HALO: IMPROVING FORECAST ACCURACY THROUGH HETEROSCEDASTIC ESTIMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Heteroscedastic forecasting, where a network estimates a scale parameter alongside a location parameter, is normally motivated by uncertainty quantification. This paper shows it also improves the point estimate, in contrast to reported negative results for heteroscedastic estimation outside time series. Halo is a modification that reuses an existing deep forecaster's architecture, giving it a second output for the scale of its implied distribution and training it under the matching negative log likelihood. Adapting three state-of-the-art models — a transformer, a graph network paired with a variational autoencoder, and a single-layer convolutional network — under both Gaussian and Laplacian losses demonstrates the phenomenon. On the five electricity price markets of a standard forecasting benchmark, Halo improves MSE and MAE in 28 of 30 model-market-metric comparisons, cutting average MSE by 2.6% to 16.5% and average MAE by 1.7% to 11.0%. Two findings emerge: (1) whether the scale estimate comes from a second projection head or from a full parallel network matters far less than whether the network estimates scale, and (2) the improvement holds under the hyperparameters already tuned for the point-estimate baseline, so retuning is optional.

1 INTRODUCTION

Time series forecasting with deep neural networks (DNNs) poses some unique challenges relative to other DNN problems, like computer vision and natural language processing. On the surface, it shares similarities with language models, where the input is a sequence of words (represented as tokens) and the output is the next sequence of words that follows the input. In time series, the input is a sequence of data points, and the output is the next sequence of data points. Unlike language modeling, however, the data points themselves carry no semantic meaning the way words do (Nie et al., 2022). For many real-world time series, the past values of the time series itself are less informative about the future values than past values of related (exogenous) time series (Granger, 1969; Box et al., 2015). For example, in forecasting stock price returns, past values of the stock price tend to have less influence on future returns than do exogenous time series like interest rates and company fundamentals (Chincarini & Kim, 2022). There are other challenges as well: since time series are rarely stationary, the older data a model trains on may have a different distribution than the newer data it runs inference on (Kim et al., 2021; Liu et al., 2022). Additionally, model training has to avoid leakage, where data from the test or validation set influences the trained model and gives false confidence in its forecasting ability. While obvious forms of leakage are easy to stop, more subtle forms can go unnoticed and skew results, like an exogenous time series sampled in the future, relative to the endogenous time series being predicted (Hewamalage et al., 2023).

In this paper, we introduce a technique for modifying an existing time-series deep learning architecture to improve accuracy. The basic idea is to use heteroscedastic forecasting, where the forecaster learns scale parameters for the output, in addition to location parameters. Making heteroscedastic forecasts is useful by itself, since it helps quantify the uncertainty of a forecast. We make a stronger claim, which is that learning the scale parameter helps the forecaster improve its location estimate. We call this technique *Halo*, to emphasize that finding the halo (distribution) around the point estimate makes it easier for the network to locate the point estimate itself. This paper explores the subtleties of the Halo method, like how to architect the location and scale estimates, and how heteroscedastic estimation interacts with nonstationarity adjustments. The result is a technique that boosts performance on several state-of-the-art (SOTA) models.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2 Related Work

Neural networks have been used for time series forecasting since at least the 1980s (Lapedes & Farber, 1987; Werbos, 1988). In the 2010s, deep neural networks became popular, with recurrent neural network (RNN) approaches like DA-RNN (Qin et al., 2017), LSTNet (Lai et al., 2018), and ES-RNN (Smyl, 2020). TCN (Borovykh et al., 2017) showed that convolutional neural networks (CNNs) could outperform RNNs for forecasting tasks, and N-BEATS (Oreshkin et al., 2019) showed how to build effective DNNs without using RNNs or CNNs. Like other areas of deep learning, transformers (Vaswani et al., 2017) have become popular in time series forecasting. Some early transformer models like FEDFormer (Zhou et al., 2022), Autoformer (Wu et al., 2021), and Informer (Zhou et al., 2021) were shown to be less effective than simple linear models for forecasting tasks (Zeng et al., 2023). PatchTST (Nie et al., 2022) showed that tokenizing patches of time series (subsequences), rather than tokenizing individual data points in a series, led to more meaningful tokens, and effectively unlocked transformers as a valuable tool for time series forecasting. TimeXer (Wang et al., 2024) effectively combined patching with exogenous time series for more general-purpose time series forecasting. GCGNet (Li et al., 2026) was able to combine patching, a graph neural network (GNN), and a variational autoencoder (VAE) to outperform TimeXer. By default, GCGNet has access to future exogenous inputs, exactly the type of leakage that Hewamalage et al. (2023) argues against. Nonetheless, the authors showed that even when the future exogenous inputs are not available, it can outperform TimeXer, though its performance is degraded. Zhou et al. (2025) created a competitive model, CrossLinear, from patching and a single-layer CNN alone.

Heteroscedastic estimation in neural networks dates back to the 1990s (Nix & Weigend, 1994). While point estimates are most common in the time-series deep learning literature, DeepAR (Salinas et al., 2020) does take a probabilistic modeling approach. Interestingly, Stirn et al. (2023) showed a result that on the surface seems to contradict the main claim of this paper: on four neural network models, heteroscedastic estimation led to worse point estimation. None of these models were time series forecasters, and all were much smaller in scale than the models considered in this paper. This suggests potential future work identifying exactly when heteroscedastic estimation does and does not improve point estimates, since it does not seem to be a universal result in neural networks.

3 Halo

The basic forecasting problem this paper works through is as follows. A network takes inputs $(x_{endo}, x_{exo}) \in \mathbb{R}^T \times \mathbb{R}^{T \times C}$ and generates a forecast $\hat{y} \in \mathbb{R}^H$. x_{endo} represents the last T data points for the *endogenous signal* to be forecast. $\hat{y}$ is an estimate of $y \in \mathbb{R}^H$, the next H data points in the signal. Letting s represent the underlying endogenous signal of interest, at time t, $(x_{endo}(1), \ldots, x_{endo}(T)) = (s(t-T+1), \ldots, s(t))$ and $(y(1), \ldots, y(H)) = (s(t+1), \ldots, s(t+H))$. T is the *lookback window* of the forecast, and H is the *forecast horizon*. x_{exo} represents the last T data points of C *exogenous inputs*, signals related to the endogenous signal, which can help predict its future values. C is the number of *exogenous channels*.

The network is a function $f : \mathbb{R}^T \times \mathbb{R}^{T \times C} \times \Phi \to O$, where Φ is the space of network parameters and O is the output space. The network learns its parameters during training and holds them constant during inference. For point estimates, $O = \mathbb{R}^H$, and $\hat{y} = f(x_{endo}, x_{exo}, \phi)$. For heteroscedastic estimates, $O = \mathbb{R}^H \times (0, \infty)^H$. If the distribution parameters for location and scale are a and b respectively, write $(\hat{a}, \hat{b}) = f(x_{endo}, x_{exo}, \phi)$. For example, for a Gaussian distribution with mean μ and variance σ^2, $(\hat{\mu}, \hat{\sigma}^2) = f(x_{endo}, x_{exo}, \phi)$.

Simplifying assumptions:

- Assume the endogenous signal is a single channel. This is for clarity, and not a fundamental limitation of heteroscedastic estimation.
- Assume each exogenous signal has the same lookback window as the endogenous signal. Again, this is to keep exposition simple, and not a fundamental limit of the approach.
- Assume future exogenous inputs are not visible to the forecaster. Some recent architectures like GCGNet (Li et al., 2026) and DAG (Qiu et al., 2025) have let the forecaster see future exogenous inputs, under the view that some exogenous inputs can have highly accurate forecasts. This paper avoids that, in the interest of limiting leakage (Hewamalage et al.,

2023), since in general future exogenous inputs may carry information about yet-to-be-predicted endogenous values. Also, a forecaster can treat predictions of future values made at time t, about time $t + h$, as exogenous values at time t, so this is not a fundamental limitation.

- Assume the distribution used for heteroscedastic loss is parameterized by a location and scale parameter a and b. That is the case for the examples considered in this paper, but not a requirement for heteroscedastic loss. For example, a Weibull distribution has three parameters and an exponential distribution has only a single parameter.

3.1 HETEROSCEDASTIC LOSS

This subsection explores several loss functions of the form $L\left((y_1, \ldots, y_N), (\hat{y}_1, \ldots, \hat{y}_N)\right)$, where each y_i and $\hat{y}_i$ are members of $\mathbb{R}^H$. That represents a loss over N samples and a forecast horizon H. To keep the exposition clear, a loss function written like:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{1}$$

is really shorthand for

$$\frac{1}{H}\sum_{t=1}^{H}\left[\frac{1}{N}\sum_{i=1}^{N}(y_i(t) - \hat{y}_i(t))^2\right]. \tag{2}$$

That is to say that in practice, the loss is averaged across the forecast horizon, but to keep equations short and readable, they are written in this subsection as if the forecast horizon was 1, and there was only a single output value to consider.

The two most common loss functions in deep learning time-series literature are mean squared error (MSE) and mean absolute error (MAE). These are defined by

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{3}$$

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|. \tag{4}$$

Using either of these as loss functions for training has a probabilistic interpretation. Minimizing either loss function corresponds to minimizing the negative log likelihood (NLL) function for some assumed distribution $P(Y \mid X)$, where Y is the output random variable and X is the input random variable. More specifically, given learned model parameters ϕ, write x_i for the input pair (x_{endo}, x_{exo}) of sample i, and treat the estimate $\hat{y}_i = f(x_i, \phi)$ of y_i as the location parameter for some implied distribution function. Here f is the network function.

If this distribution is Gaussian $\mathcal{N}(\mu, \sigma^2)$, with some mean μ that depends on the input, and some constant, but unknown, variance σ^2, then the NLL given N data points is

$$\text{NLL} = \frac{N}{2}\log(2\pi) + \frac{1}{2}\sum_{i=1}^{N}\log(\sigma^2) + \sum_{i=1}^{N}\frac{(y_i - \mu_i)^2}{2\sigma^2}. \tag{5}$$

In this case, with $\hat{\mu}_i = \hat{y}_i = f(x_i, \phi)$ and σ^2 constant, the ϕ that minimizes the MSE is the ϕ that minimizes the NLL.

Similarly, if the distribution is Laplacian $\mathcal{L}(\mu, b)$ with location μ and constant, but unknown, scale b, then minimizing the MAE is equivalent to minimizing the NLL

$$\text{NLL} = N\log(2) + \sum_{i=1}^{N}\log(b) + \sum_{i=1}^{N}\frac{|y_i - \mu_i|}{b}. \tag{6}$$

Removing the assumption that the variance/scale parameter is constant, and modifying the network so that instead of estimating $\hat{\mu}_i = f(x_i, \phi)$ it estimates both location and scale, like $(\hat{\mu}_i, \hat{\sigma}_i^2) = f(x_i, \phi)$ in the Gaussian case, gives a heteroscedastic model. Heteroscedastic models can be useful in their own right. For instance, a heteroscedastic model can generate probabilistic forecasts, giving ranges of possible values rather than a single one. This paper provides evidence that heteroscedastic estimates are useful even when only the point estimate matters, by helping the network make better point estimates of the location parameter. Empirically, when the network learns both location and scale, it adjusts its point estimate to account for variability. That is, the network learns to account for variance.

Note that for the Gaussian distribution, this paper uses the β-NLL function introduced by Seitzer et al. (2022) to find the mean and variance parameters:

$$\beta\text{-NLL} = \frac{1}{2N}\sum_{i=1}^{N}\text{sg}\left(|\sigma_i|\right)\log(\sigma_i^2) + \frac{1}{2N}\sum_{i=1}^{N}\text{sg}\left(|\sigma_i|\right)\frac{(y_i - \mu_i)^2}{\sigma_i^2}. \tag{7}$$

Here $\text{sg}(\cdot)$ is the stop-gradient operator. It acts as an identity function in forward-pass operations, but stops gradients from flowing in backward-pass operations. The stop gradient ensures that minimizing this function is equivalent to minimizing the NLL, and Seitzer et al. (2022) showed that it is more stable during backpropagation updates than the traditional Gaussian NLL. It is equivalent because it has the same optimal point, even though it has different gradients. There, the stop-gradient term carries an exponent β, which the authors recommend setting to 0.5. No β term appears in the equation because it is already fixed at 0.5, which is what leaves $\text{sg}(|\sigma_i|)$ in its place. No such modification is needed for the Laplacian distribution.

One final note: choosing the loss function to minimize is ultimately making a choice about the assumed distribution $P(Y \mid X)$. The right choice depends on the underlying data. If the right choice is unclear, the loss function itself can be treated as a hyperparameter. In this case, the NLL function applied to the validation set can identify which distribution better fits the data. It needs to be the full NLL function, not a function that produces the same minimum, like MSE or β-NLL in place of the Gaussian NLL in Equation 5. Otherwise, the comparison risks a scale mismatch across distributions.

3.2 From point estimates to heteroscedastic estimates

One way to go from point estimates to heteroscedastic estimates is to run two copies of the same basic network architecture in parallel, one for the location parameter and one for the scale parameter. See Figure 1. Note that the softplus : $\mathbb{R}^H \to (0, \infty)^H$ operator ensures all scale estimates are positive, with $\text{softplus}(c) = \ln(1 + e^c)$. While this is viable, it doubles the network size, and can waste network parameters effectively relearning the same representation twice.

A common architecture feeds x_{endo} and x_{exo} through embedding functions, then transforms them into an internal representation with multiple layers of the core architecture, and finally applies a projection to map the representation onto outputs. In this case, creating two projections is often sufficient to achieve an effective heteroscedastic model using a single learned representation of the inputs and distinct learned projection functions. See Figure 2. In practice, both architectures generally outperform the non-heteroscedastic architecture. Which of the two makes better estimates is close to even, so if efficiency is a constraint, the dual-head architecture is safe, and if not, hyperparameter tuning can test both architectures as variants.

3.3 Nonstationarity adjustments

Liu et al. (2022) introduced Series Standardization as a mechanism to mitigate some of the effects of nonstationarity. The idea is to "standardize" the input x_{endo} with:

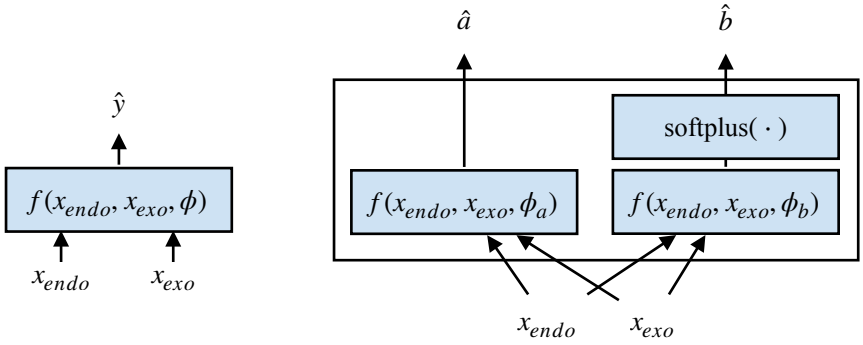

Figure 1: Running the point-estimate network on the left twice in parallel produces the heteroscedastic network on the right.

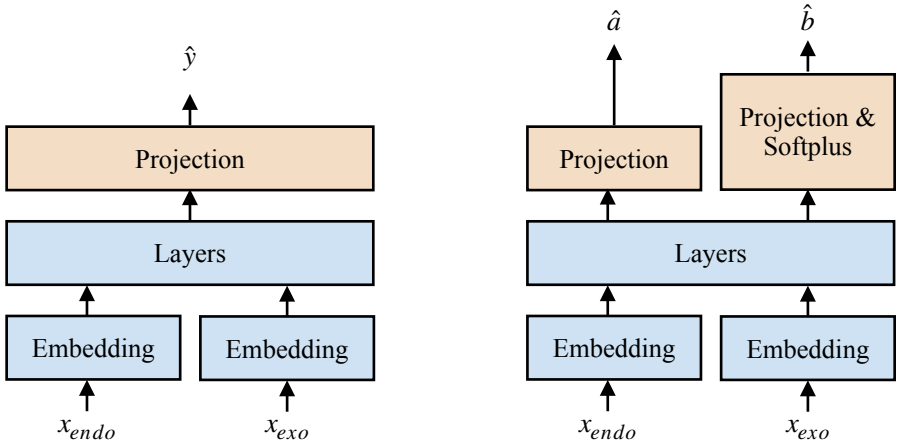

Figure 2: Giving the point-estimate network on the left a second projection head produces the heteroscedastic network on the right.

$$m = \frac{1}{T}\sum_{t=1}^{T} x_{endo}(t) \tag{8}$$

$$v = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(x_{endo}(t) - m)^2} \tag{9}$$

$$x'_{endo} = \frac{1}{v}(x_{endo} - m) \tag{10}$$

The network then takes x'_{endo}, and a final step transforms its output $\hat{y}'$ back with:

$$\hat{y} = \hat{y}'v + m \tag{11}$$

All the models in this paper use Series Standardization. This conflicts with the Halo method as described above, because the input standard deviation v has to adjust the scale estimate too. Thus, if $\hat{a}'$ is the location estimate the heteroscedastic network emits, and $\hat{b}'$ is the scale parameter after softplus, then the final outputs are:

$$\hat{a} = \hat{a}'v + m \tag{12}$$

$$\hat{b} = \hat{b}'v \tag{13}$$

The experiments below use these adjusted outputs. The models also apply a nonstationarity adjustment to the exogenous inputs, but the specific adjustment varies across the models, and it does not interact with Halo in any meaningful way.

4 EXPERIMENTS

Three SOTA models adapted with Halo demonstrate its ability to boost forecast accuracy: TimeXer (Wang et al., 2024), GCGNet (Li et al., 2026), and CrossLinear (Zhou et al., 2025). TimeXer and GCGNet are both complex, multi-layer models adapted with the dual-head architecture described above, whereas CrossLinear is a single-layer CNN adapted by running two networks in parallel. TimeXer is a transformer-based architecture, while GCGNet uses a GNN combined with a VAE for its architecture. The authors of TimeXer and CrossLinear trained their models with MSE loss, while the GCGNet authors trained their models with MAE loss, so these three examples enable testing the Halo method with both Gaussian and Laplacian distributions.

The datasets for testing are the electricity price forecasting (EPF) benchmarks described in detail by Lago et al. (2021). This is a set of five datasets with different exogenous and endogenous signals targeting electricity prices across different markets: Nord Pool (NP), the Pennsylvania-New Jersey-Maryland interconnection (PJM), and the Belgian (BE), French (FR), and German (DE) markets of EPEX SPOT. The datasets themselves are diverse, because of the different signals and different pricing models they track. More importantly, the original authors of TimeXer, GCGNet, and CrossLinear have already identified a good set of hyperparameters on those datasets, with tuned hyperparameters reused here for the baselines. First, the headline results: as stated above, Halo generally boosts performance across the five markets. See Table 1.

Table 1: Each of the three models is compared side-by-side with and without Halo applied. Halo generally boosts performance, with the only exception being a single market where TimeXer does better without Halo. Here "Avg" represents the average MSE/MAE for each model across all five markets.

	TimeXer + Halo		**TimeXer**		**GCGNet + Halo**		**GCGNet**		**CrossLinear + Halo**		**CrossLinear**	
	MSE	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
NP	**0.2377**	**0.2643**	0.2529	0.2755	**0.2294**	**0.2525**	0.2498	0.2700	**0.2300**	**0.2599**	0.2468	0.2764
PJM	**0.0837**	**0.1726**	0.0853	0.1760	**0.0706**	**0.1618**	0.0753	0.1694	**0.0781**	**0.1721**	0.0940	0.1878
BE	**0.3591**	**0.2425**	0.3890	0.2521	**0.3672**	**0.2516**	0.3912	0.2673	**0.3689**	**0.2519**	0.3770	0.2569
FR	**0.3593**	**0.1903**	0.3784	0.1930	**0.3582**	**0.2006**	0.4273	0.2345	**0.3700**	**0.1883**	0.3804	0.2014
DE	0.4790	0.4256	**0.4527**	**0.4220**	**0.4112**	**0.3981**	0.5765	0.4804	**0.4432**	**0.4179**	0.4495	0.4203
Avg	**0.3037**	**0.2591**	0.3117	0.2637	**0.2873**	**0.2529**	0.3440	0.2843	**0.2981**	**0.2580**	0.3095	0.2686

Note that the hyperparameters the original authors used varied per model and per market. Because there is no reason to assume the same hyperparameters will be optimal when Halo is applied, the Halo-modified models received a hyperparameter search of their own on the validation dataset, which fixed the hyperparameters used on the test datasets above. To see the effect of this, look at Table 2, which includes the test results TimeXer with Halo would have given if it had kept the original authors' configuration instead of retuning. Note that in some cases the newly tuned parameters helped, but in others they did not, though in both the tuned and the untuned cases, Halo generally improved results. Running the hyperparameter search on the final test data would have made at least one of the headline numbers look better: the MSE on the DE market, though this would have been a form of "statistical cheating".

Section 3.2 describes the two different architectures for Halo: a parallel architecture for simple models and a dual-head architecture for more complex models. Hyperparameter tuning tested both architectures. For CrossLinear, on the validation set, the parallel architecture clearly outperformed the dual-head architecture, and that is the variant behind the headline numbers. Table 3 includes test

Table 2: For MSE/MAE, the best numbers are shown in bold, and the second-best numbers are underlined. The left column runs TimeXer + Halo on the hyperparameter configuration TimeXer's authors found. The middle column retunes those hyperparameters for Halo. The right column runs TimeXer itself, on its authors' tuned hyperparameters.

	TimeXer + Halo, untuned		**TimeXer + Halo, tuned**		**TimeXer**	
	MSE	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
NP	**0.2361**	**0.2621**	0.2377	0.2643	0.2529	0.2755
PJM	**0.0786**	**0.1721**	0.0837	0.1726	0.0853	0.1760
BE	0.3745	0.2448	**0.3591**	**0.2425**	0.3890	0.2521
FR	0.3710	0.1921	**0.3593**	**0.1903**	0.3784	0.1930
DE	**0.4514**	0.4243	0.4790	0.4256	0.4527	**0.4220**
Avg	**0.3023**	0.2591	0.3037	**0.2591**	0.3117	0.2637

results with the dual-head architecture on CrossLinear. Here the difference between the two was less decisive than on the validation set.

Table 3: Comparing CrossLinear with parallel Halo, dual-head Halo, and no Halo. For MSE/MAE, the best numbers are shown in bold, and the second-best numbers are underlined.

	CrossLinear + parallel Halo		**CrossLinear + dual-head Halo**		**CrossLinear**	
	MSE	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
NP	0.2300	**0.2599**	**0.2298**	0.2626	0.2468	0.2764
PJM	0.0781	**0.1721**	**0.0780**	0.1741	0.0940	0.1878
BE	**0.3689**	**0.2519**	0.3936	0.2604	0.3770	0.2569
FR	0.3700	**0.1883**	**0.3520**	0.1897	0.3804	0.2014
DE	0.4432	**0.4179**	**0.4417**	0.4220	0.4495	0.4203
Avg	**0.2981**	**0.2580**	0.2990	0.2618	0.3095	0.2686

For the other two models, the difference was less pronounced in hyperparameter tuning, so the less expensive dual-head architecture is the one behind those headline numbers. The same "what-if" scenario tested both Halo architectures on TimeXer. See Table 4.

Table 4: Comparing TimeXer with parallel Halo, dual-head Halo, and no Halo. For MSE/MAE, the best numbers are shown in bold, and the second-best numbers are underlined.

	TimeXer + parallel Halo		**TimeXer + dual-head Halo**		**TimeXer**	
	MSE	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
NP	**0.2288**	**0.2589**	0.2377	0.2643	0.2529	0.2755
PJM	0.0842	0.1726	**0.0837**	**0.1726**	0.0853	0.1760
BE	0.3754	0.2511	**0.3591**	**0.2425**	0.3890	0.2521
FR	0.3597	**0.1863**	**0.3593**	0.1903	0.3784	0.1930
DE	**0.4510**	0.4248	0.4790	0.4256	0.4527	**0.4220**
Avg	**0.2998**	**0.2587**	0.3037	0.2591	0.3117	0.2637

5 CONCLUSION

This paper introduces Halo, which gives an existing DNN forecaster a second output for the scale parameter of its implied distribution and trains it under the matching negative log likelihood. The modification needs no architecture-specific design, and it improves the point estimate. Across three architectures — a transformer, a GNN paired with a VAE, and a single-layer CNN — and five electricity price markets, Halo improved MSE and MAE in 28 of 30 comparisons, and improved every model's average on both metrics. Average MSE fell between 2.6% and 16.5%, and average MAE between 1.7% and 11.0%, depending on the model.

Two practical findings came out of the experiments. First, how the scale estimate attaches to the network matters less than going from point estimates to heteroscedastic estimates in the first place. The two architectures described in Section 3.2 finish within a percent and a half of each other on both models, and in the same direction each time: the parallel architecture is slightly ahead of the dual-head architecture on TimeXer, and slightly ahead again on CrossLinear, but never by enough to choose on accuracy alone. The dual-head architecture costs roughly half the parameters of the parallel one, so that is the one to prefer, with the parallel architecture worth testing as a variant when the budget allows. Second, the hyperparameters tuned for a point-estimate model are often sufficient for its Halo variant. Retuning on the validation set helped in two markets and hurt in three, on both metrics, and left the test averages within half a percent of where they started, so retuning is optional, not a prerequisite for the improvement.

The result sits against Stirn et al. (2023). Both findings can hold at once: electricity prices carry the time-varying volatility that gives a scale estimate something to learn, and the models in that study had no such signal to work from. Testing Halo on other benchmarks and horizons, and across seeds, is the next step.

AI USE STATEMENT

In this work, we used generative AI tools for implementing the models and experiment code, for reviewing that code (in addition to human review), for translating pseudo-LaTeX into compile-safe LaTeX, and for drafting portions of this paper. We have not used generative AI tools to generate data or to produce any reported result; all numbers come from executed training runs.

REPRODUCIBILITY STATEMENT

Section 3 specifies the method, and Appendix A gives the full training setup — optimizer, schedule, epoch budget, early stopping, hardware, and seeding. The EPF datasets are public (Lago et al., 2021). Anonymized code for the models, training loop, and table-generating scripts is included at `https://anonymous.4open.science/r/HaloForecaster-1C3D/`, and it carries the per-market hyperparameters. All results are single runs from a fixed base seed.

REFERENCES

Anastasia Borovykh, Sander Bohte, and Cornelis W. Oosterlee. Conditional time series forecasting with convolutional neural networks. arXiv:1703.04691, 2017.

George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

Ludwig B. Chincarini and Daehwan Kim. *Quantitative Equity Portfolio Management: An Active Approach to Portfolio Construction and Management*. McGraw Hill, 2nd edition, 2022.

Clive W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3):424–438, 1969.

Hansika Hewamalage, Klaus Ackermann, and Christoph Bergmeir. Forecast evaluation for data scientists: common pitfalls and best practices. *Data Mining and Knowledge Discovery*, 37(2): 788–832, 2023.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

Jesus Lago, Grzegorz Marcjasz, Bart De Schutter, and Rafał Weron. Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark. *Applied Energy*, 293:116983, 2021.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2018.

Alan Lapedes and Robert Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, 1987.

Zhengyu Li, Xiangfei Qiu, Yuhan Zhu, Xingjian Wu, Jilin Hu, Chenjuan Guo, and Bin Yang. GCGNet: Graph-consistent generative network for time series forecasting with exogenous variables. In *International Conference on Learning Representations*, 2026.

Yong Liu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Non-stationary transformers: Exploring the stationarity in time series forecasting. In *Advances in Neural Information Processing Systems*, volume 35, pp. 9881–9893, 2022.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. arXiv:2211.14730, 2022.

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.

Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. arXiv:1905.10437, 2019.

Yao Qin, Dongjin Song, Haifeng Cheng, Wei Cheng, Guofei Jiang, and Garrison W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 2627–2633. AAAI Press, 2017.

Xiangfei Qiu, Yuhan Zhu, Zhengyu Li, Xingjian Wu, Bin Yang, and Jilin Hu. DAG: A dual correlation network for time series forecasting with exogenous variables. arXiv:2509.14933, 2025.

David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3): 1181–1191, 2020.

Maximilian Seitzer, Arash Tavakoli, Dimitrije Antic, and Georg Martius. On the pitfalls of heteroscedastic uncertainty estimation with probabilistic neural networks. arXiv:2203.09168, 2022.

Slawek Smyl. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1):75–85, 2020.

Andrew Stirn, Hans-Hermann Wessels, Megan Schertzer, Laura Pereira, Neville E. Sanjana, and David A. Knowles. Faithful heteroscedastic regression with neural networks. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Yuxuan Wang, Haixu Wu, Jiaxiang Dong, Guo Qin, Haoran Zhang, Yong Liu, Yunzhong Qiu, Jianmin Wang, and Mingsheng Long. TimeXer: Empowering transformers for time series forecasting with exogenous variables. In *Advances in Neural Information Processing Systems*, volume 37, pp. 469–498, 2024.

Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems*, volume 34, pp. 22419–22430, 2021.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 2023.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 2021.

Pengfei Zhou, Yunlong Liu, Junli Liang, Qi Song, and Xiangyang Li. CrossLinear: Plug-and-play cross-correlation embedding for time series forecasting with exogenous variables. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*. PMLR, 2022.

A Implementation Details

Every model trains on a single Apple M2 Max system (30-core integrated GPU, 12 CPU cores, 32 GB unified memory) running macOS 26.6, where PyTorch 2.13 dispatches to the GPU through the Metal Performance Shaders (MPS) backend. The M2 Max shares one pool of memory between CPU and GPU, so the 32 GB figure is the whole budget for a run rather than a dedicated accelerator memory. Training runs for up to 50 epochs, with an early stopping patience of 5. That budget matches GCGNet, whose authors trained for more epochs than TimeXer and CrossLinear, and it puts all three models on roughly equal footing. The optimizer is Adam, with the initial learning rate left as a tunable hyperparameter. That rate holds for the first four epochs, then falls to 0.9 of its previous value at each epoch after that. Every reported number is a single training run. A fixed base seed (2021) combines with the model and the market to give each run its own seed, so no result is an average over restarts, and the paper reports no variance across seeds. All hyperparameter tuning uses a validation hold-out set, and every reported result comes from a separate test hold-out set, touched only after tuning finished. Since this is time series data, the training data comes chronologically first, then the validation data, then the test data.