A Conformal Prediction-Based Framework for CPU Load Forecasting: A Black-Box Approach

Edin Jelačić, Cristina Seceleanu,
Peter Backeman, Ning Xiong, Tiberiu Seceleanu
School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
{edin.jelacic, cristina.seceleanu,
peter.backeman, ning.xiong, tiberiu.seceleanu}@mdu.se

Axel Jantsch
Institute of Computer Technology
Vienna University of Technology
Vienna, Austria
axel.jantsch@tuwien.ac.at

***Abstract*—To address safety concerns in industrial systems, we propose a framework for forecasting CPU load with respect to a predetermined threshold, allowing customers to add tasks from a predefined library. Existing tools, akin to Windows Task Manager, provide limited insights due to their aggregate nature and high computational overhead. Our approach uses conformal prediction for rapid uncertainty-aware forecasts and Shapley value analysis to quantify individual task contributions to the CPU load. This proof-of-concept framework improves system safety assessment by addressing key research questions in load prediction and validation, paving the way for refined measurement methodologies in industrial applications.**

***Index Terms*—conformal prediction, Shapley, CPU, forecasting, load.**

I. Introduction

Industrial hardware systems power critical applications in fields like cybersecurity, telecommunications, power distribution and automatic control. These are devices such as programmable logic controllers (PLCs), embedded systems, and electronic control units (ECUs). Operating in real-time, these devices demand reliability and specialized expertise. Manufacturers typically deliver these systems pre-configured with essential tasks such as control algorithms, measurements, signal processing routines, and kernel-level processes. The systems are designed to run immediately upon delivery. However, when customers need to add or modify tasks to suit evolving operational conditions, the increasing complexity can overload the device's processing unit(s) and compromise real-time performance [1], [2].

To avoid such undesirable situations, we propose a user-friendly machine-learning-based framework that forecasts the processor load before deploying any new task configuration. Our approach is inspired by monitoring tools like the *Windows Task Manager* and UNIX's *top*/*htop*, tools that directly measure the proportion of CPU time used by tasks versus idle time [3], [4]. However, unlike these tools that provide online measurements of load, we utilize statistical methods to provide offline load forecasts and an understanding of individual task impacts. We employ *conformal prediction* [5] for reliable uncertainty quantification and *Shapley value analysis* [6] to explain each model prediction. Although industrial use-case forecasting has received attention in the literature [7]–[9], the mentioned combination of techniques has not been applied to CPU load forecasting before. Our framework provides detailed load information for individual cores in multi-core systems and quantifies the impact that each task has on overall computing load. Due to practical constraints on the procedure of extracting the data from the real device, such as excessive time and labor required, our experiments focus on a limited set of configurations. This limitation motivates our framework's design, which offers continuous, non-intrusive analysis without the need for deep architectural knowledge.

A. Objectives and Contributions

Our goal is to create a statistical framework for uncertainty-aware explainable black-box processor load prediction for an embedded system processing unit (or units, generally in the multi-core case). This research goal is rooted in an industrial problem important to a large partner company, with the target of processor load forecasting with a degree of confidence that quantifies the certainty of a device setup to not cause the total load crossing a certain load threshold. Based on this need, we extract prominent research topics that we tackle in the paper. Our integrated framework combines conformal prediction with Shapley values to provide statistically sound prediction intervals alongside clear, interpretable insights into feature contributions. Additionally, we have implemented a user-friendly GUI, which enhances the practical application of our framework by simplifying its deployment and use in operational settings, thus addressing the customer-side device programmability queries. The applicable workflow of the framework that we propose is presented in fig. 1.

II. Our Methodology

A. Data Extraction and Preprocessing

To test our framework, we utilize an experimentally attained processor load dataset from a unit manufactured and pre-

The authors are partially supported by the Swedish Knowledge Foundation via the projects PerFlex (*Performant and Flexible digital Systems through Verifiable Artificial Intelligence*), grant nr. 20220033, and IGP-IDS (*International guest professor of Intelligent Distributed Systems*), grant nr. 20230147.

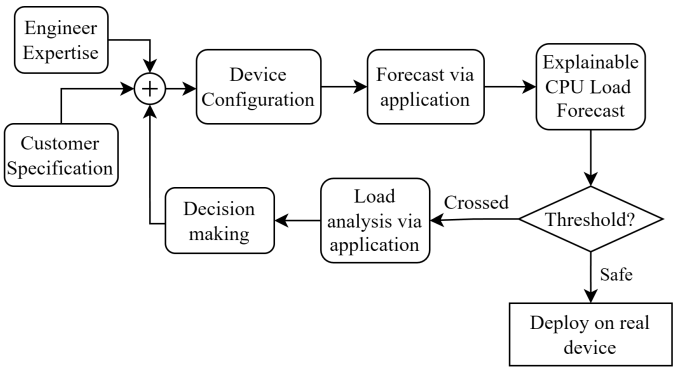


Fig. 1: Workflow of the framework, from expertise and specification to deployment

programmed by our industrial partner. The device in question is a dual-core real-time measurement instrument with a task set extensible by proprietary software provided by the manufacturer. We loaded the instrument onto a test environment that mimicks how it operates in customer facilities. For our experiments, we selected a limited subset of 27 tasks from the full range of tasks running on the device, which is in the hundreds. There also exist many other tasks running on the instrument as part of its operating system and some of them interact with some of the ones we have selected, which may present unobserved confounding features for our statistical approach. Formally, letting $\mathbb{B} = \{0, 1\}$, we represent our dataset as $\mathbb{B}^{27}$, where each component corresponds to one of the 27 tasks. Each vector $\vec{f}_{27\times 1} \in \mathbb{B}^{27}$ represents a unique configuration of tasks being active or inactive during testing. Note that, due to practical constraints, our experiments are conducted on this finite set of configurations, toggled via a script that sequentially activates and deactivates individual functions using the manufacturer's software, as well as performs logging and raw data pickup.

The device is equipped with a basic internal CPU load measurement system. The measurement system produces relatively little overhead on the device, around 1% as reported by the manufacturer, and does not impact the load measurement process. Its margin of error is $\pm 2\%$, with an updating frequency of 0.5 Hz, and it reports three components independently:

- Overall CPU (Dual Core) load,
- Processor 0 (Core 0) load,
- Processor 1 (Core 1) load.

An example of the measurement system's logs is as per single row:

```
0533394390 14-06:12:04.038558
279: CPU load: 45.48 %
CPU0: 36.01 %   CPU1: 54.96 %
free: 109.03 % of UP
```

This row states that the overall processor load, as measured by the device at 6 hours, 12 seconds and approximately 4 seconds, is 45.48%, the processor 0 load is 36.01%, processor 1 load is 54.96% The raw data was in need of several preprocessing steps. CPU load logs are filtered to remove configuration-switching spikes, reordered to reflect the actual configuration sequence, and aggregated — each configuration is sampled for about one minute at 0.5 Hz —to yield the metrics in table I, with the resulting data stored in CSV format. Moreover, we must stress that, due to the way in which the experimental setup works, the datapoints we acquire are **independent** and **identically distributed** (IID), which is an otherwise difficult to make assumption on data, opening the door to later conformal treatment. In this work, we focus on the **mean** load of each configuration, i.e., for our dataset with N entries, for each $f_i, i \in [1, N]$, we focus on $\bar{L}_0(f_i)$, $\bar{L}_1(f_i)$ and $\bar{L}_{dual}(f_i)$ and their numerical relationships, as well as the relationships of their confidence intervals to the threshold load values.

TABLE I: Data attained from instrument log processing

Processor	Avg. load $\bar{L}$	Std. deviation σ	Minimum load L_{min}	Maximum load L_{max}
0	L_0	σ_0	$L_{0,min}$	$L_{0,max}$
1	L_1	σ_1	$L_{1,min}$	$L_{1,max}$
Dual	L_{dual}	σ_{dual}	$L_{dual,min}$	$L_{dual,max}$

B. Conformal Prediction Framework

Conformal prediction (CP) is a statistical method used in machine learning, to predict the probability of correctly classifying new observations [10]–[12]. This method is a statistically sound approach to creating rigorous uncertainty sets (in the case of classification tasks) or intervals (in the case of regression tasks, which we concern ourselves with in this work). These intervals are statistically guaranteed to contain the true value of the sample with a predetermined probability, which is generally chosen to be 90%. Furthermore, CP takes a black-box approach, that is, the intervals generated by this approach are valid without having any prior knowledge of the model trained to model the distribution of the underlying data, as long as the scoring function utilized by CP models appropriately how well the prediction of a sample "conforms" to the training data. This means that we can apply CP to many types of regression models successfully, gradient-boosted trees, random forests and neural networks alike, which gives us breadth in terms of adequate model selection and explainability, as well as provide quantification of uncertainty in situations such as ours, where one must be able to decide how "safe" a certain task configuration is with regards to CPU overloading. This gives the engineer a statistically sound insight into what can be put into production. This is even more significant since the procedure of testing is tedious, laborious and time-consuming, so it is to be avoided if possible. For instance, let us assume that we have 5 task configurations to predict. In fig. 2 we see a direct comparison between mere point forecasting and CP interval prediction. We see all the possibilities with respect to the relationship between the predicted load values and the threshold, which is 70 in this case. The *safe* CPU load zone is marked with green, the *unsafe* zone with red. In the point forecast case, configuration 1 appears clearly safe, whereas configuration 3 appears clearly unsafe. However, the situation is not so clear for configurations 2, 4 and 5. Without a

prediction interval, it is impossible to include a quantification of uncertainty into the decision process of whether to allow these configurations on the machine. However, once CP is applied, we can clearly estimate how "certainly" safe or unsafe a particular configuration is. We see that configuration 2 is deemed likely safe, whereas configuration 4, despite seeming unsafe at first, may likely produce a non-overloading load. On the other hand, configuration 5 is deceptively safe, given that CP produces bounds that are mostly in the *unsafe* zone. This is the way in which CP helps inform point forecasting, however the benefits go the other way as well. Point forecasts inform the user of the direction of the interval to put more emphasis on, when deciding whether to implement a task configuration on the device.

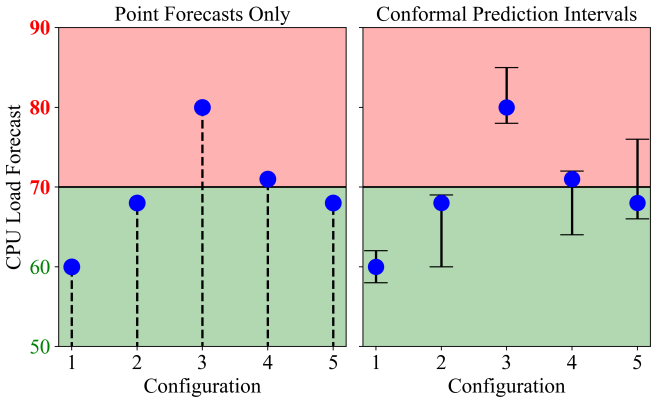


Fig. 2: Point forecasting of CPU load versus intervals around predictions generated by conformal prediction

Normally the paradigm of regression learning requires the splitting of the dataset that models our underlying distribution, into a *training* dataset and a *test* dataset. However, to be able to create the CP intervals, the *training* dataset is once again split, and from the training data we reserve a moderately sized representative sample that we term the *calibration* dataset. Upper and lower quantile regressors (QRs) are trained on the training samples so that, by definition, there is a 5% probability that the true label values fall below the lower quantile regressor (0.05), and a 5% probability that they exceed the upper quantile regressor (0.95). QRs are used as a base, and are in subsequent steps conformalized to statistical certainty. Although in implementation this step differs slightly between models, the statistical guarantee remains identical between them. In general, for the miscoverage error, we denote and set $\alpha = 0.1$, so that the coverage rate is $\mathbb{P} = 1 - \alpha = 0.9$.

We denote the features $X \in \mathbb{B}^{27}$, $X_i = \vec{f_i}$, $i = 1, ..., N$ and $y = \{y_1, y_2, \ldots, y_N\}$ of vectors in $\mathbb{R}^3$ such that $y_i = (\bar{L}_{0,i}, \bar{L}_{1,i}, \bar{L}_{dual,i})$ is the vector of mean loads for the i-th observation. We take N_{cal} samples from the training dataset for calibration. Let $X_{train} \subset \mathbb{B}^{27}$ and $y_{train} \subset y$ have length $N_{train} = |X_{train}|$, and $X_{test} = X \setminus X_{train}$ and $y_{test} = y \setminus y_{train}$ have length $N_{test} = N - N_{train} - N_{cal}$.

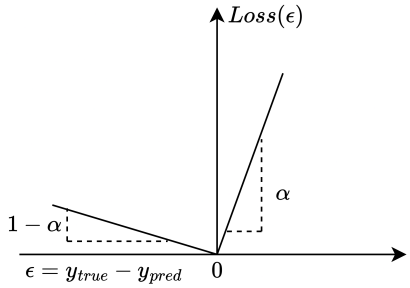


Fig. 3: Pinball loss (Quantile loss) function

Finally, let $X_{cal} = \mathbb{B}^{27} \setminus (X_{train} \cup X_{test})$ and $y_{cal} = y \setminus (y_{train} \cup y_{test})$ have length N_{cal}.
Our conformal prediction framework proceeds as follows:

Step 1: Train the Base Point Forecast Model $\mathbf{M}$ We begin by training a neural network model $\mathbf{M}$ on the training data (X_{train}, y_{train}), so that for each observation X_i the model outputs the prediction $\mathbf{M}(X_i) = \hat{y}_i \approx y_i$. Training is performed over multiple epochs using mean squared error (MSE) as the loss function.

Step 2: Fit Upper and Lower Quantile Regressors For each output component (i.e. for $\bar{L}_0, \bar{L}_1, \bar{L}_{dual}$), we train two quantile regressors on the calibration set:

- A **lower QR** to estimate the $\alpha/2$ quantile of the residuals,
- An **upper QR** to estimate the $1 - \alpha/2$ quantile.

We denote these regressors QR_{lower} and QR_{upper}. They are fitted using the training features (X_{train}, y_{train}) and the corresponding residuals for each output on the asymmetric pinball loss (or quantile loss) metric, whose specific graph can be seen in fig. 3. Although they are normally usable for the purposes of estimating confidence intervals, in their present state they do not provide coverage guarantees and hence we will conformalize them.

Step 3: Compute Residuals Following the guidelines for split conformal prediction outlined in [10], our conformal score $s(x, y)$ is based on the difference between the predicted label value and the nearest quantile value of the calibration samples, as per equation:

$$s(x, y) = \max\left(QR_{lower}(x) - y, y - QR_{upper}(x)\right),$$

with $x \in X_{cal}, y \in y_{cal}$. Once the conformal scores are calculated, the corresponding quantiles for the core 0, core 1 and dual core predictions are calculated as: $\hat{q} = \text{Quantile}\left(s_1, s_2, ...s_n; \frac{\lceil (n+1)(1-\alpha) \rceil}{n}\right)$[1].

Step 4: Construct Prediction Intervals
For a new input x, the model produces a point forecast $\hat{y} = \mathbf{M}(x)$. From the quantile regressors we then get the non-conformalized intervals $[QR_{lower}, QR_{upper}]$, which we then

[1]In [10] the authors suggest adding an empirical correction term $1/(n+1)$ to α (n being the size of the calibration dataset) to correct for the fact that the CP procedure entails a finite sample. Namely, since the quantiles of the score function (in our case the residuals) are estimated from a finite sample, the empirical quantile selection is discrete rather than, ideally, continuous. To properly adjust for this discreteness, instead of using the theoretical $(1 - \alpha)$ quantile directly, we use $\lceil (n + 1)(1 - \alpha) \rceil / n$.

conformalize by adding the corresponding quantiles $\hat{q}$ to form valid prediction intervals according to:

$$\text{Lower bound} = QR_{lower}(x) - \hat{q}, \tag{1}$$

$$\text{Upper bound} = QR_{upper}(x) + \hat{q}. \tag{2}$$

In other words, we expand the quantile regression to achieve the desired coverage.

These intervals, by design, satisfy the CP guarantees, ensuring that the true output is contained within the interval $[\text{Lower bound}, \text{Upper bound}]$ with probability at least $1-\alpha$.

For example, assume that our CP model has been fully trained and calibrated and that we are presented with 100 fresh samples to predict. The samples are binary vectors $\vec{f} \in \mathbb{B}^{27}$. The model first generates a point forecast of CPU load for each sample using the underlying distribution approximator (i.e. neural network). Concurrently, the pre-trained quantile regressors compute the lower and upper residual adjustments which are added to the point forecasts to form statistically valid prediction intervals. If our miscoverage rate α was set to 0.9, our prediction intervals are constructed to achieve a 90% coverage rate. Statistically, this means that out of our 100 fresh samples, approximately 90 will have their true CPU load values fall within the predicted intervals.

C. Shapley Value Integration

Given the systematic nature of our framework in acquiring the dataset—through experimental control and clear separation of individual inputs—a natural question arises: can we measure individual function contributions solely based on the system's inputs and outputs, without delving into the model's internal details? For notational simplicity, let us denote the $\mathbf{M}$ model's output as a 3×3 matrix, where the each column is vector of Upper bound, prediction and Lower bound values for core 0, core 1 and dual core as

$$\begin{aligned}\mathbf{M}(\vec{f}) &= \left[\bar{L}_{0\downarrow}^{\uparrow}, \bar{L}_{1\downarrow}^{\uparrow}, \bar{L}_{dual\downarrow}^{\uparrow}\right] \\ &= \begin{bmatrix} \bar{L}_0^{\uparrow} & \bar{L}_1^{\uparrow} & \bar{L}_{dual}^{\uparrow} \\ \hat{\bar{L}}_0 & \hat{\bar{L}}_1 & \hat{\bar{L}}_{dual} \\ \bar{L}_{0\downarrow} & \bar{L}_{1\downarrow} & \bar{L}_{dual\downarrow} \end{bmatrix}.\end{aligned} \tag{3}$$

To attain estimates for individual task contribution to the load intervals and predicted load values, we integrate Shapley values into our framework in the form of the SHAP library [13]. Our model diagram then becomes as per fig. 4.

Shapley values are an approach to explaining outputs of machine learning models rooted in cooperative game theory. Each of the model inputs, that is, each one of the tasks (with binary value 0 or 1) gets its own Shapley value that it relates to each of the output components from the matrix in eq. (3). Therefore, the total output of our framework amounts to

$$\overbrace{27}^{\text{\# of inputs}} \times \underbrace{3}_{\text{core 0, 1, dual}} \times \overbrace{3}^{\text{upper, lower, prediction}} = 243$$

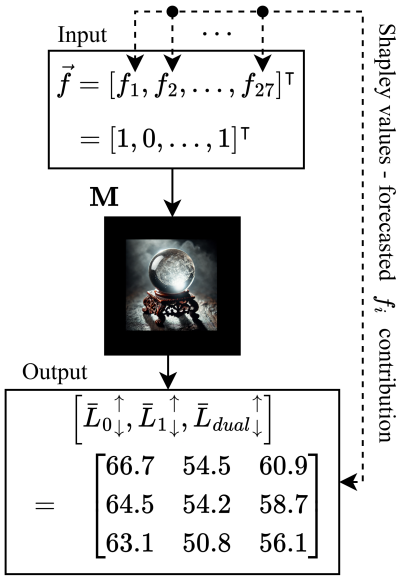


Fig. 4: Framework outline

individual values. The way Shapley values work is by providing a principled way to quantify the contribution of each parameter by averaging its marginal impact over all possible feature subsets. In our context, this means that for each prediction, we compute the Shapley value ϕ_i for every feature i, which reflects how much the presence of that feature alters the forecasted CPU load and the interval.

For a set of features Φ and a prediction function γ, the Shapley value for feature $i \in \Phi$ is calculated as the weighted average of the increase of the output value when i is active with the group of features $S \subseteq \Phi$ versus when i is inactive with group S. This averaging is done over all possible subsets S where i is not in, and formally the definition is:

$$\overbrace{\phi_i}^{\text{i's Shapley}} = \sum_{S \subseteq \Phi \setminus \{i\}} \underbrace{\frac{|S|!(|\Phi| - |S| - 1)!}{|\Phi|!}}_{\text{S's weight}} \overbrace{\Big[\gamma(S \cup \{i\}) - \gamma(S)\Big]}^{\text{i's marginal contribution}}, \tag{4}$$

where $\gamma(S)$ is the model output when only the features in subset S are active [14]. Although calculating this sum exactly is computationally expensive (in fact, calculating Shapley values in general is an NP-hard problem [15]), efficient approximations such as KernelSHAP or TreeSHAP from the SHAP library allow us to estimate these values in practice.

In our framework, the conformal prediction method guarantees that the overall prediction interval has a positive length. By design, the upper bound is always higher than the lower bound. However, when we decompose the interval using Shapley values, we calculate the marginal contribution of each feature to the lower and upper bounds independently. This means that a given feature may have a stronger impact on one bound than the other. For instance, if for a specific feature f_1

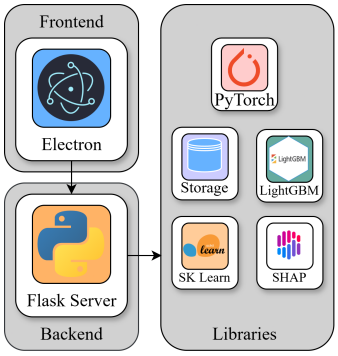


Fig. 5: Utilized application stack

we obtain

$$\phi_{1\downarrow}(\vec{f}) = 5.0 \;\&\; \phi_1{}^{\uparrow}(\vec{f}) = 4.0,$$

it indicates that f_1 contributes more to the lower bound than to the upper bound. In practical terms, f_1 is pushing the lower bound upward, i.e., reducing the safety margin more significantly than it is elevating the upper bound. This distinction is important because it offers deeper insight into the model's behavior: even though the overall interval remains valid and of positive length, the individual contributions reveal that certain features may disproportionately influence the conservative (lower) aspect of the interval. Such insights are valuable in safety-critical applications, where understanding the source of prediction conservatism can guide more informed decisions.

By applying these methods, we obtain local explanations that detail the influence of each parameter on the CPU load forecast. This not only aids in understanding model behavior but also highlights which features are most critical to system performance, thereby supporting more informed decision-making in industrial applications.

D. System Integration & GUI Application

For the integration of this system our target was a portable and platform-independent, easy-to-use mechanism with which experienced device engineers can estimate CPU loads and function contributions at the click of a few buttons with minimal time spent learning a new methodology of interfacing with the underlying algorithms. The application is composed in the Electron front-end framework and runs a Python Flask server on the back-end. Communication between the two is achieved via HTTP requests. The application stack is shown in fig. 5.

The entire application stack is in this stage kept to a minimum, as the core of the development is focused on the underlying algorithms. This also keeps development overhead low and strikes a moderate balance between modularity (extensibility) and code maintainability. Upon opening the application, the user is welcomed with the screen shown in fig. 6. The user can choose whether to train a new model with customized hyperparameters or load an existing model from the application storage. If the user chooses to train a new model, they can choose from the implemented model types.

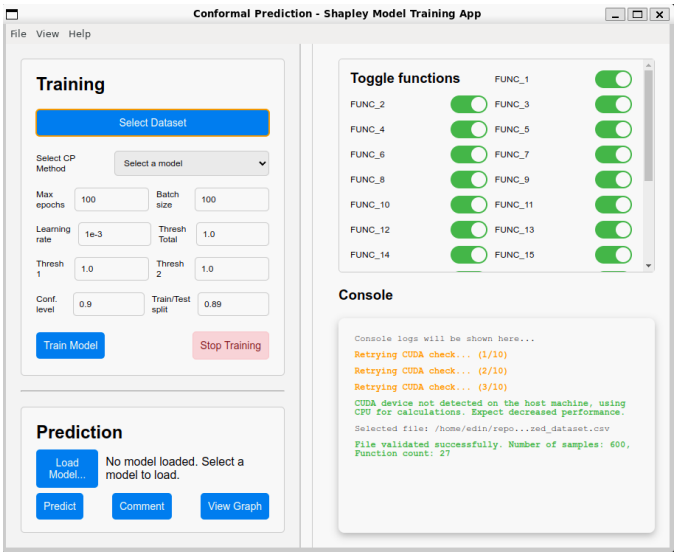


Fig. 6: Initial GUI screen - The left hand side is divided into Training and Prediction sections. The Training section uses user input for various general model training parameters, the Prediction section controls the loading of a trained model, annotations and graph window. The right hand side is divided into the Toggle section where tasks are toggled On/Off for prediction and the Console section where various information, warnings and errors are shown to the user.

The core idea of the GUI for our framework is to interactively provide actionable insights into the device under development, while abstracting away low-level ML-related design decisions (and still providing the optional functionality to perform them).

The selection of the model opens the modal window wherein the user specifies hyperparameters. An example of such a modal for the neural network model is in fig. 7. The model training is tracked throughout the console through information and warning lines presented to the user. Once the model is trained, it is stored to the application database, along with the metadata containing information about the machine used for training, the duration of the training, the hyperparameters used and any model comments left by the user. If the user chooses to load a model, the model selection is sent to the backend for server-side model verification, and if it is successful, the user is presented with console output showing the details of the loaded model's structure.

The user can switch the available tasks on/off in the right-hand side of the screen and execute predictions. Each prediction, along with its metadata, is stored in the application storage. The user can choose to view the graphs for the predictions, whereupon he is presented with three screens. An example of a screen for Core 0 with 5 measurements may be seen at fig. 8, with analogues for Core 1 and Dual Core. Each screen represents predictions for Core 0, Core 1 and Dual Core respectively with mean load intervals and predictions, as well as individual task Shapley values for the lower contribution, prediction contribution and upper contribution.

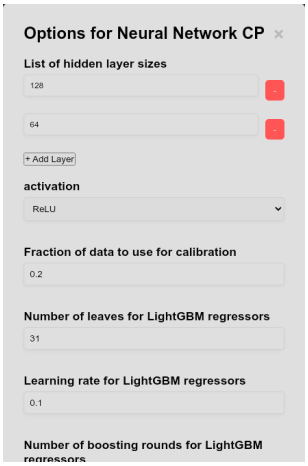


Fig. 7: Modal that opens inside the GUI once a model type is selected. Offers model-specific hyperparameters for input. Default values are always set to what has been (through our experiments) deemed satisfactory, but setting these values has been left to the user.

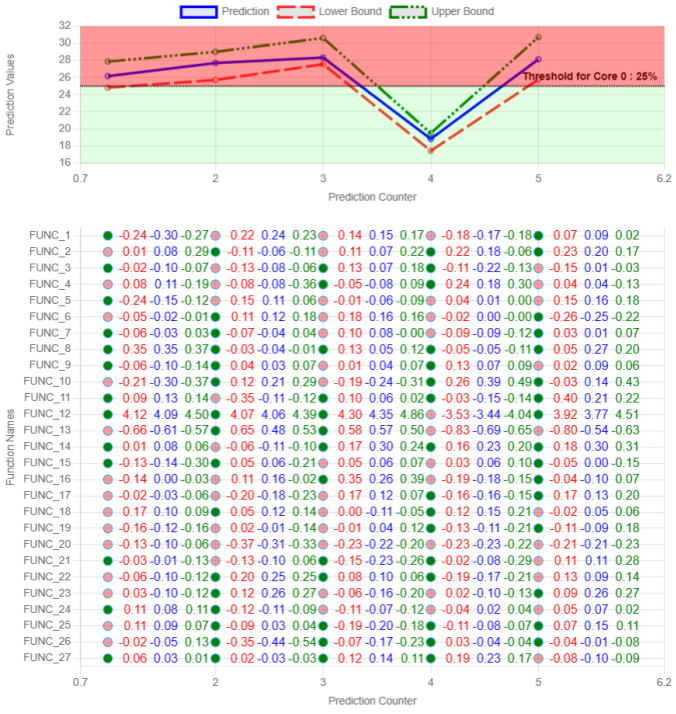


Fig. 8: Graph window for the predicted CPU loads for Core 0. Displayed with 5 predictions and a set threshold of 25%. Active tasks are colored green, while inactive ones are colored red. Individual task contributions are shown as a triplet: contribution to lower bound, contribution to predicted value and contribution to the upper bound.

Shapley value charts allow users to understand which tasks have significant impact on the predicted loads. For example:

- A task with a **high positive contribution** to the upper bound indicates that its presence increases the estimated maximum load, such as *FUNC_12* in the first prediction in fig. 8,
- Conversely, a task with a **low or negative contribution** suggests it has less influence or even reduces the predicted load, such as *FUNC_20* in fig. 8.

By analyzing these contributions across all 27 features, we can identify which tasks are critical in shaping the model's predictions. This information is particularly valuable for understanding why certain prediction intervals are wider or narrower and how specific tasks contribute to the overall safety margins (lower bounds) or performance expectations (upper bounds).

The matrix in eq. (3) provides a structured view of these contributions, where each task's Shapley value reflects its marginal impact on the predicted values. For instance, if a task has unequal contributions to the lower and upper bounds, it highlights an asymmetric effect indicating that the task influences one aspect of the prediction more strongly than the other.

Ultimately, this integration enables users to interpret not only what the model predicts but also why specific tasks are driving those predictions. This transparency is crucial for making informed decisions in safety-critical applications where understanding the root causes of prediction intervals is essential. For example, assume that we must implement functionality achievable by either loading task A or task B onto the device and the set is already predicted to be operating close to the CPU threshold. Both tasks may take the predicted load over the threshold, perhaps similarly so, but if task A has a lower contribution to the upper bound of the prediction than task B, we can quantify via Shapley values the inherent risk of choosing to implement the functionality with task B over task A w.r.t. the threshold. In this case, despite similar predicted load, task B is deemed more dangerous than task A, and we would proceed with implementation using task A.

III. Experimental Setup

A. Dataset Description

In this study, we utilize a dataset collected from the aforementioned industrial hardware system. The dataset consists of 27 binary input features representing task configurations, as outlined previously in the paper, along with corresponding CPU load measurements. The dataset contains 7129 samples, wherein 25% or 1784 samples are left out for testing the approach, 37.5% or 2673 utilized as the calibration dataset for the split conformal prediction algorithm and the remaining 37.5% (2673) for the predictor training. For the corresponding task configurations we record the NN's outputs: point predictions, lower bounds, and upper bounds. These measurements allow us to assess both the performance of our predictive framework and the interpretability of feature contributions.

B. Experimental Design

Our evaluation framework is two-fold:

- **Coverage, Interval length and Prediction Error:** We assess the predictive performance of our conformal prediction model using standard metrics:
  - **Coverage Rate:** The percentage of true CPU load values that fall within the predicted intervals, ideally meeting the specified confidence level (e.g., 90%).
  - **Interval Length:** The size of the intervals themselves, as the smaller they are while retaining coverage guarantees, the higher the model's certainty about the outcome.
  - **Prediction Error:** Metrics such as Mean Squared Error (MSE) are used to quantify the accuracy of the point predictions.
- **Shapley Value Evaluation:** Evaluating the quality of Shapley explanations is challenging. In our experiments, we use the following strategies to validate and interpret the Shapley values computed for each function:
  - **Additivity Check:** For each prediction, we verify that the sum of the baseline prediction and the computed Shapley values closely approximates the model's output. This confirms the internal consistency of the Shapley decomposition.
  - **Sensitivity Analysis:** Features with high Shapley values should cause larger changes when perturbed, thereby supporting the validity of the feature attributions. This is particularly significant given that we realized, both from our experimental testing and discussions with the partner experts, tasks *FUNC_12* and *FUNC_13* are some of the most load impacting on the device. The observation of this peculiarity must be maintained throughout testing.
  - **Ablation Study:** By selectively masking tasks one by one (removing their variance from the dataset) and analyzing the impact on the model's prediction, we compare the observed prediction shifts to the corresponding Shapley values. If our Shapley value analysis is correct, these value shifts should not be so significant as to impact the contributions of non-masked tasks.

IV. Results

We first establish a baseline using the unmodified testing dataset. Our framework predicts lower, median, and upper CPU loads for Core 0, Core 1, and Dual Core configurations, and computes Shapley values for each of the 27 tasks for each setup. Next, we perform 27 separate ablations by masking each task in the training, calibration, and testing data one at a time. These ablations simulate scenarios in which some functions present on the device are excluded from the analysis. If our framework continues to operate robustly under these ablations, it implies resilience for the real system when additional functions are implemented. Accordingly, all performance metrics are compared between the baseline and the ablated models.

During discussions with industry experts, it was suggested that *FUNC_12* should serve as a benchmark, as it is anticipated to be the most impactful task. Capturing its influence accurately is crucial for validating our approach, and conversely, if capturing its influence on the system failed to materialize, it would serve as a signal that the framework is flawed.

A. Coverage, Interval Length, and Prediction Error

Table II summarizes the key metrics for both the baseline and ablated models. Overall, the coverage rates remain close to the nominal level of 90% $(1 - \alpha)$ across Core 0, Core 1, and Dual Core measurements, confirming that our conformal prediction (CP) framework is valid for device utilization. Notably, ablating *FUNC_12* does not substantially affect coverage, though it does lead to a significant increase in interval length. This suggests that *FUNC_12* is critical for constraining predictive uncertainty—a result that aligns with our Shapley analysis, which identifies it as the most impactful function.

The relatively small mean interval lengths (except in the case of ablated *FUNC_12* and, to a lesser degree, ablated *FUNC_13*) and consistent MSE values indicate that removing any single function causes only minor deviations compared to the full baseline. This robustness is promising, particularly for the overall device prediction system, as it implies that even if functions not encompassed in our dataset were omitted, the system would not break down dramatically. The device measurement system, therefore, demonstrates resilience to individual task losses, ensuring reliable performance and actionable insights for end users.

In summary, while some functions (notably *FUNC_12*) have a more pronounced effect on model performance, the overall results confirm that the model is robust to the removal of single tasks, with only modest changes in predictive accuracy and uncertainty estimates under ablation.

B. Forecasting Performance

An example of our mechanism's forecasting performance is shown in fig. 9 for Core 0, with similar behavior observed for Core 1 and Dual Core. In this figure, the value of CP intervals is evident. The intervals offer a more informative prediction range than mere point estimates. For example, at prediction indices 5, 6, and 9, the point forecast falls outside the CP interval, yet the interval itself successfully captures the true value. Comprehensive results, including coverage metrics, mean interval lengths, and prediction errors over the test dataset and its ablations are summarized in table II.

C. Additivity Check

For our model, we generate a total of 9 SHAP library explainers, a total which results from combining the 3 Lower, Prediction and Upper outputs with the 3 Core 0, Core 1 and Dual Core setups. For each of these Shapley explainers, it must hold that the model's prediction of the given target value is approximately equal to the sum of the base value (implicit in the Shapley decomposition) plus the individual Shapley

TABLE II: Coverage, Mean Interval Length and Prediction Error for the model M, C0 - Core 0, C1 - Core 1, Dual - Dual Core

	Coverages			Mean Interval Lengths			MSE		
Ablated Function	**C0**	**C1**	**Dual**	**C0**	**C1**	**Dual**	**C0**	**C1**	**Dual**
1	0.89	0.89	0.89	1.32	7.94	4.66	0.05	0.47	0.17
2	0.89	0.89	0.89	1.32	7.93	4.66	0.04	0.45	0.16
3	0.89	0.89	0.89	1.27	7.74	4.58	0.04	0.46	0.16
4	0.89	0.89	0.89	1.32	7.92	4.65	0.04	0.45	0.16
5	0.89	0.89	0.90	1.32	7.93	4.66	0.06	0.48	0.17
6	0.89	0.88	0.89	1.32	7.93	4.66	0.05	0.44	0.16
7	0.89	0.89	0.89	1.27	7.94	4.67	0.06	0.47	0.17
8	0.89	0.89	0.89	1.32	7.93	4.65	0.05	0.46	0.17
9	0.89	0.88	0.89	1.32	7.93	4.67	0.04	0.47	0.17
10	0.89	0.89	0.89	1.33	7.98	4.69	0.05	0.53	0.19
11	0.89	0.89	0.89	1.32	7.92	4.65	0.04	0.47	0.17
12	0.89	0.89	0.90	2.60	17.94	10.26	2.44	70.85	23.76
13	0.88	0.89	0.89	1.48	8.07	4.75	0.26	2.66	1.06
14	0.89	0.89	0.89	1.34	7.93	4.64	0.05	0.45	0.16
15	0.89	0.89	0.90	1.33	7.93	4.66	0.04	0.47	0.16
16	0.88	0.89	0.88	1.32	7.91	4.64	0.06	0.48	0.17
17	0.89	0.88	0.89	1.32	7.90	4.63	0.05	0.49	0.17
18	0.89	0.89	0.89	1.32	7.92	4.66	0.04	0.46	0.16
19	0.89	0.89	0.89	1.32	7.92	4.66	0.04	0.45	0.16
20	0.89	0.88	0.89	1.33	8.00	4.73	0.08	0.75	0.28
21	0.89	0.88	0.89	1.32	7.93	4.66	0.06	0.47	0.16
22	0.89	0.89	0.89	1.33	7.93	4.67	0.06	0.46	0.16
23	0.89	0.88	0.89	1.34	7.93	4.66	0.07	0.48	0.18
24	0.89	0.89	0.89	1.33	7.93	4.66	0.05	0.47	0.16
25	0.89	0.89	0.89	1.35	7.99	4.71	0.06	0.47	0.16
26	0.88	0.88	0.89	1.35	8.01	4.73	0.06	0.46	0.16
27	0.89	0.88	0.89	1.32	7.93	4.67	0.04	0.46	0.16
None	0.89	0.89	0.89	1.32	7.92	4.66	0.06	0.46	0.16

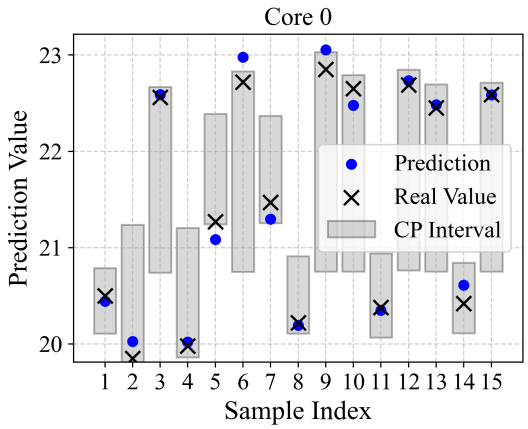


Fig. 9: Outputs of M, i.e. the point forecasts and intervals for Core 0, with the real value shown for demonstration

contributions towards the target value. The effective base value is computed as

$$\text{Base}_{est} = \mathbf{M}(x) - \sum_{i=1}^{27} \phi_i(x)$$

and if the additivity property holds, the base estimate calculated like this should be nearly constant across individual samples. In other words, the difference between the mean $\overline{\text{Base}_{est}}$ and the actual Base_{est} calculated from individual samples should be 0. This is tested in table III. We see that most ablations yield zero or near-zero mean additivity errors, indicating that the Shapley decomposition remains consistent across tasks. However, ablating *FUNC_12* or *FUNC_13* increases errors, especially for the upper bounds. Ablating *FUNC_12* or *FUNC_13* forces the model to redistribute their relatively large contributions among other tasks, which disrupts the otherwise tight match between the summed Shapley values and the model's output. As a result, the additivity error rises when either of those tasks is removed, which implies their outsized role in the model's learned representation.

TABLE III: Mean Absolute Additivity Error table for Shapley values across ablations and baseline $\left|\overline{\text{Base}_{est}} - \text{Base}_{est}\right|$

	Core 0			Core 1			Dual Core		
Ablated Function	**Lower**	**Upper**	**Pred.**	**Lower**	**Upper**	**Pred.**	**Lower**	**Upper**	**Pred.**
None	0	0	0.17	0	0	0.17	0	0	0.14
1	0	0	0.12	0	0	0.16	0	0	0.13
2	0	0	0.16	0	0	0.16	0	0	0.13
3	0	0	0.12	0	0	0.15	0	0	0.13
4	0	0	0.19	0	0	0.19	0	0	0.14
5	0	0	0.15	0	0	0.19	0	0	0.14
6	0	0	0.13	0	0	0.18	0	0	0.14
7	0	0	0.19	0	0	0.19	0	0	0.14
8	0	0	0.13	0	0	0.18	0	0	0.14
9	0	0	0.16	0	0	0.18	0	0	0.14
10	0	0	0.19	0	0	0.22	0	0	0.17
11	0	0	0.16	0	0	0.2	0	0	0.16
12	0	0	0.62	0	0	1.1	0	0	0.86
13	0	0	0.24	0	0	0.33	0	0	0.27
14	0	0	0.13	0	0	0.16	0	0	0.14
15	0	0	0.18	0	0	0.17	0	0	0.14
16	0	0	0.12	0	0	0.18	0	0	0.13
17	0	0	0.18	0	0	0.19	0	0	0.16
18	0	0	0.15	0	0	0.17	0	0	0.13
19	0	0	0.18	0	0	0.18	0	0	0.14
20	0	0	0.18	0	0	0.21	0	0	0.17
21	0	0	0.16	0	0	0.16	0	0	0.13
22	0	0	0.16	0	0	0.2	0	0	0.15
23	0	0	0.12	0	0	0.16	0	0	0.13
24	0	0	0.15	0	0	0.14	0	0	0.11
25	0	0	0.13	0	0	0.16	0	0	0.12
26	0	0	0.14	0	0	0.2	0	0	0.15
27	0	0	0.16	0	0	0.18	0	0	0.13

D. Sensitivity Analysis over Ablations

Upon reviewing the frameworks outputs over the baseline and the ablations, we may conclude that *FUNC_12*, *FUNC_13*, *FUNC_20* and *FUNC_10* are by far the most impactful for the CPU loads in the machine, which may be seen in fig. 10. We will focus primarily on these four tasks in the Shapley value analysis, more specifically, on these tasks' contributions when active. The results may be observed in figs. 11a to 11c.

Observing the percentage changes as compared to baseline gives us insights into the effect of ablating individual measurement features. Since the baseline contribution for *FUNC_20* is relatively small, when *FUNC_12* is ablated, even a modest absolute increase in *FUNC_20*'s contribution (0.101) translates into an outsized percentage change. Additionally, if *FUNC_12* and *FUNC_20* are correlated or interact in the model, ablating *FUNC_12* might force the model to compensate by relying much more on *FUNC_20*, leading to a dramatic increase in its attributed importance. This reallocation can cause the computed percentage change to spike, such as the 1394% observed for the upper bound contribution in fig. 11a, even though the absolute change is only 0.101. We see a similar pattern for all tasks except *FUNC_12*, such as the -1085% change in *FUNC_13*'s Upper bound contribution to Core 0 when *FUNC_12* is ablated, as well as *FUNC_20*'s upper

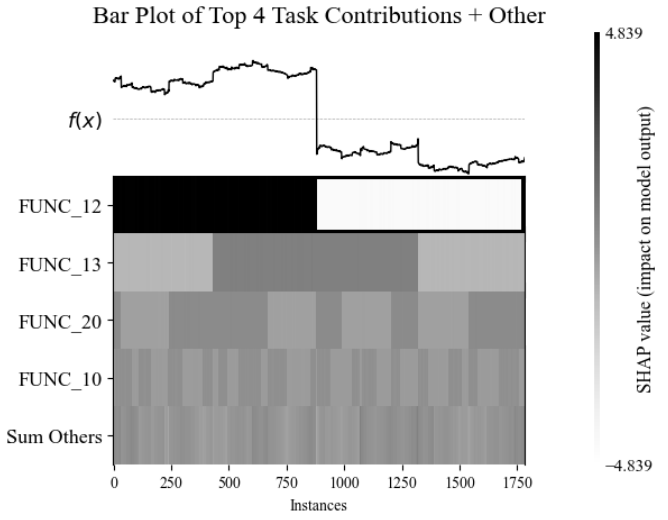


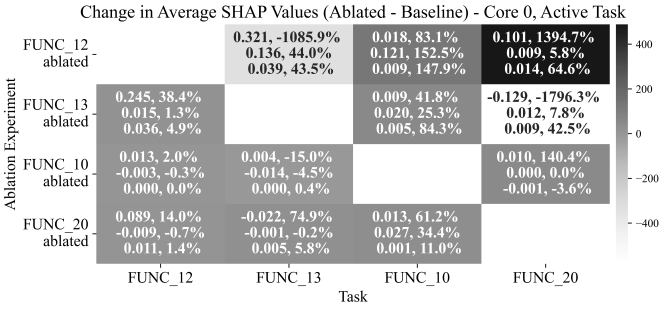
Fig. 10: Shapley value summary for predictions on Core 0, un-ablated baseline test dataset, 1784 testing samples.

bound contribution for the Dual Core and Core 1 cases at almost 20 times the baseline contribution.

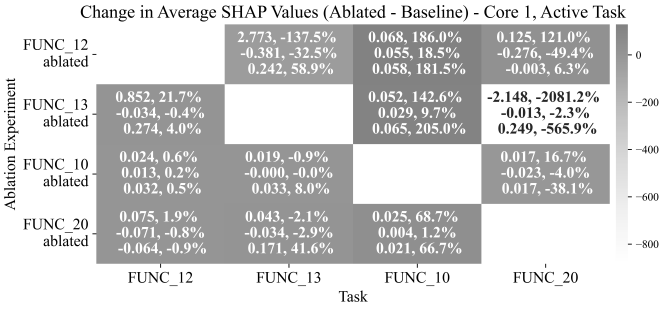
V. Discussion

From the perspective of modeling the CPU usage in an embedded device with respect to the tasks loaded onto the device, we can state that our framework and application present a usable alternative to manual testing and "rule-of-thumb" estimates of CPU load. The synergic composition of point forecasting along with conformal prediction gives us uncertainty-bounded load estimates usable in the real world as guidelines for device deployment. Furthermore, provided that we continuously utilize the device data storing and model refinement, it is an approach with potential to provide more insights with more data and more device parameters forwarded to it. Presently the application supports training a new model based on refreshed data, but we intend to implement support for continuous model refinement. Our industrial partners have found the application highly useful, providing invaluable feedback for further development. They intend to utilize a specialized version running on their premises. Based on table II, we can see that we approach 90% coverage on Core 0, Core 1 and Dual Core measurements. The mean interval length for Core 0 is within the measurement system's margin of error, whereas Core 1 and Dual Core measurements, though informative, leave some place for improvement. For example, while we do rely on the essential algorithm of split conformal regression on I.I.D. data for our estimates, there exist more informed adaptive approaches (such as [16], [17]) that could tighten these intervals further while retaining coverage. For the point forecast predictor, we can state that it is highly accurate based on the minimal MSE scores recorded on the test dataset.

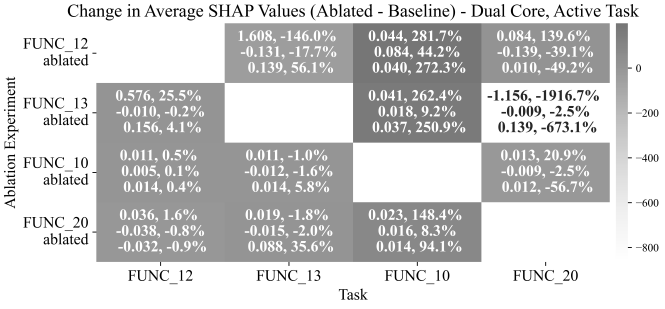
Though the contribution shifts in ablated datasets in absolute terms are within the measurement system's margin of error, they cannot be ignored in the sense of formulating a robust and



(a) Core 0 measurements.



(b) Core 1 measurements.



(c) Dual Core measurements.

Fig. 11: Change in Shapley values of most significant tasks in ablated vs. baseline cases for the task's contribution when active. Values on the bottom represent the lower bound contribution, values in the middle are the prediction contribution, and values on top are the upper bound contribution.

pervasive CPU load measurement framework. They underline the significance of getting good data from the devices that the application users are trying to model, in terms of capturing as many tasks running on the device as possible. In line with the general notion of machine learning-based approaches becoming more useful the richer the underlying datasets become, our approach would greatly benefit from deeper analysis of the device.

VI. Related Work

One of the big motivations for this work was the black-box latency estimation approach in [18], where the authors implement a conformal prediction estimator for embedded hardware platforms running machine-learning inference. Though we did not utilize a similar embedded device or measurement framework, and targeted overall load instead of latency, the

approach was inspirational in the application of conformal prediction to the task at hand. Though there have been a plethora of papers published in the field of conformal prediction in recent years [19], the application of Shapley values to conformal predictors is a relatively unexplored one. In [20] the authors apply a Shapley value calculation algorithm augmented by split conformal prediction to achieve explainable feature attributions. Though the authors here focus primarily on quantifying uncertainty of the feature attributions via a more efficient Shapley value estimate, it is in fact a target for future improvement of our own work. In [21] the authors focus on the interpretation of Shapley values for tree models in particular. This contains insights we intend to apply to our own future model implementations, since our framework leaves space for modular model improvements. There is significant room for improvement in the very approach to explainable feature attributions, as is evidenced by work on the very topic of replacing Shapley values as explanation mechanisms with *regional explanations* [22] so as to capture global, non-specific-prediction-specific significances of the features.

VII. Conclusion and Future Work

In this work, we have developed an integrated framework for CPU load forecasting that combines deep neural networks with conformal prediction and Shapley value-based interpretability. Our approach yields robust uncertainty quantification while providing clear, actionable insights into the contributions of individual tasks. Experimental results demonstrate that our model reliably predicts CPU load across a range of configurations, and that the Shapley values effectively capture the importance of each task, even under ablation conditions, underlining the critical role of certain functions in maintaining system performance. Our work awaits more extensive validation in different domains. Building on these results, as future work, we intend to pursue several avenues for enhancement. First, we plan to expand our model library by incorporating alternative methods such as Gradient Boosting Machine, Random Forest, and Quantile Random Forest based conformal prediction models, which may offer improved performance or robustness in different settings. Additionally, we intend to augment our dataset with supplementary system-level metrics, including background processes, interrupts, memory management, I/O operations, context switching, and peripheral interactions to capture a more holistic view of the factors influencing CPU load. From a model development perspective, we will explore incremental learning strategies (e.g., elastic weight consolidation) to enable our models to incorporate new data without requiring complete retraining, thereby preserving learned knowledge over time. We also envision enhancing the application interface to support intuitive, drag-and-drop task management and to facilitate real-time model validation and automatic refinement via direct interfacing with the embedded device. Finally, recognizing that our current conformal prediction implementation guarantees only marginal coverage, we will investigate training-conditional coverage methods [23], [24] to provide more specific, per-observation uncertainty guarantees. These enhancements are expected to further strengthen the safety and robustness of our forecasting framework in real-world industrial deployments.

References

[1] S. Nejati, S. Di Alesio, M. Sabetzadeh, and L. Briand, "Modeling and analysis of cpu usage in safety-critical embedded systems to support stress testing," in *Model Driven Engineering Languages and Systems: 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30–October 5, 2012. Proceedings 15*, pp. 759–775, Springer, 2012.

[2] J. Regehr and U. Duongsaa, "Preventing interrupt overload," *ACM SIGPLAN Notices*, vol. 40, no. 7, pp. 50–58, 2005.

[3] E. Ciliendo, T. Kunimasa, and B. Braswell, *Linux performance and tuning guidelines*. IBM, International Technical Support Organization, 2007.

[4] P. Yosifovich, M. E. Russinovich, A. Ionescu, and D. A. Solomon, *Windows Internals: System architecture, processes, threads, memory management, and more, Part 1*. Microsoft Press, 2017.

[5] G. Shafer and V. Vovk, "A tutorial on conformal prediction.," *Journal of Machine Learning Research*, vol. 9, no. 3, 2008.

[6] E. Winter, "Chapter 53 the shapley value," in *Handbook of Game Theory with Economic Applications*, vol. 3, pp. 2025–2054, Elsevier, 2002.

[7] A. Khosravi, S. Nahavandi, and D. Creighton, "Construction of optimal prediction intervals for load forecasting problems," *IEEE Transactions on Power Systems*, vol. 25, no. 3, pp. 1496–1503, 2010.

[8] J. De la Torre, L. R. Rodriguez, F. E. Monteagudo, L. R. Arredondo, and J. B. Enriquez, "Electricity price forecast in wholesale markets using conformal prediction: Case study in mexico," *Energy Science & Engineering*, vol. 12, no. 3, pp. 524–540, 2024.

[9] C. O'Connor, M. Bahloul, R. Rossi, S. Prestwich, and A. Visentin, "Conformal prediction for electricity price forecasting in the day-ahead and real-time balancing market," 2025.

[10] A. N. Angelopoulos and S. Bates, "A gentle introduction to conformal prediction and distribution-free uncertainty quantification," 2022.

[11] N. Dewolf, B. D. Baets, and W. Waegeman, "Valid prediction intervals for regression problems," *Artif. Intell. Rev.*, vol. 56, p. 577–613, Apr. 2022.

[12] E. Society, *Econometrica: journal of the Econometric Society*, vol. 23. Econometric Society, the University of Chicago, 1955.

[13] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[14] H. Chen, I. C. Covert, S. M. Lundberg, and S.-I. Lee, "Algorithms to estimate shapley value feature attributions," *Nature Machine Intelligence*, vol. 5, no. 6, pp. 590–601, 2023.

[15] X. Deng and C. H. Papadimitriou, "On the complexity of cooperative solution concepts," *Mathematics of operations research*, vol. 19, no. 2, pp. 257–266, 1994.

[16] N. Deutschmann, M. Rigotti, and M. R. Martinez, "Adaptive conformal regression with jackknife+ rescaled scores," *arXiv preprint arXiv:2305.19901*, 2023.

[17] S. I. Amoukou and N. J. B. Brunel, "Adaptive conformal prediction by reweighting nonconformity score," 2023.

[18] M. Wess, D. Schnöll, D. Dallinger, M. Bittner, and A. Jantsch, "Conformal prediction based confidence for latency estimation of dnn accelerators: A black-box approach," *IEEE Access*, 2024.

[19] V. Manokhin, "Awesome conformal prediction," Apr. 2022.

[20] D. Watson, J. O'Hara, N. Tax, R. Mudd, and I. Guy, "Explaining predictive uncertainty with information theoretic shapley values," *Advances in Neural Information Processing Systems*, vol. 36, pp. 7330–7350, 2023.

[21] F. Xu, Z.-J. Zhou, J. Ni, and W. Gao, "Interpretation with baseline shapley value for feature groups on tree models," *Frontiers of Computer Science*, vol. 19, no. 5, p. 195316, 2025.

[22] S. I. Amoukou, *Trustworthy machine learning: explainability and distribution-free uncertainty quantification*. PhD thesis, Université Paris-Saclay, 2023.

[23] V. Vovk, "Conditional validity of inductive conformal predictors," in *Asian conference on machine learning*, pp. 475–490, PMLR, 2012.

[24] M. Bian and R. F. Barber, "Training-conditional coverage for distribution-free predictive inference," *Electronic Journal of Statistics*, vol. 17, no. 2, pp. 2044–2066, 2023.