Time-series Forecasting for Network Utilization in Large-Scale Scientific Workflows

Veronica Yakubovich[1], Ivy Sim[2], Phillip Kim[2], Nick Monozon[1], Jinny Chung[3], Alex Sim[4], Kesheng Wu[4]
[1]Institute for Computational and Mathematical Engineering, Stanford University, {vyakubovich,nmonozon}@stanford.edu
[2]University of California, Berkeley, {ivy_sim,woosuk_kim}@berkeley.edu
[3]Department of Statistics, Stanford University, jinnyc@stanford.edu
[4]Lawrence Berkeley National Laboratory, {asim,kwu}@lbl.gov

***Abstract*—As scientific workflows grow in scale, regional data caches are increasingly crucial for minimizing redundant data transfers and network congestion. To enhance cache management, we investigate the use of predictive models to forecast regional cache utilization. Leveraging historical data from the Southern California Petabyte Scale Cache, which supports a high-energy physics experiment, we evaluate the performance of eight time-series forecasting models in predicting daily cache hits. The models assessed include CNN-LSTM, LSTM-XGBoost, Seq2Seq with attention, TimeGrad, VARMAX, DeepGPVAR, Temporal Fusion Transformer, and ARIMA. Our analysis provides insights into the effectiveness of these models in optimizing cache management policies.**

I. Introduction

Large-scale scientific collaborations, such as the Large Hadron Collider, produce massive volumes of data, increasingly straining wide-area networks by creating network congestion with frequent file transfers [1]. To mitigate these challenges, regional data caches have been deployed to store frequently accessed datasets closer to users, reducing redundant long-distance transfers and improving network efficiency.

This work aims to investigate predictive capability for optimizing caching policies to enable proactive rather than reactive content placement. By analyzing historical access patterns, user behavior, and content popularity trends, predictive models anticipate which content will be requested and pre-position it before actual demand occurs. This forward-looking approach significantly reduces cache misses, minimizes origin server load, and improves content delivery latency for end users.

Earlier investigation [1] has demonstrated the feasibility of using a small number of popular models. This work further investigates a broad set of forecasting techniques for time-series data and includes classical statistical baselines, neural networks for time-series, probabilistic models, hybrid architectures, and transformer-based models. Our analysis aims to identify architectures that balance long-term trend modeling with responsiveness to peak events, informing more adaptive caching strategies.

II. Background and Data

This study focuses on forecasting the daily hit size in the Southern California (SoCal) cache, using data collected between August 2022 and March 2024 and the six key variables listed in Table 1, measured at daily and hourly intervals.

Metric	Description
Access Count	Number of cache requests in each interval
Access Size	Total data volume of those requests
Hit Count	Number of requests served directly from cache
Hit Size	Total data volume served from cache
Miss Count	Number of requests that missed the cache
Miss Size	Total data volume transferredfrom origin

Table 1: Description of cache usage metrics for forecasting.

Each cache request is classified as a hit or a miss, with access metrics computed as the sum of hits and misses. Data preprocessing includes converting byte volumes to terabytes, filling in missing values with zeros, and removing infinite values.

We evaluate eight forecasting models designed to capture overall cache usage trends while improving peak prediction accuracy: (i) a CNN-LSTM, (ii) a hybrid LSTM-XGBoost model, (iii) a Seq2Seq, (iv) a diffusion-based model, (v) VARMAX, (vi) DeepGPVAR, (vii) a Temporal Fusion Transformer, and (viii) ARIMA. In addition to these eight methods, we include a persistence baseline ($\hat{y}_{t+1} = y_t$) that uses the previous day's hit size as the forecast for the next day. To ensure consistent evaluation, we compare performance using RMSE on the original, unscaled data.

A. CNN-LSTM

The CNN-LSTM model combines spatial and temporal features by first applying convolutional filters, then passing data to an LSTM network [2]. We use a multivariate input window of size one, providing a six-dimensional daily input vector based on the six features in Table 1. This window size was selected as larger input windows yielded a higher RMSE and were less effective at capturing short-term peaks.

A 1D convolutional layer operates across the feature dimension, learning local interactions and cross-feature dependencies, such as correlations between hit size and access counts or imbalances between hits and misses. The transformed representation feeds into a single-layer LSTM, which encodes sequential patterns and filters temporal noise. A dense layer then maps the LSTM output to predictions for the next day's cache utilization. Together, the convolution and LSTM layers allow the model to capture sudden bursts, transient drops, and patterns that emerge from the joint evolution of cache metrics. Unlike other models that incorporate external signals or engineered features (e.g. Fourier terms), this architecture relies purely on the learned internal structure, making it compact and generalizable across different caches.

B. LSTM-XGBoost

This hybrid model combines an LSTM for temporal pattern extraction with an XGBoost regressor to refine predictions, particularly for sudden spikes in cache activity [2], [3]. We train a multivariate LSTM with a window size of one, using the six daily features to generate the LSTM's hidden state. Similar to CNN-LSTM, this window was selected as larger window sizes were less effective at capturing peaks. To capture seasonality, we add Fourier terms for weekly and monthly cycles, which are concatenated with both the final hidden state of the LSTM and the raw input features.

The resulting feature vector is passed to an XGBoost model, which learns complex nonlinear interactions and sharp transitions. Decoupling temporal feature extraction from nonlinear regression enables the model to capture both smoothed trends and abrupt changes, making it effective in forecasting cache utilization metrics that display abrupt jumps like peak hit counts or sudden miss surges.

C. Seq2Seq

The Seq2Seq architecture treats cache-metric forecasting as a sequence-to-sequence translation task, mapping a window of past observations to the next day's cache metrics [4]. The encoder, a stacked LSTM, compresses the input window into a set of hidden states, which an attention-based decoder transforms into predictions. We test two attention mechanisms: additive attention, which uses a feed-forward scoring function to identify relevant timesteps, and multiheaded dot product attention, which projects queries, keys, and values into multiple subspaces to capture orthogonal temporal patterns like weekly trends or sudden spikes in parallel.

This architecture combines LSTM state compression, dynamic context weighting via attention, and a feature-wise output layer, allowing it to model both long-term trends and sudden spikes in cache behavior.

A probabilistic Seq2Seq extension replaces the final dense layer with a head that outputs a mean-scale pair (μ, σ) for each metric, enabling interval predictions. By optimizing the negative log-likelihood, the model learns when to widen or narrow its confidence bands, adding calibrated uncertainty estimates without changing the underlying Seq2Seq architecture.

D. Diffusion

The diffusion-based model applies the TimeGrad architecture, framing time series forecasting as a denoising problem [5]. Instead of directly predicting next-day cache metrics, TimeGrad learns to reverse a forward diffusion process that incrementally adds noise to historical data.

During training, noise is added to past observations over multiple diffusion steps, and a score network is trained to estimate the gradient of the log-likelihood of the clean data given the noisy input. At inference, the model starts from pure noise and iteratively denoises toward a plausible future sample, enabling it to capture uncertainty and multimodal patterns in cache behavior, especially during ambiguous or volatile periods.

We use a multivariate version of TimeGrad that jointly models all six cache usage metrics, conditioning on their shared history to capture inter-feature dependencies. While this method yields well-calibrated forecasts for average behavior, it tends to oversmooth rare but significant spikes in activity, suggesting future work on adaptive noise schedules or spike-sensitive conditioning.

E. Vector Autoregressive Moving Average Model with Exogenous Variables (VARMAX)

The VARMAX model extends the VARMA framework by incorporating exogenous variables. Let $\mathbf{Y}_t \in \mathbb{R}^n$ be endogenous variables and let $\mathbf{X}_t \in \mathbb{R}^k$ be exogenous variables:

$$Y_t = C + \sum_{i=1}^{p} \Phi_i Y_{t-i} + \sum_{j=1}^{q} \Theta_j \varepsilon_{t-j} + \sum_{\ell=0}^{s} B_\ell X_{t-\ell} + \varepsilon_t \quad (1)$$

where Φ_i, Θ_j, B_ℓ are coefficient matrices, and $\varepsilon_t \sim \mathcal{N}(0, \Sigma)$. The structure in (1) captures both internal dynamics and the influence of external factors.

F. Deep Gaussian Process Vector Autoregression (DeepGP-VAR)

DeepGPVAR generalizes VAR by modeling nonlinear temporal dependencies using stacked Gaussian Processes:

$$\mathbf{Y}_t = f(\mathbf{Y}_{t-1}, \mathbf{Y}_{t-2}, \ldots, \mathbf{Y}_{t-p}) + \varepsilon_t \quad (2)$$

where $f(\cdot)$ is a deep Gaussian Process prior, ε_t is a noise term, and the structure in (2) allows successive GP layers to abstract increasingly high-level temporal features for flexible nonlinear modeling.

G. Temporal Fusion Transformer (TFT)

The TFT [6] is a sequence-to-sequence neural model for multi-horizon forecasting that combines static features, observed history, and known future covariates using gating, attention, and interpretable components. It outputs quantile forecasts using the pinball loss and emphasizes interpretability through variable gating and attention..

Let $\mathbf{y}_t = (y_t^{(1)}, y_t^{(2)}, y_t^{(3)})^\top$ denote the three cache variables $\{\texttt{access\_count}, \texttt{hit\_size}, \texttt{miss\_size}\}$. Then covariates $x_t \in \mathbb{R}^d$ span a fixed window $L_e + L_d$, forming input $X_t \in \mathbb{R}^{d \times (L_e + L_d)}$. The model learns a nonlinear mapping:

$$\hat{Y}_{t:t+L_d-1} = f_\theta(X_t) \in \mathbb{R}^{3 \times L_d} \quad (3)$$

where the mapping in (3) summarizes the encoder–decoder structure that underlies TFT's multi-horizon predictions. TFT combines four specialized blocks: (i) a variable-selection network (VSN) that gates each feature dimension via learned static importances, (ii) a bidirectional LSTM for short-range patterns, (iii) static enrichment to inject global context (e. g. region), and (iv) a multi-head attention layer [7] that attends jointly over past encoder states and future known covariates. A position-wise gated feed-forward layer refines the attended sequence, and a final linear head outputs the $\tau \in \{0.1, 0.5, 0.9\}$ quantiles via the pinball loss $\mathcal{L}_\tau$.

Models are trained per feature using PyTorch-Forecasting [8], with encoder length $L_e = 200$, decoder horizon $L_d = 100$, softplus normalization, and region-based grouping. Hyperparameters (hidden size, dropout, attention heads, etc.) are tuned with Optuna [9]. Each trial runs at most 15 epochs and is pruned early by the median-rule.

H. Autoregressive Integrated Moving Average (ARIMA)

ARIMA is a classical univariate time–series model that differences a series d times to induce stationarity and then fits an ARMA(p, q) model to the transformed data. For each variable $y_t \in \{\texttt{access\_count}, \texttt{hit\_size}, \texttt{miss\_size}\}$, we fit an ARIMA$(p, d, q)$ model [10], [11]:

$$\Phi_p(B)\,(1-B)^d\,y_t \;=\; \Theta_q(B)\,\varepsilon_t, \qquad \varepsilon_t \overset{\text{iid}}{\sim} \mathcal{N}(0,\sigma^2), \quad (4)$$

where $\Phi_p(B)$ and $\Theta_q(B)$ are polynomials in the back-shift operator B, allowing the structure in (4) to capture level, trend (d), and short-range autocorrelation (p, q).

We use auto_arima from pmdarima [12] to minimize the corrected Akaike information criterion $\text{AICc} = -2\log\mathcal{L} + 2kT/(T-k-1)$, with $k = p+q+d+1$. The optimal (p_0, d_0, q_0) seeds a 3×3 local grid (fixing $d = d_0$), with final order selected via five-fold rolling-origin RMSE (horizon $H = 30$).

The selected order is re-estimated on the full training span via conditional maximum likelihood [10], and an H-step forecast $\{\hat{y}_{T+j}\}_{j=1}^{H}$ is produced along with $\pm 1.96\,\sigma_j$ prediction bands. Optimization completes in < 45s/variable.

III. Analysis

This section compares our eight forecasting approaches for predicting daily cache hit size in the SoCal cache from August 2022 to March 2024. Each model is trained on the first 80% of the time series and evaluated on the remaining 20%, with results summarized in Table 2. Compared to the baseline, the models vary in capturing the trend and peaks, with only some approaches outperforming it. Training times also vary: CNN-LSTM, LSTM-XGBoost, and VARMAX train in roughly 10 s, DeepGPVAR in 30-45 s, Seq2Seq in roughly 2 min, ARIMA in about 5 min, TimeGrad in 15-20 min, and TFT in 30 min.

Model	Baseline	CNN-LSTM	LSTM-XGBoost	Seq2Seq	TimeGrad
RMSE	43.8	9.9	22.5	28.8	29.6
Model	VARMAX	DeepGPVAR	TFT	ARIMA	
RMSE	35.2	49.1	58.13	61.25	

Table 2: Test RMSE on Hit Size across models

A. CNN-LSTM

The CNN-LSTM model combines a 1D convolutional layer with an LSTM to capture both cross-feature interactions and temporal structure in cache usage signals. It uses a single-day input window with six features per timestep. The architecture begins with a causal 1D convolutional layer (256 filters, kernel size of 5, ReLU activation) followed by a 0.25 dropout layer. The convolutional layer operates across the feature dimension, learning short-range relationships such as the link between hit size and miss count within a single day.

The convolution output is fed into a single-layer LSTM with 256 units and tanh activation, followed by a dense layer that outputs the forecast. The model is trained using the Adam optimizer (learning rate 0.0005) and a 0.7 quantile loss, improving upper-bound predictions crucial for peak prediction. Training runs for up to 60 epochs with early stopping based on validation loss.

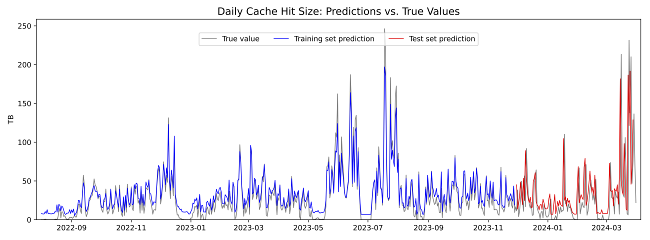

Fig. 1: Results of CNN-LSTM for Daily Cache Hit Size.

Figure 1 shows the CNN-LSTM's forecasts for daily cache hit size. The model effectively captures the trend and peak behavior and achieves the lowest RMSE among all evaluated approaches.

B. LSTM-XGBoost

The LSTM-XGBoost hybrid combines sequential modeling with nonlinear regression to forecast daily cache hit size. A multivariate LSTM processes a single-day input window of six features using 64 hidden units, tanh activation, and a dropout rate of 0.17, producing a 64-dimensional hidden state. Rather than directly predicting hit size, this hidden state is concatenated with the raw input features to form a combined feature vector for XGBoost. XGBoost models the target independently with 100 trees, a learning rate of 0.18, maximum depth of 2, and regularization to control overfitting. Subsampling and early stopping further prevent overfitting and encourage generalization.

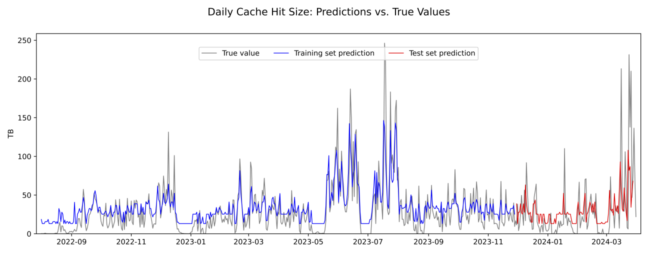

Fig. 2: Results of LSTM-XGBoost for Daily Cache Hit Size.

Figure 2 shows that the model effectively captures smooth underlying trends and some aspects of peak behavior, but is less sensitive to sharp peaks compared to CNN-LSTM. Overall, LSTM-XGBoost balances trend fitting with moderate peak prediction but lacks the sensitivity needed to capture rapid usage spikes.

C. Seq2Seq

The Seq2Seq model was trained using a 7-day history window. Its encoder consists of a 2-layer unidirectional LSTM with 128 hidden units, 0.1 dropout, and LayerNorm after each layer. The decoder uses multiheaded dot-product attention with four heads across the encoder timesteps. The resulting 128-dimensional context vector is concatenated with the top-layer hidden state, normalized, and projected to a single output for hit size. Training used AdamW and MSE loss for up to 100 epochs with early stopping patience set to 20.

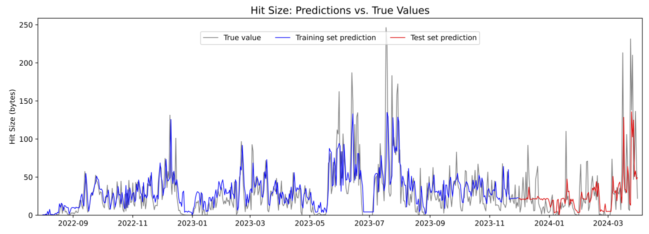

Fig. 3: Results of Seq2Seq trained for Daily Cache Hit Size.

Figure 3 shows that Seq2Seq generally captured broad weekly and monthly trends but consistently underestimated short-term peaks. The 7-day attention window tended to smooth out single-day peaks, causing sudden surges to be diluted in the context vector and leading to underestimated predictions. Reducing the attention window to 1 day did not improve performance, suggesting that, with only a single day of history, the model lacks sufficient context to distinguish true peaks from noise, and without broader temporal cues, it cannot improve upon the 7-day variant.

D. Diffusion

In our experiments, we use a multivariate version of TimeGrad to jointly forecast all six cache usage metrics, conditioning on a 7-day context window to learn shared temporal and cross-feature dependencies. The model is trained using a linear noise schedule with 32 diffusion steps.

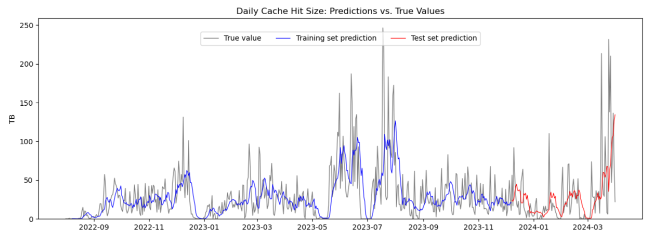

Fig. 4: Results of Diffusion for Daily Cache Hit Size.

As shown in Figure 4, the model captured overall trends and smooth variations well but struggled to predict sudden spikes in hit size. High-variance events were often smoothed over during the denoising process, leading to underestimation and higher RMSE, as shown in Table 2.

However, TimeGrad's probabilistic framework offers valuable uncertainty qualification. Notably, using the 75th percentile of generated samples as the point estimate reduced RMSE, demonstrating how selecting a more risk-aware point estimate can improve predictive accuracy on metrics with asymmetric error behavior.

Despite these advantages, diffusion-based models may be unnecessarily complex for this task. With only modest gains in accuracy, the added cost of iterative sampling and score-based training may not justify the overhead compared to simpler models like CNN-LSTM.

E. VARMAX

To assess stationarity, we applied the Augmented Dickey–Fuller (ADF) test to each series. Hit size had a p-values below the 0.05 threshold, satisfying the prerequisites for VARMAX modeling. Model orders were selected using AIC. VAR(3) yielded the lowest AIC among pure autoregressive models, while a grid search over (p, q) combinations showed VARMA(1,0,2) as the opitmal combination. Adding 7-day rolling averages of access count and access size as exogenous regressors yielded a VARMAX(1,0,2) model that further improved forecast accuracy.

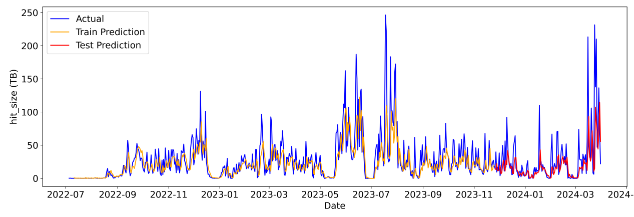

Fig. 5: Results of VARMAX for Daily Cache Hit Size.

The final models were estimated using maximum likelihood. As shown in Figure 5, VARMAX predictions closely track the observed hit size values, capturing the overall trend, but not the peaks.

F. DeepGPVAR

We implemented DeepGPVAR using the GluonTS framework with a context length of 3 (to match VARMAX lag structure), prediction length of 1 day, and a 3-layer RNN trained over 30 epochs with the Adam optimizer. Forecasts were generated via rolling prediction with a sliding window of 3 context steps.

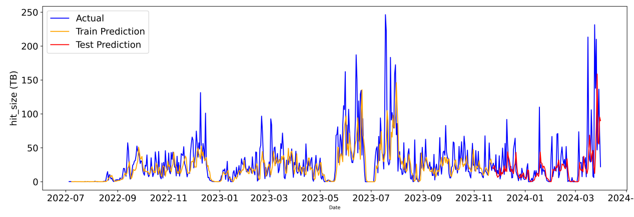

Fig. 6: Results of DeepGPVAR for Daily Cache Hit Size.

Although DeepGPVAR is a powerful deep probabilistic model capable of capturing long-range dependencies and non-linear seasonality, its RMSE is much higher than that of simpler statistical methods. It uses GP-based mechanisms and recurrent structure to model temporal patterns like seasonality and long memory. However, in our setting these capabilities offer little benefit. The result is unnecessary complexity without predictive gain. For short-range tasks, simpler autoregressive models often outperform models that learn a full latent process representation.

G. TFT

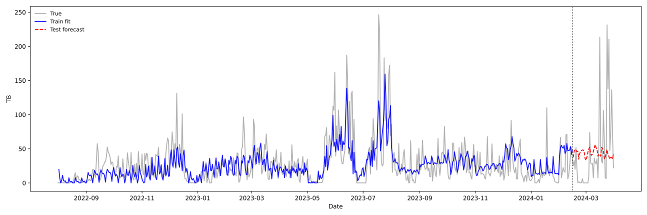

Fig. 7: Results of TFT for Daily Cache Hit Size.

The model uses an encoder length of 240 and a 30-day forecast horizon, with batch size 32, group-wise softplus normalization, and embedded categorical covariates (month, day-of-week). Hyperparameters were tuned via Optuna with early stopping, dropout ($p \in [0.1, 0.3]$), and gradient clipping. Overall, the model has a persistent train–validation gap typical of heavy-tailed behavior. Variable selection emphasizes region embedding and relative time index, and attention weights concentrate on recent lags and monthly boundaries, mirroring known traffic patterns. Notably, TFT's output quantiles widen in volatile regions, making the model potentially useful for risk-aware planning.

H. ARIMA

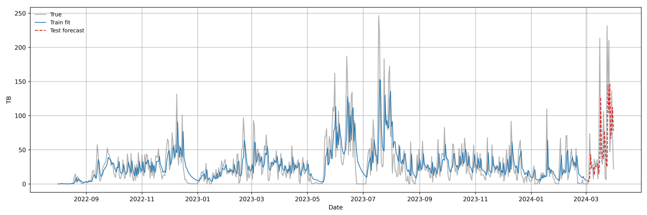

Fig. 8: Results of ARIMA for Daily Cache Hit Size.

Figure 8 shows the results of ARIMA for daily cache hit size, with an ARIMA$(1, 1, 1)$ model selected using AICc-guided stepwise search and a local 3×3 grid with rolling-origin validation. First-order differencing was applied to induce stationarity, and residual ACF/PACF plots showed no material autocorrelation up to lag 30, supporting the adequacy of the chosen specification for short-range linear dynamics. While the analytic 95% forecast bands cover much of the holdout region, they occasionally miss abrupt spikes, suggesting unmodeled regime shifts or conditional variance effects. As a result, the ARIMA model provides a transparent, interpretable linear baseline for hit size, but struggles to capture bursty, heavy-tailed behavior, motivating the need for nonlinear models.

IV. Conclusion

This study compares eight forecasting models representing a range of time-series approaches, from classical statistical methods to deep learning and hybrid models. Neural network–based models performed best in forecasting cache hits. In particular, CNN-LSTM achieved the lowest RMSE, capturing both overall trend and sharp fluctuations. LSTM-XGBoost also tracked trends and some peak behavior, though with lower sensitivity.

Probabilistic models like Seq2Seq and TimeGrad captured trends and uncertainty but often smoothed over extreme values. VARMAX and DeepGPVAR followed overall patterns but struggled with peaks, and DeepGPVAR's added complexity did not improve results compared to simpler statistical models. TFT produced quantile forecasts and highlighted temporal patterns but showed a consistent gap between training and validation. ARIMA provided a simpler, interpretable reference but struggled to capture the overall trend. The persistence baseline offered a naive reference point, and several models exceeded its performance while others did not.

These results suggest that neural architectures like CNN-LSTM, which handle both trends and spikes well, are better suited to forecast cache utilization. Although this work focused on a single cache, future work could include additional sites to support more localized forecasting strategies, as well as performing sensitivity analysis on key training conditions, such as window length, model horizon, and hyperparameters, to assess the robustness of the models under varied configurations.

Acknowledgment

This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] A. Sim, E. Wang, R. Monga, K. Wu, J. Balcas, B. White, C. Guok, I. Monga, D. Davila, F. Wurthwein, and H. Newman, "Comparing cache utilization trends for regional data caches," in *27th International Conference on Computing in High Energy & Nuclear Physics (CHEP2024)*, 2024.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997.

[3] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. ACM, Aug. 2016, p. 785–794.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.

[5] K. Rasul, C. Seward, I. Schuster, and R. Vollgraf, "Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting," 2021.

[6] B. Lim, S. Arik, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *International Journal of Forecasting*, 2021.

[7] A. Vaswani, N. Shazeer, and N. e. a. Parmar, "Attention is all you need," *Proc. NIPS*, 2017.

[8] J. Borchert, J. Fischer, and S. Günnemann, "PyTorch Forecasting: End-to-end time-series forecasting with pytorch," https://github.com/pycaret/pytorch-forecasting, 2020, accessed 2024-07-14.

[9] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next–generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'19)*, 2019, pp. 2623–2631.

[10] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. John Wiley & Sons, 2015.

[11] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*, 3rd ed. Springer, 2016.

[12] B. Smith, C. Collingwood, N. Roche, and A. Hall, "pmdarima: Arima and seasonal arima in python," https://github.com/alkaline-ml/pmdarima, 2022, version 2.0.1.