Short-Term Load Forecasting at the Device Level: Are specialized deep-learning approaches worth the effort?

Gal Zmazek
gz66721@student.uni-lj.si
Jožef Stefan Institute
Ljubljana, Slovenia

Aljaž Osojnik
aljaz.osojnik@ijs.si
Jožef Stefan Institute
Ljubljana, Slovenia

Martin Žnidaršič
martin.znidarsic@ijs.si
Jožef Stefan Institute
Ljubljana, Slovenia

Bernard Ženko
bernard.zenko@ijs.si
Jožef Stefan Institute
Ljubljana, Slovenia

Abstract

Short-horizon device-level load forecasting underpins demand-side management and local flexibility services, yet recent deep architectures have been validated almost exclusively on long-horizon benchmarks. We examine this gap on one device by adapting six architectures across five representative families—linear, hierarchical, mixer, patch-based, and exogenous-native—to a four-hour horizon at fifteen-minute resolution and evaluating them on a single commercial baseload meter with aligned weather channels, against classical baselines (persistence, linear regression, random forest, XGBoost). All models share one normalised dataset, one fixed temporal split, and one rescaling policy, isolating architecture as the variable. On this device we observe the following: first, performance across all deep architectures is tightly clustered, and a compact linear model comes within a few percentage points of the best deep architecture despite using orders of magnitude fewer parameters. Second, weather covariates benefit only those architectures purpose-built for exogenous fusion; among models with generic covariate support, identical weather inputs yield no measurable improvement or even degrade accuracy, and a capacity-matched control indicates that the effect is architectural rather than parametric.

Keywords

short-term load forecasting, device-level forecasting, exogenous covariates, weather, forecasting architectures, evaluation methodology

1 Introduction

Device-level load forecasting operates at short horizons: fifteen minutes to a few hours, consisting of only a few time steps. Deep-learning architectures for forecasting published over the last five years were developed and ranked predominantly on long-horizon benchmarks spanning hundreds of time steps over largely aggregate series. Whether their mechanisms still perform as intended at short horizons on data from a single (noisy) device is untested.

Weather appears as a natural exogenous driver of device load, however, the evidence for its impact is mixed. One study on heat-pump devices shows that transformers do not consistently beat XGBoost and tend to overfit on irregular series despite added capacity, with lagged load dominating feature importance and temperature having a negligible effect [13]. Exogenous support is also uneven across architectures: some are built around covariates [23]; several, including PatchTST [15], lack a mechanism for a second input series, so only the target's own history is consumed and incorporating weather requires modifying the architecture; while others accept covariates only over the forecast window, which presumes a weather forecast.

We explore whether recent architectures benefit from the additional information provided by historical weather measurements. We test six architectures: DLinear [25], TiDE [4], N-HiTS [2], TSMixerx [3], TimeXer [23] and PatchTST [15]; and six classical baselines: persistence, linear regression, an MLP, random forest and two XGBoost variants. We select data from one commercial meter under one persisted normalisation, with a fixed train/test split and window-rescaling policy. Each covariate-capable architecture is evaluated with and without additional past weather information.

Contributions: (i) a standardised way of comparing classical and recent architectures, training and evaluating every model under the same preprocessing and the same splits at 15 min, 1 h and 4 h horizons; (ii) a per-architecture weather comparison; and (iii) an empirical noise bound from repeated same-seed runs.

2 Related work

2.1 Classical and tree baselines

For load-forecasting, seasonal-naive baselines with calendar features remain the standard reference point [9]. For building-load forecasting, the field has shifted toward deep recurrent models [18], while tree ensembles have established themselves as a primary comparator. The supporting evidence is consistent: random forests were benchmarked against ARIMA, exponential smoothing, and neural networks [5] and achieve accuracy comparable to neural networks on building energy [1]. They are robust in their principal hyperparameter, such that an untuned forest is a fair comparator [24], and they retain an advantage at scale [14]. A gradient-boosted tree given identical window-based inputs matched eight deep architectures across nine datasets [6], and an inductive-bias advantage for tree models is well documented on medium-sized tabular problems [8]. In line with this, our tree baselines receive the same sliding-window inputs as the deep models, alongside the subsampled fit with which they are typically supplied.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
Information Society 2026, Ljubljana, Slovenia
© 2026 Copyright held by the owner/author(s).
https://doi.org/10.70314/is.2026.sikdd.110

2.2 Deep-learning architectures

Recent load-forecasting research spans multiple families of deep-learning architectures. Attention-based forecasters [11] report accuracy gains that depend strongly on aggregation level [7, 20], which motivates evaluating them directly at device level. PatchTST segments a series into patches treated as tokens and encodes channels independently [15], typically with reversible instance normalisation [10]; its results are contested by linear models [25] and by architectures that attend across variates rather than along time [12]. Among prior device-level studies, the work closest to ours forecasts 308 heat pumps day-ahead and reports strong device heterogeneity [13].

2.3 Architectures that support historical weather

We found that exogenous support hinges on implementation rather than the published architecture, which commonly differ. Models exposing *historical* covariates can take past weather into account directly [2–4, 11, 17, 23], while models exposing only *future* covariates [19, 26] take a weather forecast as input; these fall outside our comparison, as we compare architectures against past weather only. Several architectures [12, 15, 22, 25] provide no covariate path.

3 Problem setup

Let y_t be the active power of one commercial meter at 15-minute cadence and $w_t \in \mathbb{R}^{17}$ the aligned weather vector. Given $x_t = (y_{t-L:t-1}, w_{t-L:t-1})$ with $L = 192$ (48 h), predict $y_{t:t+H-1}$ with $H = 16$ (4 h). Every trained model minimises squared error, so we report the matching quantity, RMSE in watts, at the single steps $h = 1$ (15 min), $h = 4$ (1 h) and $h = 16$ (4 h), and pooled across all 16 steps. Squared error weights the large deviations that a scheduler has to cover, and the persistence row in Table 1 anchors the scale.

The target is one always-on circuit of a service station, recorded from July 2020 to March 2024 (3.6 years): mean 618 W, standard deviation 104 W on the training slice, no zero readings, and a moderate seasonal swing (July mean 1.35 times February's). The test segment (June 2023 to March 2024) has a 10% lower mean. A heat pump would be a harder test of the weather question.

The series is split chronologically 80/20 into train-plus-validation and test, the last tenth of the training portion held out for early stopping. The test segment yields 24 864 windows, strictly later than everything the models see.

4 Methods

4.1 One pipeline

Recent architectures are usually benchmarked each under its own codebase and preprocessing, which leaves open whether a reported gain comes from the architecture or from the setup. To avoid that, every model in this paper, classical or deep, is trained and evaluated the same way: on the same normalised dataset, split into train and test at the same point in time, with every input window rescaled by the same declared rule, so that among the deep models a difference comes from the architecture rather than the data handling, and each weather comparison changes exactly one thing, the presence of the 17 weather channels. The classical baselines also differ in features and normalisation, so their gap is one between approaches. Per-column statistics are fitted on the training slice and shared by every run. Per-window instance normalisation centres and scales each lookback window by its own mean and standard deviation before the model sees it, then inverts that transformation on the output. This is the operation RevIN contributes to PatchTST [10] and that most recent architectures ship in some form; it is applied by the pipeline to the target channel of every deep model, with each model's internal version disabled so nothing normalises twice. Weather keeps the global normalisation only. RevIN's learnable affine parameters are not reproduced. Classical baselines receive the global normalisation alone.

4.2 Baselines and architectures

Persistence repeats y_{t-1}. Linear regression flattens the window. Random forest and XGBoost consume engineered features (load lags, rolling statistics, calendar encodings, weather at the forecast origin) and fit one regressor per horizon step. They are fitted twice: on a 15 000-window subsample, which is the usual practice, and on the same 89 357 windows the deep models see. An MLP flattens the raw window.

Recent architectures are taken from an existing implementation [16] through a thin adapter, so they train with the same loop, splits, normalisation and loss: DLinear, TiDE, N-HiTS, TSMixerx, TimeXer and PatchTST. Each receives the raw 192-step load window, a weather arm also the 17 weather channels. Library defaults apply, with a shared 25-epoch budget and early stopping. TimeXer runs at the shared 128-dimensional, 3-layer width rather than its own default of 512, which is four times every other Transformer here.

4.3 Constructing a weather path for PatchTST

PatchTST encodes each channel independently, so attention runs over the 24 time patches within a channel and never across them, and the head keeps only the target's forecast. Feeding weather as extra channels therefore does nothing: built from the same seed, the concatenated and univariate models have identical parameters and bit-identical output.

We therefore construct a weather path by hand. Both streams, load and the 17 weather channels collapsed into one combined signal, are built the same way up to the point where they meet: each is split into 24 overlapping patches, each patch is mapped to a 128-dimensional embedding by one linear layer plus a learned positional code, and the resulting sequence of embeddings is treated as a sequence of tokens and passed through its own three-layer self-attention encoder. This gives two independently-encoded token sequences, $Z_\ell \in \mathbb{R}^{24\times128}$ for load and $Z_w \in \mathbb{R}^{24\times128}$ for weather, each having so far attended only to itself. The two constructions differ only in how Z_ℓ and Z_w are then joined:

$$\text{positional:} \quad Z = \mathrm{W}[Z_\ell \,\|\, Z_w] \tag{1}$$

$$\text{attentional:} \quad Z = Z_\ell + \mathrm{MHA}(Q{=}Z_\ell,\ K{=}V{=}Z_w) \tag{2}$$

In the positional join, called the *separate encoder* model in Tables 1 and 2, $[\cdot\|\cdot]$ concatenates the two streams at each matching patch position into a 256-dimensional vector, and W is one learned linear layer projecting it back down to 128 dimensions. In the attentional join, MHA is multi-head cross-attention: the load tokens Z_ℓ act as the query Q, the weather tokens Z_w act as both the key and value ($K = V = Z_w$), so each load patch computes a weighted combination of all 24 weather patches instead of being tied to the matching one, letting a lag be learned rather

than specified. Neither construction is novel: cross-attention between an endogenous and an exogenous stream is published as TimeXer [23] and, with two encoders, as ExoTST [21].

5 Results

5.1 Model comparison

Because each model configuration is evaluated using a single random seed, a reported difference in test RMSE may reflect run-to-run variability rather than a genuine difference between models. To avoid overinterpreting small differences, we estimate an empirical reproducibility floor. We reran the linear model and the univariate PatchTST five times each while keeping the random seed, machine, data, and training code unchanged. For repeats whose early stopping terminated at the same epoch, the observed variation is attributable to low-level computational nondeterminism rather than to differences in model or data. The five linear repeats stopped at the same epoch and spanned 0.06 RMSE, with a standard deviation of 0.03. The PatchTST repeats that stopped at the same epoch differed by 0.07 RMSE in the same-epoch pairs we inspected. The largest same-epoch pairwise gap observed was 0.075 RMSE. We round this maximum up to 0.1 RMSE and use that value as a decision floor for comparisons on this dataset. This rule is conservative and operational, not inferential. An RMSE difference smaller than 0.1 is treated as within the reproducibility range of the single-seed evaluation protocol and is not interpreted as evidence that one model is better than another. An RMSE difference larger than 0.1 is reported as an apparent difference that exceeds the observed computational-noise floor; it is not claimed to be statistically significant.

Random forest and Random forest‡ are the identical model on the identical features (the last 48 load lags plus the weather reading at the forecast time, about 65 inputs); the plain row is fit on the usual 15 000-window subsample, the ‡ row on the full 89 357-window training set every deep model sees. XGBoost appears three times because we tried two feature sets and one data-size variant: XGBoost, calendar adds calendar encodings (hour-of-day and day-of-week as sine-cosine pairs, a weekend flag) to 48 load lags and weather, about 70 inputs; XGBoost, rolling stats uses 24 load lags, the mean, standard deviation, minimum and maximum of the load over the last 6, 12 and 24 hours, and the same weather reading, 53 inputs; and XGBoost, rolling stats‡ repeats the rolling-stats feature set on the full training set instead of the subsample.

Table 1 gives the raw 15-minute results. Considering only models without weather as input, the six explored architectures span just 1.1% (69.6 to 70.4), against a gap of up to 6.3% from the best of them to the best tabular baseline. Taking into account weather variations, the gaps go from 1.1% to 5.4% on the span across architectures and from 6.3% to 7.7% on the gap to the best tabular baseline, since weather helps some architectures and hurts others. DLinear reaches 70.3 with 6,176 parameters, within 2.4% of the best model overall, PatchTST + separate encoder, which is 144 times larger and correspondingly more expensive to train. TSMixerx at 48 K parameters matches univariate PatchTST within that 0.1 RMSE margin. More data does not close the tabular gap: on all 89 357 windows the random forest gets worse (77.0 to 78.1) and XGBoost moves by 0.2. PatchTST, library, is the implementation of [16] without the changes of Methods.

Table 1: Test RMSE (W), raw 15-minute target, single seed. † predates pipeline unification. ‡ fitted on all 89 357 training windows rather than a 15 000-window subsample.

Model	Params	Overall	15 min	1 h	4 h
Persistence	–	93.9	62.6	94.0	97.8
Random forest‡	–	78.1	61.9	77.0	81.9
Random forest	–	77.0	63.2	75.9	80.6
Linear	55 K	76.0	57.0	74.6	80.4
MLP†	955 K	75.6	71.5	73.6	77.9
XGBoost, calendar†	–	74.7	55.5	73.9	77.9
XGBoost, rolling stats†	–	74.2	55.1	73.0	78.4
XGBoost, rolling stats‡	–	74.0	54.1	72.9	78.5
N-HiTS + weather	6.2 M	72.4	56.4	72.0	75.0
N-HiTS	2.9 M	70.4	52.0	69.9	73.9
DLinear	6 176	70.3	52.9	69.6	73.8
TiDE	1.3 M	70.3	52.7	69.5	73.8
TiDE + weather	2.1 M	70.3	52.7	69.4	73.8
TSMixerx	48 K	69.7	52.9	69.1	73.3
TSMixerx + weather	50 K	69.7	55.2	69.3	72.7
PatchTST	452 K	69.6	52.3	68.7	73.2
TimeXer	650 K	69.6	52.6	69.0	72.9
PatchTST, 8 layers	1.1 M	69.5	52.0	68.9	73.0
PatchTST, library	452 K	69.4	51.9	69.1	72.7
TimeXer + weather	650 K	69.0	52.3	68.4	71.8
PatchTST + cross-attn	1.1 M	68.9	52.2	68.7	71.6
PatchTST + separate enc.	887 K	**68.7**	52.3	68.4	71.5

5.2 Weather

Table 2 pairs each architecture's weather arm (with weather as input) against its univariate arm (without). All native arms receive identical weather, so a difference in gain comes from how each architecture uses it, not from having different weather to work with. Three of the six, TimeXer and our two PatchTST constructions, were built specifically for exogenous fusion, and all three gain: the separate-encoder join by 1.37%, cross-attention by 1.10%, and TimeXer by 0.87%. The other three, TSMixerx, TiDE and N-HiTS, take weather through a general-purpose covariate mechanism rather than one designed for weather specifically; of these, TSMixerx and TiDE move by less than that 0.1 RMSE margin, and N-HiTS degrades by 2.9%.

Between the two PatchTST constructions, the simpler one wins: the separate-encoder join gains 1.37% at 887 K parameters, against 1.10% for cross-attention at 1.1 M. Its extra capability, a learned lag over the weather patches, does not pay off here.

Both constructions carry more parameters than the univariate model they are compared against, so we ran a capacity-matched control: univariate PatchTST at eight encoder layers, 1 114 264 parameters against cross-attention's 1 120 040. It scores 69.51. Multiplying the parameters of the univariate model by 2.5 therefore buys 0.11 of RMSE, 0.15%, right at that 0.1 RMSE margin, while the same budget spent on a weather path buys 0.66, and 0.85 for the positional join with fewer parameters than the control. The fusion effect is six to eight times the capacity effect, so the gains in Table 2 are weather, not size. TimeXer's 0.87% needed no control: exogenous series enter as variate tokens, so adding weather adds tokens rather than weights [23].

6 Discussion

On this device, long-horizon architectures transfer but do not differentiate. Without weather, mechanisms that separate them

Table 2: Weather arm against univariate arm. Negative favours weather.

Architecture	Path	Δ	Δ %	Params
PatchTST	separate encoder	−0.96	−1.37	2.0×
PatchTST	cross-attention	−0.76	−1.10	2.5×
TimeXer	native	−0.60	−0.87	unchanged
TSMixerx	native	−0.01	−0.02	+2 K
TiDE	native	−0.01	−0.01	+800 K
N-HiTS	native	+2.01	+2.85	+3.3 M

on long-horizon benchmarks compress into a 1.1% band at four hours, with a 6.3% gap to the best tabular model. At this horizon the choice among recent architectures matters less than the choice between the recent field and a tabular model.

On this device, weather helps only where the architecture was designed around it. Here a weather benefit is a property of a particular design rather than of the task, which is consistent with reading published fusion gains as architecture-specific and with the device heterogeneity reported by Lüdecke et al. [13]. Within the constructed designs the simpler join is the better one.

Our study also has some important shortcomings and limitations. We ran a single seed and no significance test on the test errors; same-seed repeats alone differ by up to 0.07 RMSE, on the order of several differences in Table 1. We use data of one meter at one site, so no claim about other devices or sites can be made. Library defaults were used rather than per-model tuning, and the PatchTST family carries a learning rate tuned in earlier work.

7 Conclusion

On the one commercial meter we studied, recent deep-learning architectures transfer to short-horizon load forecasting but do not provide meaningful performance differentiation; the gap between models is narrow, and a 6,176-parameter DLinear model comes within 2.4% of the best. Furthermore, on this meter weather improved accuracy only in architectures with a mechanism designed for exogenous fusion, so the gain depends on the design and not only on the data.

For this device, the complexity of the larger deep-learning models is not justified by their marginal accuracy gains over a lightweight one such as DLinear; whether this holds for other, more weather-driven devices remains to be tested.

Acknowledgements

The authors acknowledge the financial support from the European Union through the SEEDS project under the Horizon Europe Programme (GA No. 101138211) and the Slovenian Research and Innovation Agency (Research Core Funding No. P2-0103).

References

[1] Muhammad Waseem Ahmad, Monjur Mourshed, and Yacine Rezgui. 2017. Trees vs Neurons: Comparison between Random Forest and ANN for High-Resolution Prediction of Building Energy Consumption. *Energy and Buildings* 147 (2017), 77–89. doi:10.1016/j.enbuild.2017.04.038

[2] Cristian Challu, Kin G. Olivares, Boris N. Oreshkin, Federico Garza Ramirez, Max Mergenthaler Canseco, and Artur Dubrawski. 2023. N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 6989–6997. arXiv:2201.12886 doi:10.1609/aaai.v37i6.25854

[3] Si-An Chen, Chun-Liang Li, Nate Yoder, Sercan O. Arik, and Tomas Pfister. 2023. TSMixer: An All-MLP Architecture for Time Series Forecasting. *Transactions on Machine Learning Research (TMLR)* (2023). arXiv:2303.06053

[4] Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan Mathur, Rajat Sen, and Rose Yu. 2023. Long-term Forecasting with TiDE: Time-series Dense Encoder. *Transactions on Machine Learning Research (TMLR)* (2023). arXiv:2304.08424 https://openreview.net/forum?id=pCbC3aQB5W

[5] Grzegorz Dudek. 2015. Short-Term Load Forecasting Using Random Forests. In *Intelligent Systems'2014*. Springer International Publishing. doi:10.1007/978-3-319-11310-4_71

[6] Shereen Elsayed, Daniela Thyssens, Ahmed Rashed, Hadi Samer Jomaa, and Lars Schmidt-Thieme. 2021. Do We Really Need Deep Learning Models for Time Series Forecasting? arXiv:2101.02118 [cs.LG]

[7] Elena Giacomazzi, Felix Haag, and Konstantin Hopf. 2023. Short-Term Electricity Load Forecasting Using the Temporal Fusion Transformer: Effect of Grid Hierarchies and Data Sources. In *Proceedings of the 14th ACM International Conference on Future Energy Systems (e-Energy)*. 353–360. doi:10.1145/3575813.3597345

[8] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data?. In *Advances in Neural Information Processing Systems 35, Datasets and Benchmarks Track*.

[9] Rob J. Hyndman and George Athanasopoulos. 2021. *Forecasting: Principles and Practice* (3 ed.). OTexts.

[10] Taesung Kim, Jinhee Kim, Yoonbeen Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. 2022. Reversible Instance Normalization for Accurate Time-Series Forecasting against Distribution Shift. In *International Conference on Learning Representations (ICLR)*.

[11] Bryan Lim, Sercan Ö. Arık, Nicolas Loeff, and Tomas Pfister. 2021. Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. *International Journal of Forecasting* 37, 4 (2021), 1748–1764. doi:10.1016/j.ijforecast.2021.03.012

[12] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. 2024. iTransformer: Inverted Transformers Are Effective for Time Series Forecasting. In *International Conference on Learning Representations (ICLR)*.

[13] Marcel Lüdecke, Timon Justi, Michel Meinert, and Bernd Engel. 2026. Comparing Load-Forecasts of Residential Heatpumps with Transformer and XGB on Field Data. In *Energy Innovation Conference (EnInnov)*. doi:10.34657/33291

[14] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2022. M5 Accuracy Competition: Results, Findings, and Conclusions. *International Journal of Forecasting* 38, 4 (2022), 1346–1364. doi:10.1016/j.ijforecast.2021.11.013

[15] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *International Conference on Learning Representations (ICLR)*.

[16] Kin G. Olivares, Cristian Challú, Azul Garza, Max Mergenthaler Canseco, and Artur Dubrawski. 2022. NeuralForecast: User Friendly State-of-the-Art Neural Forecasting Models. PyCon Salt Lake City, Utah, US. https://github.com/Nixtla/neuralforecast

[17] Kin G. Olivares, Cristian Challu, Grzegorz Marcjasz, Rafał Weron, and Artur Dubrawski. 2023. Neural Basis Expansion Analysis with Exogenous Variables: Forecasting Electricity Prices with NBEATSx. *International Journal of Forecasting* 39, 2 (2023), 884–900. doi:10.1016/j.ijforecast.2022.03.001

[18] Ioannis Patsakos, Eleni Vrochidou, and George A. Papakostas. 2022. A Survey on Deep Learning for Building Load Forecasting. *Mathematical Problems in Engineering* 2022 (2022), 1–25. doi:10.1155/2022/1008491

[19] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *International Journal of Forecasting* 36, 3 (2020), 1181–1191. doi:10.1016/j.ijforecast.2019.07.001

[20] Ayas Shaqour, Tetsushi Ono, Aya Hagishima, and Hooman Farzaneh. 2022. Electrical Demand Aggregation Effects on the Performance of Deep Learning-Based Short-Term Load Forecasting of a Residential Building. *Energy and AI* 8 (2022), 100141. doi:10.1016/j.egyai.2022.100141

[21] Kshitij Tayal, Arvind Renganathan, Xiaowei Jia, Vipin Kumar, and Dan Lu. 2024. ExoTST: Exogenous-Aware Temporal Sequence Transformer for Time Series Prediction. In *IEEE International Conference on Data Mining (ICDM)*.

[22] Shiyu Wang, Haixu Wu, Xiaoming Shi, Tengge Hu, Huakun Luo, Lintao Ma, James Y. Zhang, and Jun Zhou. 2024. TimeMixer: Decomposable Multiscale Mixing for Time Series Forecasting. In *International Conference on Learning Representations (ICLR)*.

[23] Yuxuan Wang, Haixu Wu, Jiaxiang Dong, Guo Qin, Haoran Zhang, Yong Liu, Yunzhong Qiu, Jianmin Wang, and Mingsheng Long. 2024. TimeXer: Empowering Transformers for Time Series Forecasting with Exogenous Variables. In *Advances in Neural Information Processing Systems (NeurIPS)*. https://proceedings.neurips.cc/paper_files/paper/2024/hash/0113ef4642264adc2e6924a3cbbdf532-Abstract-Conference.html

[24] Zeyu Wang, Yueren Wang, Ruochen Zeng, Ravi S. Srinivasan, and Sherry Ahrentzen. 2018. Random Forest Based Hourly Building Energy Prediction. *Energy and Buildings* 171 (2018), 11–25. doi:10.1016/j.enbuild.2018.04.008

[25] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. Are Transformers Effective for Time Series Forecasting?. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 11121–11128. doi:10.1609/aaai.v37i9.26317

[26] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11106–11115. arXiv:2012.07436