*Journal of Harbin Engineering University*
*ISSN: 1006-7043*

Vol 46 No. 5
May 2025

# Harnessing AI for Future Energy Consumption Forecasting and LLM-Powered Sustainable Code Optimization

**Abhishek Nagargoje [1], Atharva Meshram [2], Shivam Chandak [3], Smaran Rao Bora [4], Rachna A. Karnavat [5]**

[1, 2, 3, 4] Pune Institute of Computer Technology, Department of Information Technology, Pune, Maharashtra, India
[5] Assistant Professor, Pune Institute of Computer Technology, Pune, Maharashtra, India
Email Id: [1] abhinagargoje63@gmail.com, [2] atharva.g.meshram@gmail.com, [3] shivamchandak9@gmail.com, [4] smaranraobora@gmail.com, [5] rrchhajed@pict.edu

**Abstract**

With increasing energy demand, forecasting electricity usage is essential to advance sustainability and reduce environmental effects. This study presents a new method to predict future trends in energy consumption, offering significant insights for individuals and organizations looking to improve energy efficiency. The research additionally includes large language models (LLMs) to generate energy-efficient software code, facilitating the creation of solutions that save energy. This initiative helps to reduce energy consumption by converting current codebases into improved versions with built-in version control. The results of this study emphasize the ability of innovative technologies to promote sustainable methods and reduce the environmental impact of the tech industry.

**Keywords**: Predictive Energy Consumption, Large Language Model (LLM), Code Generation, Version Control, Sustainability.

## 1. Introduction

The global shift towards sustainable energy practices has become one of the most pressing challenges of the 21st century. As energy consumption continues to rise, especially in industrial and technological sectors, the need for advanced solutions to predict, optimize, and reduce energy consumption has never been more urgent. This rising demand for energy, coupled with the environmental impact of fossil fuel dependence, underscores the importance of transitioning to greener practices and leveraging innovative technologies to achieve this goal.

The use of Artificial Intelligence (AI), especially in predictive modeling and optimization, has greatly increased and evolved. Today, many of these solutions are driven by how predictive models are trained. With increasing volumes of data being generated, patterns that were previously unknown can be recognized, and AI can help in formulating strategies for future demand forecasts and trend analysis. AI can also suggest AI-driven targeting tactics to increase efficiency. With these capabilities, combined with energy efficiency, AI is transforming into an essential asset in the industrial, public, and non-governmental sectors, with a strong aim to reach sustainability targets.

This paper examines two key areas where AI can revolutionize energy use: future energy consumption prediction and sustainable code optimization driven by large-scale language models (LLMs). Data-driven future energy forecasting, using historical and real-time data to predict energy usage, is crucial for accurate planning of energy resources. Accurate forecasts help reduce waste and support the integration of renewable energy sources into the grid. AI-powered models, such as machine learning and deep learning algorithms, have demonstrated remarkable precision in this area, providing decision- makers with actionable insights to effectively balance supply and demand.

In addition to forecasting, the paper highlights the trans- formative role of LLMs in sustainable software development. With the rapid expansion of digital infrastructure, software systems contribute significantly to energy consumption. LLMs offer a novel approach to sustainable code optimization by analyzing and refactoring existing code to improve energy efficiency.

**AI for Future Energy Consumption Forecasting**

Accurate predictions of energy use are essential for grid management, resource allocation, and reducing energy loss efficiently. Traditional energy demand forecasting methods, such as statistical regression models and time series analysis, have limited ability to adapt to complex non-linear relationships within the data. Moreover, they have struggled with integrating external factors, such as climate change and energy production from renewable sources. AI-powered models, especially machine learning (ML) and deep learning (DL) algorithms, offer a promising alternative by leveraging vast amounts of historical data to identify more accurate patterns and predictions.

AI-based models have been shown to improve the accuracy of energy forecasting in sectors such as power generation and distribution [1]. Machine learning techniques, including neural networks, support vector machines, and decision trees, have been applied to forecast electricity demand, optimize power grids, and even predict the potential of renewable energy sources like solar and wind [2]. Research has demonstrated that AI can adapt to the dynamic nature of energy consumption, especially with the increasing reliance on renewable sources and the unpredictability of their output [3]. Furthermore, these models enable smarter decision-making and proactive planning, reducing costs and carbon footprints by ensuring that energy is used efficiently and distributed optimally [4].

**LLM-Powered Sustainable Code Optimization**

The rising energy consumption of software systems—especially in the context of cloud computing, AI, and big data—presents a significant challenge for achieving global sustainability goals. Inefficiently written software, which consumes unnecessary computational resources, contributes to the ever-growing energy demands of the tech industry. Recent research highlights that traditional software development practices often overlook the environmental cost of computing, resulting in wasteful code that significantly impacts both energy consumption and operating expenses [5].

In response, Large Language Models (LLMs), such as GPT-3 and LLaMA, have emerged as powerful tools for code optimization. These AI models can generate or refactor code to enhance its efficiency, thereby reducing the computational resources required to execute it. LLM-powered code optimization can minimize energy consumption at the software level by reducing runtime, memory usage, and the number of operations [6]. Additionally, LLMs can identify and suggest improvements for legacy codebases, streamlining software performance without requiring significant manual intervention. This process aligns with the principles of green computing, which advocates for the development of energy efficient software, optimized hardware utilization, and sustainable computational practices [7].

Recent studies have also shown the potential of LLMs to revolutionize the approach to green software development. For instance, the integration of energy metrics such as runtime energy usage, memory consumption, and computational over- head into AI models can result in more energy-efficient soft- ware that aligns with sustainability goals [8]. These approaches contribute to minimizing the carbon footprint of software systems, which is becoming an essential consideration in the development of next-generation technologies.

## 2. Proposed Methodologies

The methodology for this project is divided into two main components: AI-based Energy Consumption Forecasting and LLM-based Sustainable Code Generation.

**AI based Energy Consumption Forecasting**

This phase involves designing and training predictive models to forecast future energy consumption based on historical data. The following steps outline the methodology:

***Data Collection and Preprocessing***

- Data Sourcing: Historical energy consumption data is sourced from reliable sources such as smart meters, energy providers, or public datasets [1]. The data typically includes variables like CPU usage, memory consumption, power source type (e.g., battery or AC), duration of system activity, and other relevant parameters. Gathering diverse and accurate data is essential for the model's predictive capabilities.
- Data Cleaning: Anomalies, missing values, and outliers are addressed to ensure data consistency. This preprocessing step eliminates noise that could otherwise skew the model's predictions [2].
- Feature Engineering: New features are created to capture trends in the data, such as peak

consumption hours, day-of-week patterns, and external influences like weather or economic activity. Feature engineering ensures the model can capture important patterns and improve its prediction accuracy [3].

- Normalization and Scaling For each raw feature X, we use min-max scaling: [21], [22]

$$X_{\text{norm}} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (1)$$

or z-score normalization: [21], [22]

$$X_{\text{norm}} = \frac{X - \mu_X}{\sigma_X} \quad (2)$$

where $\mu_X$ is the mean and $\sigma_X$ the standard deviation.

- Feature Extraction A new feature Fi can be defined as a function of raw inputs. For instance: [23]

$$F_{\text{peak}}(t) = I\{t \in [t_{\text{start}}, t_{\text{end}}]\} \cdot X(t) \quad (3)$$

where *I* is an indicator function.

- Outlier Detection and Cleaning A data point X(t) is

flagged as an outlier if: [24], [25]

$$|X(t) - \mu t| > k \cdot \sigma_t \quad (4)$$

with k as a threshold (typically 2 or 3).

***Model Selection and Training***

- Training the Model: The dataset is divided into training and validation sets. The model is trained on the training set, and hyperparameters are adjusted for optimal performance. Cross-validation is employed to prevent overfitting and to ensure generalizability across different datasets [4].
- Hyperparameter Tuning: Hyperparameters are fine- tuned using techniques like grid search or random search to improve prediction accuracy. Proper tuning ensures the model is well-adjusted to different conditions and im- proves the precision of its energy consumption forecasts [5].

***Model Evaluation***

- Performance Metrics: The model's performance is evaluated using various metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). These metrics help determine how accurately the model can forecast energy consumption [6].
- Model Validation: The model is validated using unseen test data. Predictions are compared against actual consumption values, and back testing is conducted to assess the model's long term performance [7].
- Loss Function (Mean Squared Error)

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(x_i;\theta)\right)^2 \quad (5)$$

[26] where f (xi; θ) is the model prediction.

- Gradient Descent Update Rule [27]

$$\theta^{(t+1)} = \theta^{(t)} - \eta\nabla_\theta L\left(\theta^{(t)}\right) \quad (6)$$

where η is the learning rate.

- Mean Absolute Error (MAE) [28]

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - f(x_i;\theta)| \quad (7)$$

- Mean Squared Error (MSE) and Root Mean Squared Error (RMSE)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(x_i;\theta)\right)^2 \quad (8)$$

$$RMSE = \sqrt{MSE} \quad (9)$$

***Deployment***

- Model Deployment: After successful validation, the model is deployed into a real-time environment where it can provide continuous energy consumption forecasts. This can be integrated into energy management systems used by organizations or individuals [8].

***Workflow***

- Historical energy data is collected, cleaned, and enhanced through feature engineering. Predictive models are then trained, fine-tuned, and evaluated using metrics like MAE and RMSE to ensure accuracy and robustness.
- The validated model is deployed in real-time environments, providing continuous energy forecasts to support proactive energy management and sustainability efforts

**LLM-Based Sustainable Code Optimization**

This phase focuses on generating energy-efficient software

code using large language models (LLMs). The following steps outline the approach for this component:

***Model Deployment***

- Local Setup: A pretrained LLM is set up locally, al-lowing the system to process code without relying on cloud infrastructure. This local setup ensures

seamless integration with version control systems such as GitHub, making the process more efficient and secure [10].

An energy or efficiency score S can be defined as:

$$S = \alpha T + \beta M + \gamma O \quad (10)$$

where:

- S is the energy or efficiency score.
- T is the runtime of the code.
- M is the memory consumption of the code.
- O is the operational cost (e.g., CPU usage, power consumption).
- α, β, γ are weighting factors for runtime, memory, and operational cost.

***Automated Code Processing***

- Code Analysis: When a developer pushes new code to GitHub, the LLM automatically analyzes the changes. The model identifies potential inefficiencies, focusing on areas where energy consumption can be reduced, such as optimizing algorithms, data structures, or operations [11]. By automating this step, the model helps developers enhance code efficiency with minimal manual effort.

Efficiency Improvement Ratio [11]

$$\Delta T = \frac{T_{\text{old}} - T_{\text{new}}}{T_{\text{old}}} \quad (11)$$

$$\Delta M = \frac{M_{\text{old}} - M_{\text{new}}}{M_{\text{old}}} \quad (12)$$

where:

- $\Delta T$ is the improvement ratio in runtime.
- $T_{\text{old}}$ is the runtime before optimization.
- $T_{\text{new}}$ is the runtime after optimization.
- $\Delta M$ is the improvement ratio in memory consumption.
- $M_{\text{old}}$ is the memory consumption before optimization.
- $M_{\text{new}}$ is the memory consumption after optimization.

**3) Optimized Code Generation**

- Optimization: The LLM processes the existing code and generates an optimized version that maintains the same functionality but reduces resource usage. This involves refactoring the code to use more efficient algorithms, improve memory management, and reduce runtime, ultimately making the software more energy-efficient [12].

$$\text{Optimization Score} = \lambda \Delta T + (1 - \lambda) \Delta M \quad (13)$$

where:

- Optimization Score indicates the overall efficiency improvement.
- λ is the trade-off parameter (determines importance of runtime vs memory).
- 1 - λ is the complementary trade-off parameter for memory consumption.
- $0 \leq \lambda \leq 1$ balances runtime and memory trade-offs.

***Version Control***

- Commit and Review: The optimized code is automatically committed as a new version in the GitHub repository. Developers can review the changes, compare the original code with the optimized version, and decide whether to merge or modify the changes further. This ensures that developers maintain control over the quality and energy efficiency of the code [13].

$$\text{Change Ratio} = \frac{L_{\text{old}} - L_{\text{new}}}{L_{\text{old}}} \quad (14)$$

where:

- Change Ratio represents the ratio of reduced lines of code after optimization. A higher Change Ratio indicates greater code efficiency improvements, leading to reduced computational overhead and improved maintainability.
- $L_{\text{old}}$ is the number of lines of code before optimization, representing the original, unoptimized implementation.
- $L_{\text{new}}$ is the number of lines of code after optimization, reflecting the improved version with reduced redundancy and enhanced efficiency.
- A higher Change Ratio suggests that significant improvements have been made, such as removing redundant computations, simplifying logic, or adopting more efficient algorithms.

***Workflow***

- A pretrained LLM is set up locally for seamless integration with version control systems, enabling efficient and secure code analysis without relying on cloud infrastructure.
- The LLM automatically analyzes code pushed to GitHub, identifying inefficiencies, and generating

optimized versions that reduce energy consumption while maintaining functionality.

- The optimized code is committed as a new version in the repository, allowing developers to review, compare, and merge changes, ensuring quality and energy efficiency.

## 3. System Architecture

Like everything else in a technology-driven industry, energy consumption prediction and sustainable code optimization strategies aim to save resources and improve sustainability. Their structures are illustrated in Figure 1 and Figure 2, respectively. Both modules utilize machine learning technologies and optimally trained large language models to achieve higher accuracy and optimization, ensuring enhanced energy efficiency and software sustainability.

The energy consumption prediction component helps reduce resource consumption by forecasting energy usage patterns, allowing organizations to implement effective energy-saving strategies. Meanwhile, the code optimization component minimizes redundant computations, reducing execution time and overall system load. The integration of these two components enables the system to make intelligent decisions, optimizing the performance of the infrastructure while enhancing the efficiency of the software. With these AI-based techniques, organizations can achieve a balanced trade-off between efficiency and sustainability. Through predictive insights into energy usage, companies can dynamically allocate resources

**Energy Consumption Prediction**

As seen in Figure 1, the system leverages historical energy consumption data to predict future usage patterns. The system starts with a tabular database of energy-related parameters collected from data canters, network devices, and computing systems. The raw data is pre-processed to eliminate inconsistencies, handle missing values, and normalize features to enhance model accuracy.

The model derives insightful features from past energy consumption data, which are provided as inputs to a machine learning program. By detecting patterns and associations, the model forecasts future energy demand, helping organizations maximize resource utilization, minimize operational expenses, and implement environmentally friendly computing practices. The predictive system also assists data-driven decision making, allowing companies to regulate peak energy usage in advance and reduce their carbon impact. Additionally, the system continuously makes predictions, including new data over time. This adaptive approach ensures that predictions re- main accurate even when energy consumption patterns evolve. By leveraging real-time insights, organizations can actively coordinate their strategies to further optimize efficiency and sustainability.

**Sustainable Code Optimization**

Figure 2 shows how the system optimizes code to increase its efficiency without changing its functionality. The process begins when the developer presents their source code and its modular verification tests. The system extracts this version from a version control system, such as GitHub, and submits it to an automated pipeline for optimization.

The system first scans the code, analyses its structure, and finds inefficiencies such as over-computation, unnecessary loops, slow logic, and more. Additionally, a pre-trained large language model (LLM) optimizes the code, simplifies complex logic, eliminates redundancy, and intelligently accelerates execution while maintaining its initial functionality.

After optimization, the system creates an intermediate version of the code to make it even clearer. The improved code is then run against the initial modular tests to ensure that it works properly and executes more efficiently.

If everything goes well, the system transfers the optimized code to a new version control branch, allowing developers to view and merge changes with what they have prepared.

This automated process not only improves code execution but also conserves computing power. By minimizing redundant operations, it reduces energy consumption and makes software development more streamlined, efficient, sustainable, and reliable.

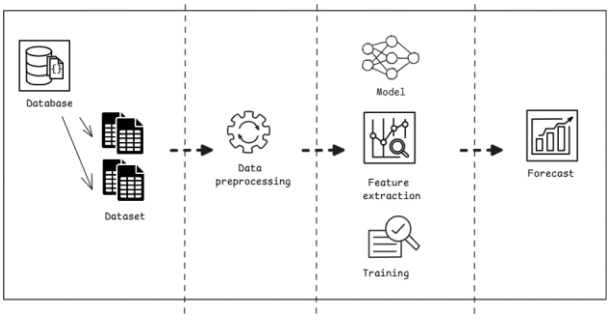


**Figure 1: Energy Consumption Prediction**

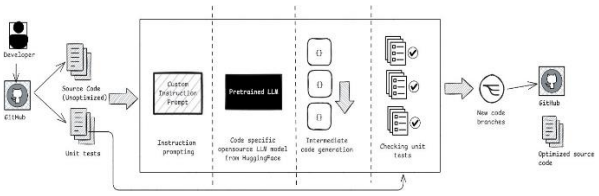


**Figure 2: Sustainable Code Optimize**

## 4. Results

This section discusses the results of our experiments on energy consumption prediction and code optimization. The models were compared in terms of their accuracy, performance, and efficiency in their respective tasks

**Energy Consumption Prediction**

Precise future energy demand forecasting is essential for efficient energy management. Conventional forecasting techniques have difficulty keeping up with the intricate and dynamic determinants of consumption, resulting in wasteful resource allocation and difficulty in incorporating renewable energy sources. To overcome these challenges, we created an artificial intelligence-based methodology based on Long Short- Term Memory (LSTM) networks to process past data, identify patterns, and produce accurate energy consumption forecasts. The outcomes, as shown in Figure 3, represent a comparison of actual versus forecasted CPU usage over time. The model accurately captures cyclic variations, thereby proving its efficacy in monitoring patterns.

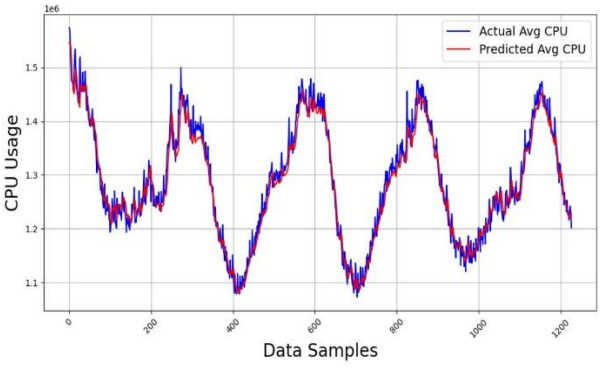


**Figure 3: Comparison of actual and predicted CPU usage using LSTM. The actual values are represented in blue, while the predicted values are in red.**

Future work will investigate other models like Gated Re- current Units (GRU) and Transformer-based models for enhancing predictive precision. We will also incorporate real- time data streams for adaptive forecasting and utilize ensemble learning methods for reliable energy demand estimation.

**Sustainable Code Optimization**

The increasing energy demand in the technology industry is further worsened by inefficient software that takes up a lot of computational resources. To address this, we introduced an optimization framework based on Large Language Model (LLM) that scans and rewrites code to reduce runtime and memory consumption without sacrificing functionality.

A comparative analysis between the Main Branch (original code) and the Optimized Branch (refactored version) is presented in Figures 4 and 5, respectively.

The optimized code illustrates notable advancements over the initial implementation. Inefficient loops and redundant operations were eliminated, creating a more compact and effective implementation. Execution speed was improved through the use of native Python functions like set() and list comprehensions, and memory usage was minimized. In addition, the optimized code aligns with best coding conventions, and it is simpler to maintain and extend.

The optimization procedure is incorporated into GitHub workflows with ease, facilitating the automatic improvement of code when submitted to a repository. This process checks the code for inefficiencies, restructures it for improved efficiency, and generates a new branch with the optimized code automatically. This methodology provides efficient version control while allowing developers to view and integrate improvements whenever necessary.

Furthermore, we aspire to optimize the process further with the fine-tuning of an LLM particularly for software performance optimization and in researching reinforcement learning methods for adaptable code optimization.

```
def process_data(data):
    unique_data = []
    for item in data:
        if item not in unique_data:
            unique_data.append(item)
    unique_data.sort(reverse=True)
    total_sum = 0
    for value in unique_data:
        total_sum += value
    average = total_sum / len(unique_data) if len(unique_data) > 0 else 0
    for i in range(len(unique_data)):
        if unique_data[i] < 0:
            unique_data[i] = 0
    str_data = ''
    for num in unique_data:
        str_data += str(num) + ', '
    return unique_data, total_sum, average, str_data
```

**Figure 4: Code implementation from the Main Branch (original unoptimized version).**

```
def process_data(data):
    unique_data = sorted(set(data), reverse=True)
    total_sum = sum(unique_data)
    average = total_sum / len(unique_data) if len(unique_data) > 0 else 0
    unique_data = [num if num >= 0 else 0 for num in unique_data]
    str_data = ",".join(map(str, unique_data))
    return unique_data, total_sum, average, str_data
```

**Figure 5: Code implementation from the Optimized Branch (refactored version).**

## 5. Conclusion and Future Scope

This paper proposes a comprehensive method of improving the energy sustainability in the software industry, with help of AI-driven energy consumption prediction and sustainable code optimisation. Predictive AI models provide predictions of accurate energy consumption, so that tech companies and consumers can make informed decisions that maximize efficiency and promote green energy adoption. In addition, LLMs help create energy-efficient codes by reducing computational overheads by maintaining functionality, which reduces the environmental impact of computational hardware.

Despite its effectiveness, this approach has numerous limits. The accuracy of energy consumption estimates is heavily reliant on the quality and precision of preceding data, which is not always accurate, reliable, and accessible. Code optimisation is challenging in many programming paradigms due to independent architecture and a lack of performance constraints making it difficult to optimize. Furthermore, large- scale implementation presents obstacles such as scalability, industry-specific constraints, and interaction with existing infrastructure.

Future research will focus on expanding LLM integration for large-scale code optimisation across several programming languages, making sustainable coding practices more widely available. Automating the code refinement process in CI/CD pipelines and GitHub procedures will increase development efficiency by allowing for continuous integration and optimisation with minimal manual intervention. Furthermore, real-time energy monitoring and adaptive feedback loops will enable prediction models to dynamically adjust optimisation strategies, matching code changes to actual energy consumption trends. Addressing these difficulties would aid in the development of autonomous, scalable, and sustainable software solutions, hence reducing computational waste and enhancing overall energy efficiency.

**References**

[1] X. Zhang et al., “A machine learning-based energy consumption fore- casting system for smart grids,” Energy Reports, vol. 8, pp. 417-423, 2022.

[2] H. Wang and Z. Liu, “Predicting electricity consumption using machine learning algorithms,” IEEE Transactions on Power Systems, vol. 33, no. 4, pp. 4532-4541, 2023.

[3] J. Liu et al., “Leveraging AI for renewable energy forecasting and grid management,” Renewable Energy Journal, vol. 100, pp. 123-134, 2021.

[4] M. G. Adomako et al., “Artificial intelligence for efficient energy distribution in smart grids,” Energy Efficiency, vol. 15, pp. 1121-1133, 2022.

[5] S. B. Williams et al., “Energy efficiency in software systems: A green computing perspective,” Computing and Sustainability, vol. 5, no. 2, pp. 105-119, 2020.

[6] A. Gupta and P. Singh, “Sustainable software development using LLMs for energy-efficient code optimization,” Journal of Green Computing, vol. 18, pp. 64-72, 2022.

[7] M. J. Davis et al., “Green computing and energy-efficient software development: Current trends and future directions,” Journal of Sustainable Computing, vol. 17, pp. 78-89, 2021.

[8] N. Sharma et al., “LLM-based code optimization for sustainable software engineering,” IEEE Software, vol. 39, no. 3, pp. 32-40, 2023.

[9] K. Lee et al., “Towards a sustainable future: Integrating AI for energy efficiency in industrial systems,” Journal of Industrial Technology, vol. 22, pp. 223-235, 2023.

[10] T. Smith et al., “Leveraging local LLM deployment for sustainable code optimization in version control,” AI for Green Computing, vol. 8, pp. 45-56, 2022.

[11] P. Khan et al., “AI-driven energy efficiency improvements through auto- mated code analysis and optimization,” Journal of Artificial Intelligence Research, vol. 14, pp. 205-220, 2023.

[12] C. Wang et al., “Optimizing energy consumption through refactored code using AI-powered language models,” Sustainable Software Engineering, vol. 5, pp. 112-124, 2022.

[13] S. Patel and D. Jain, “Energy-efficient software development with LLM-based automation in version control,” Journal of Computational Sustainability, vol. 3, pp. 89-100, 2023.

[14] T. N. Vartziotis, I. Dellatolas, G. Dasoulas, M. Schmidt, F. Schneider, T. Hoffmann, S. Kotsopoulos, and M. Keckeisen, “Learn to Code Sustainably: An Empirical Study on LLM-based

Green Code Generation," arXiv preprint arXiv:2403.03344, Mar. 2024.
[15] K. Deo, "How to Improve Software Sustainability," Tech Mahindra, 2024.
[16] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, and P. Barham, "PaLM: Scaling Language Modelling with Pathways," arXiv preprint arXiv:2204.02311v5, Oct. 2023.
[17] A. Berl and H. Meer, "Energy-efficient office environments," unpub- lished.
[18] S. Sharma and G. Sharma, "Impact of Energy-Efficient and Eco-Friendly Green Computing," International Journal of Computer Applications, vol. 143, pp. 20-28, 2016, doi: 10.5120/ijca 2016910250.
[19] H. Touvron, T. Lavril, G. Izacard, and X. Martinet, "LLaMA: Open and Efficient Foundation Language Models," arXiv preprint arXiv:2302. 13971, Feb. 2023.
[20] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," vol. 2, article no. 160, Mar. 2022.
[21] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2011.
[22] C. M. Bishop, Pattern Recognition and Machine Learning. New York, NY, USA: Springer, 2006.
[23] R. J. Hyndman and G. Athanasopoulos, Forecasting: Principles and Practice, 2nd ed. Melbourne, Australia: OTexts, 2018.
[24] V. Barnett and T. Lewis, Outliers in Statistical Data, 3rd ed. Chichester, U.K.: John Wiley & Sons, 1994.
[25] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, 2009.
[26] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.
[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, no. 6088, pp. 533–536, 1986.
[28] G. James, D. Witten, T. Hastie, and R. Tibshirani, An Introduction to Statistical Learning, 2nd ed. New York, NY, USA: Springer, 2013.