Predicting U.S. Energy Prices Using Machine Learning Models

Tran Minh Quan (21520414), Nguyen Thi Hong Ngoc (22520958), Nguyen Phan Thao Nguyen (22520976), Nguyen Trung Nguyen (22520978), and Pham Tran Da Thao (22521374)

Group 9 – Class IS403.P21
University of Information Technology, VNU-HCM, Ho Chi Minh City, Vietnam
{21520414, 22520958, 22520976, 22520978, 22521374}@gm.uit.edu.vn

June 16, 2025

Abstract. This study focuses on forecasting the prices of three energy commodities: Crude Oil, Natural Gas, and Heating Oil, using a diverse set of machine learning and statistical models. The dataset spans from January 1, 2019 to April 30, 2025 and contains over 1,600 daily price records for each commodity. Various models were applied and evaluated, including Gated Recurrent Unit (GRU), Long Short-Term Memory (LSTM), Seasonal Autoregressive Integrated Moving Average (SARIMA), Support Vector Regression (SVR), Prophet, Light Gradient Boosting Machine (LightGBM), and Transformer. Each model was fine-tuned using manual or grid search optimization to improve prediction performance. The preprocessing steps involved time series transformation, feature engineering, and normalization when necessary. Evaluation metrics such as RMSE, MAE, MAPE, and R^2 were used to assess forecasting accuracy across validation and test sets, as well as for short-term forecasts of 30, 60, and 90 days. Among the models, deep learning approaches like LSTM and Transformer showed promising results, while classical models like SARIMA and SVR performed competitively with appropriate tuning. The study demonstrates that hybrid approaches leveraging both temporal dependencies and engineered features can enhance forecasting accuracy for volatile energy markets. These findings contribute to more reliable decision-making in business analytics and energy economics.

Keywords: Time series forecasting · Energy price prediction · Machine learning · Deep learning · LSTM · GRU · SVR · SARIMA · Prophet · LightGBM · Transformer

1 Introduction

Energy prices, especially those of crude oil, natural gas, and heating oil, play a crucial role in shaping global economic dynamics. These commodities significantly affect industries, financial markets, and governmental policies. Therefore, accurately forecasting energy prices has become an essential task for investors, policymakers, and businesses aiming to make informed decisions.

In this project, we explore the application of machine learning and statistical models to forecast the prices of three major energy commodities: Crude Oil,

Natural Gas, and Heating Oil. The dataset includes over 1,600 records for each commodity, covering a period from January 1, 2019 to April 30, 2025. Statistical summaries indicate distinct characteristics among these commodities, such as high volatility in Crude Oil, skewed distribution in Natural Gas, and relatively stable patterns in Heating Oil.

Table 1: Descriptive statistics of Dataset

Statistic	**Crude Oil**	**Natural Gas**	**Heating Oil**
Count	1662	1663	1661
Mean	68,5569	3,4279	2,3169
Std	17,9029	1,6855	0,7153
Min	11,5700	1,5440	0,7036
Median	70,6450	2,8130	2,2496
Max	119,7800	9,6470	4,4874

To address the forecasting problem, we employ and compare the performance of several models, including GRU, LSTM, SARIMA, SVR, Prophet, LightGBM, and Transformer. Each model is optimized using techniques like grid search or manual parameter tuning. We assess their effectiveness using various evaluation metrics such as RMSE, MAE, MAPE, and R^2, across validation and test datasets, as well as in short-term forecasting horizons of 30, 60, and 90 days.

Through this study, we aim to identify the most effective models for energy price prediction and provide insights into the modeling strategies that best capture the temporal behavior of commodity markets.

2 Related Works

Energy price forecasting has been extensively studied due to its critical role in economic decision-making. Traditional statistical models such as ARIMA and SARIMA have been widely used for time series forecasting of crude oil, natural gas, and other commodities [9] [11]. These models capture linear dependencies and seasonal patterns but often struggle with non-linearity and volatility inherent in energy markets.

Machine learning approaches have gained popularity for their ability to model complex, non-linear relationships. Support Vector Regression (SVR) has been successfully applied to crude oil price prediction, offering robust performance on high-dimensional datasets [7] [12]. Ensemble methods like LightGBM have also demonstrated superior forecasting capabilities by leveraging gradient boosting frameworks [5].

Deep learning models [10], including LSTM and GRU networks, have shown promising results in capturing long-term dependencies in sequential data [1] [2]. More recently, Transformer-based architectures have outperformed traditional RNN-based models in various time series forecasting tasks due to their self-attention mechanisms that allow modeling of global temporal dependencies [13].

Additionally, Prophet, developed by Facebook, offers a practical and interpretable framework tailored for business time series data with seasonality and holiday effects [6]. However, its performance on highly volatile markets such as energy commodities remains a subject of ongoing research.

Despite the progress in model development, comparative studies that systematically evaluate these diverse forecasting techniques on multiple energy commodities over extended horizons are still limited. This study aims to fill this gap by benchmarking statistical, machine learning, and deep learning models on crude oil, natural gas, and heating oil price datasets.

3 Methodology

This study applies and compares seven time series forecasting models from both statistical and machine learning families: GRU, LSTM, SARIMA, SVR, Prophet, LightGBM, and Transformer. These models were selected for their proven effectiveness in temporal prediction tasks across various domains.

The dataset includes daily price records for Crude Oil, Natural Gas, and Heating Oil from January 2019 to April 2025. Before training, the data was normalized (if required by the model), and structured into sliding windows with a time step of 30 days. To examine model robustness, we experimented with three different dataset splits: 80% training – 10% validation – 10% testing (8:1:1), 70% – 15% – 15% (7:1.5:1.5), and 60% – 20% – 20% (6:2:2).

All models were evaluated under consistent preprocessing pipelines. Hyperparameters were optimized through manual or grid search depending on the model. The following subsections describe the theoretical foundation, architecture, and training configurations of each forecasting approach.

3.1 GRU (Gated Recurrent Unit)

The Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that utilizes gating mechanisms to control the flow of information, allowing it to capture temporal dependencies while mitigating the vanishing gradient problem. GRUs are computationally efficient and require fewer parameters than LSTM networks, making them suitable for medium-sized time series tasks.

A GRU cell consists of two main gates: the update gate z_t and the reset gate r_t. These gates are used to determine how much of the past information should be passed along to the future.

The GRU cell is computed as follows:

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\tilde{h}_t &= \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
$$

Where:

- x_t is the input at time step t
- h_t is the hidden state at time step t
- σ is the sigmoid activation function
- $\odot$ denotes element-wise multiplication
- W_*, U_*, b_* are learnable parameters

In our implementation, the GRU received as input a sequence of normalized daily price values over a time window of 30 days. The model output is a scalar representing the next day's predicted price. The input tensor shape was $(\texttt{batch\_size}, 30, 1)$.

Hyperparameters were optimized through grid search with the following search space:

- `hidden_size`: [32, 64]
- `num_layers`: [1, 2]
- `dropout`: [0.1, 0.3]
- `learning_rate`: [0.001, 0.0001]

The model was trained using the Mean Squared Error (MSE) loss function and the Adam optimizer. Early stopping with a patience of 10 epochs was applied to prevent overfitting. The best configuration was selected based on the lowest validation MAPE across data splits.

3.2 LSTM (Long Short-Term Memory)

Long Short-Term Memory (LSTM) networks are a specialized form of RNN designed to model long-range dependencies in sequential data. The architecture incorporates memory cells with input, forget, and output gates, allowing it to retain or discard information across time steps.

The LSTM cell consists of the following key components: the forget gate f_t, input gate i_t, candidate cell state $\tilde{c}_t$, cell state c_t, output gate o_t, and hidden state h_t. The operations of an LSTM cell at time step t are described by the following equations:

$$
\begin{aligned}
f\_t &= \sigma(W\_fx\_t + U\_fh\_t - 1 + b\_f) \\
i\_t &= \sigma(W\_ix\_t + U\_ih\_t - 1 + b\_i) \\
\tilde{c} * t &= \tanh(W\_cx\_t + U\_ch * t - 1 + b\_c) \\
c\_t &= f\_t \odot c\_t - 1 + i\_t \odot \tilde{c} * t \\
o\_t &= \sigma(W\_ox\_t + U\_oh * t - 1 + b\_o) \\
h\_t &= o\_t \odot \tanh(c\_t)
\end{aligned}
$$

Where:

- x_t is the input at time step t

- h_t is the hidden state at time step t
- c_t is the cell state at time step t
- σ is the sigmoid activation function
- $\odot$ denotes element-wise multiplication
- W_*, U_*, b_* are learnable parameters

Similar to GRU, the input to the LSTM was a sliding window of 30 days of normalized daily price data. Each sample had the shape $(\texttt{batch\_size}, 30, 1)$, and the output was a single predicted price for the following day.

The grid search for hyperparameter tuning covered:

- `hidden_size`: [32, 64]
- `num_layers`: [1, 2]
- `dropout`: [0.1, 0.3]
- `batch_size`: [32, 64]
- `learning_rate`: [0.001, 0.0005]

The model was trained for a maximum of 50 epochs using the Adam optimizer and MSE loss. Early stopping was applied with a patience of 7 epochs to reduce overfitting. The optimal configuration achieved a balance between training stability and generalization, and was selected based on validation performance.

3.3 SARIMA

SARIMA is a classical statistical model used for univariate time series forecasting with seasonality. It does not require data normalization and operates directly on the raw price series. The model was configured by grid searching the parameters:

- Non-seasonal order: (p, d, q), where $p, q \in [0, 2]$, $d = 1$
- Seasonal order: (P, D, Q, s), where $P, Q \in [0, 1]$, $D = 1$, and $s = 7$ (weekly cycle)

The general form of the SARIMA model is denoted as:

$$SARIMA(p, d, q)(P, D, Q)_s$$

The model equation is defined as:

$$\left(1 - \sum_{i=1}^{p} \phi_i B^i\right) \left(1 - \sum_{j=1}^{P} \Phi_j B^{js}\right) (1-B)^d (1-B^s)^D y_t = \epsilon^t + \left(\sum_{k=1}^{q} \theta_k B^k + \sum_{l=1}^{Q} \Theta_l (B^{ls})\right) \cdot \epsilon^t$$

Where:

- y_t: time series value at time t
- B: lag operator, where $B^k y_t = y_{t-k}$
- p, d, q: orders of the non-seasonal AR, differencing, and MA terms
- P, D, Q: orders of the seasonal AR, differencing, and MA terms
- s: seasonal period (e.g., $s = 7$ for weekly data)
- ϕ_i, θ_k: coefficients for non-seasonal AR and MA terms
- Φ_j, Θ_l: coefficients for seasonal AR and MA terms
- ϵ^t: white noise at time t

Model selection was based on the lowest Akaike Information Criterion (AIC). The optimal SARIMAX configuration was then used to forecast prices on the validation and test sets, as well as for 30, 60, and 90-day rolling predictions using '.fit()' and '.predict()' methods.

3.4 SVR (Support Vector Regression)

SVR is a kernel-based machine learning model for regression. Input data was normalized using `MinMaxScaler`, and structured with a sliding window of 30 time steps. A manual grid search was used to tune kernel type and hyperparameters. The configurations included:

- **Linear kernel:** `C = 100`
- **RBF kernel configurations:**
  - `rbf_1`: `C = 100`, `epsilon = 0.0001`, `gamma = 10`
  - `rbf_2`: `C = 10000`, `epsilon = 0.0001`, `gamma = 100`
  - `rbf_3`: `C = 1000`, `epsilon = 0.0005`, `gamma = 5`

Support Vector Regression aims to find a function $f(x)$ that deviates from the actual targets y_i by at most ε for all training data, while keeping the model as flat as possible. The function is defined as:

$$f(x) = \langle w, \phi(x) \rangle + b$$

where $\phi(x)$ maps the input x into a higher-dimensional feature space (implicitly via kernels), and w is the weight vector.

The optimization problem for SVR is formulated as:

$$\begin{aligned} \min_{w,b,\xi_i,\xi_i^*} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & y_i - \langle w, \phi(x_i)\rangle - b \le \varepsilon + \xi_i \\ & \langle w, \phi(x_i)\rangle + b - y_i \le \varepsilon + \xi_i^* \\ & \xi_i, \xi_i^* \ge 0, \quad i = 1, \ldots, n \end{aligned}$$

Here:

- ε defines the margin of tolerance where no penalty is given,
- ξ_i, ξ_i^* are slack variables for deviations beyond ε,
- C is the regularization parameter controlling the trade-off between flatness and tolerance to errors.

The kernel trick is applied to compute inner products in high-dimensional space using functions like:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \quad \text{(RBF kernel)}$$

In the case of a **linear kernel**, the function simplifies to:

$$f(x) = w^T x + b$$

and the kernel becomes a simple dot product:

$$K(x_i, x_j) = x_i^T x_j$$

This results in a linear regression with a margin of tolerance ε, optimized using the same objective and constraints as above. Linear SVR is particularly useful when the data is approximately linearly separable in its original feature space.

Models were trained and evaluated on validation sets. Predictions were inverse-transformed, and model selection was guided by MAPE.

3.5 Prophet

Prophet is a decomposable time series forecasting model developed by Facebook. It models a time series as:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

Where $g(t)$ captures trend, $s(t)$ models seasonality, $h(t)$ reflects holiday effects, and ε_t is the noise. Input data consisted of columns `ds` (date) and `y` (price). No normalization was required.

We configured Prophet with:

- `growth = 'linear'`
- Weekly and monthly seasonality enabled
- Fourier order = 6 for monthly components

Grid search was performed over:

- `changepoint_prior_scale` $[0.001, 0.01, 0.1, 0.5]$
- `seasonality_prior_scale` $[0.01, 0.1, 1.0, 10.0]$
- `changepoint_range` $[0.8, 0.9, 0.95]$

– `seasonality_mode`: `'additive'`, `'multiplicative'`

The model was trained using '.fit()' and evaluated with '.predict()' on validation, test, and 30/60/90-day forecast datasets. The best configuration minimized total RMSE.

3.6 LightGBM

LightGBM is a gradient boosting framework based on decision trees that grows trees leaf-wise. It is suitable for structured tabular data and benefits from engineered features. Input features included:

– Time-based: dayofweek, lag1, lag2, rolling3, rolling7
– Technical indicators: EMA12, EMA26, MACD, Momentum

Data preprocessing was handled via `pandas`; rows with missing values were dropped. No normalization was required.

Hyperparameters were tuned using `GridSearchCV` with 3-fold cross-validation:

– `n_estimators` $[300, 500]$
– `learning_rate` $[0.01, 0.05]$
– `max_depth` $[4, 6]$

LightGBM follows the gradient boosting principle, where at iteration m, the prediction is updated as:

$$F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$$

Here:

– $F_{m-1}(x)$ is the prediction from the previous iteration,
– $h_m(x)$ is the newly trained decision tree that fits the negative gradient (residuals),
– η is the learning rate.

LightGBM uses histogram-based algorithms and a leaf-wise growth strategy to find splits that minimize the loss more efficiently than level-wise methods. The selected model was trained on training data and used to forecast validation, test, and 30/60/90-day rolling predictions.

3.7 Transformer

Transformer is a neural network architecture originally designed for NLP tasks but adapted here for time series forecasting due to its strong capability in modeling long-range dependencies.

The input is first projected into a higher-dimensional space using a linear layer (input projection) to form `d_model`-dimensional embeddings. Since Transformers lack an inherent notion of time, positional encoding was added to preserve temporal order. The positional encoding is typically defined as:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad \mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

The architecture includes:

- **Encoder:** multi-head self-attention and feed-forward layers
- **Decoder:** masked multi-head attention, encoder-decoder attention, feed-forward layers
- **Output projection:** final linear layer mapping from `d_model` to scalar predictions

The core operation is the scaled dot-product attention, which computes the attention output as:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Here, Q (queries), K (keys), and V (values) are linear projections of the input. The multi-head attention mechanism extends this by computing h parallel attention outputs and concatenating them:

$$\begin{aligned}\mathrm{MultiHead}(Q, K, V) &= \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)W^O \\ \text{where} \quad \mathrm{head}_i &= \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

The model was trained on sequences of 30 days to predict the next price and extended to forecast 30, 60, and 90-day horizons. Optimization was done using MSE loss and the Adam optimizer.

The grid search for hyperparameter tuning covered:

- `d_model`: {64, 128}
- `n_heads`: {2, 4}
- `n_encoder_layers`: {1, 2}
- `n_decoder_layers`: {1, 2}

4 Result

Table 2: Performance metrics for Crude Oil across different dataset splits

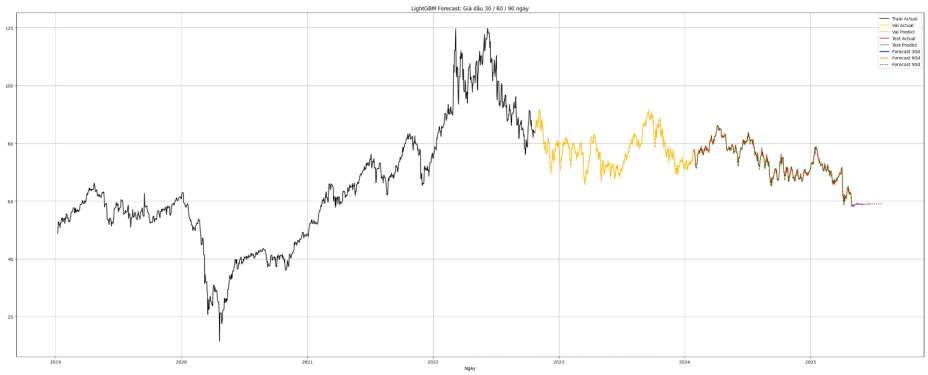
Ratio	**Algorithm**	**RMSE**	**MAE**	**MAPE**	**R^2**
6-2-2	LightGBM	0.70	0.54	0.74%	0.99
	LSTM	1.25	0.95	1.31%	0.95
	SVR	1.64	1.30	1.80%	0.92
	GRU	0.02	0.21	2.54%	0.89
	Transformer	2.89	2.31	3.14%	0.76
	Prophet	32.10	31.47	43.15%	-0.30
	SARIMA	8.27	6.90	9.78%	-1.06
7-1.5-1.5	LightGBM	0.63	0.49	0.96%	0.98
	LSTM	1.39	1.07	1.34%	0.93
	SVR	1.75	1.41	1.99%	0.88
	GRU	0.02	0.22	2.83%	0.81
	Transformer	3.21	2.61	3.27%	0.59
	Prophet	21.28	20.98	29.17%	-0.16
	SARIMA	12.18	10.55	14.35%	-0.45
8-1-1	LightGBM	0.65	0.47	0.68%	0.97
	LSTM	1.28	0.95	1.39%	0.89
	SVR	1.71	1.37	2.00%	0.84
	GRU	0.02	0.21	2.54%	0.75
	Transformer	2.86	2.25	3.21%	0.46
	Prophet	5.31	4.23	6.25%	-0.01
	SARIMA	5.08	4.34	6.16%	-0.83

Table 3: Performance metrics for Heating Oil across different dataset splits

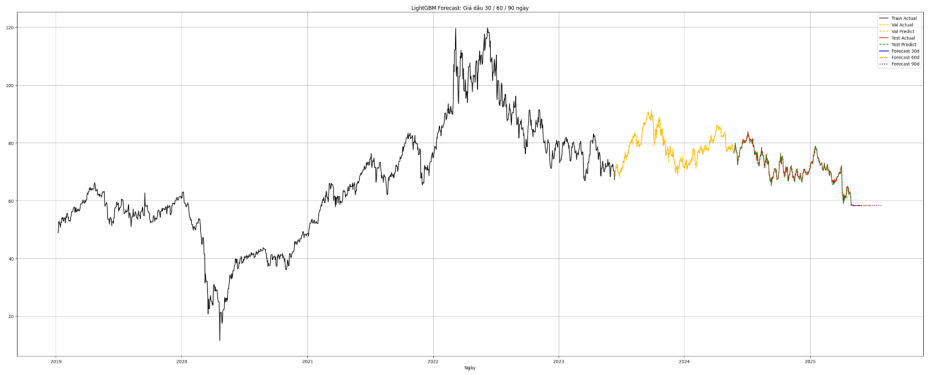
Ratio	**Algorithm**	**RMSE**	**MAE**	**MAPE**	$\mathbf{R^2}$
6-2-2	LightGBM	0.03	0.02	0.80%	0.98
	LSTM	0.04	0.03	1.08%	0.96
	GRU	0.02	0.01	3.01%	0.88
	SVR	0.11	0.10	4.16%	0.62
	Transformer	0.13	0.11	4.70%	0.47
	Prophet	0.26	0.22	9.24%	-0.82
	SARIMA	0.44	0.38	10.78%	-4.17
7-1.5-1.5	LightGBM	0.02	0.02	0.71%	0.98
	LSTM	0.04	0.03	1.07%	0.95
	GRU	0.01	0.01	2.67%	0.86
	Transformer	0.11	0.09	3.87%	0.43
	SVR	0.13	0.11	4.49%	0.37
	SARIMA	0.29	0.25	11.12%	-3.14
	Prophet	1.34	1.33	57.95%	-88.63
8-1-1	LightGBM	0.02	0.02	0.74%	0.97
	LSTM	0.04	0.03	1.35%	0.89
	GRU	0.03	0.02	2.79%	0.80
	SVR	0.10	0.07	3.25%	0.48
	Transformer	0.12	0.10	3.31%	0.36
	Prophet	0.20	0.16	7.03%	-1.62
	SARIMA	0.22	0.18	7.72%	-2.02

Table 4: Performance metrics for Natural Gas across different dataset splits

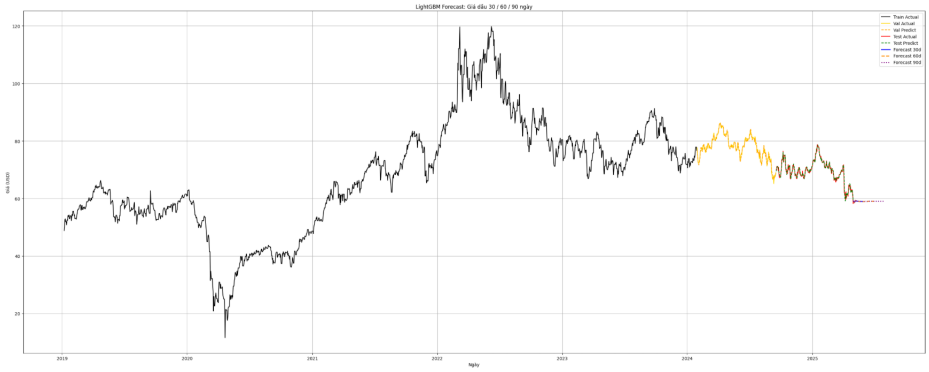
Ratio	**Algorithm**	**RMSE**	**MAE**	**MAPE**	**R^2**
6-2-2	LightGBM	0.07	0.05	1.65%	0.99
	LSTM	0.14	0.10	3.21%	0.96
	GRU	0.02	0.02	12.46%	0.94
	SVR	0.25	0.18	5.86%	0.90
	Transformer	0.30	0.23	7.88%	0.82
	SARIMA	1.60	1.29	39.87%	-3.88
	Prophet	2.70	2.13	65.92%	-12.85
7-1.5-1.5	LightGBM	0.05	0.07	1.56%	0.99
	LSTM	0.15	0.10	3.08%	0.95
	GRU	0.02	0.02	9.79%	0.91
	SVR	0.28	0.20	6.15%	0.81
	Transformer	0.33	0.27	8.10%	0.72
	SARIMA	0.79	0.64	19.16%	-0.66
	Prophet	1.48	1.29	35.49%	-4.75
8-1-1	LightGBM	0.06	0.09	1.72%	0.97
	LSTM	0.16	0.12	3.38%	0.87
	GRU	0.03	0.02	9.83%	0.77
	SVR	0.33	0.25	7.18%	0.46
	Transformer	0.27	0.27	7.52%	0.44
	Prophet	0.80	0.64	17.57%	-1.84
	SARIMA	1.03	0.90	24.89%	-3.62



(a) Split 6-2-2

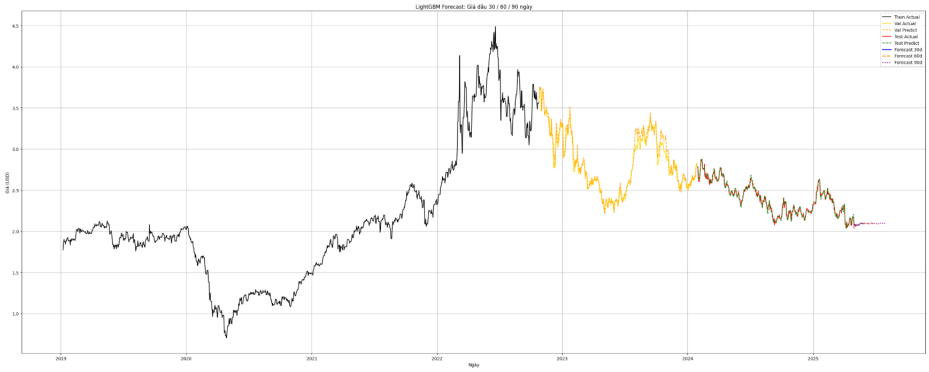


(b) Split 7-1.5-1.5

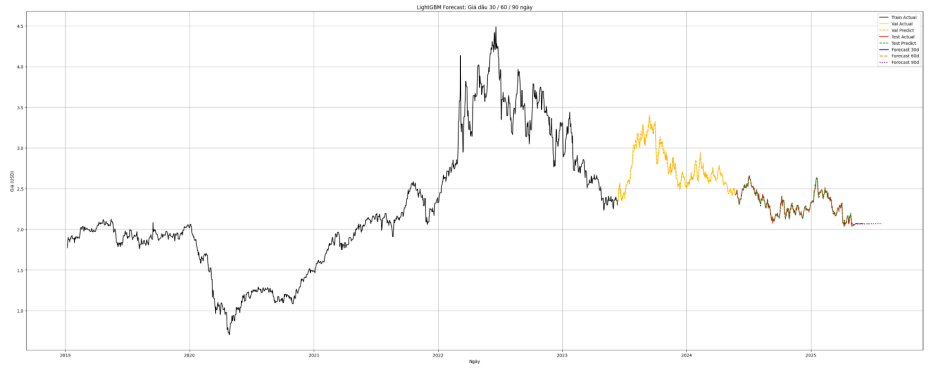


(c) Split 8-1-1

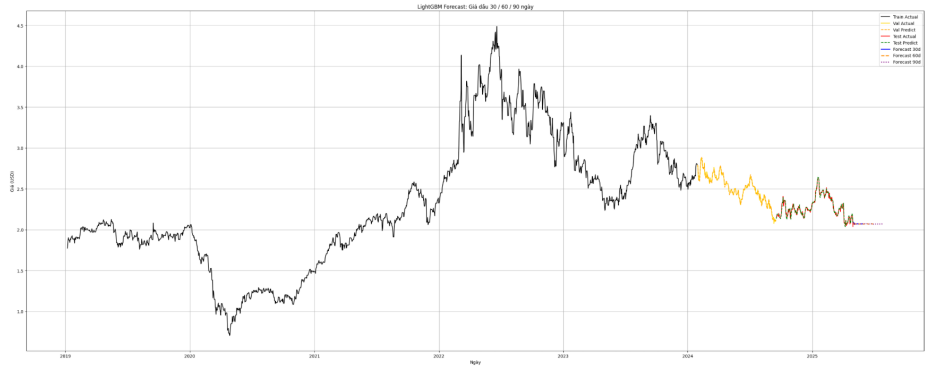
Fig. 1: Crude Oil forecast using LightGBM across different data splits



(a) Split 6–2–2

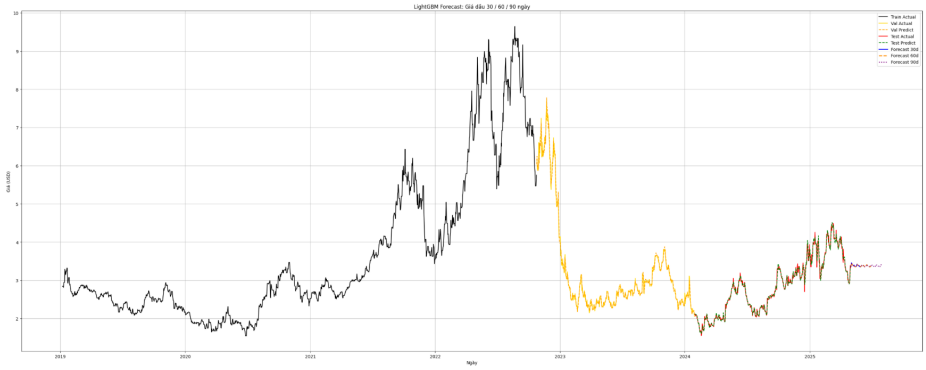


(b) Split 7–1.5–1.5

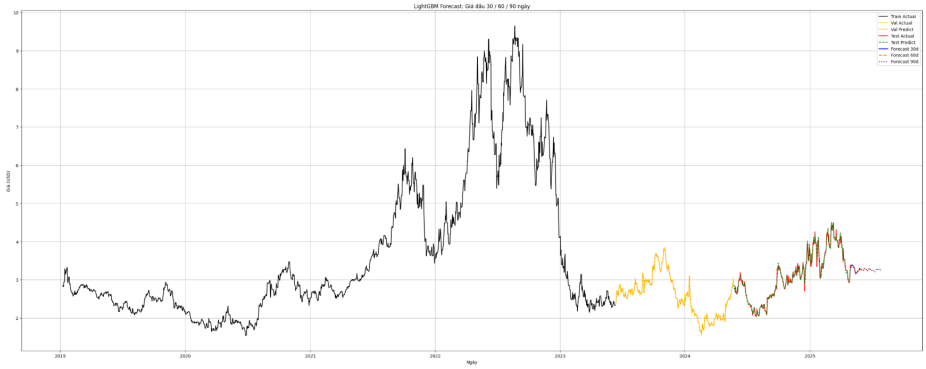


(c) Split 8–1–1

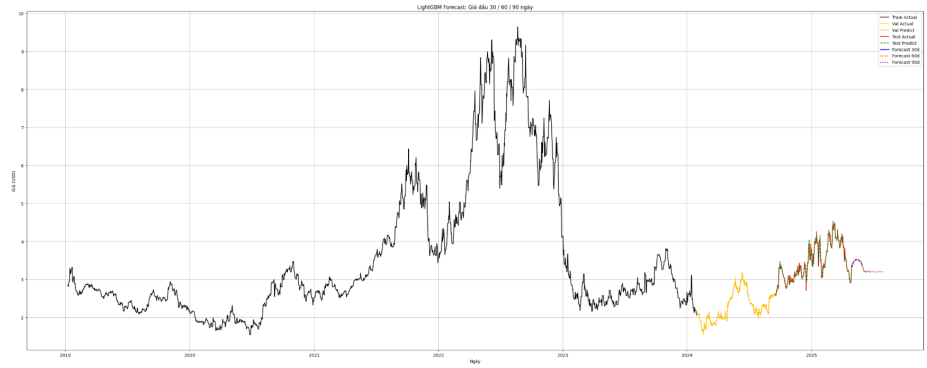
Fig. 2: Heating Oil forecast using LightGBM across different data splits



(a) Split 6–2–2



(b) Split 7–1.5–1.5



(c) Split 8–1–1

Fig. 3: Natural Gas forecast using LightGBM across different data splits

5 Conclusion

From the experimental results, LightGBM consistently achieved the highest prediction accuracy across all three energy commodities and dataset splits. LSTM also demonstrated strong performance, particularly on stable time series like Heating Oil. GRU performed reasonably well but was generally outperformed by both LightGBM and LSTM. In contrast, classical models such as Prophet and SARIMA yielded significantly higher error rates, especially on volatile data such as Natural Gas.

In future work, we plan to explore hybrid models that combine deep learning with statistical approaches, and to incorporate external factors (e.g., economic indicators, geopolitical events, or weather data) to further enhance forecasting performance and real-world applicability.

References

1. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
2. K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.
3. A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.
4. H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 11106–11115.
5. G. Ke *et al.*, "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 3146–3154.
6. S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.
7. A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004.
8. F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
9. G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Wiley, 2015.
10. B. Lim and S. Zohren, "Time-series forecasting with deep learning: A survey," *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2194, p. 20200209, 2021.
11. J. D. Hamilton, "Causes and consequences of the oil shock of 2007–08," *Brookings Papers on Economic Activity*, vol. 2009, no. 1, pp. 215–283, 2009.
12. X. Dong, L. Han, Y. Wang, and C. Xu, "Forecasting crude oil price volatility with mixture models and SVR," *Energy Economics*, vol. 70, pp. 273–285, 2018.
13. S. Li, W. Jin, and M. Zhou, "Enhancing the locality and breaking the memory bottleneck of Transformer on time series forecasting," in *Advances in Neural Information Processing Systems*, 2019.