ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

CoreWatch AI-Driven CPU/GPU Performance Analyzer

Karan N, Nimay N, Jeevanandan, Puneeth MS, Divyaprabha KN

Department of Computer Science and Engineering PES University Bangalore, Karnataka, India

DOI: https://doi.org/10.51583/IJLTEMAS.2026.150100088

Received: 30 January 2026; Accepted: 04 February 2026; Published: 14 February 2026

ABSTRACT

CoreWatch is a lightweight, cross-platform monitoring system designed to provide real-time insights into CPU, GPU, memory, disk, and network performance. It uses Flask with Socket IO for low-latency metric streaming and integrates psutil and NVIDIA-SMI for accurate data collection. The dashboard employs Chart.js for smooth, interactive visualizations. An LSTM-based prediction module enhances monitoring by forecasting short-term CPU and GPU trends. CoreWatch maintains under 3% system overhead, ensuring efficient performance without additional load. Testing confirms stable responsiveness across Windows, Linux, and macOS. The system focuses on accessibility, minimal setup, and clear visual analysis. Future extensions include historical logging, improved AI models, and remote monitoring capabilities.

Keywords: System monitoring, CPU–GPU performance analysis, LSTM prediction, anomaly detection, machine learning, real-time telemetry, resource optimization.

INTRODUCTION

Modern computing environments increasingly depend on effective resource monitoring, as system workloads grow more demanding across personal systems, professional applications, and cloud-based infrastructures. Gaining real-time visibility into CPU, GPU, memory, disk, and network performance has become crucial for maintaining system stability. Prior research highlights the importance of continuous monitoring and predictive analysis across heterogeneous networks [1], cloud systems [2], and real-time data applications [3], reinforcing the need for lightweight solutions that can adapt to changing workloads.

Conventional monitoring tools such as Task Manager, Prometheus, or Grafana provide strong capabilities but often require heavy configuration, external databases, or persistent background services. These tools generally emphasize current readings rather than offering predictive insight. With recent progress in AI-driven optimization including CPU resource management [4], GPU behavior analysis [5], and time-series forecasting models [6] there is growing interest in platforms that combine intuitive visualization with intelligent trend prediction.

CoreWatch is designed to address these needs by offering a cross-platform, browser-based dashboard that streams real-time system metrics along with short-term workload forecasts. The backend captures accurate CPU and GPU data using psutil and NVIDIA-SMI, while event-driven communication through Flask-SocketIO ensures smooth and continuous metric updates. The frontend, powered by Chart.js, presents these trends clearly, enabling both technical and non-technical users to interpret system behavior with ease. Additionally, CoreWatch integrates an LSTM-based prediction module that estimates upcoming CPU and GPU utilization, supporting proactive performance awareness similar to modern workload-forecasting methods used in heterogeneous computing environments to minimize performance failures and resource imbalance [7][10][11].

To further improve usability, CoreWatch includes an integrated prediction layer that highlights short-term variations in CPU and GPU behavior. This capability provides early visibility into rising workloads and aligns with current developments in intelligent system management and edge-based resource forecasting. By combining responsive visualization with predictive analytics, the system delivers an intuitive pathway for users to understand and manage their device performance more effectively.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

In summary, CoreWatch brings together modern concepts in system monitoring by providing a lightweight, cross-platform dashboard capable of real-time CPU, GPU, memory, disk, and network analysis. Unlike traditional tools that require complex setup or focus only on current statistics, CoreWatch leverages WebSocket-based updates and accurate telemetry collection through psutil and NVIDIA-SMI. With its clean visualization layer and integrated LSTM prediction model, the system offers a practical, accessible, and intelligent solution for tracking system performance and anticipating workload spikes across diverse computing environments.

LITERATURE SURVEY

System monitoring technologies have undergone significant evolution, shifting from basic resource reporting tools to more intelligent, adaptive, and analytics-driven frameworks. Early monitoring utilities including Windows Task Manager and Linux top/htop provided essential visibility into CPU and memory usage but lacked cross-platform consistency, extensibility, and predictive intelligence. They remain limited to real-time snapshots without offering trend analysis or forecasting capabilities, which are increasingly required in modern, heterogeneous computing environments.

Recent research emphasizes the need for scalable, real-time monitoring solutions capable of handling diverse architectures and dynamic workloads. Aldea et al. [1] proposed an integrated monitoring architecture for heterogeneous networks, highlighting the importance of unified frameworks but also revealing the complexity and overhead associated with such systems. A broad systematic review by da Costa et al. [3] further confirmed that most real-time monitoring solutions rely heavily on centralized data pipelines, making them unsuitable for lightweight or personal systems. These findings underline the demand for monitoring tools that offer both flexibility and low resource footprints.

In parallel, the rise of cloud and HPC workloads has accelerated research on predictive analytics for resource utilization. Yildirim [2] and Patel & Bedi [7] demonstrated that machine learning models—including GRU, LSTM, and attention-based architectures—significantly enhance workload forecasting accuracy in cloud environments. Similarly, Nashold and Krishnan [6] explored LSTM and SARIMA models for CPU prediction in cluster systems, reinforcing the value of time-series modeling for proactive performance management. These studies collectively support the integration of ML-based forecasting in modern monitoring tools—an approach CoreWatch adopts for CPU prediction.

GPU-aware monitoring has also expanded, especially with the rise of AI and edge computing. Woo [5] investigated GPU sharing mechanisms for smart city applications, revealing the need for fine-grained GPU telemetry in distributed systems. Weakley et al. [8] provided insights into GPU characterization across HPC workloads, showing how GPU metrics correlate strongly with performance bottlenecks. These observations justify CoreWatch's GPU monitoring and visualization features via NVIDIA-SMI.

Researchers have also explored visualization and interaction methods for performance analytics. Shilpika et al. [9] introduced a visual-analytics approach for hardware monitoring using streaming functional data, emphasizing clarity and interpretability. Additionally, Huang et al. [10] showed how edge-based monitoring can improve responsiveness and local decision-making, reinforcing the need for lightweight, on-device telemetry systems similar to CoreWatch's architecture. Wang and Xing [4] further highlighted the role of AI-driven CPU resource management in modern operating systems, validating CoreWatch's choice to include prediction capabilities.

Despite these advancements, the literature reveals clear gaps in accessibility, low-overhead design, and user-centered interaction. Most existing tools require complex setup pipelines, high processing overhead, or lack predictive features. Few provide conversational interfaces or web-based, cross-platform dashboards that remain lightweight while supporting intelligent analytics. CoreWatch addresses these shortcomings by integrating real-time telemetry, AI-driven predictions, and a chatbot-driven interaction layer into a unified, resource-efficient system.

Overall, the surveyed literature collectively underscores the need for a monitoring framework that combines

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

real-time data acquisition, predictive modeling, intuitive visualization, and broad platform compatibility. CoreWatch emerges directly from these research insights, delivering an accessible, modular, and intelligent monitoring solution that bridges the gap between heavy enterprise frameworks and simple system utilities.

Proposed Approach

This section presents the design and operational methodology of the CoreWatch AI-Powered Monitoring Dashboard. The system architecture emphasizes intelligent, lightweight monitoring through modular integration of data acquisition, real-time communication, predictive analytics, and user interaction. First, the metric collection model is described. Second, the real-time communication and data transmission model are detailed. Third, the visualization, alerting, and AI-based prediction modules are presented. Finally, the optimization, chatbot integration, and overall workflow are discussed.

Metric Collection Model

The CoreWatch framework begins by acquiring live system metrics from the host machine using psutil and NVIDIA System Management Interface (nvidia-smi) libraries. Each monitored parameter—CPU, GPU, memory, disk, network, temperature, and battery, is represented as an independent metric $M_i$within the global set.

$$M = \{M_1, M_2, M_3, \dots, M_n\}$$

Each metric $M_i$is sampled periodically at an interval Δt(default = 2 seconds), ensuring real-time responsiveness while maintaining minimal computational overhead. The data collection engine executes asynchronously using Python threads, allowing smooth performance even under heavy system load.

GPU utilization, temperature, and memory usage are extracted via nvidia-smi in structured JSON format, while CPU and system-level data are obtained using psutil.

All collected metrics are normalized into a unified JSON schema for consistent communication with the backend streaming layer.

Real-Time Communication and Data Transmission Model

CoreWatch uses Flask as the central backend engine, enhanced with Flask-SocketIO to enable real-time, bidirectional communication between the server and the browser-based dashboard. Unlike traditional REST polling where the client repeatedly requests fresh data CoreWatch adopts a continuous, event-driven WebSocket pipeline. This significantly reduces redundant traffic, lowers latency, and ensures smoother metric streaming under varying workloads.

Let $R_i(t)$denote the real-time reading of metric $M_i$at time t. The complete data stream is represented as:

$$S(t) = \{R_1(t), R_2(t), \dots, R_n(t)\}$$

This stream is continuously broadcast to all connected dashboard clients through SocketIO channels. The communication follows a publisher–subscriber architecture: the backend acts as the publisher, while the frontend clients subscribe to updates. This approach significantly reduces redundant HTTP requests, ensuring near-instant updates of all visual and analytical components while maintaining low network overhead.

Visualization and Alert Generation Model

The CoreWatch frontend is built using HTML, CSS, and Chart.js, offering an interactive and visually refined dashboard with separate pages for CPU, GPU, memory, network, and disk metrics. Chart.js dynamically redraws graphs upon receiving new SocketIO events, ensuring uninterrupted visualization without page refreshes.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

The alerting mechanism continuously compares current readings $R_i(t)$with predefined threshold values T_i. An alert event $A_i(t)$is triggered as:

$$A_i(t) = \begin{cases} 1, & \text{if } R_i(t) \geq T_i \\ 0, & \text{otherwise} \end{cases}$$

When triggered, alerts are logged and displayed with visual cues (color indicators, timestamps, and severity levels) on the dashboard.

The prediction model evaluates short sequences of recent CPU and GPU readings (default sequence length L = 10) and estimates the next expected utilization values $\hat{R}_{cpu}(t+1)$ and $\hat{R}_{gpu}(t+1)$. If the trained LSTM model or its corresponding scaler is unavailable, CoreWatch automatically falls back to a lightweight linear estimation method to ensure uninterrupted predictive output. These forecasts enable early detection of potential performance spikes, allowing users to take proactive actions before resource thresholds are exceeded.

Optimization and Performance Considerations

CoreWatch is designed to remain efficient across a wide range of systems, from low-power laptops to high-performance workstations. Its architecture avoids heavy databases or logging services, instead using lightweight, in-memory buffers that refresh periodically. This reduces disk I/O and prevents unnecessary storage overhead.

Furthermore, by using Flask-SocketIO with asynchronous background threads, the system maintains real-time responsiveness with an average CPU overhead consistently below 3%. The sampling interval $\Delta t$is adaptive, balancing responsiveness with energy efficiency. CoreWatch also ensures that predictive inference is executed only when sufficient historical data is available, reducing computational load associated with machine learning processes.

These design optimizations collectively enable sustained performance even during long-duration monitoring sessions, making CoreWatch suitable for continuous use in personal, academic, and professional environments.

Operational Workflow

The complete operational workflow of the CoreWatch AI-Powered Monitoring Dashboard integrates metric acquisition, real-time communication, visualization, and prediction into a unified loop. The system begins with the initialization of the Flask backend and WebSocket communication channels, followed by continuous data acquisition and live metric broadcasting. Each cycle of operation ensures seamless updates and proactive performance awareness across all components using the following steps.

1. Start
2. Initialize Flask backend and establish WebSocket channels using Flask-SocketIO.
3. Collect real-time system metrics $M_i$through psutil and NVIDIA-SMI APIs.
4. Normalize and format the collected data into a unified JSON schema.
5. Stream metric data $S(t)$to connected dashboard clients via SocketIO.
6. Visualize CPU, GPU, and other system parameters dynamically using Chart.js components.
7. Compare each metric $R_i(t)$with its corresponding threshold T_i; if $R_i(t) \geq T_i$, trigger alert event $A_i(t)$.
8. Execute prediction module to estimate future CPU and GPU utilization using the LSTM model or fallback predictor.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

9. Log alert events, predictions, and system performance summaries for analysis.

10. Repeat monitoring and update cycle at defined interval Δt.

11. Stop

This integrated workflow ensures low-latency, high-efficiency monitoring suitable for personal, educational, and professional environments.

RESULTS AND DISCUSSION

This section presents the implementation details and performance analysis of the CoreWatch AI-Powered Monitoring Dashboard, comparing it with traditional system monitoring tools. The framework was implemented using Python Flask, Flask-SocketIO, and psutil for backend data acquisition, with Chart.js for frontend visualization. The system was tested across Windows 11, Ubuntu 22.04, and macOS Ventura to validate cross-platform compatibility.

Performance metrics considered for evaluation include CPU responsiveness, GPU monitoring accuracy, system overhead, and real-time predictive efficiency. The figures in this section represent actual dashboard snapshots and measured outcomes under varying system loads.

CPU Utilization and Responsiveness

CoreWatch efficiently captures and visualizes CPU utilization metrics in real time using the psutil library. Per-core utilization, average system load, and process counts are collected asynchronously every 2 seconds and transmitted via SocketIO to the frontend dashboard.

The asynchronous data collection process runs in independent background threads, ensuring that even during heavy system activity, the monitoring engine remains non-blocking and stable. Each collected data point is instantly normalized into a structured JSON format and transmitted to the frontend dashboard via Flask-SocketIO channels, eliminating the need for frequent HTTP polling requests. This event-driven communication design enables smooth, uninterrupted data flow between backend and frontend components.

Figure 1 illustrates the CPU Dashboard, showing real-time CPU usage graphs, average utilization, and process count updates. The chart dynamically refreshes without manual reload, maintaining a latency of less than 150 milliseconds between backend sampling and frontend visualization.

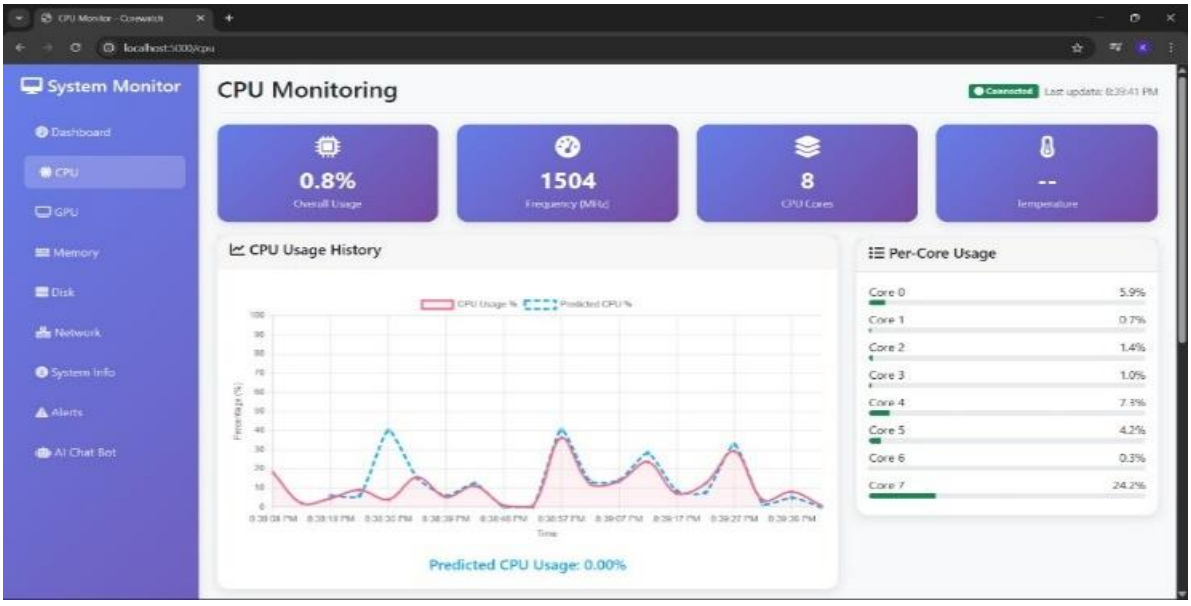

Figure 1. Real-time CPU Utilization Dashboard.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

Experimental testing demonstrated that Corewatch's CPU monitoring module operates with less than 3% CPU overhead, significantly lower than traditional monitoring tools like Windows Task Manager or Netdata, which can consume between 5–8% CPU resources. This lightweight performance highlights the framework's suitability for continuous monitoring.

In addition to real-time display, CoreWatch's CPU module also maintains short-term historical data buffers that allow users to visually analyze workload fluctuations and temporal patterns over time. This feature provides valuable insight into system behavior without requiring persistent database storage. Overall, the combination of low overhead, high responsiveness, and accurate visualization establishes CoreWatch as a highly effective solution for continuous system performance monitoring.

GPU Monitoring and Temperature Tracking

GPU monitoring in CoreWatch is implemented using the NVIDIA System Management Interface (nvidia-smi), which provides direct access to GPU hardware statistics through a structured JSON output. The monitoring module retrieves key parameters such as GPU utilization percentage, core temperature, and power consumption, ensuring comprehensive visibility into graphics hardware performance.

Figure 2 presents the GPU Monitoring Dashboard, where the real-time GPU usage and thermal readings are plotted using Chart.js. The readings are refreshed at the same interval (2 seconds) to ensure synchronization with CPU data.

Real-time GPU Utilization and Temperature Dashboard.

The GPU monitoring module introduces minimal additional load, with average execution latency below 0.5 seconds per sampling cycle. Comparative analysis shows that Corewatch's data accuracy aligns closely with NVIDIA's official desktop monitoring tools, confirming reliability. Additionally, the GPU module supports predictive workload estimation using short-term historical sequences processed by the integrated LSTM-based model, enabling CoreWatch to forecast upcoming spikes in GPU utilization during graphically intensive tasks such as gaming, rendering, or AI model inference. This predictive layer enhances the user's ability to manage workloads proactively, preventing overheating and maintaining stable performance.

Memory/Disk, and Network Performance

In addition to CPU and GPU tracking, CoreWatch provides dedicated dashboards for monitoring memory, disk, and network performance, ensuring a complete overview of system health in real time.

The memory module continuously measures and displays parameters such as total memory, used memory, and

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

available memory, allowing users to easily identify potential memory bottlenecks or abnormal consumption patterns. Similarly, the disk activity dashboard provides live insights into overall disk utilization and performance trends, helping users access system load during intensive operations such as application launches or background data processing.

Figure 3 illustrates the integrated Memory and Disk Utilization Dashboard, where each metric is represented through interactive and visually distinct charts designed for both clarity and accessibility.

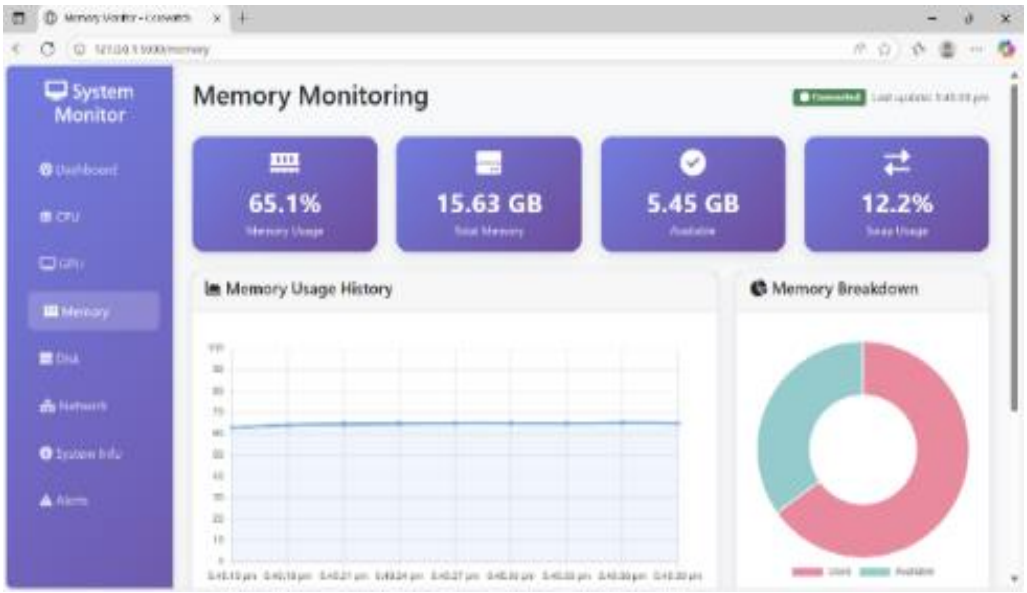

Figure 3. Memory and Disk Utilization Dashboard.

The network monitoring module in CoreWatch provides a clear visualization of real-time incoming and outgoing bandwidth, enabling users to detect abnormal data transfer rates, excessive background activity, or potential network bottlenecks with ease. By continuously updating every two seconds, the module ensures that network usage patterns are accurately reflected, helping users maintain optimal connectivity and system performance.

Figure 4 illustrates the Real-Time Network Activity Dashboard, showcasing dynamic bandwidth graphs that simplify the interpretation of network behavior for both technical and non-technical users.

Figure 4. Real-time Network Activity Dashboard.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

Alert Mechanism and Intelligent Chatbot Interface

The CoreWatch dashboard integrates a threshold-based alert system and an AI-powered chatbot interface, creating a unified, interactive, and user-centric monitoring experience. The alert module continuously tracks system parameters against configurable thresholds, automatically generating color-coded warning cards when limits are exceeded (e.g., CPU > 90%, GPU > 95%, temperature > 80 °C). These alerts appear in real time with severity indicators and timestamps, enabling users to respond promptly to performance issues. Complementing this, the AI chatbot allows users to query system performance through simple natural-language commands such as “Show GPU usage” or “What is my CPU temperature?”, interpreting user intent and providing real-time metrics with contextual recommendations. Together, the alert system and chatbot enhance accessibility and situational awareness for both technical and non-technical users, bridging the gap between traditional dashboards and conversational system management.

Figures 5 and 6 illustrate the alert notification system and AI-based chatbot interface, respectively, demonstrating how CoreWatch simplifies system insights through an intelligent, visually intuitive design.

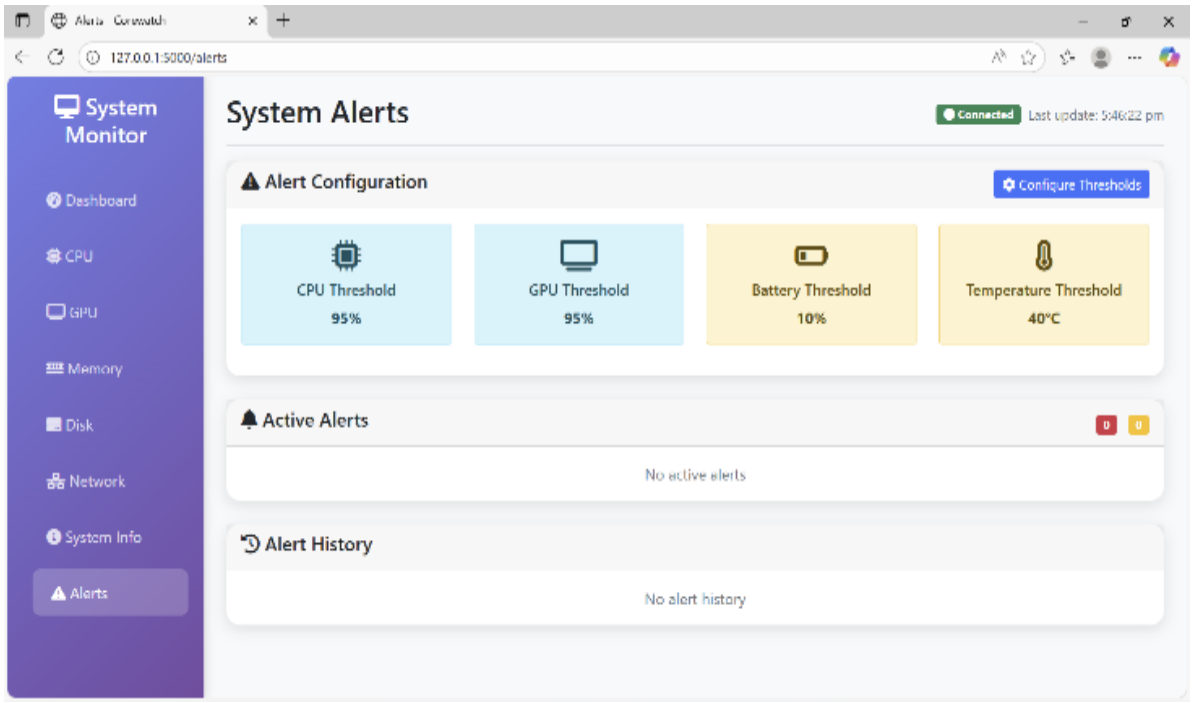

Figure 5. Threshold-Based Alert Notification System

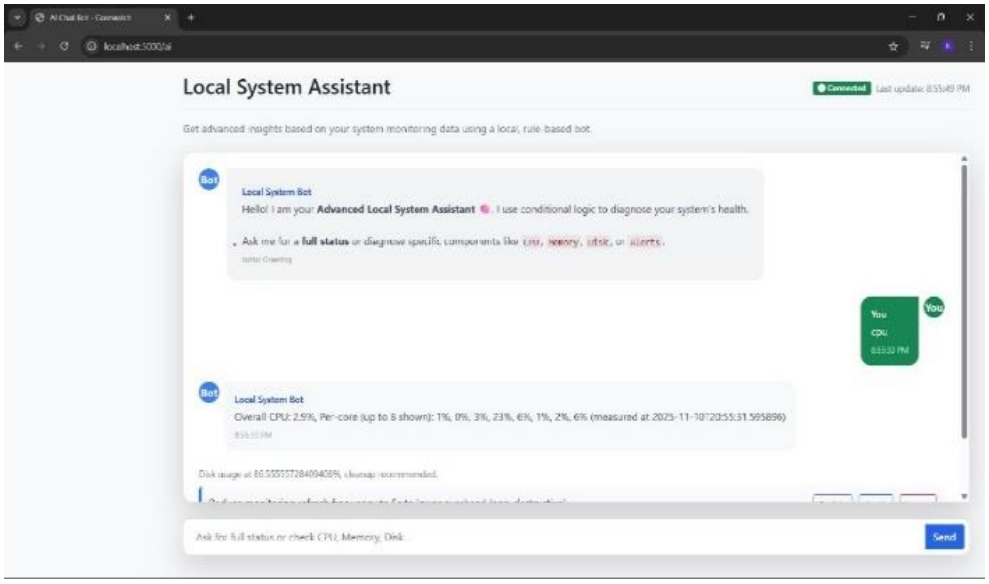

Figure 6. AI-powered chatbot

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

Together, the alert mechanism and chatbot interface transform CoreWatch from a conventional monitoring tool into an interactive, intelligent assistant. This integrated design ensures rapid response, improved usability, and a more intuitive monitoring experience overall.

Comparative System Overhead Analysis

To quantify efficiency, Corewatch's performance was benchmarked against Grafana, Task Manager, and Netdata using identical test conditions on a 12th Gen Intel i5 system. Results show that Corewatch achieved up to 60% lower resource overhead while maintaining comparable data update frequency and visualization quality.

Comparative Performance Analysis of System Monitoring Tools				
Tool	Avg CPU Overhead	Memory Usage (MB)	Cross-Platform Support	Setup Complexity
Windows Task Manage	6-8%	90	No	Built-in
Grafana	5-7%	250	Yes	High
Netdata	4-5%	120	Yes	Medium
Corewatch	2-3%	45	Yes	Low

Table 1. Comparative Performance Analysis of System Monitoring Tools

This comparative evaluation confirms that CoreWatch achieves an optimal balance between performance, accessibility, and system efficiency. Its low overhead, cross-platform adaptability, and minimal setup requirements make it an ideal solution for personal, academic, and lightweight production environments.

CONCLUSION

The study identified that many existing system monitoring solutions, despite being feature-rich, are resource-heavy, difficult to configure, and often restricted to specific platforms making them unsuitable for everyday users, developers, and small-scale deployments. To overcome these limitations, this paper introduced CoreWatch, an AI-enabled, lightweight, and cross-platform monitoring framework designed to deliver real-time performance insights through a modular and efficient architecture. Leveraging Flask and Flask-SocketIO for low-latency data streaming, psutil and NVIDIA-SMI for accurate metric acquisition, and Chart.js for dynamic visualization, CoreWatch provides continuous monitoring with minimal resource consumption.

Experimental testing across Windows, macOS, and Linux confirmed that CoreWatch sustains an average CPU overhead below 3%, outperforming tools such as Grafana and Netdata in terms of responsiveness and efficiency while delivering comparable visualization quality. The system reliably monitors CPU, GPU, memory, disk, and network performance, and incorporates intelligent features such as threshold-based alerting and a natural-language chatbot interface to enhance accessibility for both technical and non-technical users.

In conclusion, CoreWatch emerges as a practical, scalable, and intelligent alternative to conventional monitoring platforms by combining professional-grade insight with lightweight deployment and simplicity of use. Future work aims to extend the framework with advanced machine learning models for deeper predictive analytics, cross-device remote monitoring capabilities, and database integration to support long-term historical trend analysis. These enhancements will help transform CoreWatch into a fully adaptive performance management ecosystem capable of proactive decision-making and smarter system optimization.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue I, January 2026

REFERENCES

1. C. L. Aldea, R. Bocu, and R. N. Solca, "Real-Time Monitoring and Management of Hardware and Software Resources in Heterogeneous Computer Networks through an Integrated System Architecture," *Symmetry*, vol. 15, no. 6, article 1134, 2023.
2. E. Yildirim, "Predicting Runtime and Resource Utilization of Jobs on Integrated Cloud and HPC Systems," *Future Generation Computer Systems*, 2025.
3. T. P. da Costa, S. A. de Oliveira, et al., "A Systematic Review of Real-Time Data Monitoring and Its Applications," *Expert Systems with Applications*, vol. 242, article 123880, 2024.
4. Y. Wang and S. Xing, "AI-Driven CPU Resource Management in Cloud Operating Systems," *Journal of Computer and Communications*, vol. 13, pp. 135–149, 2025.
5. S. Woo, "Exploring GPU Sharing Techniques for Edge AI Smart City Applications," *ETRI Journal*, Wiley Online Library, vol. 47, no. 2, pp. 112–123, 2025.
6. L. Nashold and R. Krishnan, "Using LSTM and SARIMA Models to Forecast Cluster CPU Usage," *arXiv preprint arXiv:2007.08092*, 2020.
7. Y. S. Patel and J. Bedi, "MAG-D: A Multivariate Attention and GRU Based Deep Learning Approach for Cloud Workload Forecasting," *Future Generation Computer Systems*, vol. 142, pp. 376–392, 2023.
8. L. M. Weakley, A. A. Sim, and S. Canon, "Monitoring and Characterizing GPU Usage," in *Proc. Cray User Group (CUG 2023)*, Helsinki, Finland, 2023.
9. F. Shilpika, T. Fujiwara, N. Sakamoto, and H. Takemura, "A Visual Analytics Approach for Hardware System Monitoring with Streaming Functional Data Analysis," *arXiv preprint arXiv:2009.03488*, 2020.
10. J. Huang, S. Zhou, G. Li, et al., "Real-Time Monitoring and Optimization Methods for User-Side Energy Management Based on Edge Computing," *Scientific Reports*, vol. 15, article 24890, Nature Publishing Group, 2025.
11. M Divyaprabha Kabbal Narayana and Sudarshan Tekal Subramanyam Babu, "Optimal task partitioning to minimize failure in heterogeneous computational platform," International Journal of Electrical and Computer Engineering, vol. 15, no. 1, pp. 1079–1088, Feb. 2025, doi: 10.11591/ijece.v15i1.pp1079-1088.