0.40 \pm 0.07$ | $0.40 \pm 0.07$ | $0.09 \pm 0.03$ | $0.09 \pm 0.03$ |
| energy | $0.09 \pm 0.02$ | $0.09 \pm 0.02$ | $0.0030 \pm 0.0009$ | $0.0032 \pm 0.0011$ |
| kin8nm | $0.59 \pm 0.02$ | $0.59 \pm 0.02$ | $0.071 \pm 0.005$ | $0.072 \pm 0.005$ |
| naval-propulsion-plant | $0.055 \pm 0.004$ | $0.48 \pm 0.02$ | $0.012 \pm 0.006$ | $0.012 \pm 0.007$ |
| power-plant | $0.072 \pm 0.007$ | $0.072 \pm 0.007$ | $0.048 \pm 0.006$ | $0.049 \pm 0.006$ |
| wine-quality-red | $0.63 \pm 0.11$ | $0.63 \pm 0.11$ | $0.59 \pm 0.09$ | $0.59 \pm 0.10$ |
| yacht | $0.35 \pm 0.10$ | $0.36 \pm 0.10$ | $0.004 \pm 0.002$ | $0.004 \pm 0.003$ |

| | ConditionalFlow | |
|---|---|---|
| | Before | Flow |
| bostonHousing | $0.13 \pm 0.06$ | $0.13 \pm 0.06$ |
| concrete | $0.09 \pm 0.03$ | $0.09 \pm 0.03$ |
| energy | $0.0022 \pm 0.0006$ | $0.0027 \pm 0.0007$ |
| kin8nm | $0.081 \pm 0.013$ | $0.081 \pm 0.013$ |
| naval-propulsion-plant | $0.009 \pm 0.004$ | $0.016 \pm 0.012$ |
| power-plant | $0.048 \pm 0.007$ | $0.048 \pm 0.007$ |
| wine-quality-red | $0.6 \pm 0.2$ | $0.64 \pm 0.15$ |
| yacht | $0.004 \pm 0.003$ | $0.008 \pm 0.012$ |

Reported in Table 2 are the mean and standard deviation of PICP and MSE across the 20 seeds. Due to the limited data, for calibration error with 100 bins, we computed it over the CDFs from all 20 seeds. We included PICP (the percentage of true observations between the quantiles 0.025 and 0.975) to compare against alternative approaches where we prespecify which quantiles we need.

### C.2 Recalibration

From the above section, we recalibrate the best Bayesian Ridge Regression, Conditional Gaussian, and Conditional Flow. In Table 3, we compare the calibration error before recalibration, after recalibration with isotonic regression, and after recalibration with normalizing flows.

As we can see, across all 8 UCI datasets, the calibration errors after recalibrating with normalizing flows are comparable to those from isotonic regression. More interestingly though, improvement in calibration error can be detrimental to mean performance.

# D FULL PLOTS

## D.1 RECALIBRATION

The full plots for Figure 6 are Figure 10 and Figure 11. Visually, we can see the model is well-calibrated across the full dataset since the histogram in the upper right panel is quite uniform. We can also see in Figure 10 the full visualization of the systematic bias mentioned in the paper: the percentiles are consistently overestimated for small predictions and underestimated for large ones. On the other hand, Figure 11 shows visually the 5th and 95th quantile are well calibrated. There still is some bias in the other quantiles; however, the bias is less than that found in Figure 10.

## D.2 TIME SERIES

The full plots for Figure 7 are Figure 12 and Figure 13. Visually, we can see both models are similarly calibrated across the full dataset since the histogram in the upper right panel is quite similar. However, we can see that in Figure 12, the CDFs of the second time step are strongly dependent on the CDF of the first time step. In essence, this means the LSTM was not able to fully capture the positive correlation between the two targets. On the other hand, Figure 13, though still biased, does not have this trend and the performance is more uniform across CDFs of the first time step.

# E PLOT PROOFS

An important note in the design of the mean and CDF performance plots is the view that the data we are seeing is from a mixture distribution where the features describe which component of the mixture the observation comes from. In this way, when we visualize the data, we need to account for the situation where we are marginalizing over our features. On top of this, we do not know the true $p(X)$ over our features and so we must make sure the value we are estimating can be estimated from the data.

## E.1 MEAN PERFORMANCE

Given some scalar function on the features $g(x)$ (for example, mean prediction) is in some set of values $\mathcal{V}$, the mean of the residuals (observation minus true conditional mean) in this set has a value of 0 in expectation.

$$\mathbb{E}_{x\sim p_X}\left[\mathbb{E}_{y\sim p_{Y|X}}\left[y-\mathbb{E}_{y\sim p_{Y|X}}[y]\right] \,\middle|\, g(x)\in\mathcal{V}\right]=0$$

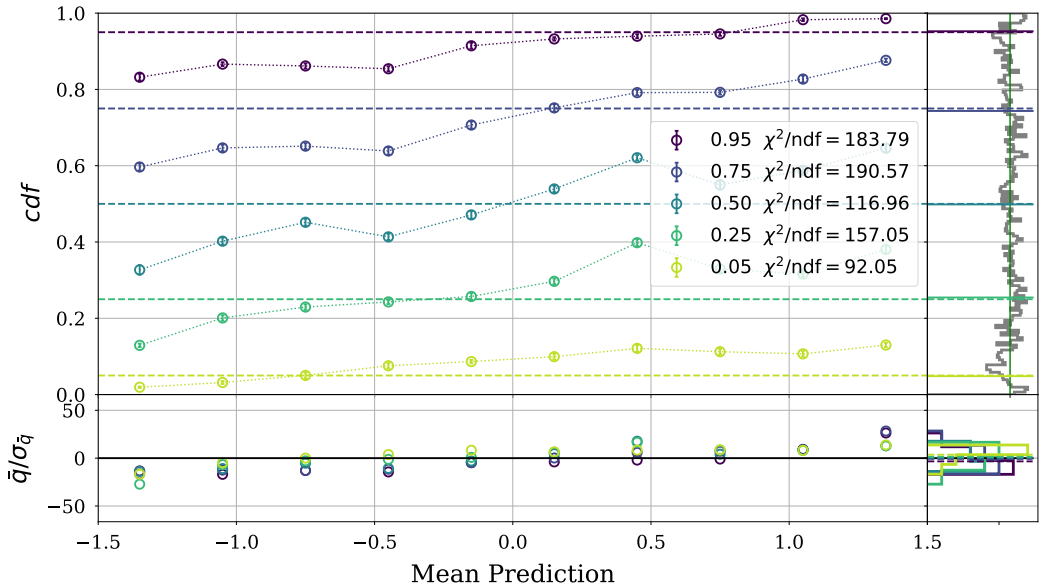
The mean performance plot is where $g(x)$ is the function to compute the mean from the features and $\mathcal{V}$ are the different vertical buckets. An important note about the above is that this is true whether the targets are continuous or discrete.

However, if we condition on $y \in \mathcal{Y}$ (observation in the x-axis), then the mean of the residual is

$$\begin{aligned} V &= \mathbb{E}_{x\sim p_X}\left[\mathbb{E}_{y\sim p_{Y|X}}\left[y-\mathbb{E}_{y\sim p_{Y|X}}[y]\middle|y\in\mathcal{Y}\right]\right] \\ &= \int dx\, p(x)\int_{y\in\mathcal{Y}} dy\, \frac{p(y|x)\left(y-\mathbb{E}_{y\sim p_{Y|X}}[y]\right)}{p(y\in\mathcal{Y}|x)} \end{aligned}$$

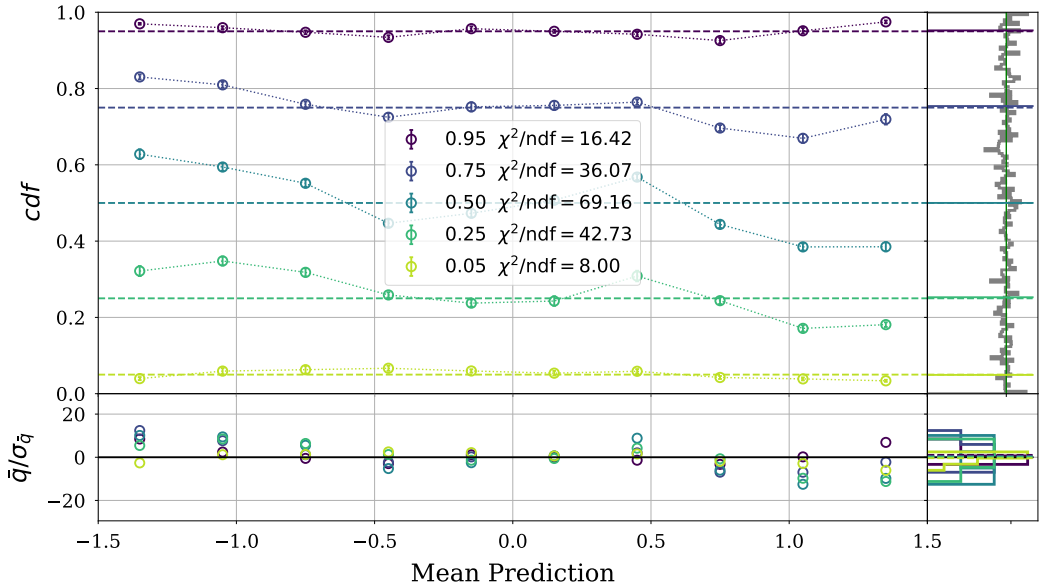
To simplify the above equation, we set $\mathcal{Y}$ to have one value, (or in terms of practice, the unrealistic but ideal scenario in which we have infinitesimally tiny buckets):

$$\begin{aligned} V &= \mathbb{E}_{x\sim p_X}\left[\mathbb{E}_{y\sim p_{Y|X}}\left[y-\mathbb{E}_{y\sim p_{Y|X}}[y]\middle|y=y_0\right]\right] \\ &= \int dx\, p(x)\left(y_0-\mathbb{E}_{y\sim p_{Y|X}}[y]\right) \end{aligned}$$




If the dot is below the line, the model on average overestimates; if the dot is above the line, the model on average underestimates.

Figure 10: CDF performance on "naval-propulsion-plant" across 20 random seeds after recalibrating using an unconditional flow.




If the dot is below the line, the model on average overestimates; if the dot is above the line, the model on average underestimates.

Figure 11: CDF performance on "naval-propulsion-plant" across 20 random seeds after recalibrating using a flow conditional on the mean predicted by the original model.

For a simple case, let $p(X)$ be a one-dimensional Bernoulli distribution with $p = 0.3$, and let $p(y|x = 1)$ be a Bernoulli with $p = 0.5$ and $p(y|x = 0)$ be a Bernoulli with $p = 1.0$. Then, the above formula applied to $y_0 = 1$ gives:

$$
\begin{aligned}
V &= y_0 - (p(x = 1)\ E[y|x = 1] + p(x = 0)\ E[y|x = 0]) \\
&= 1 - (0.3 * 0.5 + 0.7 * 1.0) \\
&\neq 0
\end{aligned}
$$

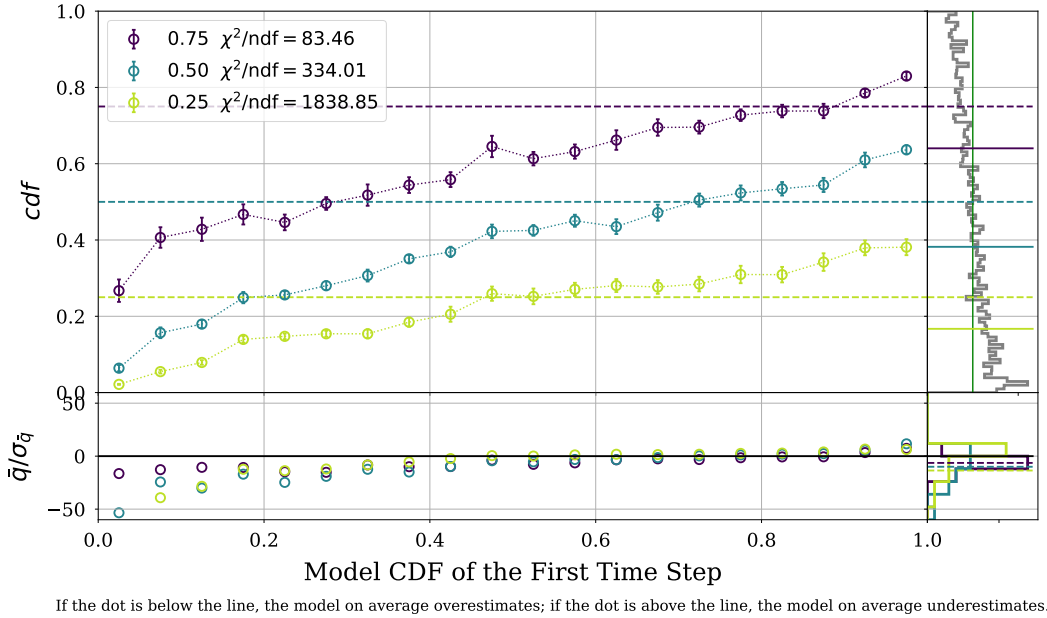



Figure 12: CDF performance of the second time step on "electricity" after recalibrating using isotonic regression on each time step. Calibration error for the second time step is $0.82$.

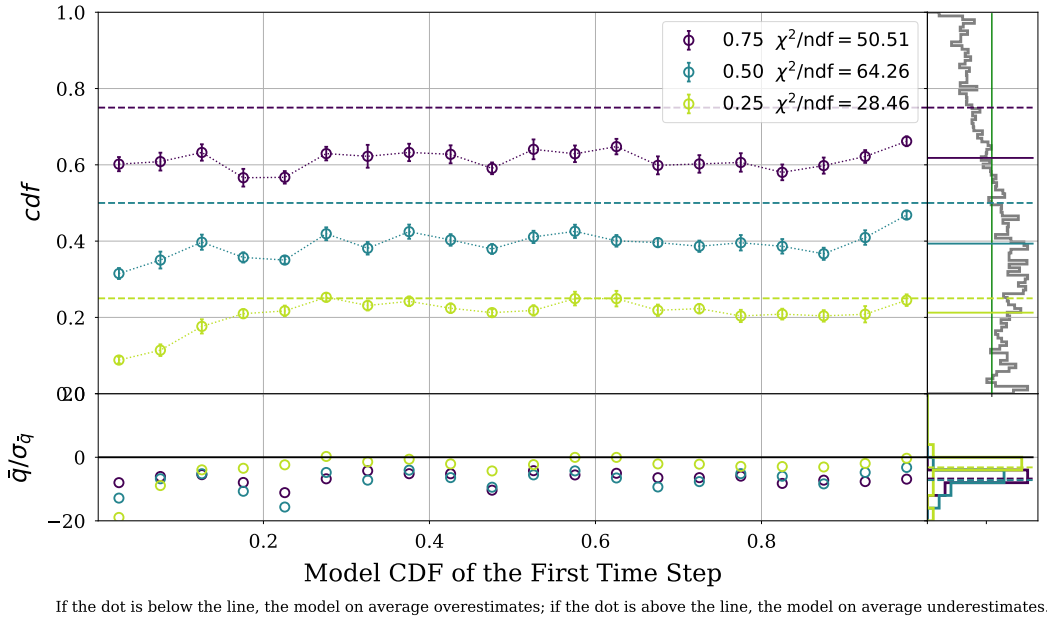
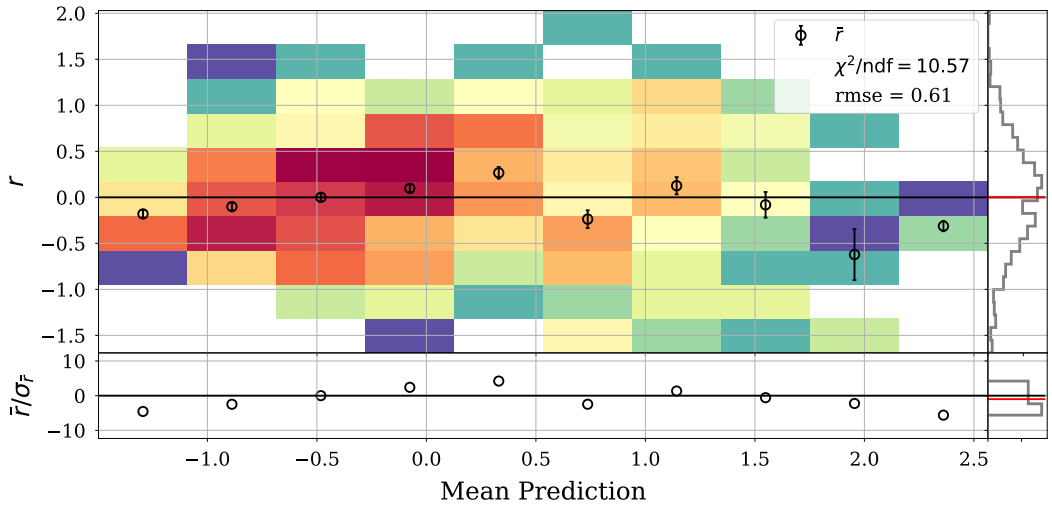



Figure 13: CDF performance of the second time step on "electricity" after recalibrating using a flow trained jointly on the CDFs of the two time steps. Calibration error for the second time step is $0.77$.

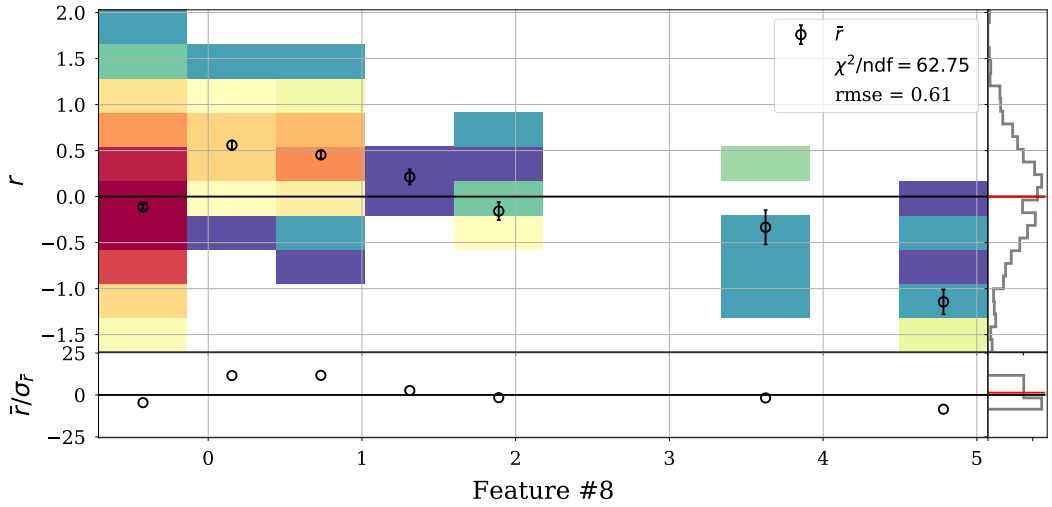
Thus, this value is not in general 0, or in other words, if we were to plot predictions versus observations, we should not expect the mean in the vertical buckets to align with a $y = x$ line.

### E.2 CDF Performance

Given some scalar function on the features $g(X)$ (for example, mean prediction) is in some set of values $\mathcal{V}$, the probability that the CDF of the observations with respect to the conditional distribution is less than some value $q$ is $q$ itself.




(a) Mean performance plot with mean prediction on the $x$-axis




(b) Mean performance plot with a feature on the $x$-axis

Figure 14: Mean performance on "concrete" with Bayesian Ridge Regression

For a mixture with one component, the above is trivially true by definition of the CDF. From this, we know the distribution of CDFs is uniform. We can interpret the situation in the previous paragraph as samples from a mixture of the conditional CDFs, or in other words, a mixture of [0,1] Uniform distributions. Since this mixture is still a [0, 1] Uniform distribution, the above claim is true.

Say we wanted to condition on $g(X) \in \mathcal{V}$ and compare specific quantile values (e.g. the median), we would need to compare the $q$-th quantile of observations where $g(X) \in \mathcal{V}$ and the $q$-th quantile of the mixture distribution.

## F Example Uses of Mean and CDF Performance Plots

In Figure 14a, we can see with Bayesian Linear Regression on the dataset "concrete", the mean performance is strong relative to the prediction. Though there is some systematic bias where the model consistently overestimates for large absolute predictions, the pulls are not statistically significant ($< 10$). However, if we change the x-axis to Feature 8 (Figure 14b), the pulls are now statistically significant. In other words, the model is biased with respect to Feature 8 in a statistically significant way.

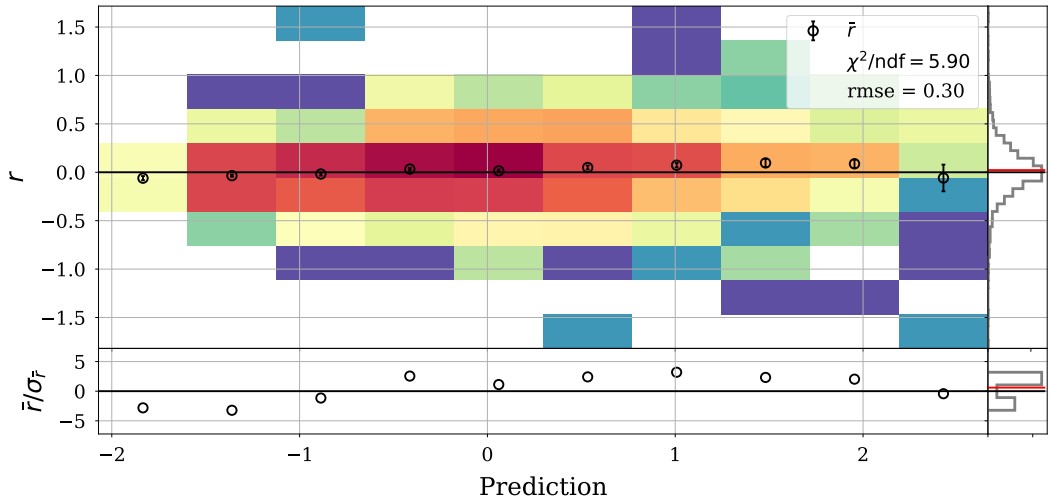
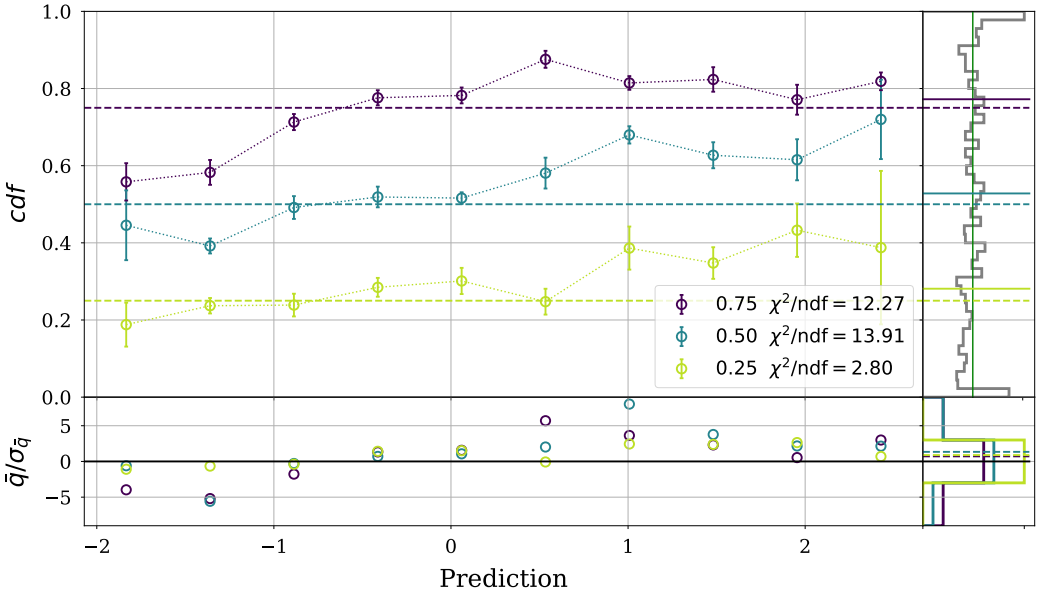



Figure 15: Mean performance on "concrete" with a Conditional Gaussian model.




If the dot is below the line, the model on average overestimates; if the dot is above the line, the model on average underestimates.

Figure 16: CDF performance on "concrete" with a Conditional Gaussian model.

In Figure 15 we can see, whereas Conditional Gaussian performs well in terms of the mean on "naval-propulsion-plant", the CDF performance is not particularly strong (Figure 16). This is expected as mean performance and CDF performance are different tasks where good performance in one does not imply good performance in the other.

## G CODE

For ease of use, we have included the Python code to generate the mean performance and CDF performance plots. The code is split into three parts: utility functions (Appendix G.1), code for the mean performance plot (Appendix G.2) and code for the CDF performance plot (Appendix G.3).

### G.1 UTILS

```
from collections import Counter, OrderedDict
import numpy as np
import scipy
import scipy.stats
from matplotlib import pyplot as plt, cm, ticker, rc, gridspec, colors, patches


def hist2d(ax, x, y, xlim=None, ylim=None, bins=50, norm=True):
    counts, xedges, yedges = np.histogram2d(
        x, y, bins=bins, range=(xlim, ylim)
    )

    xcenters = (xedges[:-1] + xedges[1:]) / 2
    ycenters = (yedges[:-1] + yedges[1:]) / 2

    if norm is True:
        slice_counts = np.sum(counts.T, 0)
        counts = counts.T / slice_counts
    else:
        counts = counts.T

    img = ax.pcolor(
        xedges,
        yedges,
        np.nan_to_num(counts),
        norm=colors.LogNorm(),
        cmap=plt.get_cmap("Spectral_r"),
    )
    return counts


def plot_bin_mean(
    ax,
    x,
    y,
    bins=50,
    xlim=None,
    color="black",
    label="",
    do_plot=True,
    mean_unc=False,
    statistic=None,
    err_func=None,
):

    if statistic is None:
        statistic = "mean"

    x_window = (x >= xlim[0]) & (x <= xlim[1])
    mean, bin_edges, binnumber = scipy.stats.binned_statistic(
        x[x_window], y[x_window], bins=bins, statistic=statistic,
    )

    if err_func is not None:
        pass
    elif mean_unc:
        err_func = lambda arr: np.std(arr, ddof=1) / np.sqrt(len(arr))
    else:
        err_func = lambda arr: np.std(arr, ddof=1)
    err, bin_edges, binnumber = scipy.stats.binned_statistic(
        x[x_window], y[x_window], bins=bins, statistic=err_func
    )
    binnumber[binnumber > len(err)] = len(err)
```

```
# for sanity. Sometimes we get a bin for NaN, because of invalid values (extremely large v
# err_counts = np.array(list(Counter(binnumber).values()))

if isinstance(bins, int):
    bins_x = bins
else:
    bins_x = len(bins) - 1
c_binnumber = Counter(binnumber)

# add guard to fix bug where binnumber exceeds bins even after
# min and max were removed
c_binnumber = {
    i: c_binnumber[i] for i in c_binnumber if i in list(range(1, bins_x + 1))
}
missing_binnumber = list(set(range(1, bins_x + 1)) - set(c_binnumber.keys()))
if len(missing_binnumber) > 0:
    for mb in missing_binnumber:
        c_binnumber.update({mb: 1})
    c_binnumber = OrderedDict(c_binnumber)
err_counts = np.array(list(c_binnumber.values()))
centers = (bin_edges[:-1] + bin_edges[1:]) / 2

if do_plot:
    _, caps, _ = ax.errorbar(
        centers,
        mean,
        yerr=err,
        color=color,
        markeredgecolor=color,
        label=label,
        fmt="o",
        markerfacecolor="none",
        markersize=10,
        markeredgewidth=2,
        capsize=2,
        elinewidth=2,
    )
    for cap in caps:
        cap.set_markeredgewidth(2)

return [centers, mean, err, err_counts]
```

### G.2 MEAN PERFORMANCE

```
import numpy as np
import scipy
import scipy.stats
from matplotlib import pyplot as plt, gridspec, patches


def model_mean_performance(
    pred,
    obs,
    feature=None,
    xlim=None,
    ylim=None,
    ylim_b=None,
    bins=None,
    bins_b=None,
    xlabel="Prediction",
    legend_loc="best",
    title="Model Mean Performance",
    norm=True,
):
    x = pred if feature is None else feature
    y = obs
    resid = obs - pred
    rmse = np.sqrt(np.mean(np.square(resid)))

    xlim = (np.min(x), np.max(x)) if xlim is None else xlim
    ylim = (np.min(y), np.max(y)) if ylim is None else ylim
    bins = int(np.sqrt(len(x)) / 10) if bins is None else bins

    fig = plt.figure(figsize=(18, 10))
    gs = gridspec.GridSpec(3, 2, height_ratios=[4, 1, 1], width_ratios=[15, 1])

    # Heatmap
    ax0 = plt.subplot(gs[0])
    img = hist2d(
        ax0, x, resid, xlim=xlim, ylim=ylim, bins=bins, norm=norm
    )

    centers, mean, mean_unc, err_counts = plot_bin_mean(
        ax0,
        x,
        resid,
        bins=bins,
        xlim=xlim,
        color="black",
        label=r"$\bar r$",
        mean_unc=True,
        do_plot=True,
        statistic="mean",
    )

    plt.setp(ax0.get_xticklabels(), visible=False)
    xlim = ax0.get_xlim()
    ylim = ax0.get_ylim()
    ax0.plot([xlim[0], xlim[1]], [0, 0], lw=2, color="black")

    # Pulls along the bottom
    ax1 = plt.subplot(gs[2], sharex=ax0)

    pulls = mean / mean_unc
    ax1.scatter(centers, pulls, color="black", facecolor="none", s=96, lw=2)
    ax1_xlim = ax1.get_xlim()
    ax1.plot([ax1_xlim[0], ax1_xlim[1]], [0, 0], lw=2, color="black")
```

```
pull_ymax = 2.0 * np.max(np.abs(ax1.get_ylim()))
ax1.set_ylim(-pull_ymax, pull_ymax)

# Pull marginal on the right
ax2 = plt.subplot(gs[3], sharey=ax1)
pulls = pulls[~np.isnan(pulls)]
ax2.hist(
    pulls,
    bins=int(np.sqrt(len(mean))) if bins_b is None else bins_b,
    lw=3,
    range=ylim_b,
    orientation="horizontal",
    histtype="step",
    color="grey",
)

# set limit for bottom
if ylim_b is None:
    ax2.set_ylim(-pull_ymax, pull_ymax)
else:
    ax2.set_ylim(ylim_b[0], ylim_b[1])

ax2_xlim = ax2.get_xlim()
ax2.plot([ax2_xlim[0], ax2_xlim[1]], [0, 0], lw=2, color="black")
ax2.plot(
    [ax2_xlim[0], ax2_xlim[1]], [np.mean(pulls), np.mean(pulls)], lw=2, color="red"
)
plt.setp(ax2.get_yticklabels(), visible=False)
plt.setp(ax2.get_xticklabels(), visible=False)
mean_reduced_chi2 = np.sum(pulls ** 2) / (len(pulls) - 1)

# Residual marginal on the right
ax3 = plt.subplot(gs[1], sharey=ax0)
ax3.hist(
    resid,
    bins=int(np.sqrt(len(y))),
    lw=3,
    range=ylim,
    orientation="horizontal",
    histtype="step",
    color="grey",
)
ax3.set_ylim(ylim)
ax3_xlim = ax3.get_xlim()
ax3.plot([ax3_xlim[0], ax3_xlim[1]], [0, 0], lw=2, color="black")
ax3.plot(
    [ax3_xlim[0], ax3_xlim[1]], [np.mean(resid), np.mean(resid)], lw=2, color="red"
)
plt.setp(ax3.get_yticklabels(), visible=False)
plt.setp(ax3.get_xticklabels(), visible=False)

# Legends
handles, labels = ax0.get_legend_handles_labels()
dummy = patches.Rectangle((0, 0), 1, 1, fc="w", fill=False, edgecolor="none", linewidth=0)
handles.append(dummy)
labels.append(r"$\chi^2 / \mathrm{ndf} = %.2f$" % mean_reduced_chi2)
handles.append(dummy)
labels.append("rmse = {:.2f}".format(rmse))
ax0.legend(handles, labels, loc=legend_loc)

# Align the plots properly with limits from the main plots
ax0.set_xlim(xlim)
ax1.set_xlim(xlim)
# Set up the grids
ax0.grid(True)
```

```
    ax1.grid(True)
    ax2.yaxis.grid(True)
    ax3.yaxis.grid(True)
    # Informative labels
    ax1.set_xlabel(xlabel)
    ax0.set_ylabel(r"$r$")
    ax1.set_ylabel(r"$\bar r / \sigma_{\bar r}$")
    if title is not None:
        ax0.set_title(title)
    # Remove space between plots
    plt.subplots_adjust(hspace=0.0, wspace=0.0)
    return fig, ax0, ax1, ax2, ax3
```

### G.3 CDF PERFORMANCE

```
import numpy as np
import scipy
import scipy.stats
from matplotlib import pyplot as plt, gridspec, patches


def model_cdf_performance(
    feature,
    cdf,
    xlim=None,
    ylim=(0, 1),
    ylim_b=None,
    bins_b=None,
    bins=None,
    xlabel="Prediction",
    legend_loc="best",
    title="Model CDF Performance",
    percentiles=[0.5],
):
    assert np.min(cdf) >= 0, "cdf cannot be negative"
    assert np.max(cdf) <= 1, "cdf cannot be larger than 1"

    x = feature
    resid = cdf

    xlim, ylim = get_lims2d(x, resid, xlim=xlim, ylim=ylim)

    if bins is None:
        bins = int(np.sqrt(len(x)) / 10)

    fig = plt.figure(figsize=(18, 10))
    gs = gridspec.GridSpec(2, 2, height_ratios=[6, 2], width_ratios=[15, 2])

    ax0 = plt.subplot(gs[0])
    ax1 = plt.subplot(gs[2], sharex=ax0)
    ax2 = plt.subplot(gs[3], sharey=ax1)
    ax3 = plt.subplot(gs[1], sharey=ax0)

    # Residual marginal on the right
    ax3.hist(
        resid,
        bins=int(np.sqrt(len(resid))),
        lw=3,
        range=ylim,
        orientation="horizontal",
        histtype="step",
        color="grey",
        density=True,
    )

    color_cycle = plt.cm.viridis(np.linspace(0, 0.9, len(percentiles)))
    pull_means = []
    legends = []
    for i, p in enumerate(np.sort(percentiles)[::-1]):
        # Dry-run to get mean_reduced_chi2
        centers, means, errs, err_counts = plot_bin_mean(
            ax0,
            x,
            resid,
            bins,
            xlim=xlim,
            mean_unc=True,
            statistic=lambda x: np.quantile(x, p),
```

```
            do_plot=True,
            err_func=lambda x: scipy.stats.mstats.mjci(
                x,
                prob=p,
            ),
            color=color_cycle[i],
        )

        ax0.set_ylim(*ylim)
        ax0.set_xlim(*xlim)
        ax0.plot(
            centers[~np.isnan(means)],
            means[~np.isnan(means)],
            c=color_cycle[i],
            linestyle="dotted",
        )

        # Pulls along the bottom
        pulls = (means - p) / errs
        pulls = pulls[~np.isnan(pulls)]
        mean_reduced_chi2 = np.sum(pulls ** 2) / (len(pulls) - 1)
        legends.append(
            r"$%.2f$  $\chi^2 / \mathrm{ndf} = %.2f$" % (p, mean_reduced_chi2)
        )
        ax1.scatter(centers, pulls, color=color_cycle[i], facecolor="none", s=96, lw=2)

        pulls = pulls[~np.isnan(pulls)]
        ax2.hist(
            pulls,
            bins=int(np.sqrt(len(means))) if bins_b is None else bins_b,
            lw=3,
            orientation="horizontal",
            histtype="step",
            color=color_cycle[i],
            range=ylim_b,
        )

        plt.setp(ax2.get_yticklabels(), visible=False)
        plt.setp(ax2.get_xticklabels(), visible=False)

        ax0.plot(
            [xlim[0], xlim[1]],
            [p, p],
            lw=2,
            color=color_cycle[i],
            linestyle="dashed",
        )
        pull_means.append(np.mean(pulls))

    ax0.legend(legends)
    ax1_xlim = ax1.get_xlim()
    ax1.plot([ax1_xlim[0], ax1_xlim[1]], [0, 0], lw=2, color="black")

    pull_ymax = 2.0 * np.max(np.abs(ax1.get_ylim()))
    ax1.set_ylim(-pull_ymax, pull_ymax)

    # set limit for bottom
    if ylim_b is None:
        ax2.set_ylim(-pull_ymax, pull_ymax)
    else:
        ax2.set_ylim(ylim_b[0], ylim_b[1])

    ax2_xlim = ax2.get_xlim()
    ax2.plot([ax2_xlim[0], ax2_xlim[1]], [0, 0], lw=2, color="black")
    for i, pull_mean in enumerate(pull_means):
```

```
        ax2.plot(
            [ax2_xlim[0], ax2_xlim[1]],
            [pull_mean, pull_mean],
            lw=2,
            color=color_cycle[i],
            linestyle="dashed",
        )

    xlim = ax0.get_xlim()
    ylim = ax0.get_ylim()

    ax3.plot(
        np.ones_like(np.linspace(np.min(resid), np.max(resid), 1000)),
        np.linspace(np.min(resid), np.max(resid), 1000),
        color="green",
    )

    ax3.set_ylim(ylim)
    ax3_xlim = ax3.get_xlim()
    for i, p in enumerate(np.sort(percentiles)[::-1]):
        ax3.plot(
            [ax3_xlim[0], ax3_xlim[1]],
            [p, p],
            lw=2,
            color=color_cycle[i],
            linestyle="dashed",
        )
        ax3.plot(
            [ax3_xlim[0], ax3_xlim[1]],
            [np.quantile(resid, p), np.quantile(resid, p)],
            lw=2,
            color=color_cycle[i],
        )

    plt.setp(ax0.get_xticklabels(), visible=False)
    plt.setp(ax3.get_yticklabels(), visible=False)
    plt.setp(ax3.get_xticklabels(), visible=False)

    # Align the plots properly with limits from the main plots
    ax0.set_xlim(xlim)
    ax1.set_xlim(xlim)

    # Set up the grids
    ax0.grid(True)
    ax1.grid(True)
    ax2.yaxis.grid(True)
    ax3.yaxis.grid(True)

    # Informative labels
    ax1.set_xlabel(xlabel)
    ax0.set_ylabel(r"$cdf$")
    ax1.set_ylabel(r"$\bar{q} / \sigma_{\bar q}$")

    if title is not None:
        ax0.set_title(title)

    # Remove space between plots
    plt.subplots_adjust(hspace=0.0, wspace=0.0)

    plt.figtext(
        0.5,
        -0.01,
        "If the dot is below the line, the model on average overestimates; "
        "if the dot is above the line, the model on average underestimates.",
        horizontalalignment="center",
```

```
        fontsize=16,
    )

    return fig, ax0, ax1, ax2, ax3
```