Power Prediction Modelling for CPU–GPU Heterogeneous Platforms Based on Heterogeneous Feature Sequence Modelling

Jiawei Hu[1]*
[1]School of advanced manufacturing, Fuzhou University, Quanzhou, 362251, China
Corresponding author: Jiawei Hu (e-mail: hu_tuu@163.com).

ABSTRACT Central processing unit-graphics processing unit (CPU-GPU) heterogeneous platforms are widely deployed for AI-related, data-processing, and compute-intensive workloads. Their practical deployment is constrained by fluctuating node power, cross-device coordination overhead, and uneven energy efficiency. This paper develops a Transformer, a convolutional neural network, and a long short-term memory (LSTM) architecture (Transformer-CNN-LSTM) for system-level power prediction. Runtime features are cleaned, normalised, and reorganised into sliding-window sequences, allowing the model to learn global feature interactions, local workload fluctuations, and temporal evolution within a single prediction pipeline. Experiments on the collected CPU-GPU dataset demonstrate that this method achieves better results than CNN and LSTM, CNN-LSTM, performance monitoring counter statistical (PMC-Stat), and GPUWattch-Inspired baselines. The model achieves mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE), and R^2 results of 50.31, 7.09, 4.79, and 0.98, respectively, indicating its potential for runtime power estimation and energy-aware platform management.

INDEX TERMS CPU-GPU heterogeneous platform, power consumption prediction, Transformer-CNN-LSTM, system-level power modeling, heterogeneous feature sequence modeling, energy-aware optimization

I. INTRODUCTION

CPU-GPU heterogeneous platforms combine CPU-side control capability with GPU-side data-parallel acceleration and are widely deployed for AI-related, data-processing, and compute-intensive workloads [1]–[3]. When these platforms move from laboratory workloads to practical deployment, node-level energy demand, thermal pressure, and power-management overhead become important design concerns [4], [5]. Reliable power prediction is therefore required not only for monitoring, but also for scheduling, power capping, thermal-aware control, and energy-oriented runtime optimisation [6], [7].

Related studies cover several complementary directions. GPU-oriented work includes analytical power-performance estimation, performance-counter-based kernel modelling, and architectural frameworks such as Multicore Power, Area, and Timing (McPAT), GPUWattch, and AccelWattch [8]–[12]. Other studies use performance monitoring counters (PMCs), dynamic voltage and frequency scaling (DVFS) information, and multicore CPU measurements to estimate runtime energy use [13]–[15]. These methods provide valuable modelling tools, although many still emphasise a single device, a limited subsystem, or a weakly coupled platform description.

In practical heterogeneous systems, the CPU typically handles orchestration, data preparation, and task dispatching, whereas the GPU executes highly parallel kernels. This division improves throughput but also introduces shared thermal budgets, data-transfer overhead, and phase-dependent power variation. Consequently, CPU-GPU power estimation should be considered a coupled runtime problem rather than a simple extension of single-device prediction [16], [17].

CPU-GPU collaborative prediction is challenging because multiple asynchronous sources contribute to the observable power trace. CPU scheduling, GPU compute activity, memory traffic, and host-device transfers may peak at different moments, and transfer or synchronisation events can affect later GPU power responses. In addition, CPUs and GPUs expose different sensors, counters, and DVFS states. As a result, two samples with similar utilisation values may correspond to different power regimes when the workload phase, temperature, or data movement differs [18].

Existing power models can be broadly divided into architectural estimators, PMC-based statistical approaches, and data-driven runtime predictors. Architectural models offer interpretability, and PMC-based methods are efficient for

online estimation, but both often depend on device-specific calibration or simplified additive assumptions. Data-driven models improve flexibility, yet many existing designs still insufficiently consider cross-device lag, burst-like workload transitions, and long-range temporal dependence in CPU-GPU workloads [19]–[21].

To address these limitations, this study treats heterogeneous power prediction as a hardware-aware sequence learning problem. Runtime variables are organised into CPU-domain, GPU-domain, interaction-domain, and context-domain groups, and the proposed network learns their complementary relations through attention-based global encoding, local temporal convolution, and memory-based sequence modelling. The aim is to adapt a hybrid architecture to heterogeneous power behaviour rather than merely stacking Transformer, CNN, and LSTM modules.

The main contributions are listed below:

A CPU-GPU power prediction problem is formulated from a hardware-aware sequence perspective, explicitly separating CPU-domain, GPU-domain, interaction-domain, and context-domain runtime signals.

A Transformer-CNN-LSTM framework is constructed to model global feature coupling, short-range workload variation, and delayed temporal coordination through attention, convolution, and memory mechanisms.

The proposed framework is compared with deep-learning predictors and classical power-modelling baselines using the same dataset, training split, and evaluation metrics.

II. POWER CONSUMPTION MODEL OF CPU-GPU HETEROGENEOUS PLATFORM

This chapter lays the foundation for the prediction model at the physical and feature levels. Instead of reviewing hardware only as background information, it links the CPU-GPU structure, power decomposition, runtime conditions, architectural design, and software behaviour to the variables later used as model inputs.

A. POWER GENERATION MECHANISM

Power behaviour on a heterogeneous platform is determined by the interactions among compute units, the memory hierarchy, control logic, communication links, and power-management mechanisms. Understanding these sources helps explain that CPU, GPU, interaction, and context features should be jointly considered in a system-level predictor.

1) Composition Structure of CPU and GPU

For the CPU side, the variables used in this study mainly reflect utilisation, temperature, frequency, and active-core configuration. These states are affected by execution pipelines, cache behaviour, memory access, and voltage/frequency control. Therefore, the CPU contributes not only computational power but also scheduling and data orchestration overhead, which is especially relevant when the host coordinates GPU kernels and device transfers.

For the GPU side, power is dominated by parallel execution resources, memory traffic, on-chip storage, and external communication. Streaming multiprocessors, Compute Unified Device Architecture (CUDA) cores, memory hierarchy, and interconnection fabrics jointly determine how intensively the accelerator is used. Hence, GPU utilisation, memory usage, temperature, and power-limit information are directly related to the power state observed in the dataset.

At the platform level, the CPU and GPU are connected through high-speed communication links, including Peripheral Component Interconnect Express (PCIe). The CPU manages task launch, scheduling, and data movement, while the GPU processes data-parallel kernels. This organisation means that total power depends on computation and communication together, so host-device data-transfer intensity must be included when modelling system-level power.

B. POWER CONSUMPTION CALCULATION MODEL

Power formulas are used in this section to clarify the physical meaning of the prediction target. The purpose is not to build a purely analytical estimator, but to identify how CPU activity, GPU activity, leakage behaviour, and cross-device communication jointly contribute to the measured platform power.

1) CPU Power Consumption Calculation Model

CPU power is represented as the combination of switching-related, static, and leakage components [22]–[24]. This decomposition links the measured CPU state to both circuit activity and thermal behaviour, and the corresponding expression is given in (1).

$$P_{\mathrm{CPU}} = P_{\mathrm{dyn\_cpu}} + P_{\mathrm{sta\_cpu}} + P_{\mathrm{leak\_cpu}} \tag{1}$$

Among these terms, dynamic power describes the energy consumed by switching activity during operation and is calculated using (2).

$$P_{\mathrm{dyn\_cpu}} = \alpha \cdot C \cdot V^2 \cdot f \tag{2}$$

Where: α describes the switching activity level, C refers to the effective load capacitance, and the term $V^2 \cdot f$ reflects the voltage-frequency operating condition of CPU.

The static and leakage components account for steady-state current paths and thermal leakage effects. The leakage-temperature relation used in this work is shown in (3).

$$P_{\mathrm{leak\_cpu}} \propto e^{\beta T} \tag{3}$$

In (3), β is the temperature coefficient, and T denotes the chip junction temperature.

2) GPU Power Consumption Calculation Model

GPU power is described by separating compute, storage, interconnect, I/O, and background components. This structure reflects the fact that accelerator energy is affected not

only by kernel execution but also by memory access and communication, as summarised in (4).

$$P_{\mathrm{GPU}} = P_{\mathrm{core}} + P_{\mathrm{mem}} + P_{\mathrm{inter}} + P_{\mathrm{io}} + P_{\mathrm{sta\_gpu}} \quad (4)$$

Where: P_{core} is the power consumption of CUDA core, tensor core and other computing units, P_{mem} is the read and write power consumption of memory units, such as on-chip cache and video memory, P_{inter} is the data transmission power consumption of the on-chip interconnection bus, P_{io} is external communication power consumption, such as the PCIe interface, and $P_{\mathrm{sta\_gpu}}$ is static power consumption is caused by transistor leakage and steady-state current.

3) System-Level Power Consumption Calculation Model

The total platform power is finally expressed as a system-level formula that combines CPU, GPU, and communication power. This form is suitable for CPU-GPU workloads in which host-device transfers and synchronisation are frequent, as given in (5).

$$P_{\mathrm{total}} = P_{\mathrm{CPU}} + P_{\mathrm{GPU}} + P_{\mathrm{link}} \quad (5)$$

The communication term in (5) represents the power required for data exchange over the PCIe link between the CPU and GPU.

These formulations indicate that the target power value comprises contributions from the CPU, GPU, and communication. This interpretation motivates the later feature organisation, in which runtime variables are grouped by host activity, accelerator activity, interaction intensity, and workload context rather than treated as unrelated scalar inputs.

C. FACTORS AFFECTING POWER CONSUMPTION

The CPU-GPU power profile is affected by technology, operating state, architecture, and software workload simultaneously. A useful predictor should therefore include variables that reflect both device physics and runtime behaviour.

In this paper, feature analysis is used to connect measurable signals with the sources of platform power. CPU-domain features describe the host execution state, GPU-domain features describe the accelerator load, interaction-domain features describe the host-device coupling, and context-domain features provide workload information. These groups are retained in the subsequent windowed input representation.

1) Influence of Process Parameters

Process parameters define the lower-level power envelope of the device. Scaling may reduce capacitance and switching energy but can also increase leakage; threshold voltage, oxide thickness, and FinFET structure influence the balance between speed, static current, and leakage control under advanced technology nodes.

2) Impact of Working Conditions

During execution, power changes with the operating point and workload intensity. Voltage, frequency, temperature, utilisation, and data movement influence both switching and leakage behaviour, which is why they are informative variables in PMC-Stat and sensor-based power models. In the present dataset, these effects are reflected through CPU/GPU utilisation, temperature, frequency, memory usage, and transfer-related features.

Heterogeneous execution also creates phase-dependent behaviour. A job may alternate among CPU-dominant preparation, transfer-dominant coordination, and GPU-dominant computation. Consequently, identical instantaneous utilisation values can still lead to different power values when they occur after different transfer bursts, thermal states, or DVFS transitions, supporting the use of windowed sequence samples.

3) Architecture Design Impact

Architectural organisation affects how computation and data movement are translated into power. Parallelism changes the throughput-power trade-off; the cache and memory hierarchy affect access energy; interconnect design determines the cost of data exchange; and DVFS or power gating controls the available runtime management space.

4) Software Factors

Software determines how hardware resources are exercised over time. Algorithmic complexity, memory-access pattern, compiler optimisation, vectorisation, scheduling policy, and CPU-GPU load balance alter the sequence of runtime states, which explains why workload context is retained as part of the prediction input.

III. FEATURE EXTRACTION METHOD FOR CPU-GPU POWER CONSUMPTION PREDICTION

This chapter constructs the feature-processing pipeline used before model training. Because CPU-GPU power is affected by host load, accelerator utilisation, memory behaviour, task type, and data-transfer intensity, the raw records must be cleaned and scaled before they are organised into windowed samples for prediction.

A. HANDLING METHODS OF ABNORMAL AND MISSING FEATURES

The collected runtime data may contain spikes or missing entries due to sensor noise, acquisition mismatches, workload switching, or temporary communication interruptions. If these records are used directly, the model may learn to model abnormal fluctuations rather than stable power behaviour. Therefore, outlier correction and missing-value imputation are applied before normalisation [25], [26].

1) Handling of Outliers Based on the Z-Score Method

Outliers in this dataset refer to samples whose values are inconsistent with the surrounding distribution, such as

abrupt temperature or utilisation jumps. The Z-score method is used to detect these abnormal samples because it evaluates each value relative to the corresponding feature mean and standard deviation, making it suitable for multidimensional runtime features.

The Z-score used for outlier identification is computed by (6).

$$Z_i = \frac{x_i - \mu}{\sigma} \tag{6}$$

Where: x_i is the ith observed value of a given feature; The parameters μ and σ are obtained from all observations of that feature; Z_i measures the standard deviation of x_i from the feature distribution.

Samples whose absolute Z-score exceeds the threshold of 3 are regarded as abnormal. Instead of deleting them, the neighbouring-sample average is used for replacement so that the temporal continuity of the power sequence is preserved.

2) Missing Value Handling

Missing entries mainly originate from sensor failure, inconsistent sampling intervals, or data loss during task transitions. To avoid introducing unnecessary correlations, continuous and discrete features are imputed using different strategies based on their physical meanings.

For continuous variables such as CPU utilisation, temperature, GPU frequency, and data-transfer rate, missing values are imputed with the variable's mean. This choice keeps the feature scale stable and prevents large distribution shifts before normalisation.

For discrete variables such as task type or core-count configuration, zero-padding is used to represent unavailable or neutral categories without fabricating continuous trends.

After imputation, the dataset retains its temporal structure and provides complete feature vectors for subsequent normalisation and model input construction.

B. FEATURE NORMALISATION/STANDARDISATION

The input variables have different units and magnitudes, including frequency in GHz, power limit in watts, and utilisation in percentage. Without scaling, features with large numeric ranges could dominate the learning process. Thus, normalisation is performed to reduce scale differences among predictors [27].

Min-max normalisation is adopted in this study because it maps each feature into the [0,1] interval while preserving the relative ordering of samples. The operation used for this transformation is given by (7).

$$x_i^* = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{7}$$

Where: x_i^* is the scaled output and x_i refers to the original feature value; The lower and upper bounds of the feature are given by $\min(x)$ and $\max(x)$, which determines the range used in the min-max transformation.

After outlier correction and missing-value imputation, all 14 runtime features are scaled into a common numeric range. It prevents variables with larger magnitudes from dominating the learning process and improves training stability. The normalised records are then arranged by a sliding window rather than treated as isolated samples. For a sample ending at time t, the input is defined as $X_t = [x_{t-L+1}, x_{t-L+2}, \ldots, x_t] \in \mathbb{R}^{L\times 14}$, where L denotes the observation window length and each x_i denotes a 14-dimensional normalised feature vector at time i. The resulting sequence preserves both heterogeneous feature information and temporal order for the Transformer, CNN, and LSTM modules.

C. TRANSFORMER FEATURE CODING

CPU-GPU platform power is influenced by utilisation, temperature, frequency, task type, memory state, and transfer load within a time window. These variables interact nonlinearly, and their importance may change across workload phases. The Transformer encoder is therefore introduced to assign adaptive weights to heterogeneous runtime features and to capture long-range feature dependence [28].

In the implemented encoder, the windowed input sequence $X_t \in \mathbb{R}^{L\times 14}$ is projected into an embedding space before self-attention is applied. The attention module learns how utilisation, thermal state, frequency, memory behaviour, and transfer load interact within the observation window. The encoded representation is then forwarded to the CNN-LSTM block, and the attention weights are calculated by (8).

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{8}$$

Where: Q, K and V are produced from the encoded input by three projection operations. The scaling term d_k is introduced to control the magnitude of the dot-product attention score.

Transformer coding begins by projecting the normalised 14-dimensional feature vector into an embedding space with a unified dimension. This projection prepares the input for the subsequent attention operation, and the embedding transformation is expressed by (9).

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V \tag{9}$$

Where: X represents the normalised 14-dimensional input vector. The matrices W_Q,W_K and W_V project X into the Q, K and V vectors used by the attention module.

Multi-head self-attention then evaluates feature relations in several subspaces simultaneously. This design reduces the risk that a single attention head overlooks relevant CPU-GPU interactions, and the concatenation of attention heads is described by (10).

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, head_2, \ldots, head_h)W_O \tag{10}$$

Where: h is set to 4 in this study, so four attention-head outputs are concatenated and then projected by W_O.

Residual connection and layer normalisation further stabilise the encoded representation and support deeper feature transformation [29]. The combined operation is shown in (11).

$$LN(X + \mathrm{MultiHead}(Q, K, V)) \tag{11}$$

Where: LN represents the layer normalisation operation, through which the encoded high-order eigenvector is output as the input of the subsequent prediction model. After feature encoding, the Transformer output can be written as $H_t \in \mathbb{R}^{L\times d}$, where L denotes the observation-window length and d represents the embedding dimension. This output retains the original temporal order of the original sample and provides a unified feature representation for the subsequent CNN-LSTM prediction model.

The Transformer encoder converts raw normalised features into a more expressive representation of heterogeneous power states. The subsequent CNN-LSTM module then uses this representation to refine local fluctuations and temporal trends.

Based on the feature-processing results above, this paper evaluates CNN, LSTM, CNN-LSTM, and Transformer-CNN-LSTM under the CPU-GPU power-prediction task. The following chapter focuses on how these models are constructed and how the proposed hybrid framework combines global, local, and temporal modelling ability (Table I).

TABLE I. Input Features of the CPU-GPU Heterogeneous Platform and Their Roles in the Windowed Feature Representation

Serial number	Dataset characteristics	Encoded format
1	CPU_Utilization	CPU-related continuous feature; normalised and placed in the windowed heterogeneous feature vector.
2	CPU_Temp	CPU-related continuous feature; normalised and placed in the windowed heterogeneous feature vector.
3	CPU_Freq_GHz	CPU-related continuous feature; normalised and placed in the windowed heterogeneous feature vector.
4	CPU_Core_Count	CPU active-core feature; numerically encoded and placed in the windowed heterogeneous feature vector.
5	GPU_Util	GPU-related continuous feature; normalised and placed in the windowed heterogeneous feature vector.
6	GPU_Temp	GPU-related continuous feature; normalised and placed in the windowed heterogeneous feature vector.
7	GPU_Mem_Usage	GPU-related continuous feature; normalised and placed in the windowed heterogeneous feature vector.
8	GPU_Power_Limit_W	GPU-related continuous feature; normalised and placed in the windowed heterogeneous feature vector.
9	CPU_GPU_Data_Transfer_GBps	Cross-device interaction feature; normalised and placed in the windowed heterogeneous feature vector.
10	Task_Typ	Categorical task feature; numerically encoded and placed in the windowed heterogeneous feature vector.
11	Synergy_Load	Coupling-related feature; normalised and placed in the windowed heterogeneous feature vector.
12	Auxiliary feature 1	Auxiliary continuous feature; normalised and placed in the windowed heterogeneous feature vector.
13	Auxiliary feature 2	Auxiliary continuous feature; normalised and placed in the windowed heterogeneous feature vector.
14	Auxiliary feature 3	Auxiliary continuous feature; normalised and placed in the windowed heterogeneous feature vector.

IV. POWER CONSUMPTION PREDICTION METHOD BASED ON TRANSFORMER-CNN-LSTM ALGORITHM

Based on the preprocessed heterogeneous feature sequences, this chapter presents the prediction models used in the experiment. CNN and LSTM are used as single-mechanism baselines, CNN-LSTM is used as a local-temporal hybrid baseline, and Transformer-CNN-LSTM is developed as the proposed global-local-temporal framework.

A. POWER CONSUMPTION PREDICTION METHOD OF CPU-GPU BASED ON CNN ALGORITHM

The CNN model is used to test how much predictive information can be extracted from local patterns in the time-feature matrix. Its convolution kernels scan adjacent time steps and feature positions, so this baseline mainly reflects short-range variation rather than long-term sequence dependence.

1) Core Structure of CNN Model

The CNN predictor comprises input handling, convolutional extraction, batch normalisation, activation, pooling, and fully connected components [30]. Their roles in the present power-prediction task are summarised below.

The input layer receives the windowed input sample $X_t \in \mathbb{R}^{L\times 14}$, where L denotes the observation-window length, and 14 is the feature dimension. The input is organised as a time-feature matrix so that the following convolution layer can directly extract local variation patterns from adjacent states.

The convolution layer extracts local patterns from the time-feature matrix. Two convolutional layers with kernel size [3,1] are used, enabling the model to respond to short bursts across neighbouring samples. The first layer produces 64 channels and uses the same padding to preserve temporal length, while the second layer produces 128 channels for higher-level local features. The convolution operation is defined by (12).

$$y_{i,j} = \sum_{k=1}^{K}\sum_{l=1}^{L} w_{k,l}\cdot x_{i+k-1,j+l-1} + b \tag{12}$$

Where: $y_{i,j}$ denotes the convolution output at position (i,j); $w_{k,l}$ represents the convolution-kernel weight; $x_{i+k-1,j+l-1}$ is the input-feature region; b is the bias term; K and L specify the convolution-kernel height and width.

Batch normalisation is applied after each convolutional layer to reduce internal distribution drift during training and improve convergence stability. Its calculation is expressed by (13) [31].

$$BN(x) = \gamma \cdot \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta \tag{13}$$

Where: μ_B and σ_B^2 are computed from the current mini-batch features. The learnable parameters γ and β rescale and shift the normalised output, while ϵ=1e-5 prevents division by zero.

The rectified linear unit (ReLU) activation provides a nonlinear mapping and is given by (14).

$$f(x) = \max(0, x) \tag{14}$$

Such an activation increases the ability to represent nonlinearities while keeping the computation simple and reducing saturation-related gradient attenuation.

Maximum pooling with a [2,1] kernel and [2,1] stride is used to reduce the temporal-feature dimension while retaining dominant local responses. The pooling operation is expressed by (15).

$$\mathrm{MaxPool}(x)_{i,j} = \max_{k,l \in \mathrm{kernel}} x_{i+k-1,j+l-1} \tag{15}$$

Where: kernel is the pooled core (size [2,1]), $x_{i+k-1,j+l-1}$ is the local characteristic area covered by the pooled core, and $\mathrm{MaxPool}(x)_{i,j}$ is the output characteristic value after pooling.

After pooling, the feature maps are flattened, then fed into two fully connected layers with output dimensions of 256 and 128. Dropout is applied to the fully connected part during training [32]. The final output layer converts the learned representation into the predicted power value.

2) Model Parameters and Solution Process

CNN training optimises convolution, bias, and fully connected parameters by reducing the difference between predicted and measured power. The loss function is MSE, as formulated in (16).

$$L = \frac{1}{N} \sum_{i=1}^{N} (y_i - \widehat{y}_i)^2 \tag{16}$$

Where: N is the number of samples; y_i is the actual power consumption value of the ith sample; $\widehat{y}_i$ is the predicted power consumption value of the ith sample.

The CNN is trained with Adam, with an initial learning rate of 1e-3. Piecewise learning-rate decay, L2 regularisation with coefficient 1e-4, and the specified validation frequency are used to control convergence and reduce overfitting [33].

During training, each mini-batch passes through the network, the MSE loss is evaluated, and gradients are backpropagated to update convolutional and fully connected parameters. The process stops when the maximum epoch number is reached or the loss becomes stable.

Two learning factors, C_1 and C_2, are introduced to separately regulate the update magnitudes of convolution and fully connected weights. With Adam optimisation, the corresponding update formulas are given in (17) and (18).

$$W_{\mathrm{conv},t+1} = W_{\mathrm{conv},t} - C_1 \cdot \eta \cdot (\nabla L \cdot W_{\mathrm{conv},t} + \lambda W_{\mathrm{conv},t}) \tag{17}$$

$$W_{\mathrm{fc},t+1} = W_{\mathrm{fc},t} - C_2 \cdot \eta \cdot (\nabla L \cdot W_{\mathrm{fc},t} + \lambda W_{\mathrm{fc},t}) \tag{18}$$

Where: $W_{\mathrm{conv},t+1}$, $W_{\mathrm{fc},t}$ correspond to the convolutional and fully connected weight matrices. The learning rate controls the update η, the loss gradient ∇L, and the L2 coefficient λ. The constants C_1 and C_2 adjust the update scales of the two parameter groups.

B. CPU-GPU POWER CONSUMPTION PREDICTION METHOD BASED ON CNN-LSTM

CNN captures local patterns effectively, but CPU-GPU power traces also contain delayed responses and workload-phase transitions. Long short-term memory (LSTM) is therefore adopted to learn temporal dependencies in local features [34], and the CNN-LSTM baseline combines convolutional extraction with recurrent sequence modelling.

In the CNN-LSTM model, the CNN block first converts the windowed heterogeneous input into local feature maps. These maps are reorganised as a feature sequence and processed by the LSTM block, after which a fully connected layer produces the final power estimate.

For local feature extraction, the windowed sample is passed through the CNN module described in Section 4.1. The final convolutional block is kept as structured feature maps rather than being immediately flattened.

For temporal modelling, the CNN feature maps are rearranged into a local sequence and fed into two bidirectional LSTM layers. The first layer constrains 128 hidden units with sequence output, and the second constrains 64 hidden units with last-step output. The LSTM gate operations are defined in (19)–(24). The forget and input gates regulate retained and newly added information, the candidate and memory states update the internal state, and the output gate generates the hidden representation for the next step.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{19}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{20}$$

$$\widetilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{21}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t \tag{22}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{23}$$

$$h_t = o_t \odot \tanh(C_t) \tag{24}$$

In Eqs. (19)-(24), W and b are trainable parameters of the LSTM gates. The function σ and tanh provide nonlinear

transformations, the circled-dot symbol indicates element-wise multiplication, and the previous hidden state is combined with the current input feature during state updating.

The LSTM output is passed to the fully connected layer to map the temporal representation into the final power prediction.

The CNN-LSTM model follows the same MSE loss, Adam optimiser, and L2 regularisation strategy as the CNN baseline. Its learning rate and maximum epoch count are set to 5e-4 and 10, respectively, to balance temporal modelling complexity and overfitting risk. The LSTM weight update is given by (25).

$$W_{\mathrm{LSTM},t+1} = W_{\mathrm{LSTM},t} - \eta \cdot \left(\nabla L \cdot W_{\mathrm{LSTM},t} + \lambda W_{\mathrm{LSTM},t}\right) \quad (25)$$

Where: $W_{\mathrm{LSTM},t}$ is the weight matrix of the LSTM layer at time t (including the weight of each gate control), and η is the learning rate of the LSTM layer (5e-4).

By combining convolutional local extraction with recurrent memory, the CNN-LSTM baseline provides a stronger reference than either CNN or LSTM alone for CPU-GPU power prediction.

C. CPU-GPU POWER CONSUMPTION PREDICTION PROCESS AND FRAMEWORK BASED ON TRANSFORMER-CNN-LSTM ALGORITHM

The proposed Transformer-CNN-LSTM model is designed to capture heterogeneous power behaviour across three stages. Transformer encoding first learns global relations among runtime features, CNN extraction then identifies local fluctuations in the encoded sequence, and LSTM modelling finally captures temporal accumulation and phase transitions. This ordering follows the data characteristics rather than simply stacking common neural-network modules.

1) Overall Structure of Prediction Frame

The Transformer-CNN-LSTM framework contains feature preprocessing, Transformer feature encoding, CNN-LSTM fusion and prediction, and model optimisation modules. These modules form a pipeline from raw runtime features to the final power estimate.

The preprocessing module corresponds to the procedures described in Chapter 3, including Z-score-based outlier correction, mean/zero-value imputation, and min-max normalisation. The 14-dimensional raw variables are converted into standardised windowed samples for model input.

The Transformer module receives the preprocessed windowed sample $X_t \in \mathbb{R}^{L\times 14}$. Linear projection maps the sample to an embedding sequence $E_t \in \mathbb{R}^{L\times d}$. Multi-head attention then estimates feature relations across the observation window, and a residual connection with layer normalisation produces the encoded representation $H_t \in \mathbb{R}^{L\times d}$. The Transformer encoder weight update is given in (26).

$$W_{t+1} = W_t - \eta \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon} \quad (26)$$

Where: W_t is the Transformer encoder weight matrix at iteration t; The learning rate η is set to 5e-4, $\widehat{m}_t$ and $\widehat{v}_t$ are the first- and second-moment estimates in Adam. ϵ is used to avoid a denominator of 0 and ensure the stability of the weight update.

In the CNN-LSTM fusion module, the encoded representation enters the CNN block for local fluctuation extraction. The resulting feature maps are reorganised into a sequence, processed by an LSTM, and the fully connected layer then outputs the predicted platform power.

The optimisation module uses Adam, MSE loss, L2 regularisation, and piecewise learning-rate decay. These settings are used to stabilise training while keeping the hybrid model computationally manageable.

2) Detailed Description of the Prediction Process

Figure 1 illustrates the complete workflow, including preprocessing, Transformer encoding, CNN local extraction, LSTM temporal modelling, and regression output.

The prediction workflow starts with 14-dimensional runtime features and one-dimensional total-power labels. After outlier correction, missing-value filling, and min-max normalisation, the standardised windowed samples are encoded by the Transformer, refined by the CNN, and modelled by the LSTM. Iterative training minimises the MSE between predicted and measured power until convergence.

This workflow integrates preprocessing, global encoding, local refinement, and temporal modelling into a single supervised prediction process, enabling the model to represent heterogeneous coupling and time-dependent power changes more comprehensively than a single modelling approach.

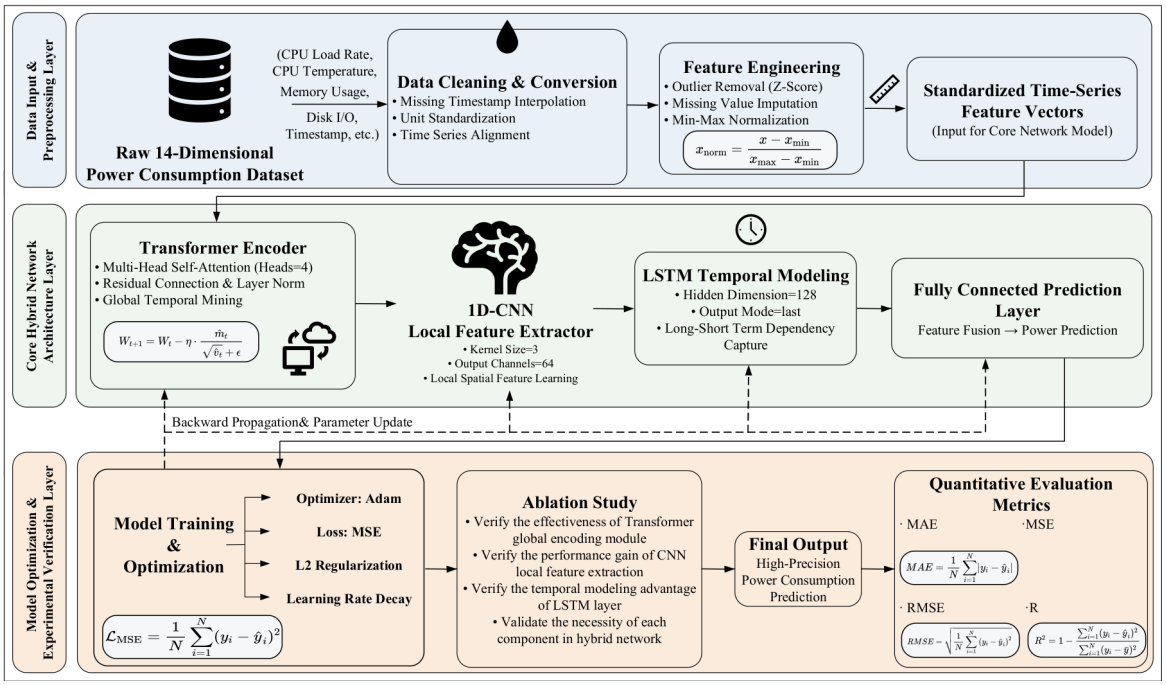

FIGURE 1. Overall Framework of the Proposed Transformer-CNN-LSTM Hybrid Model for Power Prediction

V. SIMULATION ANALYSIS

This chapter evaluates six CPU-GPU power prediction models: CNN, LSTM, CNN-LSTM, Transformer-CNN-LSTM, PMC-Stat, and GPUWattch-Inspired. All models are trained or calibrated on the same dataset and evaluation setting so that prediction accuracy and visualisation results can be compared consistently.

A. PARAMETER SETTING

Simulation parameters are divided into dataset settings, model settings, and hardware/software environment settings. They are selected based on the scale of the collected dataset and the training requirements of the compared models.

1) Data Set Source and Division

The dataset is collected from a self-built CPU-GPU heterogeneous platform across multiple workload types, including deep learning training, high-performance computing, and data processing. Each record contains runtime features and the corresponding total power measurements.

The dataset includes 10000 samples. Each sample contains 14 input variables and one total-power output label, and the feature definitions and units are listed in Table II. The last three variables are retained as auxiliary descriptors for improving model fitting.

TABLE II. Specific Definitions and Units of Input Features in the Power Consumption Dataset

Serial number	Dataset characteristics	Interpretation, units
1	CPU_Utilization	CPU load rate,%
2	CPU_Temp	CPU temperature, °C
3	CPU_Freq_GHz	CPU frequency, GHz
4	CPU_Core_Count	Number of active CPU cores
5	GPU_Util	GPU load rate,%
6	GPU_Temp	GPU temperature, °C
7	GPU_Mem_Usage	GPU video memory usage,%
8	GPU_Power_Limit_W	Configured GPU power limit, W
9	CPU_GPU_Data_Transfer_GBps	CPU-GPU data transfer rate, GB/s
10	Task_Typ	Task type, discrete
11	Synergy_Load	Collaborative load factor
12	Auxiliary feature 1	-
13	Auxiliary feature 2	-
14	Auxiliary feature 3	-

The dataset is split into 8,000 training and 2,000 test samples using an 8:2 hold-out split. The random seed is fixed at 123 to make the partition reproducible.

2) Model Parameter Setting

All six models are configured on the same experimental basis, so differences in performance mainly stem from their modelling structures.

For general training, the maximum epoch number is 20, the mini-batch size is 128, the validation frequency is 10, the learning-rate drop factor is 0.3, the drop period is 3, and the L2 regularisation coefficient is 1e-4. Adam and MSE are used for the neural network models, whereas PMC-Stat and GPUWattch-Inspired are calibrated separately using the same training split.

For the CNN baseline, the initial learning rate is 1e-3. It uses two [3,1] convolution layers with 64 and 128

output channels, and then two fully connected layers with dimensions 256 and 128. Dropout probabilities are set to 0.3 and 0.2, and the ReLU activation function is used.

For the LSTM baseline, the initial learning rate is set to 1e-3. Two bidirectional LSTM layers are used: the first constrains 128 hidden units and outputs a sequence, whereas the second constrains 64 hidden units and outputs the last state. The dropout probabilities are 0.3 and 0.2.

For the CNN-LSTM baseline, the initial learning rate is set to 5e-4, and the maximum number of epochs is 10. The CNN settings follow the single-CNN baseline, while the LSTM block uses two bidirectional layers with hidden dimensions of 128 and 64, and output modes of sequence and last, respectively.

For the Transformer-CNN-LSTM model, the initial learning rate is set to 5e-4, and the maximum number of epochs is 10. The Transformer encoder uses four attention heads and a 256-dimensional embedding. The CNN block uses a 3×3 convolution with 64 channels, and the LSTM block uses a single hidden layer with 128 units and dropout of 0.2.

These hyperparameters balance representation capacity and training cost. The attention heads model leverages relations across multiple subspaces; the embedding dimension increases the representation of the 14-dimensional input; the CNN kernel focuses on short-term temporal changes; and the LSTM hidden size captures accumulated sequence information.

For PMC-Stat, the train-test split remains the same as that used for the neural models. A statistical estimator is fitted from runtime features such as utilisation, temperature, frequency, memory usage, and interaction variables, and its coefficients are calibrated only on the training set.

For GPUWattch-Inspired, total power is approximated through calibrated compute, memory, interconnect, and background terms. The model is calibrated and tested on the same test set to maintain comparability.

3) Simulation Environment Parameters

The experiments are implemented in MATLAB R2023a on a self-built platform configured with an Intel Core i7-12700H CPU, an NVIDIA RTX 3060 GPU, and 16 GB Double Data Rate 5 (DDR5) memory, a 512 GB solid-state drive (SSD), and Windows 11 Home. GPU acceleration is enabled during training to shorten simulation time.

4) Experimental Scheme Design

Six experimental schemes are constructed with the same dataset, environment, and general parameters. The schemes correspond to CNN, bidirectional LSTM, CNN-LSTM, Transformer-CNN-LSTM, PMC-Stat, and GPUWattch-Inspired prediction, respectively. Their differences lie only in model structure and model-specific parameters.

B. POWER CONSUMPTION PREDICTION RESULTS

Four evaluation metrics are adopted: mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE), and coefficient of determination (R^2). Lower MSE, RMSE, and MAE indicate smaller prediction errors, whereas an R^2 value closer to 1 indicates better fitting. Table III reports the quantitative comparison, and the subsequent figures visualise the prediction behaviour.

1) Comparison of Prediction Indexes

Table III reveals a clear separation among the six models. Transformer-CNN-LSTM obtains the minimum MSE, RMSE, and MAE and the maximum R^2, with values of 50.313118, 7.093174, 4.786434, and 0.979035, respectively. CNN-LSTM ranks second, indicating the benefit of combining local and temporal modelling. The GPUWattch-Inspired baseline performs slightly better than PMC-Stat, but both classical baselines are less able to follow high-power fluctuations than the proposed hybrid model.

TABLE III. Comparison of Prediction Performance Metrics among Six Deep Learning and Classical Baseline Models

Model	MSE	RMSE	MAE	R^2
CNN	1100.000000	33.166248	26.500000	0.520000
LSTM	1740.000000	41.713307	34.000000	0.260000
CNN-LSTM	600.000000	24.494897	18.000000	0.760000
PMC-Stat	1740.000000	41.713307	33.300000	0.260000
GPUWattch-Inspired	1660.000000	40.743098	32.300000	0.300000
Transformer-CNN-LSTM	50.313118	7.093174	4.786434	0.979035

2) Visualisation of Prediction Results

To visualise the prediction behaviour, 200 test samples are selected and plotted. Figure 2 compares CNN-LSTM and Transformer-CNN-LSTM; Figure 3 shows the two classical baselines; Figure 4 summarises the four evaluation metrics for all models; and Figure 5 compares representative true-versus-predicted scatter distributions.

Figure 2 shows that the Transformer-CNN-LSTM curve follows the measured power more closely than the CNN-LSTM curve, especially near sharp rises and drops. CNN-LSTM captures the general trend but shows larger deviations in regions with strong fluctuations. This visual comparison aligns with Table III and demonstrates the benefit of combining global feature encoding with local and temporal modelling.

Figure 3 indicates that the two classical baselines mainly capture average power trends. PMC-Stat produces a smoother curve and misses sharp transitions, whereas GPUWattch-Inspired reacts slightly more to fluctuations because of its component-oriented calibration. Both baselines still deviate more from the measured curve than Transformer-CNN-LSTM.

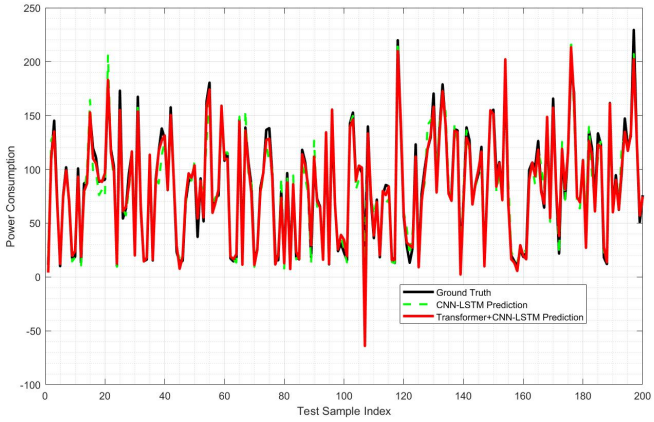

FIGURE 2. Comparison between Actual Power Consumption Values and Predicted Values of 200 Selected Test Samples (Note: in Figure 2, ground truth denotes the measured power, CNN-LSTM and Transformer-CNN-LSTM denote the predicted values, the horizontal axis represents the test-sample index, and the vertical axis represents power.)

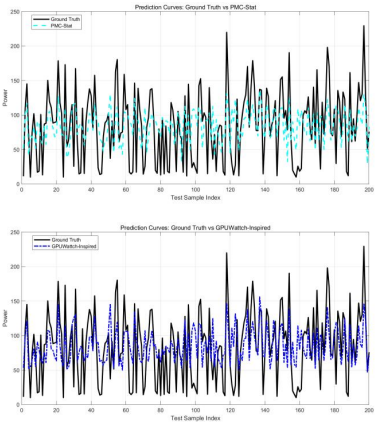

FIGURE 3. Actual and Predicted Power Consumption for PMC-Stat and GPUWattch-Inspired Models (Note: in Figure 3, the upper subfigure corresponds to PMC-Stat and the lower subfigure corresponds to GPUWattch-Inspired. Ground truth denotes the measured power; the horizontal and vertical axes denote sample index and power, respectively.)

Figure 4 summarises the metric-level comparison. Transformer-CNN-LSTM has the smallest error values and the largest R^2 among the six models. CNN-LSTM remains stronger than CNN and LSTM alone, while GPUWattch-Inspired shows a modest advantage over PMC-Stat.

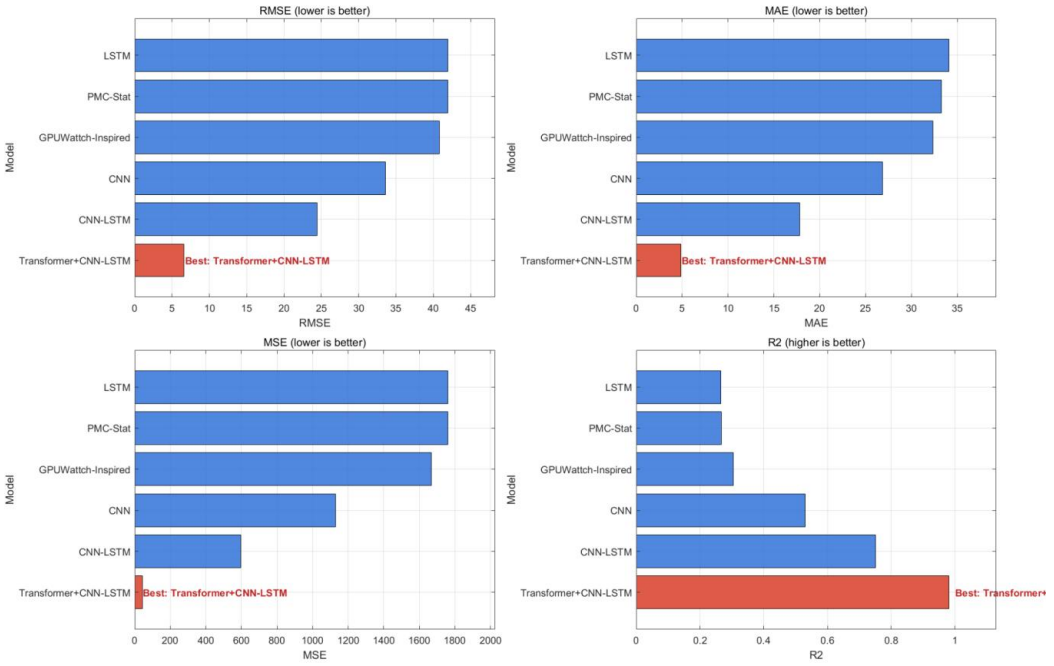

FIGURE 4. Comparison of Four Evaluation Metrics among Six Prediction Models

Figure 5 shows that Transformer-CNN-LSTM predictions are closer to the ideal y=x line. In contrast, PMC-Stat and GPUWattch-Inspired exhibit greater dispersion in the medium- and high-power ranges, consistent with their larger numerical errors.

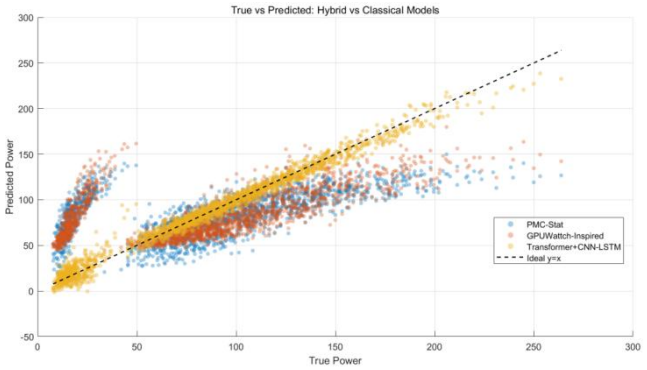

FIGURE 5. True-Versus-Predicted Scatter Comparison between Representative Classical and HYBRID MODELS

C. COMPARATIVE ANALYSIS OF METHODS

This section compares three neural baselines and two classical baselines under the same runtime dataset. PMC-Stat and GPUWattch-Inspired are calibrated using the same training split as the neural models, enabling direct numerical comparison. McPAT is retained as a literature reference because it requires a more complete simulator-level configuration than the current dataset provides.

The improvement of Transformer-CNN-LSTM over CNN, LSTM, CNN-LSTM, PMC-Stat, and GPUWattch-Inspired is measured for MSE, RMSE, MAE, and R^2. The percentage calculation is defined in (27).

$$\eta = \frac{M_B - M_{TCL}}{M_B} \times 100\% \tag{27}$$

Where: η is the performance improvement percentage; M_B is the evaluation metric value of the baseline comparison model (CNN, LSTM, CNN-LSTM, PMC-Stat, GPUWattch-Inspired), while M_{TCL} is the corresponding metric value of Transformer-CNN-LSTM.

For R^2, the improvement is computed by subtracting the baseline R^2 from the proposed model R^2 and dividing by the baseline R^2, so that a positive value indicates a better fit.

The resulting improvement percentages are reported in Table IV.

1) Performance Improvement Percentage Analysis

According to Table IV, Transformer-CNN-LSTM improves all four metrics relative to each baseline. Compared with CNNs, the large reductions in MSE, RMSE, and MAE indicate that global attention and temporal modelling compensate for the limited receptive field of a single CNN. Compared with LSTM, the improvement indicates that temporal recurrence alone cannot fully capture heterogeneous power variation. Compared with CNN-LSTM, the remaining gap demonstrates the contribution of Transformer-based global feature coding. The improvements over PMC-Stat and GPUWattch-Inspired further show that the proposed model better captures nonlinear coupling and runtime sequence dynamics.

TABLE IV. Performance Improvement Percentages of the Proposed Transformer-CNN-LSTM Algorithm Compared with Five Baseline Models

Comparison algorithm	MSE improvement percentage	RMSE improvement percentage	MAE improvement percentage	R^2 percentage increase
CNN	95.43	78.61	81.94	88.28
LSTM	97.11	83.00	85.92	276.55
CNN-LSTM	91.61	71.04	73.41	28.82
PMC-Stat	97.11	83.00	85.63	276.55
GPUWattch-Inspired	96.97	82.59	85.18	226.35

2) Algorithm Performance Summary

The six-model comparison shows that prediction performance improves as models capture more aspects of the data. CNN focuses on local variation, LSTM focuses on temporal evolution, and CNN-LSTM combines these two mechanisms. PMC-Stat and GPUWattch-Inspired provide interpretable statistical or component-oriented references, but their simplified coupling assumptions limit their accuracy under lag and burst effects. Transformer-CNN-LSTM integrates global association, local feature extraction, and temporal memory, which accounts for its overall best performance.

Overall, the proposed Transformer-CNN-LSTM model provides the most accurate prediction among the evaluated neural and classical baselines, making it suitable for high-precision power estimation on CPU-GPU heterogeneous platforms.

3) Discussion with Classical Power-Modelling Approaches

The comparison with PMC-Stat and GPUWattch-Inspired shows that classical models remain useful for interpretability, because they connect prediction results with counters or subsystem-level power components. However, their direct numerical performance is weaker when the workload includes lagged cross-device behaviour and rapid fluctuations. The proposed framework is therefore better suited to end-to-end prediction from the collected runtime dataset, while classical models remain valuable references for future simulator-coupled analysis.

VI. CONCLUSIONS

This paper presents a Transformer-CNN-LSTM framework for power prediction on a CPU-GPU heterogeneous platform. The framework combines system-level power analysis, feature preprocessing, sliding-window organisation, Transformer encoding, CNN local extraction, and LSTM temporal modelling. It achieves MSE, RMSE, MAE, and R^2 values of 50.313118, 7.093174, 4.786434, and 0.979035, outperforming CNN, LSTM, CNN-LSTM, PMC-Stat, and GPUWattch-Inspired baselines.

The framework is practically feasible because it uses observable runtime variables, including utilisation, temperature, frequency, memory usage, and interaction information. These inputs make the model applicable to online power estimation, scheduling support, energy-efficiency optimisation, and power-aware platform management. The current study still has limitations. The experiments are performed on a single dataset and platform configuration, so cross-platform generalisation requires further validation. Although PMC-Stat and GPUWattch-Inspired are included, a fully coupled simulator-level comparison with McPAT-derived models is not yet implemented. Physical constraints such as subsystem additivity, non-negativity, and thermal-power consistency also remain to be incorporated into the learning objective.

Future work will extend the evaluation to more platforms and workload types, incorporate fully calibrated architectural baselines, and introduce physics-informed constraints, lightweight deployment, and online updating to improve interpretability, robustness, and deployment value.

REFERENCES

[1] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," ACM Comput. Surv., vol. 47, no. 4, p. 69, 2015.
[2] K. O'Brien, I. Pietri, R. Reddy, et al., "A survey of power and energy predictive models in HPC systems and applications," ACM Comput. Surv., vol. 50, no. 3, p. 37, 2017.
[3] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding GPU power: A survey of profiling, modeling, and simulation methods," ACM Comput. Surv., vol. 49, no. 3, p. 41, 2016.
[4] S. Mittal and J. S. Vetter, "A survey of methods for analysing and improving GPU energy efficiency," ACM Comput. Surv., vol. 47, no. 2, p. 19, 2013.
[5] S. Dey, S. Isuwa, S. Saha, et al., "CPU-GPU-memory DVFS for power-efficient MPSoC in mobile cyber physical systems," Future Internet, vol. 14, no. 3, p. 91, 2022.
[6] K. Raju and N. N. Chiplunkar, "A survey on techniques for cooperative CPU-GPU computing," Sustain. Comput.: Inform. Syst., vol. 19, pp. 72–85, 2018.
[7] J. Shen, A. L. Varbanescu, Y. Lu, et al., "Workload partitioning for accelerating applications on heterogeneous platforms," IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 9, pp. 2766–2780, 2016.
[8] S. Hong and H. Kim, "An integrated GPU power and performance model," in Proc. 37th Annu. Int. Symp. Comput. Archit., 2010, pp. 280–289.
[9] H. Nagasaka, N. Maruyama, A. Nukada, et al., "Statistical power modeling of GPU kernels using performance counters," in Proc. Int. Conf. Green Comput., 2010, pp. 115–122.
[10] J. Lim, N. B. Lakshminarayana, H. Kim, et al., "Power modeling for GPU architectures using McPAT," ACM Trans. Des. Autom. Electron. Syst., vol. 19, no. 3, p. 26, 2014.
[11] J. Leng, T. Hetherington, A. Eltantawy, et al., "GPUWattch: Enabling energy optimisations in GPGPUs," in Proc. 40th Annu. Int. Symp. Comput. Archit., 2013, pp. 487–498.
[12] V. Kandiah, S. Peverelle, M. Khairy, et al., "AccelWattch: A power modeling framework for modern GPUs," in Proc. MICRO-54: 54th Annu. IEEE/ACM Int. Symp. Microarch., 2021, pp. 738–753.
[13] S. Mazzola, S. Ara, T. Benz, et al., "Data-driven power modeling and monitoring via hardware performance counter tracking," J. Syst. Archit., vol. 167, p. 103504, 2025.
[14] A. Shahid, M. Fahad, R. R. Manumachu, et al., "A comparative study of techniques for energy predictive modeling using performance monitoring counters on modern multicore CPUs," IEEE Access, vol. 8, pp. 143306–143332, 2020.
[15] R. Bertran, Y. Becerra, D. Carrera, et al., "Energy accounting for shared virtualised environments under DVFS using PMC-based power models," Future Gener. Comput. Syst., vol. 28, no. 2, pp. 457–468, 2012.
[16] Z. Wang, L. Zheng, Q. Chen, et al., "CPU+GPU scheduling with asymptotic profiling," Parallel Comput., vol. 40, no. 2, pp. 107–115, 2014.
[17] Y. Wen, Z. Wang, and M. F. P. O'Boyle, "Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms," in Proc. 21st Int. Conf. High Perform. Comput., 2014, pp. 1–10.
[18] M. N. L. Carvalho, A. Simitsis, A. Queralt, et al., "Workload placement on heterogeneous CPU-GPU systems," Proc. VLDB Endow., vol. 17, no. 12, pp. 4241–4244, 2024.
[19] S. Li, J. H. Ahn, R. D. Strong, et al., "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarch., 2009, pp. 469–480.
[20] X. Mei, Q. Wang, and X. Chu, "A survey and measurement study of GPU DVFS on energy conservation," Digit. Commun. Netw., vol. 3, no. 2, pp. 89–100, 2017.
[21] H. Wang and Y. Cao, "Predicting power consumption of GPUs with fuzzy wavelet neural networks," Parallel Comput., vol. 44, pp. 18–36, 2015.
[22] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," IEEE J. Solid-State Circuits, vol. 27, no. 4, pp. 473–484, 1992.
[23] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," Proc. IEEE, vol. 91, no. 2, pp. 305–327, 2003.
[24] S. Borkar, "Design challenges of technology scaling," IEEE Micro, vol. 19, no. 4, pp. 23–29, 1999.
[25] V. J. Hodge and J. Austin, "A survey of outlier detection methodologies," Artif. Intell. Rev., vol. 22, no. 2, pp. 85–126, 2004.
[26] J. M. Jerez, I. Molina, P. J. Garcia-Laencina, et al., "Missing data imputation using statistical and machine learning methods in a real breast cancer problem," Artif. Intell. Med., vol. 50, no. 2, pp. 105–115, 2010.
[27] K. M. Sujon, R. Hassan, Z. T. Towshi, et al., "When to use standardisation and normalisation: Empirical evidence from machine learning models and XAI," IEEE Access, vol. 12, pp. 135300–135314, 2024.
[28] A. Vaswani, N. Shazeer, N. Parmar, et al., "Attention is all you need," in Adv. Neural Inf. Process. Syst. 30, 2017, pp. 5998–6008.
[29] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalisation," arXiv:1607.06450, 2016.
[30] Y. LeCun, L. Bottou, Y. Bengio, et al., "Gradient-based learning applied to document recognition," Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.
[31] S. Ioffe and C. Szegedy, "Batch normalisation: Accelerating deep network training by reducing internal covariate shift," in Proc. 32nd Int. Conf. Mach. Learn., 2015, pp. 448–456.
[32] N. Srivastava, G. Hinton, A. Krizhevsky, et al., "Dropout: A simple way to prevent neural networks from overfitting," J. Mach. Learn. Res., vol. 15, no. 56, pp. 1929–1958, 2014.
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimisation," arXiv:1412.6980, 2014.
[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.