WORTEX: Worst-Case Execution Time and Energy Estimation in Low-Power Microprocessors Using Explainable ML

Hugo Reymond
Univ. Rennes, INRIA, CNRS, IRISA, Rennes, France

Abderaouf Nassim Amalou
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Isabelle Puaut
Univ. Rennes, INRIA, CNRS, IRISA, Rennes, France

Abstract

Real-time and energy-constrained systems heavily rely on estimates of the worst-case execution time (WCET) and worst-case energy consumption (WCEC) of code snippets to ensure trustworthy operation. Designing architecture-specific analytical models for time and energy is often challenging and time-consuming. In situations where analytical models are unavailable or incomplete, machine learning (ML) techniques emerge as a promising solution to build WCEC/WCET models. This paper introduces WORTEX, a toolkit for WCEC/WCET estimation of basic blocks based on ML techniques. To ensure the real-world applicability of its models, WORTEX extracts large datasets of basic blocks from real programs and precisely measures their energy consumption/execution time on the physical target platform. The dataset is used to train various WCEC/WCET models using different ML techniques. Experimental results on simple and time-predictable hardware show that even the most basic ML techniques provide accurate results, that never underestimate actual values. We also discuss the use of explainability techniques to gain trustworthiness for the models.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases Worst-Case Execution Time (WCET), Worst-Case Energy Consumption (WCEC), Machine Learning, Explainable ML models

Digital Object Identifier 10.4230/OASIcs.WCET.2024.1

Supplementary Material *Dataset*: `https://zenodo.org/doi/10.5281/zenodo.11066622` [22]

Funding *Hugo Reymond*: This work has received a French government support granted to the Labex CominLabs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference ANR-10-LABX-07-01.
Abderaouf Nassim Amalou: This work was supported by the Agence Nationale de la Recherche under grant number ANR-22-CE92-0066.

Acknowledgements The authors would like to thank Hector Chabot for his essential work on the creation of this paper dataset.

1 Introduction

Worst-case execution time (WCET) and worst-case energy consumption (WCEC) estimation techniques play a crucial role in the validation of real-time systems. They provide upper bounds of WCEC/WCET and allow to guarantee correct operation of real-time embedded devices. Estimations of WCEC/WCET are particularly useful in battery-less devices, that harvest energy from their environment, store it in a capacitor, and execute short bursts of computation using the stored energy. For such devices, WCEC/WCET estimations can be used by static checkpointing strategies [6, 21, 31], that select checkpoint locations in the code to ensure that there is enough energy to reach the next checkpoint.

The state-of-the-art methods for statically estimating the WCET/WCEC of a program commonly use the Implicit Path Enumeration Technique (IPET) [18, 13]. IPET estimates WCET/WCEC by finding the longest path in the program's Control Flow Graph (CFG), using Integer Linear Programming (ILP) to avoid enumerating all paths in the CFG. The IPET approach requires time and energy consumption data for the basic blocks [1] of the program, which are usually derived from detailed architectural documentation of the processor. Such information is not always available due to intellectual property restrictions. In cases where they are accessible, it would require a detailed analysis of the documentation and extensive validation of the obtained model, which is time-consuming. Moreover, the documentation often does not include energy consumption information. As an alternative to an analytical model, it is possible to directly measure the basic block time/energy consumption and provide it as an input to IPET. However, the applicability of this technique is limited in practice, because the measurement campaign has to ensure that every basic block is executed sufficiently often to identify its WCET/WCEC. Moreover, this process needs to be performed for every modification of the program.

To overcome these shortcomings, a new class of approaches has been introduced, that leverages machine learning (ML) to estimate the WCEC/WCET of individual basic blocks. These approaches involve training an ML model using a large dataset of representative and varied basic blocks. The WCEC/WCET of individual basic blocks can then be used as inputs to IPET to calculate WCET/WCEC at the program level.

Previous works in this area feature some limitations. Some use small datasets or operate at the source code or intermediate code level [5, 11, 12, 30], which affects the model's accuracy. All previous ML models, including the most elaborated ones [3, 2] act as black boxes and lack interpretability of the results. To address these challenges, we propose WORTEX, for WORst-case execution Time and Energy consumption estimation using eXplainable machine learning, a toolkit for WCEC/WCET estimation, that aims at overcoming the limitations of existing techniques. WORTEX uses large and diverse datasets from AnghaBench [7] to train the models. Furthermore, we leverage a local model-agnostic explainable artificial intelligence technique [20] and discuss how such a technique can be used to understand the predictions, find bias in the training dataset, and therefore gain confidence in the model.

As a case study, we selected the MSP430FR5969 low-energy processor [27], which is extensively used in the battery-less community. To evaluate our solution, we conducted experiments at both the basic block and program levels. To do so, WORTEX generated models were integrated into the Heptane static WCET estimation tool [10].

Overall, our contributions are the following:

- We propose WORTEX, a toolkit for WCEC/WCET estimation of basic blocks using ML techniques. The simple ML models integrated in WORTEX (linear regression, gradient boosting, multi-layer perceptron neural network) were selected because of their very fast prediction, and good accuracy for the simple MSP430 platform. The loss function used when training the models is tailored to the prediction of worst-case values.
- We open-source a dataset of 30 000 basic blocks for the training of ML models [22]. All of them have been measured and have energy consumption and execution time information.
- We discuss the use of the LIME explainable AI technique [23] to validate the ML models and gain confidence in the trained models.

[1] A basic block is a CFG node, and is defined as a sequence of consecutive instructions with a single entry point and no branching inside

The remainder of the paper is organized as follows. Section 2 first compares WORTEX with related works. Section 3 then gives an overview of WORTEX in an architecture-independent way. The components of WORTEX that are specific to targeted architecture (MSP430) are described in Section 4. Section 5, presents experimental results. Finally, Section 6 discusses the use of explainable AI to validate the ML models. Section 7 concludes and outlines future works.

2 Related work

This section reports previous research leveraging machine techniques for WCET and WCEC estimation.

ML-based WCET estimation. Several methods to estimate the WCET using Machine Learning (ML) have been proposed [5, 1, 16, 17, 5, 3, 11, 2]. The studies reported in [5, 1, 11, 12] utilize worst-case event counts (number of multiplications, number of memory accesses...) extracted from the intermediate representation of the code to train different ML models, which are then used to calculate the WCET of a program in early development phases. Similarly, Kumar's research, described in [16, 17] estimate WCET by examining features taken from the program's source code. All these methods ignore important information about how the generated code and hide the effects of compilation by operating at the source code or intermediate code level, which might affect timing predictions. In contrast, the research presented in [3, 2], like WORTEX, extract features from the binary code and use ML techniques to predict the WCET of individual basic blocks. All the approaches cited above treat the ML techniques as a black boxes. WORTEX add to them the use of explanations [20] to gain more confidence in the predictions.

ML-based WCEC estimation. While numerous studies have explored the use of ML for estimating average-case energy/power consumption (as reviewed in [8]), the application of ML to estimate WCEC was only introduced in [12] and [30]. The work by Huybrechts et al. [12] specifically focuses on intermediate code representation. In contrast, WORTEX enhances the accuracy of estimation by operating on binary code. Moreover, Huybrechts et al. ML-based WCEC estimation technique typically relies on a limited number of benchmarks for training, whereas WORTEX leverages a large collection of basic blocks, ensuring a superior quality of training. In the study [30], simulation (cycle accurate model) or FPGA synthesis (Register Transfer Level RTL) is employed to capture the performance counters (such as cache misses and branch predictor misses) and the corresponding energy consumption on their target. By creating a dataset through this process, the authors employ linear regression as an energy model that can forecast energy consumption when provided with performance counters for code snippets. Contrary to this kind of approach, WORTEX does not need the RTL nor the cycle-accurate model to create an energy model that makes it usable for any target.

3 Overview of WORTEX

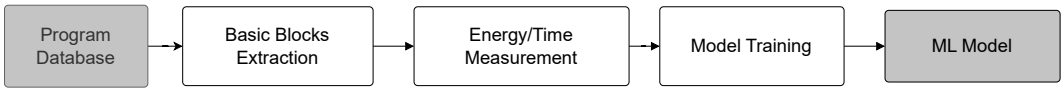


Figure 1 WORTEX's prediction workflow.

WORTEX comprises three components, depicted Figure 1. The first component *Basic Blocks Extraction* is responsible for extracting a large and unique dataset of basic blocks from different programs. The second component measures the energy consumption and execution times of all basic blocks from this dataset. The third component *Model Training* trains an ML model and deploys explainability [20] to understand the impact of each instruction on basic block WCEC/WCET. The obtained ML model can predict the WCEC/WCET of previously unseen basic blocks. Our tool offers flexibility in the choice of machine learning models, that can be used in different tools with minimal integration efforts. We have integrated the generated models into the Heptane static WCEC/WCET estimation tool [10]. This section describes the different components of WORTEX.

3.1 Dataset generation

To accurately predict WCEC/WCET using ML techniques, it is crucial to train the model with an appropriate dataset. This section outlines the requirements for generating such a dataset. Their implementation for a specific architecture (MSP430FR5969, from Texas Instruments) will be described in Section 4.2. The requirements that we consider important for accurate training are the following:

- **RD1: representative and diverse dataset**. This requirement aims at training the ML models on a set of basic blocks that provide good coverage of the instruction set of the target architecture but also of the code constructs of the programming language used. This requirement is met in WORTEX by using the AnghaBench benchmark suite [7] to train the models. AnghaBench contains 1 million compilable programs, mined from the largest public C repositories on GitHub [2].
- **RD2: balanced dataset**. Compilers tend to generate basic blocks (e.g. for function entry, exit, array indexing) that are identical or almost identical to other basic blocks (e.g. same instructions but different registers). Such duplicated basic blocks do not improve the quality of training and should be removed from the dataset that is used during training.
- **RD3: ability to execute basic blocks in isolation**. Basic blocks are executed in isolation to measure their energy consumption and execution time. Executing basic blocks out of their original context may lead to exceptions (e.g. divisions by zero, invalid memory accesses). A pre-processing phase has to be applied to each basic block to ensure that it executes without error. For example, the contents of the registers used for indirect accesses to memory have to be controlled to avoid indirect memory accesses.

3.2 Energy and time measurement

WORTEX is designed to produce models for *worst-case* timing and energy consumption. Hence, the measurement process must meet the following two requirements:

- **RM1: measurement in worst-case scenario:** The execution context of basic blocks, in particular in architectures with caches and pipelines, largely influences the execution time and energy consumption of basic blocks. For instance, cache misses cause longer execution times and higher energy consumption than cache hits. Enforcing RM1 highly depends on the targeted architecture, it is addressed in Section 4.3.
- **RM2: precise measurements:** The precision of measurements obviously has an impact on the quality of the energy/timing estimations. Addressing RM2 is particularly challenging because basic blocks usually comprise a small number of instructions, which makes measurements tricky on individual basic blocks. This is detailed in Section 4.3, by performing measurements on a series of basic blocks instead of individual basic blocks.

[2] AnghaBench dataset: `https://github.com/brenocfg/AnghaBench`

3.3 Model creation

Using the collected dataset, WORTEX includes three ML models capable of conservatively estimating basic block's WCEC/WCET:

- **Quantile Linear Regression (QLR) [15].** It is a statistical technique that shows how the input variables (basic block) affect linearly the output variable (WCEC/WCET). They offer a more detailed view compared to classic linear regression by looking at the full range of possibilities for the dependent variable (WCEC/WCET), not just its average.
- **Gradient Boosting (GB) [26].** Gradient boosting constructs a model based on a set of decision trees. In contrast to linear regression GB can capture non-linear relationships.
- **Multi-Layer Perceptron (MLP) [25].** An MLP is a type of artificial neural network featuring layers of interconnected nodes or neurons. It is particularly skilled at modeling intricate, non-linear relationships in the data.

Deploying the ML models is performed in two classical phases: the *training phase* and the *estimation phase*.

Training. During the training phase, WORTEX trains ML models on a large dataset to derive a WCET/WCEC model at the basic block granularity. Training is performed once for each specific micro-architecture (in our use case, the MSP430FR5969). The algorithms learn from *features* extracted from the basic blocks. Due to the simplicity of the targeted architecture, WORTEX currently uses *static* features. The features are mainly the proportions of different types of machine instructions across their various operand addressing modes divided by the total number of instructions in the basic block, for example [29]. The algorithms analyze *features* extracted from basic blocks, utilizing all MSP430 machine instructions across their various operand addressing modes. By representing instruction types as proportions, the timing/energy model created by WORTEX becomes independent of the basic block's length.

To mitigate the risk of underestimation, we leverage a *quantile loss* function, that penalizes overestimations and underestimations asymmetrically. The quantile loss evaluates the disparity between predicted and actual quantiles and for a given quantile q where $0 < q < 1$ is defined as:

$$L_q(y, \hat{y}) = \sum_{i=1}^{n} (q - \mathbf{1}\{y_i - \hat{y}_i < 0\}) \cdot (y_i - \hat{y}_i)$$

where:

- y_i and $\hat{y}_i$ are respectively the true value and the predicted for the i-th observation in the dataset
- $\mathbf{1}\{y_i - \hat{y}_i < 0\}$ is an indicator function that equals 1 if the condition $y_i - \hat{y}_i < 0$ is true and 0 otherwise.

Prediction. WORTEX estimates the WCEC/WCET at the basic block level. To estimate WCEC/WCET at the program level, the program is split into basic blocks, the features of basic blocks are extracted and the ML model is then applied. The estimated energy/timing can then be given as input to a static WCEC/WCET analysis tool.

4 Specifics of WORTEX on MSP430

This Section describes the components of WORTEX that are specific to the low-power microcontroller MSP430FR5969.

4.1 The MSP430FR5969 microcontroller

The MSP430FR5969 is an energy-efficient and low-cost microcontroller based on a 16-bit CPU, popular for applications powered by ambient energy. The micro-architecture of the MSP430X CPU series is simple. The architecture has no data cache, no branch predictor, and features a very simple pipeline with only 3 stages. The MSP430FR5969 features two main memories: Volatile Memory (VM), more precisely 2KB of SRAM, and non-volatile memory (NVM), more precisely 64kB of non-volatile ferromagnetic RAM (FRAM)[28]. The FRAM controller uses a small 2-way associative cache that has a 64-bit line size (4 16-bit instructions) to store pre-fetched instructions. In our use case, the code is stored in NVM and the VM contains the program data and stack.

4.2 Dataset generation

WORTEX first cross-compiles the numerous C sources from the AnghaBench benchmark suite into assembly language. At this stage, the generated basic blocks do not yet meet requirements RD2 and RD3 described in Section 3.1: many redundant basic blocks exist in the data set (RD2), and some of them may cause errors when executed in isolation (RD3).

Regarding the elimination of redundant basic block (RD2), preliminary experiments on the MSP430FR5969 have shown that the specific general-purpose register used in an instruction has negligible impact on its WCEC/WCET. Therefore, basic blocks are pre-processed to use a specific general-purpose register for all instructions. Duplicated basic blocks are then removed from the dataset.

Further pre-processing is required to ensure that basic blocks can be executed in isolation (requirement RD3) without addressing errors. This is achieved as follows:

- For *absolute addressing*, each accessed symbol/address is replaced with a predetermined location in NVM.
- For *indirect accesses*, where the address to be accessed is stored in a register, we face an additional challenge. In such cases, we need to have control over the value stored in the register itself. To address this, we have selected one of the registers to be a *controlled base register* used as a target for every indirect access. This register holds the address of a chosen memory location in the VM. We further replace any written instruction targeting this register with another one.

Simulation shipped with the MSP430 GNU Debugger (GDB) was used to check that no run-time error occurs and that all memory accesses fall in the VM.

4.3 Energy and timing measurement

Measurement in worst-case scenario (RM1). Enforcing the worst-case execution scenario on the MSP430FR5969 architecture is achieved through a control of its two main sources of variability, its instruction cache and its pipeline.

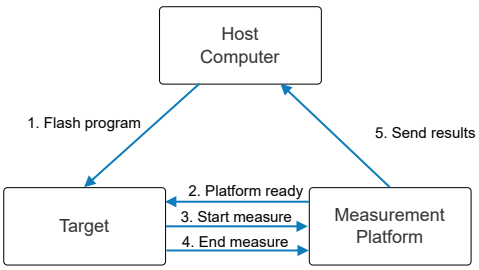
The worst-case scenario regarding the instruction cache is enforced by controlling the layout of basic blocks to maximize the number of cache misses. The end of the first instruction of each basic block is aligned on a cache-line boundary, as depicted in Figure 2a. Fetching the first instruction (line 9) will thus trigger a first cache miss. Then, the second instruction (line 10), belonging to another cache line, will also generate a cache miss, loading the subsequent instructions.

Regarding pipeline effects, adding JMP (unconditional jump) and NOP instructions before the basic block induces a pipeline flush (lines 5-7), as the NOP instruction, while loaded in the pipeline, will not be executed.

```
1    START_MEASURE
2   JMP .BB
3 . balign 8
4 ------- Cache line frontier -------
5    NOP
6    NOP
7    NOP
8 .BB:
9    PUSHM .W #2, R10          CACHE MISS
10 ------- Cache line frontier -------
11   SUB.W #10 , R10           CACHE MISS
12   MOV.W R4 , 6( R15)        CACHE HIT
13   MOV.W R13 , 4( R15 )      CACHE HIT
14   JMP .END                  CACHE HIT
15 ------- Cache line frontier -------
16 .END:
17   END_MEASURE
```

Duplicate n times

(a) Ensuring the worst case scenario for the MSP430FR5969 instruction cache and pipeline.



(b) Basic block measurement process.

Figure 2 Measuring a basic block energy consumption and execution time.

Precision of measurements (RM2). To measure the energy consumption and execution time of the MSP430FR5969, we need to be able to handle the scale difference between the measurement tool and the basic block execution time. This is achieved by duplicating the basic block multiple times while making sure to keep the worst-case memory layout as explained previously. Furthermore, to synchronize the basic block execution with the measurement platform, we add instructions to signal the start and the end of the execution of the basic block using General-Purpose Input/Output ports (GPIOs), (START_MEASURE and END_MEASURE on Figure 2a).

The measurement process involves three actors, as depicted in Figure 2b: the target (MSP430 under analysis), a measurement platform, and the host computer. The host computer initiates the process by flashing the instrumented code on the target (1). Once flashed, the target actively waits for the measurement platform *Platform Ready* signal (2). As it receives it, it sends back a signal *Start Measure* and starts executing the code (3). As soon as the *Start Measure* signal is received, the measurement platform measures the energy consumption of the target, until it receives the signal *End Measure* (4), marking the end of basic block execution. The measurement platform then computes the basic block execution time (defined by the time between *Start Measure* and *End Measure*). Finally, the measurement platform sends the measurements to the computer (5), which, in turn, will flash a new program to the target (1).

5 Experimental evaluation

Section 5.1 first describes the experimental setup used. Then WORTEX is evaluated according to different metrics: quality of predictions at the BB level (Section 5.2) and quality of predictions at the program level (Section 5.3)

5.1 Experimental setup

The target platform for our experiments is the MSP430FR5969 microcontroller. All code is stored in NVM and the VM is used to store program data and stack. The basic blocks, extracted from the AnghaBench benchmark suite [7] are compiled using the GCC MSP430

compiler version 9.3.1 with no optimization (-O0). Obtained basic blocks are pre-processed and filtered as explained in Section 4. Basic blocks are duplicated 50 times. The measurement platform, similar to the one used in [4], includes the following components:

- A stable power supply (N6705A [14]) providing 3.3V to power the MSP430.
- A shunt resistor with a resistance of 4.7Ω to measure the current consumed by the MSP430. This resistor has been chosen to ensure a low voltage drop (around $4.7mV$) and is connected to an operational amplifier (INA214CIDCKR) to amplify the voltage drop amplitude, allowing precise measurement. The amplified voltage is then measured with an ADS8661 analog-to-digital converter.
- A relay array to isolate the MSP430 from the computer when measuring energy. It prevents any energy interference coming from the JTAG connection.
- A bare metal Raspberry PI (RPI) 3B+ is used for synchronization with the MSP430, for controlling the isolation of the MSP430, and for time/current consumption measurement.

Section A.1 in the Appendix explains how we deduce the energy consumption from the measured current.

For each (duplicated) basic block, we perform 100 measurements and retain the highest value. Subsequently, we divide this value by the duplication factor to get the basic block energy consumption and execution time. A dataset of 30 000 unique basic blocks with their energy consumption and execution time was used, where 80% of the basic blocks were used for training and cross-validation and 20% were used for testing.

5.2 Analysis of WCET and WCEC predictions at BB level

Table 1 qualifies the results of the predictions at the BB level for the three models (QLR, GB, and MLP), by employing quantile values of 0.99 and 0.99999[3]. The quality of predictions is evaluated using two metrics.

- Mean Average Percentage Error (MAPE): MAPE $= \frac{1}{n}\sum_{i=1}^{n}\left|\frac{A_i-P_i}{A_i}\right| \times 100\%$, where A_i is the actual value, P_i is the predicted value and n is the number of basic blocks. The lower the MAPE, the more accurate the model, disregarding under/overapproximations.
- Underestimations: percentage of basic blocks whose WCEC/WCET is underestimated: Underestimation (%) $= \frac{N_{\text{underestimated}}}{N_{\text{total}}} \times 100$

Table 1 Comparison of MAPE and Underestimation Percentages for WCET and WCEC Predictions at Quantiles 0.99 and 0.99999 Across Models.

Model	MAPE				Underestimation (%)			
	Quantile 0.99		Quantile 0.99999		Quantile 0.99		Quantile 0.99999	
	WCET	WCEC	WCET	WCEC	WCET	WCEC	WCET	WCEC
QLR	56.7 %	58.0 %	101.3 %	**97.3 %**	0.69 %	**0.52 %**	**0 %**	**0 %**
GB	42.1 %	39.8 %	93.0 %	**97.3 %**	**0.56 %**	0.65 %	**0 %**	**0 %**
MLP	**8.2 %**	**16.2 %**	**41.1%**	99.1 %	0.65 %	1.07 %	**0 %**	**0 %**

The results show that the MLP model outperforms QLR and GB in terms of accuracy, with the lowest MAPE for both time and energy and both quantile values, except at WCEC for 0.99999 quantile where the QLR and GB model are more accurate. As we shift from

[3] A quantile of 0.99 (resp. 0.99999) means that we aim at predictions that are larger or equal to the actual WCEC/WCET in 99% (resp. 99.999%) of the cases.

quantile 0.99 to quantile 0.99999, accuracy degrades for all models, as shown by the increase in the MAPE metric. However, this degradation comes with a sharp decrease in the ratio of underestimations (no underestimation is actually observed), which is essential when estimating WCEC/WCET. More complete results about the quality of predictions for the different techniques are given in Section A.2 in the appendix.

5.3 Analysis of WCET and WCEC predictions at program level

This experiment aims at evaluating the error made by WORTEX when it is used by the Heptane static WCET analysis tool to estimate WCEC/WCET of entire programs, from the Mälardalen benchmark suite [9]. In this experiment, the WCET/WCEC predicted by WORTEX (for 0.99999 quantile) is fed to Heptane that uses the standard Implicit Path Enumeration Technique (IPET, [18]). We selected a small subset of benchmarks for which we can generate input data that systematically execute the longest execution path found by Heptane, ensuring that the pessimism comes from WORTEX and not from Heptane.

Table 2 Observed vs predicted execution time/energy consumption (time in µs, energy in nJ).

Program	Maximum observed		QLR based estimations		GB based estimations		MLP based estimations	
	Time	Energy	Time	Energy	Time	Energy	Time	Energy
bs	293	358	507	516	445	459	334	376
fibcall	1 035	1 059	1 131	1 132	1 181	1 214	1 197	1 380
lcdnum	919	1 019	1 506	1 516	1 372	1 484	1 077	1 260
nsichneu	24 672	25 179	36 252	37 256	33 367	35 141	27 248	31 967
Overestimation		mean	48.3%	36.9%	37.6%	32.0%	**14.3%**	21.5%
		min	9.2%	6.9%	14.1%	14.6%	10.4%	**5.0%**

Table 2 shows the maximum observed execution time and energy consumption for each program (executed its worst-case input) and the prediction from the different ML techniques. Regardless of the benchmarks studied, we can make several observations. First, thanks to the loss used during the training phase, the prediction always overestimates the observed execution time and energy consumption. Second, MLP-based techniques significantly outperform QLR and GB. Third, the overestimation for the best-performing technique MLP is reasonable (14% on average for time, 21% for energy).

5.4 Inference time

One of the interests of using ML techniques for WCEC/WCET estimation if that the predictions are fast enough to be used in WCET/WCEC analysis tools. Table 3, reports the average inference time of one basic block for each technique.

Table 3 Average inference time per basic block, on an Intel i7-11850H CPU.

Method	QLR	GB	MLP
Average time (µs)	1.95	48.95	35.88

Regardless of the chosen technique, the inference time is within the micro-second range. As anticipated, the QLR model, due to its simplicity, exhibits the shortest inference time, surpassing the other techniques by approximately a factor of 20. In contrast, GB and MLP show higher inference time but offer better precision.

6 Discussion on explainable AI

Most ML techniques, while making accurate predictions, are black-box techniques, meaning their users cannot easily understand *why* they make their decisions. *Explainability* [20] techniques help understand why a model makes certain predictions, by revealing the factors that influence the estimations. Explainability helps detect misbehaviors of ML models, caused for example by biased datasets or sub-optimal selection of features. Once the misbehaviors are detected and corrected, confidence can be gained in the models.

Several techniques for explainability exist for black-box models. Among them, we focus in this paper on LIME [23] (Local Interpretable Model-agnostic Explanations). LIME creates interpretable models (e.g., linear regression) that approximate the predictions of the black-box model around a specific data point of interest. It achieves this by generating a synthetic dataset (i.e., neighborhood points) and training a simple, interpretable model on it (a linear regression in our case). By analyzing the behavior of this *new simple local model*, we can gain insights into the factors influencing the model's decision at that particular data point.

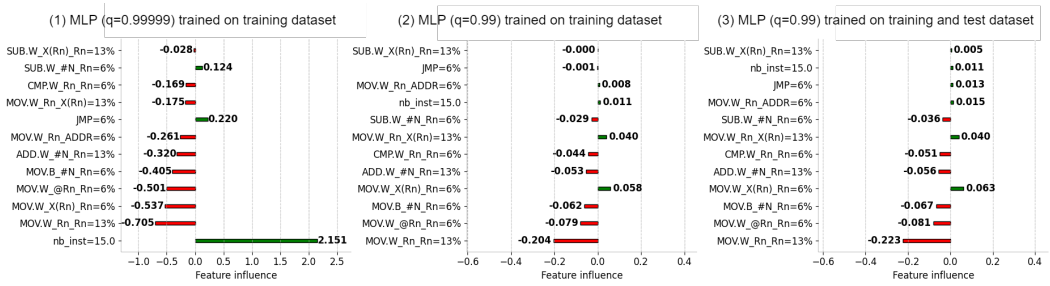


Figure 3 LIME Explanations for MLP on a BB under different training settings (quantile value, training dataset).

Figure 3 illustrates the explainability of the MLP model generated by LIME for a single BB sample. This MLP is trained under different configurations of quantile loss and dataset compositions to investigate the model's decisions. The figure is divided into three generated LIME explanations, where the x-axis represents the direction of influence that each feature has on the model's prediction (which we retrieve from a linear regression local model): positive values are associated with an increase in the predicted outcome, while negative values suggest a decrease. The y-axis lists the non-null features of the BB: instructions (type and addressing mode), with a percentage indicating their occurrence rate within the BB.

In Part (1) of Figure 3, the MLP is trained with a quantile loss of 0.99999. The results show a significant negative influence of certain features like `MOV.W_X(Rn)_Rn` and `MOV.W_Rn_X(Rn)`, which is counter-intuitive considering the known indirect access mode instruction latencies for the MSP430 [29]. This suggests that the extreme quantile value, aiming to capture the worst-case predictions, may be skewing the model's focus towards only the `nb_inst` (number of instructions) feature to make a conservative prediction independently of the actually executed instructions.

Part (2) of Figure 3 presents the LIME explanations when the MLP is trained with a quantile loss of 0.99. Here, the influence of features appears to align more closely with the MSP430 instruction latencies [29], except for `SUB.W_X(Rn)_Rn`. Although it should have the same influence factor as the `MOV.W_X(Rn)_Rn` instruction, it does not. Upon investigation, we found that the subtraction instruction appears less frequently than the `MOV` instruction within our training dataset.

Finally, to investigate whether the hypothesis that the rarity of the `SUB.W_X(Rn)_Rn` instruction influenced the explanation, we explore in part (3) of Figure 3 LIME explanations with post-augmentation of the training dataset (used in the previous two experiments) with the test set, under the same quantile loss of 0.99. This data enrichment leads to a further refined interpretation of feature influences, correcting earlier misalignments between LIME explanation and the MSP430 documentation [29], for instance `SUB.W_X(Rn)_Rn` has now a positive influence.

These findings highlight the importance of selecting an appropriate quantile loss value ensuring the comprehensiveness and the empirical safety of the ML models. They also show that XAI [20] techniques can help us diagnose our model and dataset.

7 Conclusion

This paper has introduced WORTEX, a toolkit for WCEC/WCET estimation of basic blocks using ML techniques. WORTEX was shown to produce safe yet precise WCEC/WCET estimates for low-power processors. As future work, we believe that integrating WORTEX in a compiler toolchain would allow us to explore program optimizations with both time and energy in mind would be interesting. In addition, further exploring other explainability techniques like SHAP [19] or Anchors [24] will help the design of WCEC/WCET estimation techniques on more complex machine learning techniques, for example, Transformers [2].

References

1 Peter Altenbernd, Jan Gustafsson, Björn Lisper, and Friedhelm Stappert. Early execution time-estimation through automatically generated timing models. *Real-Time Systems*, 2016. doi:10.1007/s11241-016-9250-7.

2 Abderaouf Nassim Amalou, Elisa Fromont, and Isabelle Puaut. CAWET: Context-Aware Worst-Case Execution Time Estimation Using Transformers. In *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, 2023. doi:10.4230/LIPIcs.ECRTS.2023.7.

3 Abderaouf Nassim Amalou, Isabelle Puaut, and Gilles Muller. WE-HML: hybrid WCET estimation using machine learning for architectures with caches. In *IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2021. doi:10.1109/RTCSA52859.2021.00011.

4 Gautier Berthou, Kevin Marquet, Tanguy Risset, and Guillaume Salagnac. Accurate power consumption evaluation for peripherals in ultra low-power embedded systems. In *Global Internet of Things Summit (GIoTS)*, 2020. doi:10.1109/GIOTS49054.2020.9119593.

5 Armelle Bonenfant, Denis Claraz, Marianne De Michiel, and Pascal Sotin. Early WCET prediction using machine learning. In *17th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2017. doi:10.4230/OASIcs.WCET.2017.5.

6 Jongouk Choi, Larry Kittinger, Qingrui Liu, and Changhee Jung. Compiler-directed high-performance intermittent computation with power failure immunity. In *28th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'22)*, 2022. doi:10.1109/RTAS54340.2022.00012.

7 Anderson Faustino Da Silva, Bruno Conde Kind, José Wesley de Souza Magalhães, Jerônimo Nunes Rocha, Breno Campos Ferreira Guimaraes, and Fernando Magno Quinão Pereira. Anghabench: A suite with one million compilable c benchmarks for code-size reduction. In *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021. doi:10.1109/CGO51591.2021.9370322.

8 Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 2019. doi:10.1016/j.jpdc.2019.07.007.

9 Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The Mälardalen WCET benchmarks: Past, present and future. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2010. doi:10.4230/OASIcs.WCET.2010.136.

10 Damien Hardy, Benjamin Rouxel, and Isabelle Puaut. The heptane static worst-case execution time estimation tool. In *17th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2017. doi:10.4230/OASIcs.WCET.2017.8.

11 Thomas Huybrechts, Siegfried Mercelis, and Peter Hellinckx. A new hybrid approach on WCET analysis for real-time systems using machine learning. In *18th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2018. doi:10.4230/OASIcs.WCET.2018.5.

12 Thomas Huybrechts, Philippe Reiter, Siegfried Mercelis, Jeroen Famaey, Steven Latré, and Peter Hellinckx. Automated testbench for hybrid machine learning-based worst-case energy consumption analysis on batteryless iot devices. *Energies*, 2021. doi:10.3390/en14133914.

13 R. Jayaseelan, T. Mitra, and X. Li. Estimating the worst-case energy consumption of embedded software. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2006. doi:10.1109/RTAS.2006.17.

14 KEYSIGHT. N6700 Modular Power System Family. URL: https://www.keysight.com/fr/en/assets/7018-05443/data-sheets/5992-1857.pdf.

15 Roger Koenker. *Quantile regression*, volume 38. Cambridge university press, 2005.

16 Vikash Kumar. Deep neural network approach to estimate early worst-case execution time. In *IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021. doi:10.1109/DASC52595.2021.9594326.

17 Vikash Kumar. Estimation of an early WCET using different machine learning approaches. In *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, 2022. doi:10.1007/978-3-031-19945-5_30.

18 Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In *SIGPLAN workshop on Languages, compilers, & tools for real-time systems*, 1995. doi:10.1145/216636.216666.

19 Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 2017.

20 Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. Interpretable machine learning–a brief history, state-of-the-art and challenges. In *ECML PKDD 2020 Workshops*, 2021. doi:10.1007/978-3-030-65965-3_28.

21 Hugo Reymond, Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou, Isabelle Puaut, and Erven Rohou. SCHEMATIC: compile-time checkpoint placement and memory allocation for intermittent systems. In *IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2024*, 2024. doi:10.1109/CGO57630.2024.10444789.

22 Hugo Reymond, Hector Chabot, Abderaouf Nassim Amalou, and Isabelle Puaut. MSP430FR5969 basic block worst case energy consumption (WCEC) and worst case execution time (WCET) dataset, April 2024. doi:10.5281/zenodo.11066623.

23 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *SIGKDD international conference on knowledge discovery and data mining*, 2016. doi:10.1145/2939672.2939778.

24 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, 2018. doi:10.1609/aaai.v32i1.11491.

25 David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 1986. doi:10.1038/323533a0.

26 Robert E. Schapire. The boosting approach to machine learning: An overview. *Nonlinear estimation and classification*, 2003. doi:10.1007/978-0-387-21579-2_9.

27 Texas Instruments. MSP430FR5969 user's guide. URL: https://www.ti.com/lit/ug/slau367p/slau367p.pdf.

28 MSP430FR5969 product information. URL: https://www.ti.com/product/MSP430FR5969.

29 Msp430 family instruction set summary. URL: https://www.ti.com/sc/docs/products/micro/msp430/userguid/as_5.pdf.

30 Simon Wegener, Kris K. Nikov, Jose Nunez-Yanez, and Kerstin Eder. EnergyAnalyzer: Using Static WCET Analysis Techniques to Estimate the Energy Consumption of Embedded Applications. In *21th International Workshop on Worst-Case Execution Time Analysis (WCET 2023)*, 2023. doi:10.4230/OASIcs.WCET.2023.9.

31 Bahram Yarahmadi and Erven Rohou. Compiler Optimizations for Safe Insertion of Checkpoints in Intermittently Powered Systems. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2020. doi:10.1007/978-3-030-60939-9_12.

A Appendix

A.1 Computing the energy consumption from the current consumption

The energy consumption of a basic block is computed from voltage and current consumption using equation 1.

$$E = \int U \times i(t)dt \tag{1}$$

As we work with discrete samples, equation 1 becomes equation 2:

$$E = \sum_{k=0}^{n} U \times i(k) \times t_{sample} = U \times t_{sample} \times \sum_{k=0}^{n} i(k) \tag{2}$$

t_{sample} defines the time needed to get a sample, is supposed to be the same for each sample, and is computed as shown in Equation 3.

$$t_{sample} = \frac{T_{measure}}{n_{sample}} \tag{3}$$

A.2 Detailed results on BB prediction

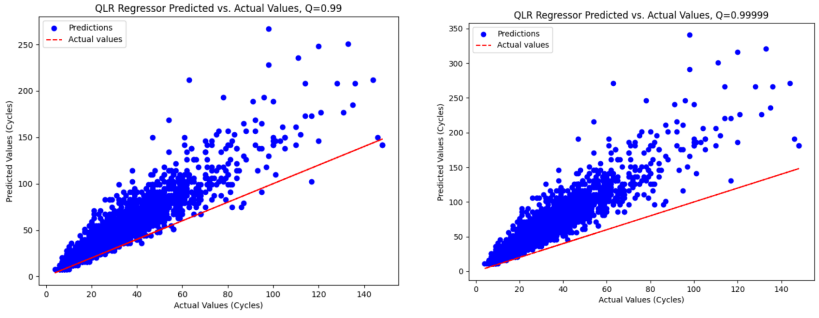


Figure 4 Predictions using QLR for quantile values of 0.99 (left) and 0.99999 (right).

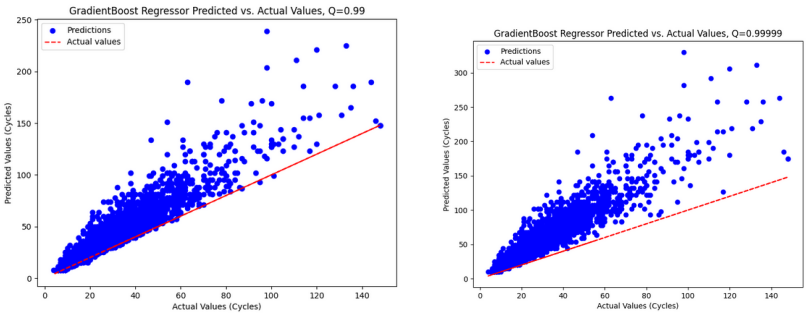


Figure 5 Predictions using GB for quantile values of 0.99 (left) and 0.99999 (right).

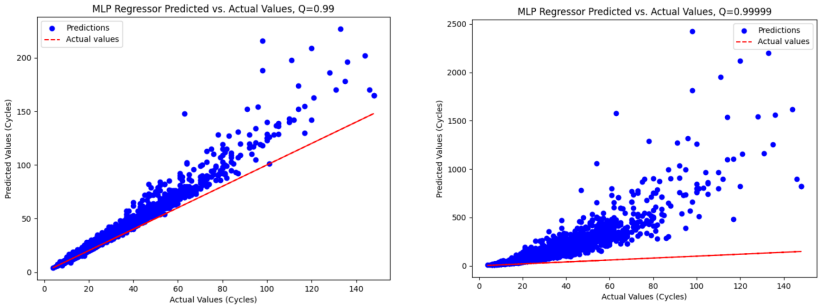


Figure 6 Predictions using MLP for quantile values of 0.99 (left) and 0.99999 (right).