# HighRPM: Combining Integrated Measurement and Sofware Power Modeling for High-Resolution Power Monitoring

Xinxin Qi[1], Juan Chen[1*], Yong Dong[1*], Yuan Yuan[1], Tao Xu[1], Rongyu Deng[1], Zekai Li[1], Kexing Zhou[1], Zheng Wang[2]
[1] NUDT, China [2] University of Leeds, United Kingdom
{qixinxin19,juanchen,yongdong,yuanyuan,xt.0320,dengrongyu21,zekaili,zhoukexing}@nudt.edu.cn,z.wang5@leeds.ac.uk

## ABSTRACT

In an era where power and energy are the first-class constraints of computing systems, accurate power information is crucial for energy efficiency optimization in high-performance computing (HPC) systems. Existing power monitoring techniques rely on either software-centric power models that suffer from poor accuracy or integrated hardware measurement schemes that have a low reading update frequency and coarse granularity. These result in a low spatiotemporal resolution for power monitoring. This paper introduces HighRPM, a new method for accurately measuring power consumption on HPC systems. HighRPM combines coarse-grained power sensor readings and software power modeling techniques to improve temporal and spatial resolutions. To provide high-frequent power readings in the temporal domain, HighRPM employs statistical modeling and machine learning techniques to predict the long-term power trend and the short-term fluctuations in power consumption. To improve spatial coverage, HighRPM takes low-time resolution node-level power consumption and uses a neural network to distribute the power readings to lower-level computing components like CPUs and memory components. We evaluate HighRPM by applying it to both ARM-based and X86-based platforms. Experimental results show that HighRPM improves time resolution by 10 times, provides accurate readings for CPUs and memory, and reduces error by 7-24% compared to other power modeling methods.

## CCS CONCEPTS

• **General and reference → Power modeling**.

## KEYWORDS

spatiotemporal resolution, power monitoring, power model, integrated measurement

## 1 INTRODUCTION

Accurate and timely power monitoring is essential for effective energy management and optimization in high-performance computing (HPC) systems [28]. Power readings help the system quickly respond to changes in workload demand and behavior, which is important for efficient workload scheduling, reducing energy consumption, preventing overheating, and maintaining system stability [16, 18].

Existing power monitoring solutions generally fall into three categories: (i) direct measurement using external power instruments [19], (ii) integrated measurement through power sensors, baseboard management controller (BMC), or FPGA components, and (iii) software-centric power modeling techniques like Intel's running average power limit (RAPL) interface [30]. However, none of these approaches adequately balance accuracy, spatiotemporal resolution, scalability, and deployment costs, which are critical for the broad adoption of power monitoring [20, 21].

High-frequency power meters can provide accurate and high-resolution power readings through direct measurement but are impractical for large-scale deployment due to high cost and scalability issues. Integrated measurements are accurate but struggle to provide timely power readings due to the long read-out delay. Additionally, power sensors may not always be available in HPC systems or computing components. On the other hand, software-based power modeling techniques are cost-effective and can provide high-resolution readings, but their accuracy is limited due to their volatile nature [32, 34].

In this paper, we ask the question "can we bring the best of integrated measurement (IM) and power modeling techniques to build a better power monitoring solution?". To answer this question, we develop new techniques to enhance the temporal and spatial resolution of IM. We achieve higher temporal resolution by estimating power readings between two IM readings. At the same time, we enhance spatial resolution by distributing the node-level IM readings to lower-level individual components like CPU and memory. Our approach utilizes IM to improve the accuracy of software-based power modeling, while leveraging statistical modeling and machine-learning techniques to improve the spatiotemporal resolutions of IM-based power monitoring. By integrating hardware IM and software modeling techniques, we create a bi-directional interaction that combines the strengths of individual solutions to overcome the limitations of individual technology components.

While promising, translating our high-level idea into a practical system is non-trivial. Power consumption often follows trends over a long-term sampling window [20], but detecting outliers caused by workload behavior and phase changes is challenging as these patterns are difficult to predict in advance. Additionally, software-based power modeling techniques often rely on hardware performance monitoring counters (PMC) to monitor the system and workload behavior, but PMC readings can be noisy and have complex, non-linear relationships with component-level power readings. Thus, distributing the node-level power measurement to individual hardware components requires robust power modeling techniques to account for the complex relationship between PMC and power consumption.

We present HighRPM[1], a power monitoring framework that combines node-level IM and power modeling while overcoming the aforementioned challenges. HighRPM has two components: **T**emporal **R**esolution **R**estoration (TRR) models to improve the temporal resolution of power reading and a **S**patial **R**esolution

*Corresponding Author

[1]HighRPM = **High**-**R**esolution **P**ower **M**easurement

**R**estoration (SRR) model to distribute the node-level power readings to CPU and memory components.

Specifically, we enhance the temporal resolution of node-level IM by creating two models using statistical modeling and machine learning. Our first model, StaticTRR, interpolates power readings *offline* using readings collected during program execution to provide a more detailed analysis of a program's energy and power characteristics. The StaticTRR model first employs spline interpolation to determine the node-level power trend by fitting a curve to IM readings taken during program execution. It then uses a residual model based on PMC readings to estimate fluctuations and power readings beyond the estimated power trend. With StaticTRR in place, we then develop DynamicTRR to estimate dynamic power readings using IM readings. Our DynamicTRR uses a long short-term memory (LSTM) network to model the power data time series and is trained offline using power readings from training programs. The trained model can then be fine-tuned online and applied to any *unseen* programs on the target hardware system.

To improve the spatial resolution of node-level IM, we develop models that distribute node-level power information to individual hardware components, such as CPUs and memory subsystems. Previous power modeling techniques are primarily designed for systems lacking node-level power consumption information. They typically estimate node power consumption first and then obtain the power breakdown of computing components with the assistance of a power model. However, we observed that node power consumption information could significantly improve the accuracy of component power consumption estimation. As a result, we employ node-level IM, which is widely available and easily deployed on HPC systems, to establish a bi-directional power modeling workflow. We utilize a lightweight multi-layer perceptron model to accurately distribute node-level power readings to components to describe the nonlinear relationship between node-level and component-level power consumption.

We implement and evaluate HighRPM on an ARM-based multi-core platform that integrates hardware BMC with software interfaces to provide node-level IM power readings with a sampling rate of less than 0.1 sample per second (i.e., $0.1Sa/s$). We test HighRPM on 96 benchmarks and compare it with 12 prior methods (See Table 4). Extensive evaluation results show that HighRPM can effectively improve resolution by a factor of 10× and reduce the mean absolute percentage error (MAPE) by 7%-24% compared to prior component power modeling methods.

This paper presents the first work to address the low-resolution problem of general-purpose IM in HPC systems. It makes the following contributions:

- It is the first to combine IM readings and software-based power modeling techniques to improve the spatiotemporal resolution of IM power readings;
- It presents two TRR modeling methods for historical power consumption analysis and runtime power monitoring.

## 2 RELATED WORK

### 2.1 Power Monitoring Schemes

Power monitoring schemes should balance between accuracy, spatiotemporal resolution, scalability, and deployment cost [19]. High temporal resolution detects rapid power changes and improves optimization algorithms. High spatial resolution monitors systems, components like CPUs and memory to enhance component-level optimizations. High accuracy ensures effective operation and management; scalability is vital for larger systems. It is crucial to meet these constraints while minimizing the hardware costs.

Existing power monitoring solutions fall into three categories: direct measurement, integrated measurement, and software-centric power modeling techniques. Unfortunately, none of these techniques achieve a satisfactory balance between accuracy, spatiotemporal resolution, scalability, and deployment cost. Specifically, direct measurements such as OMEGAWATT [7], WATTSUP [8], or ZES LMG450 [9], are precise methods of connecting each host to a digital power meter. However, as the system scales, it is impractical to attach a power meter to each host, which is inconvenient and causes high-deployment costs. Since direct measurement does not apply to large-scale system monitoring, this section details the other two schemes (see also Table 1).

### 2.2 Integrated Measurement

The integrated measurement uses energy sensors to provide power readings like voltage and currents [11, 17, 19]. According to the different interfaces adopted, integrated measurement can be achieved using customized integrated measurement (CIM) or general integrated measurement (GIM). For example, PowerMon [11] and PowerMon2 are typical CIM devices built into commodity servers. Both approaches rely on dedicated power measurement devices, increasing hardware expenditures and measurement equipment complexity. HAEC is another CIM approach [21], with a sampling rate of up to $500kSa/s$. But it is only suitable for a single node rather than a cluster of multiple nodes. Other works [24] utilize a dedicated embedded computer to collect power measurements, incurring additional hardware overhead. Overall, CIM offers several advantages, such as high accuracy, good spatiotemporal resolution and scalability. However, the major drawback is its high hardware costs that increase as the system scales up.

In contrast to CIM, GIM is a widely available solution for HPC systems. For instance, the intelligent platform management interface (IPMI) is a popular integrated measurement solution used in most servers and computing systems worldwide. However, despite its widespread use, IPMI-based solutions suffer from long readout delays. Typically, they provide power consumption readings at intervals of 10 seconds or more, which equates to a sampling rate of less than 0.1Sa/s. This negatively impacts real-time energy efficiency optimization strategies. Additionally, IPMI-based solutions are limited by the hardware-exposed sensor interfaces and cannot provide detailed information on computing components like CPUs and memory. This limitation particularly affects servers that lack sensors, as they cannot even provide fine-grained component power consumption data.

### 2.3 Software-centric Power Modeling

Sofware-based power models often utilize hardware performance counters (PMC) to estimate the power consumption of systems. This approach offers several advantages over hardware-based schemes,

**Table 1: Comparisons of power monitoring solutions. [+] indicates a high indicator and [-] indicates a low indicator.**

| | | Example | Temporal Resolution | Spatial Resolution | Accuracy | Scalability | Cost |
|---|---|---|---|---|---|---|---|
| Integrated Measurement | Customized (CIM) | PowerMon2 [11] | ++ | + | ++ | + | ++ |
| | General (GIM) | HDEEM [19] | - | - | ++ | - | + |
| Power Model | Vendor-Specific (VPM) | RAPL [30] | ++ | ++ | ++ | ++ | - |
| | General (GPM) | Linear/Non-Linear (Table 4) | ++ | ++ | + | ++ | - |
| **Combination** | **General** | **HighRPM** | ++ | ++ | ++ | ++ | - |

including high spatiotemporal resolution, low cost, and scalability. Power models can be classified into two categories based on their generality: vendor-specific power models (VPMs) and general-purpose power models (GPMs).

Vendor-specific power models (VPMs) obtain power consumption data by directly reading model-specific registers (MSRs) through the operating system. VPMs are commonly used in mainstream commercial servers with comprehensive software stacks. For example, RAPL [30] is a VPM designed for Intel® Sandy Bridge and later processors, while APM [10] is used for AMD processors. Despite their higher spatiotemporal resolution, VPMs are limited to products from the same manufacturer and cannot be applied to machines from other manufacturers.

To address this limitation and enhance applicability, power models known as CPMs (constructed with PMCs) have been proposed. Singh et al.[35], for instance, developed a power model based on multiple linear regression using PMCs. Powell et al.[29] utilized a linear regression model based on PMCs to estimate the activity factor and subsequently predict power consumption. Apart from linear methods, machine learning (ML)-based techniques are employed to explore non-linear relationships between variables. For instance, Song et al.[36] proposed a power and energy model that utilized a configurable back-propagation artificial neural network (BP-ANN) for modeling. Sagi et al.[33] used negative feedback neural networks to model power consumption in multi-core processors. ML-based approaches generally offer improved accuracy compared to linear methods. However, they also introduce challenges such as increased complexity in model implementation, repeated experimental testing, the need for substantial training data, and careful selection of kernel programs [27]. These considerations require significant attention when employing ML-based power modeling methods.

Our work combines GIM and GPM to leverage their strengths. Relying solely on VPM or combining GIM and VPM has limitations for two reasons. Firstly, VPM's customization, like RAPL, may lack generality, limiting its compatibility with different CPU models. Secondly, VPM-only approaches, such as RAPL, cannot monitor multiple processors simultaneously. In contrast, our proposed HighRPM achieves a balance between resolution, accuracy, scalability, and cost-effectiveness while providing comprehensive system-wide coverage. Additionally, HighRPM offers greater generalizability compared to other techniques, overcoming the limitations of VPM and accommodating diverse systems.

## 3 MOTIVATION

Our work aims to improve the spatial and temporal resolution of power monitoring. Increasing the spatial resolution allows us to provide power readings for individual components like the CPU and the memory subsystems. Increasing the temporal resolution

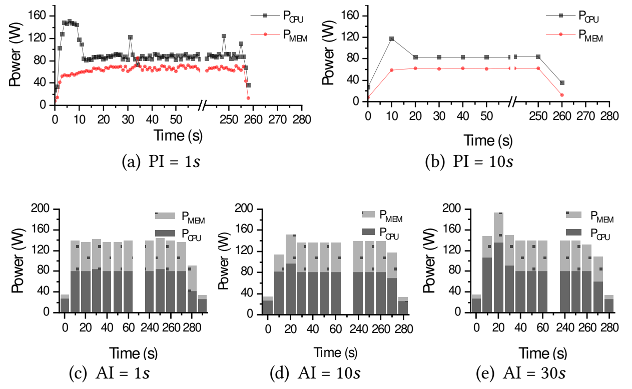
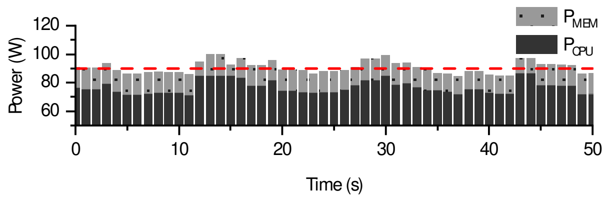


**Figure 1: Power changes of CPU and DRAM when optimizing the Graph500 [3] benchmark with different power capping frequency (AI) under different power reading frequencies (PI).**

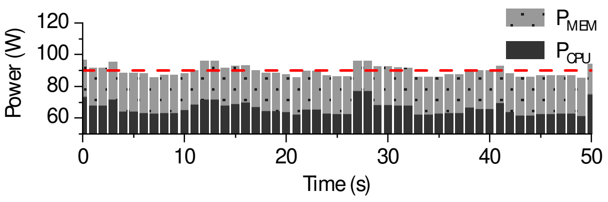
provides higher frequent power readings to support power optimization techniques like power capping.

As an example, Fig. 1 describes the power changes when running the Graph500 (BFS) benchmark [3] under different power reading intervals (PI) and power capping frequencies (AI). The hardware platform is a multi-core development board with a 16-core ARMv8 CPU. A coarser-grained power reading, e.g., when PI increases from one reading every 1$s$ to one reading every 10$s$, as (a)→(b) shows, can prevent the power monitoring system from capturing the sudden changes (e.g., the spiking points in Fig. 1(a)) of the power consumptions. Failing to react quickly to power changes can lead to overheating or increased energy consumption by not enforcing power capping. For example, when the time between two power capping actions increases 1$s$ to 30s (as (c)→(d)→(e) shows), the peak power of the application would increase to 50$W$, which also prolongs the peak power consumption. In such a scenario, the energy consumption of the program increased by 1.1$KJ$ (37.3$KJ$→38.4$KJ$).

In another example, Fig. 2 shows power readings while executing FFT and Stream [25] without power capping. While the node-level average power consumption of the benchmarks is around the 90$W$ line (the power consumption of other peripherals is observed to be a constant 25$W$), their CPU and memory consumption exhibit noticeable divergences. Specifically, the CPU power consumption dominates the power consumption of the computing node when running the computation-intensive FFT benchmark. In contrast, the RAM power consumption dominates the node-level power consumption when running the memory-intensive Stream benchmark. As can be seen from the example, the power distribution of individual components can vary depending on the running applications. Since knowing the power distribution of individual components is useful for determining the power optimization direction [37],



(a) Graph500



(b) Stream

**Figure 2: The power changes of FFT and Stream when running on a node of the ARM-based platform.**

there is a need to break down the coarse-grained node-level power reading to individual components.

## 4 OUR APPROACH

### 4.1 Overview

HighRPM enhances the temporal and spatial resolutions of power monitoring by leveraging node-level IPMI-based integrated measurement and software power modeling techniques. The framework, as depicted in Fig. 3, consists of two stages: the initial learning stage and the active learning stage.

During the **initial learning stage**, HighRPM employs available initial samples to train models for both temporal resolution restoration (TRR) and spatial resolution restoration (SRR) tasks. The objective is to generate high-quality restored samples. Subsequently, in the **active learning stage**, the initial and restored samples are combined to create a new sample set. A sampler randomly selects reinforcement samples from this set, which are then used to fine-tune HighRPM.

When deployed, HighRPM can be installed as a service on the control node of the target HPC system and shared with other computing nodes. The framework operates on each node, capturing power variations and hardware characteristics through reinforcement samples. This design accounts for power variations between nodes, making HighRPM highly effective for both single-node and multi-node scenarios.

HighRPM consists of two core models: temporal resolution restoration (TRR) and spatial resolution restoration (SRR). In the TRR model, spline interpolation is used to estimate the missing node power consumption, resulting in $P_{splined}$. The difference between the actual node power consumption $P_{Node}$ and $P_{splined}$ is used to build a PMC-based residual model called ResModel. PMCs serve as features in the dataset $D_{StaticTRR}$, enabling the estimation of node power consumption. For the SRR model, the predicted node power consumption $P'Node$ from the TRR model, along with PMCs, is input into a multi-layer perceptron (MLP) model. The MLP model consists of an input layer, hidden layer, and output layer, and is used to predict CPU power consumption ($P'CPU$) and memory power consumption ($P'_{MEM}$).

### 4.2 Temporal Resolution Restoration Model

Node power consumption data is characterized by long-term trends determined by program loops and unforeseen short-term fluctuations. Recovering temporal resolution requires distinguishing between components of power consumption attributed to long-term trends and those due to short-term fluctuations when the node power consumption is unknown. To address this challenge, we introduce two TRR models: *StaticTRR* and *DynamicTRR*. StaticTRR is ideal for historical power log analysis and exploratory tasks, while DynamicTRR is more suitable for real-time power monitoring of servers or computing clusters. By combining these two approaches, our framework effectively tackles the challenges of recovering temporal resolution and predicting power consumption in a robust and versatile manner.

*4.2.1 StaticTRR.* Spline interpolation is a popular method for data completion, using variable splines to create a smooth curve through known points. However, interpolation techniques like splines and ARIMA can only estimate missing data points based on long-term trends and may not capture short-term fluctuations accurately [31].

To address this limitation, StaticTRR combines spline interpolation with a ResModel, which is a PMC-based residual model. The residual error represents the difference between observed and interpolated values. StaticTRR assumes that these residuals arise from short-term power variability rather than interpolation errors. This assumption is supported by the high accuracy achieved by HighRPM (discussed in Section 6). By using PMCs as features, which track hardware events during program execution, the ResModel effectively identifies abnormal variability locations and estimates mutation amplitude.

To train and build StaticTRR start with the PMCs obtained at specific time intervals and the node power consumption measurements recorded in chronological order. Then, we divide our data samples into two initial sets: set $A$, which consists of all labeled samples (i.e., those with known power consumption values for compute nodes), and set $B$, which contains all unlabeled samples (i.e., those where node power consumption is unknown). The complete data set $D$ is formed by combining these two sets. We then follow a number of steps to train the StaticTRR model, described as follows.

**Building spine model** We utilize the initial data set $A$ to establish the spline model by selecting 50% of it as the training set. Subsequently, we input $A$ and $B$ into the built spline model to obtain an estimate of power consumption ($P_{splined}$). Then, we replace the power consumption in the initial dataset to obtain the datasets $A_{splined}$ and $B_{splined}$.

**Building Resmodel** For datasets $A$ and $A_{splined}$, we select 50% of them as the training set, use $ABS(P_{splined} - P_{Node})$ as the prediction target, and PMCs as the features to build a ResModel using the decision tree (DT) algorithm. We tested all the linear and nonlinear methods listed in Table 4 but found that DT worked best, hence it was used to train the ResModel. Afterward, the ResModel is applied to the initial datasets $A$ and $B$ to obtain the power consumption estimates $P_{residual}$. Finally, we replace the power consumption in the initial datasets to obtain datasets $A_{residual}$ and $B_{residual}$.

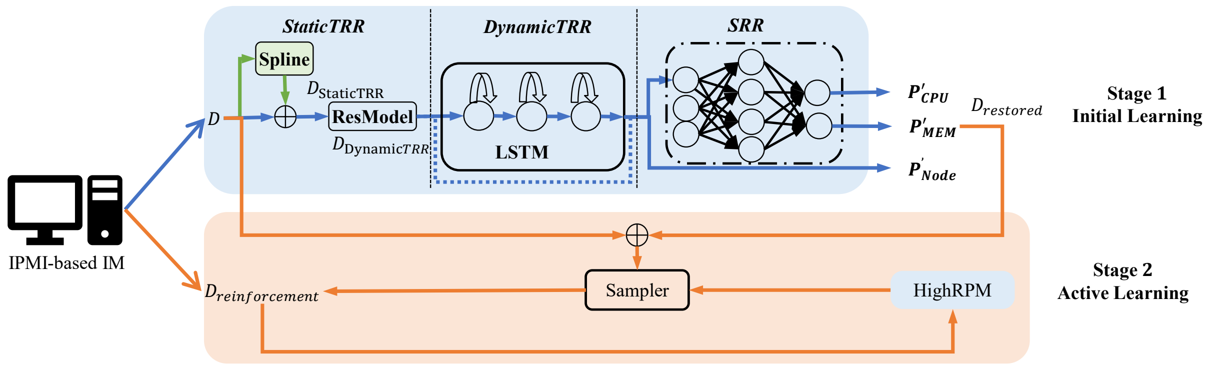


**Figure 3: Overview of HighRPM.**

**Post-processing.** We further apply operations on the power consumption estimates obtained from the spline model and the ResModel to improve the model accuracy of StaticTRR as Algo. 1 shows. Then, we replace the power consumption in the initial dataset as $P_{StaticTRR}$ to obtain the restored dataset $D_{StaticTRR}$. Operations 1 and 2 post-process the outputs of the spline and ResModel, respectively, reducing the model error to a certain extent. Compared with the spline model, the ResModel is slightly inferior in accuracy and robustness, especially the accuracy is very doubtful when the estimated value $P_{residual}$ is greater than the bottom limit of power consumption $P_{bottom}$ or less than the upper limit of power consumption ($P_{upper}$). Operation 3 is established to consider the two models synergistically to maximize model accuracy.

*4.2.2 DynamicTRR.* In many power monitoring scenarios, there can be a significant time lag between consecutive readings. For example, after obtaining an IM reading at time $t_0$, the next reading can only be obtained at time $t_{10}$. This introduces a prediction problem rather than a fitting problem for obtaining $P_{Node}$ in the interval $[t_0, t_{10}]$. As shown in Fig. 4, spline interpolation cannot be used for the samples at the $(n-1)$-th and $n$-th moments. While StaticTRR is a fitting method that captures trends and sudden changes in power consumption using known points, it is not suitable for predicting future points beyond the last known sampling point.

To address this, we propose DynamicTRR, a dynamic method designed for real-time power monitoring. It leverages clever dataset construction and a lightweight LSTM model. We build DynamicTRR in two steps, described as follows.

**Training data collection** The DynamicTRR training data, as shown in Fig. 4, can be represented as a matrix with $n$ rows and $m+2$ columns. Each row corresponds to a sample $s^{(i)}$, where $C_1^{(i)}...C_2^{(m)}$ are $m$ PMCs at moment $i$ and $P_{Node}^{(i)}$ is the node power consumption at that moment. The label for each sample is the real node power consumption at moment $i$, denoted as $P_{Node}^{(i)}$. To capture trend characteristics, we create a new sample $s'^{(i)}$ with consecutive $miss\_interval$ samples.

The shape of $s'^{(i)}$ is ($miss\_interval$,$m$+1) instead of $(1, m+1)$. For each $s'^{(i)}$, the label becomes $< P_{Node}^{(i)}, P_{Node}^{(i+1)}, ..., P_{Node}^{(i_{miss}-1)} >$. We construct multiple $s'^{(i)}$ to form the training dataset $D_{DynamicTRR}$, which contains $(n - miss\_interval + 1)$ samples.

**Algorithm 1: Post-Processing Algorithm of StaticTRR**

```
Input: $P$, $P_{splined}$, $P_{upper}$, $P_{bottom}$.
Output: $P_{trr}$.
1  for $i$ = 1 to $n$ do
2      // Operation 1 if $P_{splined}[i] \geq 30\% * (P_{upper} - P_{bottom})$
        then
3          $P_{splined}[i - miss\_interval/2 : i + miss\_interval/2] =$
            $P_{splined}[i]$.
4      end
5      // Operation 2 if $P_{residual}[i] \geq P_{upper}$ then
6          $P_{residual}[i] = P_{splined}[i]$.
7      end
8      // Operation 3 if $P_{residual}[i] \leq P_{bottom}$ then
9          $P_{residual}[i] = P_{splined}[i]$.
10     end
11     if $abs(P_{splined}[i] - P_{residual}[i]) \leq \alpha *$
        $min(P_{spline}[i], P_{residual}[i])$ then
12         $P_{trr}[i] = P_{splined}[i]$.
13     end
14     if $abs(P_{splined}[i] - P_{residual}[i]) \geq \alpha *$
        $min(P_{splined}[i], P_{residual}[i])$ &
        $abs(P_{splined}[i] - P_{residual}[i]) \leq \beta *$
        $min(P_{splined}[i], P_{residual}[i])$ then
15         $P_{trr}[i]$=0.5*($P_{splined}[i]$+$P_{residual[i]}$).
16     end
17     if $abs(P_{splined}[i] - P_{residual}[i]) \geq \beta *$
        $min(P_{splined}[i], P_{residual}[i])$ then
18         $P\_trr[i]$=$P_{splined}[i]$.
19     end
20 end
```

This method is based on the interpolated values obtained by StaticTRR. We observed that spline interpolation had better fitness than other power modeling methods, especially those relying solely on PMCs. This indicates the importance of power consumption information in ensuring model accuracy, as spline relies on accurately predicting the trend of $P_{Node}$. To utilize this valuable information,

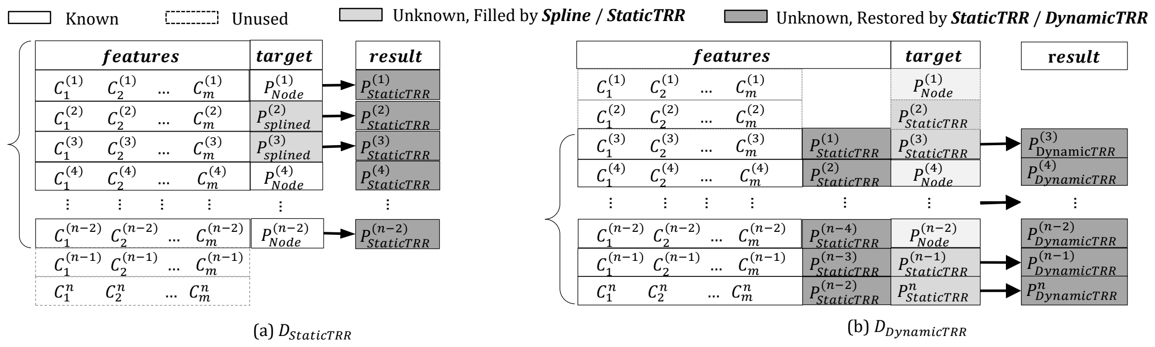

Known
Unused
Unknown, Filled by Spline / StaticTRR
Unknown, Restored by StaticTRR / DynamicTRR
features
target
result
(a) $D_{StaticTRR}$
(b) $D_{DynamicTRR}$

**Figure 4: Dataset as input for TRR. $C_*^*$ denotes $PMC_*^*$. In the example depicted in this figure, the *miss_interval* is 2.**

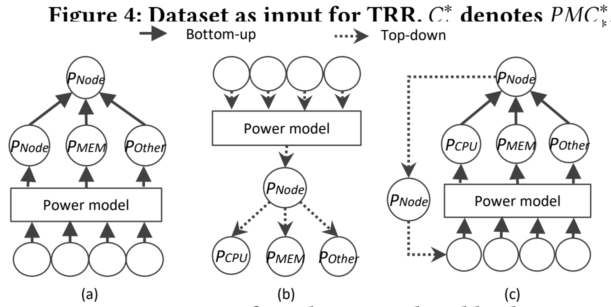


**Figure 5: Comparisons of unidirectional and bi-directional power modeling approaches.**

we include $P'_{Node}$ at the $(i-1)$-th moment as a feature when constructing the sample. Notably, $P'_{Node}$ at the $(i-1)$-th moment is always available and can be determined from either the observed value or the spline model.

**Model training and fine-tuning** After creating the training dataset $D_{DynamicTRR}$, we utilize a compact LSTM model with an input layer, two hidden layers, and a fully connected layer to construct a power model. To ensure that each sample $s'$ includes a measured $P_{Node}$, we set the sliding window size as *miss_interval*.

At each moment $i$, DynamicTRR handles the samples and models differently depending on whether the IM readings can be obtained at the previous moment $(i-1)$ (measured or predicted). If $P_{Node}^{(i-1)}$ is the predicted value, we use it directly to create the sample $s^{(i)}$, while the previous $miss\_interval - 1$ samples form $s'^{(i)}$. This $s'^{(i)}$ is input into the trained model to predict the value $P_{Node}^{i}$.

If $P_{Node}^{(i-1)}$ is the measured value, we also use it to create $s^{(i)}$, along with the previous $miss\_interval - 1$ samples forming $s'^{(i)}$. Importantly, we fine-tune the existing model using $s'^{(i)}$ and input it into the refined model to obtain the predicted values.

## 4.3 Spatial Resolution Restoration Model

Two workflows for power modeling to calculate compute nodes and components' power usage are bottom-up and top-down. Bottom-up power modeling [12] (Fig. 5(a)) focuses on the target system's micro-architecture, using micro-benchmarks for system status collection, component-level power modeling, and summing up each component's power consumption for node power consumption. This method, however, demands specific domain knowledge and can be complex. Top-down power modeling [35] (Fig. 5 (b)) views the system as a "black box" and doesn't necessitate domain knowledge. It includes feature selection, modeling, and power breakdown for each component's power consumption. It's simple, quick, deployable, and hardware-independent. Both methods are unidirectional, but the order in which power consumption is obtained distinguishes them. This step-by-step one-way modeling approach greatly simplified the problem of deriving the component power consumption to some extent.

However, we observed that the correlation between $P_{Node}$ and $P_{MEM}$ is strong enough to improve the accuracy of our model significantly, which has been verified in Sec. 6. Therefore, to take advantage of this new information and correlation, this paper extends the unidirectional model by incorporating node-level IM readings, creating a directed graph with a cycle as shown in Fig. 5 (c). This distinguishes our model from other unidirectional models.

The main part of SRR is a shallow MLP model using node power information. It consists of an input layer, a hidden layer, and an output layer. The units that make up the input layer are $P_{Node}$ output by the TRR model and multiple PMCs, the hidden layer contains multiple neural units, and the output layer consists of two units, representing $P_{CPU}$ and $P_{MEM}$, respectively. SRR has a very simple model structure, but its predictive effect is excellent (See Sec. 6.2 for details.).

## 5 EXPERIMENTAL SETUP

### 5.1 Evaluation Platforms

We evaluate and implement HighRPM on an ARM-based system that utilizes a BMC on a plug-in containing multiple nodes. Each compute node is equipped with 64-core ARMv8 processors and 128GB DDR4. Although this system provides only node power consumption with a low temporal resolution of 0.1 $Sa/s$, we are able to capture CPU and memory power consumption at a desired higher temporal resolution (>1 $Sa/s$) using a special direct measurement method, as shown in Fig. 6. It is important to note that this direct measurement method is unsuitable for large-scale deployments due to the increasing hardware cost as the system size grows. To evaluate its generalization ability, we also applied HighRPM to an x86 platform, as detailed in Sec. 6.3.

### 5.2 Measurement

We collect the real power consumption as the ground truth in two ways, as shown in Fig. 6. For one, the BMC is used to read the power consumption from the power chip using an integrated

**Table 2: Hardware performance counters used by HighRPM.**

| Unit | PMC Events | Description |
|---|---|---|
| Core | CPU_CYCLES | Cycles |
| | INST_RETIRED | Instructions |
| | BR_PRED | Branch instructions |
| | UOP_RETIRED | Micro-operations |
| Lx Cache /(L1, L2 or L3) | L1I_Cache_LD | Load instructions |
| | L1I_Cache_ST | Store instructions |
| | LxD_Cache_LD | Load instructions |
| | LxD_CACHE_ST | Store instructions |
| Main Memory | BUS_ACCESS | Bus access |
| | MEM_ACCESS | Memory access |

**Table 3: Combination of seen (unseen) sets used for evaluation. The number of programs included in the benchmark suite is denoted in parentheses.**

| Training set | Test set |
|---|---|
| SPEC(43), PARSEC(36), HPCC(12), Graph500(2), HPL-AI(1), SMG2000(1) | HPCG(1) |
| SPEC(43), PARSEC(36), HPCC(12), Graph500(2), HPL-AI(1), HPCG(1) | SMG2000(1) |
| SPEC(43), PARSEC(36), HPCC(12), Graph500(2), SMG2000(1), HPCG(1) | HPL-AI(1) |
| SPEC(43), PARSEC(36), HPCC(12), SMG2000(1), HPL-AI(1), HPCG(1) | Graph500(2) |
| SPEC(43), PARSEC(36), SMG2000, HPL-AI(1), HPCG(1), Graph500 | HPCC(12) |
| SPEC(43), SMG2000(1), HPL-AI(1), HPCG(1), Graph500(2), HPCC(12) | PARSEC(36) |
| SMG2000(1), HPL-AI(1), HPCG(1), Graph500(2), HPCC(12), PARSEC(36) | SPEC(43) |

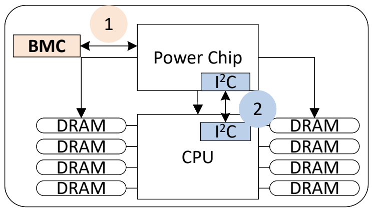


**Figure 6: Power measurement mechanism used in our valuation.**

measurement method based on IPMI. For another, on our ARM development board, we cascaded the CPU and power supply using a jumper wire, allowing the CPU to read the current flowing through different voltage domains by directly accessing the registers *0x8b* and *0x8c*, thus being able to obtain the real-time processor power consumption $P_{CPU}$ and memory power consumption $P_{MEM}$ at a sampling rate of 1 $Sa/s$. It has a power reading error of $0.1W$, which is more accurate than the vendor-provided power tools that have an error range of $1W$. Besides $P_{CPU}$ and $P_{MEM}$, we also consider the peripherals power consumption $P_{Other}$. When performing power modeling, $P_{Other}$ is set as constant at $25W$, which is measured by running no workload on the compute node. All experiments have shown that $P_{Other}$ varies very little, within just under $1W$. We set it to a constant to avoid the need for more sophisticated measurement techniques and to provide sufficient model accuracy.

Aside from collecting power consumption data, performance data is also required for modeling purposes. We collect PMC events listed in Table 2 utilizing a Linux loadable kernel module at a sampling rate of 1 $Sa/s$. The readings from various per-core counters are aggregated to achieve the desired performance data. All measurements are obtained by running experiments in standalone mode to minimize disturbances from fluctuations in system performance.

## 5.3 Benchmarking Workloads

To train our power model, we utilize empirical data from 96 benchmarks, including 43 from `SPEC CPU 2017` [4], 36 from `PARSEC` [6], 12 from `HPCC` [25], 2 from `Graph500` [3], as well as `HPL-AI` [23], `SMG2000` [14], and `HPCG` [5]. These benchmarks span from scientific to consumer applications and stress CPU and memory through their compute or memory intensiveness. We validate data through five separate runs for each parameter set, reporting average values. Every benchmark operates for 60 seconds to an hour, with a relative error in elapsed time under 5%, confirming stable performance.

**Table 4: Baseline model settings.**

| Type | Model | Abbreviation | Hyperparameters |
|---|---|---|---|
| Linear | Linear Regression [29] | LR | automatic options |
| | Lasso Regression [26] | LaR | automatic options |
| | Ridge Regression [38] | RR | solver=auto |
| | SGD Regression [38] | SGD | squared_error, max_iter=10000 |
| Nonlinear | Decision Tree [13] | DT | squared_error |
| | Random Forest [38] | RF | #trees=10 |
| | Gradient Boosting Tree [38] | GB | #trees=10 |
| | K Nearest Neighbor | KNN | #neighbors=3, algo.=auto |
| | Support Vector Machine [26] | SVM | automatic options |
| | Neural Network [33, 36] | NN | #hidden_size=30, max_iter=10000 |
| RNN | Gated Recurrent Unit | GRU | #units=2 |
| | Long Short-Term Memory [31] | LSTM | #units=2 |

We employ 5-fold cross-validation and two training set construction methods. The first assesses seen programs with samples related to the target program, and the second evaluates unseen programs, excluding such samples. We have grouped different programs from the same benchmark suite into seven sets. One set is randomly chosen as the target program(s) and the rest serve as the training set. We compile 1000 samples from each set in order, resulting in a 6000-sample training set (unseen) and a 6300-sample one (seen), plus a 1000-sample (unseen) and 700-sample (seen) test set. The combinations are detailed in Table 3. Average results for seen and unseen sets are given in Sec. 6 due to page constraints.

## 5.4 Baselines

We compare HighRPM against 12 baseline models, including four linear methods, six nonlinear methods, and two recurrent neural network (RNN) models. It should be noted that we adopted the same setups for all baseline models that undergo fine-tuning. The linear and nonlinear models are constructed using algorithms from the *scikit-learn* package, while the two RNN models are built based on the structure of HighRPM, with GridSearch used to tune the hyperparameters in each cross-validation. The hyperparameters of each model are provided in Table 4.

## 5.5 Metric

We use the Mean Absolute Percentage Error (MAPE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and coefficient of determination ($R^2$) between the observed power consumption and the predicted power consumption to evaluate the accuracy of TRR and SRR models. MAPE and RME are metrics used to measure the relative error of models, while MAE displays the absolute error. $R^2$ is used to measure the robustness of the models.

## 6 EVALUATION

This section first presents the evaluation results of TRR and SRR, followed by a comprehensive evaluation of HighRPM on an X86-based system. Additionally, we investigate its sensitivity to *miss_interval* (the sampling rate), its supportability for different frequencies, and hyperparameters, as well as its potential for extension to other peripherals. Finally, we point out the limitation of HighRPM.

Table 5: Comparisons between TRR and alternative models.

| Type | Model | Seen application | | | Unseen application | | |
|---|---|---|---|---|---|---|---|
| | | MAPE (%) | RMSE | MAE | MAPE (%) | RMSE | MAE |
| Linear | LR | 22.50 | 7.26 | 6.86 | 21.74 | 9.85 | 8.42 |
| | LaR | 22.50 | 7.26 | 6.86 | 21.74 | 9.85 | 8.42 |
| | RR | 22.49 | 7.25 | 6.86 | 21.72 | 9.84 | 8.40 |
| | SGD | 22.51 | 7.25 | 6.87 | 21.71 | 9.83 | 8.38 |
| Nonlinear | DT | 18.61 | 8.03 | 5.34 | 22.32 | 12.76 | 10.55 |
| | RF | 20.21 | 6.64 | 6.02 | 22.32 | 12.76 | 10.55 |
| | GB | 20.45 | 7.62 | 7.12 | 19.48 | 10.02 | 8.91 |
| | KNN | 28.22 | 10.58 | 9.62 | 21.59 | 10.32 | 8.75 |
| | SVM | 17.40 | 4.71 | 4.35 | 22.35 | 10.39 | 8.69 |
| | NN | 20.39 | 6.00 | 5.64 | 21.96 | 9.99 | 8.47 |
| RNN | GRU | 11.59 | 5.62 | 5.55 | 15.94 | 10.79 | 9.33 |
| | LSTM | 9.63 | 4.71 | 4.66 | 11.85 | 7.50 | 5.87 |
| TRR | DynamicTRR | 4.46 | 3.19 | 2.78 | 4.38 | 3.18 | 2.05 |

Table 6: Comparisons among TRR models.

| Model | Seen application | | | Unseen application | | |
|---|---|---|---|---|---|---|
| | MAPE (%) | RMSE | MAE | MAPE (%) | RMSE | MAE |
| **Spline** | 2.21 | 1.85 | 0.93 | 2.45 | 2.43 | 1.25 |
| **StaticTRR** | 4.02 | 2.70 | 1.99 | 3.87 | 2.66 | 1.64 |
| **DynamicTRR** | 4.46 | 3.19 | 2.78 | 4.38 | 3.18 | 2.05 |

## 6.1 TRR Performance

Table 5 reports the TRR performance when the temporal resolution is recovered by 10× (from 0.1Sa/s to 1Sa/s) compared to IPMI-based integrated measurement. As evident from the table, TRR delivered a notable enhancement, with staticTRR metrics of 4.02% (MAPE), 2.70 (RMSE), and 1.99 (MAE), and DynamicTRR metrics of 4.46% (MAPE), 3.19 (RMSE), and 2.78 (MAE).

### 6.1.1 *For unseen applications.*

When performing power prediction for previously unseen applications, conventional PMC-based models tend to exhibit a decrease in performance, indicating weak robustness. However, Table 5 demonstrates that TRR is significantly less affected by previously unseen applications, indicating strong robustness. This is due to incorporating a spline model in HighRPM, which accurately models power consumption trends even for unseen applications. Conversely, in PMC-based models, the learned PMC variation patterns may not be directly transferable to unseen applications. Thus, based on our evaluation of unseen applications, resetting HighRPM via re-execution of the initialization step is unnecessary if the workload is non-repeated and possesses power characteristics that undergo temporary fluctuations.

### 6.1.2 *Comparisons with common power models.*

Compared to alternative approaches, TRR displays a reduction in MAPE of 6%-18%, a decrease in RMSE of 1.5-7.5, a decrease in MAE of 2-7, and a $R^2$ that remains above 0.9. There are two reasons for TRR's superiority over other power models. Both spline and TRR incorporate node power consumption data into their models. Although the information regarding node power consumption is sparse, it effectively characterizes the long-term trend of power consumption behavior. Secondly, the precision of PMC-based power models relies heavily on ingeniously designed feature engineering. However, in this paper, we utilize the same PMCs for all fine-tuned models without further exploring feature engineering.

### 6.1.3 *Comparisons with spline model.*

While the spline offers a good fit, it fails to track short-term power consumption variations and predict future usage. Our proposed TRR model effectively overcomes these issues. Fig.7 shows that the spline model is more precise at a *miss_interval* of 10s, but its ability to capture short-term power consumption changes diminishes as the $miss_interval$ increases, failing in extreme cases. Despite Table6 showing TRR slightly underperforming spline in terms of MAPE, RMSE, and MAE, the difference is not statistically significant, likely due to power consumption fluctuations in the PMC-based TRR model. This minor discrepancy is acceptable as it strengthens the model's capability to capture abrupt and unpredictable power changes accurately.

## 6.2 SRR Performance

The evaluation results suggest that SRR achieves a high degree of accuracy. Specifically, as shown in Table 7, when predicting $P_{CPU}$ for known applications, the metrics are impressive, with values of 7.65% (MAPE), 3.06 (RMSE), and 2.43 (MAE). When predicting $P_{MEM}$, the metrics are even more impressive, with values of 5.31% (MAPE), 0.77 (RMSE), and 0.50 (MAE).

### 6.2.1 *Comparisons with baseline models.*

Similar to the evaluation experiments for TRR, we conducted a comparative analysis of SRR against 12 baseline models. Our results reveal that, compared to other methods, SRR achieves a reduction in MAPE of 7%-24%, a reduction in RMSE of 3-10, and a reduction in MAE of 2-8 $W$. The superior predictive capabilities of SRR can be attributed to the inclusion of $P_{Node}$, which is a variable that aggregates the power consumption of all components within the computing node. There exists a strong correlation between the power consumption of these components. To further analyze the effect of $P_{Node}$ on SRR's performance, we conducted a comparison of SRR with and without the inclusion of $P_{Node}$. Our analysis, as presented in Table 8 significantly enhances the accuracy of the SRR model.

### 6.2.2 *Evaluation on unseen applications.*

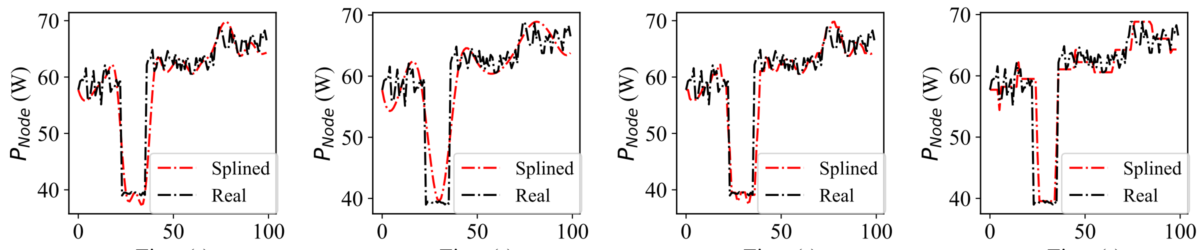
From Table 7, it is evident that most other PMC-based models experience a decrease in accuracy when applied to different target applications. In contrast, SRR consistently maintains its predictive accuracy for $P_{CPU}$, even in the case of unknown applications. However, when dealing with unfamiliar applications, there is a slight decrease in model accuracy for predicting $P_{MEM}$. This can be attributed to the limited range of variation in $P_{MEM}$, where even minor changes can have a significant impact on the MAPE values. Nevertheless, it is important to note that the MAEs demonstrate that the error rate in power prediction using SRR remains within a margin of 2W.

## 6.3 Exvaluation on x86 System

So far, our evaluation was conducted on ARM systems. In this experiment, we apply HighRPM to an x86 system designed to resemble the hardware configuration of Tianhe-1A [39]. The cluster comprises 64 compute nodes, each equipped with Intel® Xeon® E5-2660 v2 processors [2]. Intel processors offer robust support for RAPL, which exhibits high accuracy in power measurement. Hence, we employ the *perf* tool [1] to capture performance monitoring events, specifically the */power/energy-pkg/* and */power/energy-ram/* metrics, at intervals of 1 second. Previous studies have verified that the use of PMC sampling does not introduce unacceptable inaccuracies [22]. Given RAPL's capability to provide power consumption data with high resolution in both time and space, we intentionally introduce data sparsity by configuring a miss_interval of 10 seconds

**Table 7: Comparisons between SRR with alternative models.**

| Type | Model | Seen application | | | | | | Unseen application | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $P_{CPU}$ | | | $P_{MEM}$ | | | $P_{CPU}$ | | | $P_{MEM}$ | | |
| | | MAPE(%) | RMSE | MAE | MAPE(%) | RMSE | MAE | MAPE(%) | RMSE | MAE | MAPE(%) | RMSE | MAE |
| Linear | LR | 20.97 | 8.28 | 6.79 | 9.97 | 1.42 | 1.04 | 34.99 | 14.80 | 12.42 | 23.45 | 3.47 | 2.91 |
| | LaR | 20.54 | 8.28 | 6.70 | 9.84 | 1.51 | 1.04 | 31.94 | 13.77 | 11.43 | 20.85 | 3.13 | 2.59 |
| | RR | 20.96 | 8.28 | 6.79 | 9.97 | 1.43 | 1.04 | 34.72 | 14.69 | 12.32 | 23.42 | 3.46 | 2.91 |
| | SGD | 21.02 | 8.29 | 6.80 | 10.17 | 1.46 | 1.07 | 34.91 | 14.69 | 12.34 | 23.95 | 3.51 | 2.96 |
| Nonlinear | DT | 20.68 | 9.54 | 6.68 | 12.46 | 1.77 | 1.27 | 24.43 | 10.21 | 8.24 | 23.23 | 3.26 | 2.80 |
| | RF | 18.14 | 7.42 | 5.70 | 9.57 | 1.46 | 1.00 | 20.75 | 9.31 | 7.30 | 21.75 | 3.28 | 2.72 |
| | GB | 18.37 | 8.20 | 6.12 | 9.16 | 1.41 | 0.97 | 21.62 | 9.31 | 7.32 | 20.83 | 3.16 | 2.60 |
| | KNN | 16.66 | 7.74 | 5.48 | 9.60 | 1.44 | 1.00 | 24.29 | 11.02 | 8.41 | 22.12 | 3.30 | 2.73 |
| | SVM | 15.57 | 7.46 | 5.42 | 9.23 | 1.38 | 0.98 | 22.16 | 10.32 | 8.11 | 21.07 | 3.23 | 2.64 |
| | NN | 15.03 | 6.26 | 4.83 | 8.78 | 1.22 | 0.90 | 19.16 | 8.57 | 6.42 | 20.75 | 3.17 | 2.60 |
| RNN | GRU | 31.44 | 13.98 | 10.20 | 8.88 | 1.25 | 0.91 | 20.83 | 9.63 | 7.35 | 21.49 | 3.32 | 2.71 |
| | LSTM | 28.45 | 13.02 | 9.47 | 8.39 | 1.27 | 0.88 | 22.38 | 10.75 | 8.18 | 24.82 | 3.80 | 3.13 |
| **SRR** | | 7.65 | 3.06 | 2.43 | 5.31 | 0.77 | 0.50 | 7.00 | 2.93 | 2.13 | 16.49 | 2.48 | 2.04 |



(a) miss_interval=5s (b) miss_interval=10s (c) miss_interval=5s (d) miss_interval=10s

**Figure 7: The impact of different *miss_interval* on the spline model (sub-figures a and b) and StaticTRR (sub-figures c and d).**

**Table 8: Comparisons of with/without $P_{Node}$ as a feature.**

| | | With $P_{Node}$ | | | Without $P_{Node}$ | | |
|---|---|---|---|---|---|---|---|
| | | MAPE(%) | RMSE | MAE | MAPE(%) | RMSE | MAE |
| Seen app. | $P_{CPU}$ | 7.65 | 3.06 | 2.43 | 30.46 | 11.08 | 9.95 |
| | $P_{MEM}$ | 5.31 | 0.77 | 0.50 | 21.56 | 2.37 | 2.10 |
| Unseen app. | $P_{CPU}$ | 7.00 | 2.93 | 2.13 | 29.00 | 12.47 | 10.26 |
| | $P_{MEM}$ | 16.49 | 2.48 | 2.04 | 34.00 | 4.88 | 4.20 |

(equivalent to a sampling rate of 0.1 Sa/s). This deliberate sparsity allows us to evaluate the effectiveness of HighRPM by comparing the power consumption readings obtained from RAPL with the corresponding predicted values.

Table 9 quantifies the spatiotemporal resolution restoration capabilities of HighRPM for unseen applications on our x86 platform. Results indicate that DynamicTRR, a component of TRR, outperforms other methods in terms of minimizing errors in temporal resolution restoration, with a MAPE 4%-10% lower than alternative approaches. This highlights the superior resolution restoration capabilities of DynamicTRR. For spatial resolution restoration, TRR achieves a reduction in MAPEs of up to 5% for $P_{CPU}$ and over 25% for $P_{MEM}$. The incorporation of node power information in its models gives TRR a significant advantage over RF, KNN, GRU, and LSTM when predicting $P_{MEM}$, while its accuracy in predicting $P_{CPU}$ is comparable to these alternatives.

A comparison between Table 7 and Table 9 reveals a slight increase in error metrics when operating on the X86-based platform. This can be attributed to the platform's higher CPU frequency (2.6GHz on X86-based vs. 2.2GHz on ARM-based), which presents a greater challenge in achieving high prediction accuracy. The results also demonstrate that the SRR component consistently exhibits higher prediction errors in $P_{MEM}$ for both ARM and x86 platforms, likely because of a lack of adequate features for accurate prediction.

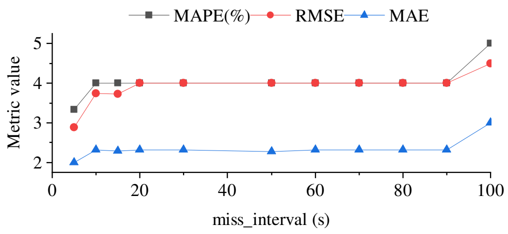


**Figure 8: Sensitivity of HighRPM to different miss_interval.**

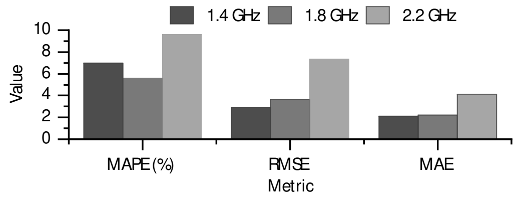
Although integrating node power consumption, which is directly linked to $P_{MEM}$, enhances the accuracy of $P_{MEM}$ prediction significantly, achieving a comparable MAPE as $P_{CPU}$ prediction remains an arduous task. Apart from feature selection and the model itself, the relatively narrow range of variation in $P_{MEM}$ may result in significant fluctuations in MAPE values, even with minor changes.

## 6.4 Discussions

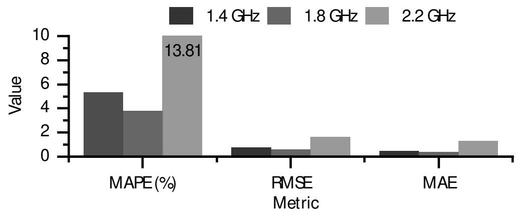
*6.4.1 Sensitivity to miss_interval (the sampling rate).* In the evaluation, we used a miss_interval of 10$s$ to achieve a target sampling rate of $1Sa/s$. However, HighRPM is flexible and can support higher temporal resolutions. In this section, we conduct a sensitivity analysis on the same ARM-based platform by setting different miss_intervals. We evaluate the MAPE (Mean Absolute Percentage Error) of node power prediction. The sensitivity analysis results are depicted in Fig. 8. Notably, the MAPE remains relatively consistent within the range of 10$s$ to 100$s$. This can be attributed to the accurate representation of power consumption trends through the use of splines, the incomplete nature of node power consumption data in DynamicTRR, and the continuous calibration of HighRPM during the active learning stage.

**Table 9: Evaluation results of HIGHRPM on unseen applications on x86 system.**

| Type | Model | Temporal Resolution | | | Spatial Resolution | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $P_{Node}$ | | | $P_{CPU}$ | | | $P_{MEM}$ | | |
| | | MAPE (%) | RMSE | MAE | MAPE (%) | RMSE | MAE | MAPE (%) | RMSE | MAE |
| Linear | LR | 11.48 | 33.79 | 18.57 | 15.31 | 22.56 | 14.75 | 35.77 | 20.59 | 18.35 |
| | LaR | 11.48 | 33.79 | 18.57 | 15.95 | 23.23 | 14.71 | 36.53 | 23.28 | 19.88 |
| | RR | 11.33 | 33.42 | 18.09 | 14.98 | 22.1 | 14.38 | 36.47 | 20.92 | 18.25 |
| | SGD | 11.31 | 33.28 | 17.96 | 15.07 | 22.59 | 14.96 | 36.03 | 20.94 | 18.19 |
| Nonlin. | DT | 10.62 | 39.82 | 18.15 | 11.03 | 18.86 | 12.92 | 32.35 | 21.05 | 17.54 |
| | RF | 10.10 | 37.56 | 16.33 | 9.77 | 16.43 | 11.56 | 31.49 | 20.51 | 17.33 |
| | GB | 9.47 | 36.46 | 15.58 | 18.36 | 26.84 | 16.22 | 39.82 | 22.64 | 20.35 |
| | KNN | 10.31 | 37.94 | 16.30 | 10.74 | 17.39 | 11.21 | 28.03 | 18.35 | 15.51 |
| | SVM | 9.62 | 36.03 | 15.13 | 18.88 | 27.84 | 16.52 | 32.33 | 20.59 | 17.84 |
| | NN | 11.16 | 35.80 | 20.69 | 14.30 | 21.31 | 13.16 | 28.64 | 19.01 | 16.17 |
| RNN | GRU | 7.24 | 29.15 | 12.65 | 10.10 | 28.70 | 11.10 | 19.44 | 8.34 | 6.47 |
| | LSTM | 15.06 | 43.01 | 21.93 | 9.53 | 29.33 | 9.67 | 21.43 | 8.93 | 7.23 |
| **TRR** | Spline | 4.38 | 25.55 | 9.10 | – | – | – | – | – | – |
| | StaticTRR | 5.20 | 25.42 | 10.03 | – | – | – | – | – | – |
| | DynamicTRR | 3.48 | 8.59 | 4.77 | – | – | – | – | – | – |
| **SRR** | SRR | – | – | – | 9.94 | 25.02 | 12.40 | 10.64 | 5.54 | 3.44 |



(a) $P_{CPU}$



(b) $P_{MEM}$

**Figure 9: The impact of different frequency levels.**

*6.4.2 Frequency.* We set three CPU frequency levels, *min* (1.4GHz), *mid* (1.8GHz), and *max* (2.2GHz), to explore the sensitivity of HIGHRPM to CPU frequency changes. Fig. 9 gives the MAPEs of $P_{CPU}$ and $P_{MEM}$ when the program `Graph500` runs at different frequency levels. As seen from Fig. 9, HIGHRPM can accurately predict instantaneous CPU and memory power consumption at all frequency levels. And the higher the frequency, the lower accuracy of $P_{CPU}$ and $P_{MEM}$. The higher the frequency, the CPU activity increased, so the more difficult the modeling. But even the highest (10% for $P_{CPU}$ and 14% for $P_{MEM}$) is far lower than other modeling methods, 5%-21% for $P_{CPU}$ and 5%-15% for $P_{MEM}$.

*6.4.3 Hyperparametric analysis.* We now analyze how sensitive HIGHRPM is to the model hyperparameters. For TRR, particularly DynamicTRR, we experiment with LSTM layers ranging from 1 to 100 to determine the ideal network structure for leveraging node information. The results indicate that accuracy initially increases and then decreases as the number of layers increases, with the best performance observed at two layers. This suggests that a simpler network structure with fewer layers effectively exploits node information, leading us to adopt a simple MLP for SRR modeling. For SRR, we vary the number of hidden layers and observe that the influence of node power consumption on model accuracy diminishes with deeper hidden layers. These findings inform the establishment of the optimal model structure described in Sec. 4.

*6.4.4 Extension to peripheral devices.* Although this paper focuses on CPUs and memory components, we would like to explore the potential extension of HIGHRPM to other peripherals. We believe that the concepts can be applied to peripheral devices that have PMCs, as power modeling for these peripherals also relies on PMCs, albeit with different monitoring units. For instance, GPU power modeling can benefit from HIGHRPM, as it shares similarities with CPU power modeling. However, GPU power modeling often requires additional considerations specific to the architecture [15]. Adapting HIGHRPM to GPU would involve adjusting the model to utilize GPU performance counters and collecting training data on the target platform. However, the methodology for training and using the models would remain largely unchanged. In future research, we plan to explore resolution recovery for other peripheral devices.

*6.4.5 Hardware cost and overhead.* As a software-centric technique, HIGHRPM does not incur additional hardware costs. Furthermore, HIGHRPM can work in both single and multi-node setups. Additionally, it took less than 10 minutes to train our model offline and less than 2 seconds to fine-tune the offline-trained model. Taking an application with a runtime of 10 minutes as an example, active learning led to a performance loss of approximately 3% after 10 adjustments, with a fine-tuning time of around 2 seconds. The prediction latency is less than 1 *ms* at both node and component levels.

*6.4.6 Limitations.* On rare occasions, the miss_interval of may fluctuate due to issues such as network congestion, which in turn can pose a challenge to the overall robustness of HIGHRPM. Specifically, for DynamicTRR, multiple samples (#samples=miss_interval) are combined to generate a single sample that should contain one measured $P_{Node}$. If some of the input samples do not contain the actual $P_{Node}$ value, it could negatively impact the accuracy of the final prediction.

## 7 CONCLUSIONS

We have presented HighRPM, a novel approach to improving the resolution for power reading of CPU and memory components. It combines coarse-grained power sensor readings and modeling techniques to improve the temporal and spatial resolution of power modeling. To provide high-frequent power reading in the temporal domain, HighRPM develops TRR models to predict the long-term power trend and the short-term fluctuations in power consumption. To improve spatial coverage, HighRPM uses a bi-directional method to distribute the node-level power readings to the computing components.

## REFERENCES

[1] [n. d.]. Linux Perf. http://perf.wiki.kernel.org/.

[2] 2001. Intel® Xeon® Processor E5-2660 v2. https://ark.intel.com/content/www/us/en/ark/products/75272/intel-xeon-processor-e52660-v2-25m-cache-2-20-ghz.html.

[3] 2017. Graph500. https://graph500.org/.

[4] 2017. SPEC CPU 2017. http://www.spec.org/cpu2017/.

[5] 2020. High Performance Conjugate Gradients. https://www.hpcg-benchmark.org/.

[6] 2020. The Princeton Application Repository for Shared-Memory Computers. https://parsec.oden.utexas.edu/.

[7] 2023. OMEGAWATT. http://www.omegawatt.fr/.

[8] 2023. WATTSUP. http://www.wattsupmeters.com/.

[9] 2023. ZES LMG450. https://www.zes.com/en/Products/Precision-Power-Analyzers/LMG450.

[10] AMP. 2012. "AMD Opteron 6200series processors Linux tuning guid.

[11] Daniel Bedard et al. 2010. PowerMon: Fine-grained and integrated power monitoring for commodity computer systems. In *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*. 479–484. https://doi.org/10.1109/SECON.2010.5453824

[12] Ramon Bertran et al. 2013. A Systematic Methodology to Generate Decomposable and Responsive Power Models for CMPs. *IEEE Trans. Comput.* 62, 7 (2013), 1289–1302. https://doi.org/10.1109/TC.2012.97

[13] Andrea Borghesi and Andrea Bartolini et al. 2016. Predictive Modeling for Job Power Consumption in HPC Systems. 181–199. https://doi.org/10.1007/978-3-319-41321-1_10

[14] Carnes Brian. 2001. The SMG2000 Benchmark Code.

[15] Robert A. Bridges, Neena Imam, and Tiffany M. Mintz. 2016. Understanding GPU Power: A Survey of Profiling, Modeling, and Simulation Methods. *ACM Comput. Surv.* 49, 3, Article 41 (sep 2016), 27 pages. https://doi.org/10.1145/2962131

[16] Juan Chen and Xinxin Qi et al. 2021. More bang for your buck: Boosting performance with capped power consumption. *Tsinghua Science and Technology* 26, 3 (2021), 370–383. https://doi.org/10.26599/TST.2020.9010012

[17] Rong Ge and Xizhou Feng et al. 2010. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Transactions on Parallel and Distributed Systems* 21, 5 (2010), 658–671. https://doi.org/10.1109/TPDS.2009.76

[18] Neha Gholkar, Frank Mueller, and Barry Rountree. 2019. Uncore Power Scavenger: A Runtime for Uncore Power Conservation on HPC Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '19)*. Association for Computing Machinery, New York, NY, USA, Article 27, 23 pages. https://doi.org/10.1145/3295500.3356150

[19] Daniel Hackenberg and Thomas Ilsche et al. 2014. HDEEM: High Definition Energy Efficiency Monitoring. In *2014 Energy Efficient Supercomputing Workshop*. 1–10. https://doi.org/10.1109/E2SC.2014.13

[20] Thomas Ilsche et al. 2015. Power measurements for compute nodes: Improving sampling rates, granularity and accuracy. In *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*. 1–8. https://doi.org/10.1109/IGCC.2015.7393710

[21] Thomas Ilsche et al. 2019. Power measurement techniques for energy-efficient computing: reconciling scalability, resolution, and accuracy. *Computer Science - Research and Development* 34 (03 2019). https://doi.org/10.1007/s00450-018-0392-9

[22] C. Isci and M. Martonosi. 2003. Runtime power monitoring in high-end processors: methodology and empirical data. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* 93–104. https://doi.org/10.1109/MICRO.2003.1253186

[23] Dongarra Jack et al. 2019. HPL-AI Mixed-Precision Benchmark. https://icl.bitbucket.io/hpl-ai/.

[24] A. Libri, A. Bartolini, and L. Benini. 2021. DiG: Enabling out-of-Band Scalable High-Resolution Monitoring for Data-Center Analytics, Automation and Control (Extended). *Cluster Computing* 24, 4 (dec 2021), 2723–2734. https://doi.org/10.1007/s10586-020-03219-7

[25] Piotr R Luszczek et al. 2006. S12-The HPC challenge (HPCC) benchmark suite. In *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, November 11-17, 2006, Tampa, FL, USA*.

[26] John C. McCullough et al. 2011. Evaluating the Effectiveness of Model-Based Power Characterization. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference* (Portland, OR) *(USENIX ATC'11)*. USENIX Association, USA, 12.

[27] Kenneth O'brien, Ilia Pietri, Ravi Reddy, Alexey Lastovetsky, and Rizos Sakellariou. 2017. A Survey of Power and Energy Predictive Models in HPC Systems and Applications. *ACM Comput. Surv.* 50, 3, Article 37 (jun 2017), 38 pages. https://doi.org/10.1145/3078811

[28] Pavlos Petoumenos, Lev Mukhanov, Zheng Wang, Hugh Leather, and Dimitrios S Nikolopoulos. 2015. Power capping: What works, what does not. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 525–534.

[29] Michael D. Powell, Arijit Biswas, Joel S. Emer, Shubhendu S. Mukherjee, Basit R. Sheikh, and Shrirang Yardi. 2009. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. 289–300. https://doi.org/10.1109/HPCA.2009.4798264

[30] E. Rotem et al. 2012. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro* 32, 2 (2012), 20–27.

[31] Mark Sagi, Martin Rapp, Heba Khdr, Yizhe Zhang, Nael Fasfous, Nguyen Anh Vu Doan, Thomas Wild, Jörg Henkel, and Andreas Herkersdorf. 2021. Long Short-Term Memory Neural Network-based Power Forecasting of Multi-Core Processors. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1685–1690. https://doi.org/10.23919/DATE51398.2021.9474028

[32] Mark Sagi and Nguyen Anh Vu Doan et al. 2022. Fine-Grained Power Modeling of Multicore Processors Using FFNNs. *Int. J. Parallel Program.* 50, 2 (apr 2022), 243–266. https://doi.org/10.1007/s10766-022-00730-9

[33] Mark Sagi et al. 2020. Fine-Grained Power Modeling of Multicore Processors Using FFNNs. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Alex Orailoglu, Matthias Jung, and Marc Reichenbach (Eds.). Springer International Publishing, Cham, 186–199.

[34] Mark Sagi et al. 2020. A Lightweight Nonlinear Methodology to Accurately Model Multicore Processor Power. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 3152–3164. https://doi.org/10.1109/TCAD.2020.3013062

[35] Karan Singh et al. 2009. Real Time Power Estimation and Thread Scheduling via Performance Counters. *SIGARCH Comput. Archit. News* 37, 2 (jul 2009), 46–55. https://doi.org/10.1145/1577129.1577137

[36] Shuaiwen Song, Chunyi Su, Barry Rountree, and Kirk W. Cameron. 2013. A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 673–686. https://doi.org/10.1109/IPDPS.2013.73

[37] Feihao Wu and Juan Chen et al. 2019. A Holistic Energy-Efficient Approach for a Processor-Memory System. *Tsinghua Science and Technology* 24, 4 (august 2019), 468–483.

[38] Xingfu Wu et al. 2022. Performance and power modeling and prediction using MuMMI and 10 machine learning methods. *Concurrency and Computation: Practice and Experience* (08 2022). https://doi.org/10.1002/cpe.7254

[39] X. J. Yang and XK. Liao et al. 2011. The TianHe-1A Supercomputer: Its Hardware and Software. *Journal of Computer Science Technology* (2011).