Intelligent Automation & Soft Computing

Tech Science Press

DOI: 10.32604/iasc.2023.037898
Article

An OP-TEE Energy-Efficient Task Scheduling Approach Based on Mobile Application Characteristics

Hai Wang*, Xuan Hao, Shuo Ji*, Jie Zheng, Yuhui Ma and Jianfeng Yang

State-Province Joint Engineering and Research Center of Advanced Networking and Intelligent Information Services, School of Information Science and Technology, Northwest University, Xi'an, 710127, China

*Corresponding Authors: Hai Wang. Email: hwang@nwu.edu.cn; Shuo Ji. Email: jishuo@nwu.edu.cn

Received: 20 November 2022; Accepted: 12 April 2023; Published: 23 June 2023

Abstract: Trusted Execution Environment (TEE) is an important part of the security architecture of modern mobile devices, but its secure interaction process brings extra computing burden to mobile devices. This paper takes open portable trusted execution environment (OP-TEE) as the research object and deploys it to Raspberry Pi 3B, designs and implements a benchmark for OP-TEE, and analyzes its program characteristics. Furthermore, the application execution time, energy consumption and energy-delay product (EDP) are taken as the optimization objectives, and the central processing unit (CPU) frequency scheduling strategy of mobile devices is dynamically adjusted according to the characteristics of different applications through the combined model. The experimental result shows that compared with the default strategy, the scheduling method proposed in this paper saves 21.18% on average with the Line Regression-Decision Tree scheduling model with the shortest delay as the optimization objective. The Decision Tree-Support Vector Regression (SVR) scheduling model, which takes the lowest energy consumption as the optimization goal, saves 22% energy on average. The Decision Tree-K-Nearest Neighbor (KNN) scheduling model with the lowest EDP as the optimization objective optimizes about 33.9% on average.

Keywords: Trusted execution environment; energy efficiency optimization; CPU scheduling governor; machine learning

1 Introduction

With the popularization of mobile devices and the development of network technology, the security of mobile devices has become a hot topic. ARM TrustZone technology [1,2] separates System on Chip (SOC) hardware and software resources into a TEE for running key System resources [3] and a Rich Execution Environment (REE) [4] for running other system resources through the combination of hardware and software. When a process executes, REE and TEE have separate physical address spaces. REE only has access to its own corresponding space, while TEE has access to the physical address spaces of both environments [5,6]. Among them, the Application running in REE is called Client Application (CA), and the Application running in TEE is called Trusted Application

(TA) [7]. The purpose of TEE is to provide a trusted, isolated environment in which sensitive data and assets can be stored and trusted code executed to protect these sensitive assets and TA from software attacks generated within the REE. TEE has been widely used in DRM, mobile financial services, security authentication, enterprise and cloud service providers. So far, there have been many different implementation forms of trusted execution environment in the industry, such as chip-based implementation forms (Intel SGX [8], Qualcomm's QSEE [9]). Based on the TrustZone technology (Trustonic Kinibi TEE [10], Open Enclaves [11], Android Trusty TEE [12]), as well as the open source trusted execution environment, etc.

OP-TEE is an open source TEE solution developed and maintained by Linaro [13]. The solution complies with the Global Platform TEE standard and supports HiKey and common ARMv7 and ARMv8 platforms, making it the first choice for research on TEE. In OP-TEE, a CA and a TA interact with each other through the Universally Unique Identifier (UUID), as shown in Fig. 1, Where, inside the red dotted line are TEE interfaces:

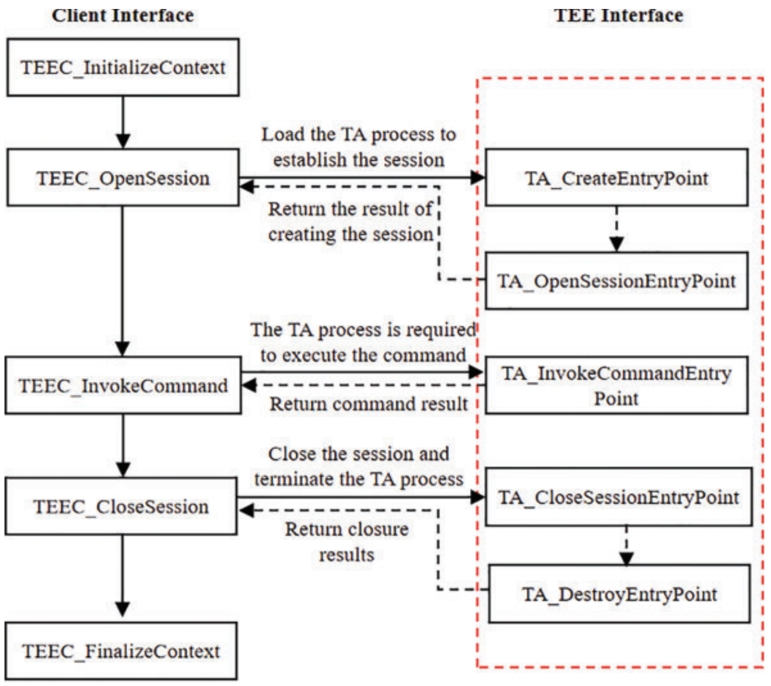
Client Interface
TEE Interface
TEEC_InitializeContext
TEEC_OpenSession
Load the TA process to establish the session
TA_CreateEntryPoint
Return the result of creating the session
TA_OpenSessionEntryPoint
The TA process is required to execute the command
TEEC_InvokeCommand
TA_InvokeCommandEntry Point
Return command result
Close the session and terminate the TA process
TEEC_CloseSession
TA_CloseSessionEntryPoint
Return closure results
TA_DestroyEntryPoint
TEEC_FinalizeContext

Figure 1: The interaction process between CA and TA

(1) CA calls the TEEC_InitializeContext() to initialize the trusted execution environment configuration, and opens the op-tee driver file to get the operation handle.
(2) CA calls the TEEC_OpenSession() to establish a communication channel with a specific TA, and uses UUID to specify TA.
(3) TA process and session are created in the trusted end, and session results are created back.
(4) CA calls TEEC_InvokeCommand() to pass the execution command to the TA process, asking it to perform the appropriate sensitive operation. A concrete operation is encapsulated into a structure, including the operation type, shared memory, session parameters, etc. The TA process executes the command and returns the result to the CA.

(5) After a successful CA call, if the TA does not need to be called again, the session needs to be closed and the TA process needs to be terminated. Ends the initial trusted execution environment configuration, ending communication between client applications and trusted applications after all shared memory resources have been freed.

Anway et al. [14] measured the interaction time of each interface of the trusted program in OP-TEE of Raspberry Pi 3B through experiments, as shown in Table 1.

Table 1: The interaction time of each interface

API name	Latency in us
TEEC_InitializeContext()	200
TEEC_OpenSession()	17000
TA_InvokeCommand()	250
TEEC_CloseSession()	1200
TEEC_FinalizeContext()	100

As can be seen from the table analysis, the interaction process with the trusted application needs at least 18000us, which is very necessary for the optimization study of the additional overhead generated by the secure interaction of trusted computing for the mobile devices with limited energy consumption. In this paper, a benchmark is firstly designed and implemented for OP-TEE considering that the widely used performance benchmarks is only suitable for rich execution environments. Then the machine learning models are used to optimize energy efficiency in combination with the CPU frequency scheduling governors. Specifically, taking the execution time, energy consumption and energy-delay product (EDP) of the program as the optimization objectives, the optimal CPU frequency scheduling strategy was respectively predicted based on the optimization objectives and program characteristics. The experimental results show that compared with the default powersave governor, the proposed scheduling methods in this paper save 34.14% time, 31.12% energy and 33.97% EDP on average. The scheduling strategies supported by the experimental platform Raspberry Pi 3B in this paper are shown in Table 2.

Table 2: CPU scaling governors on Raspberry Pi 3B

Governor	Description
Performance	Set CPU to use the maximum clock rate.
Ondemand	Timely sampling and calculating the CPU load, switching to the maximum clock frequency when the load is greater than the threshold, otherwise switching to the minimum clock frequency.
Powersave	Set CPU to use the minimum clock rate.
Conservative	Timely sampling and CPU load calculation, frequency switching is more gentle than ondemand.
Schedutil	The kernel scheduler is used to receive CPU utilization information as the basis for dynamically regulating CPU frequency.

In this paper, under different CPU frequency scheduling governors, the calculation amount of the computation-intensive application instance was changed to measure the execution time, energy consumption and EDP of the program. The results are shown in Tables 3–5 where the left-most column represents the CPU frequency scheduling governors and the other columns represent the calculations of computing the first 10, 100, 1000, 10000, 100000 primes. Through comparative analysis, we find that: (1) the optimal CPU scheduling governor for the same application is not fixed according to different indexes; (2) With the increase of task computation, the optimal CPU scheduling governor changes.

Table 3: The time of changing the range of task calculation in different governors (unit:ms)

Governor	10	100	1000	10000	100000
Performance	262.85	294.96	579.92	1706.14	11904.05
Ondemand	263.11	316.91	598.54	1913.61	12044.66
Powersave	286.08	317.75	642.51	2528.59	18234.85
Conservative	275.32	310.98	616.39	1870.51	11933.78
Schedutil	268.06	315.18	608.84	2032.96	12005.45

Table 4: The energy of changing the range of task calculation in different governors (unit:j)

Governor	10	100	1000	10000	100000
Performance	0.455	0.529	1.008	2.998	22.102
Ondemand	0.449	0.520	1.007	3.263	22.276
Powersave	0.431	0.491	0.981	3.904	29.034
Conservative	0.447	0.510	1.044	3.191	22.233
Schedutil	0.430	0.524	1.026	3.409	22.245

Table 5: The EDP of changing the range of task calculation in different governors

Governor	10	100	1000	10000	100000
Performance	0.120	0.156	0.585	5.115	263.102
Ondemand	0.116	0.165	0.603	6.246	268.312
Powersave	0.123	0.156	0.630	9.872	529.426
Conservative	0.123	0.158	0.644	5.969	265.325
Schedutil	0.115	0.165	0.625	6.931	267.065

2 Related Work

TEE is an important part of the security architecture of modern mobile devices, but its secure interaction process brings extra computing burden to mobile devices. Amacher et al. used simulation software and hardware tools to evaluate the performance of three different CPU frequency scheduling strategies (ondemand, performance and powersave) for ARM TrustZone [15]. Liu et al. proposed a program analysis and transformation techniques for the large time cost caused by TEE's complex program transformation [16]. Mukherjee et al. proposed an approach which integrated multiple trusted execution parts to minimize I/O traffic and reduce the number of switches between REE and TEE, thus reducing TEE execution overhead and increasing predictability by [14]. Yang proposed to configure trusted execution environment parameters, interrupt processing and shared memory allocation to optimize the performance of TrustZone applications. However, it is not universal due to that this method has great changes to the underlying hardware devices and has different implementation methods for different hardware devices [17]. Bailleu et al. designed and implemented a performance measuring tool, Tee-perf, to identify and optimize the application programs in TEE [18]. It used the flame graph to check the occupied time of each instruction during program execution and optimized the instruction with the largest occupied time. However, since the instruction with the largest time cost is changing, Tee-perf is not universally applicable.

Recently, machine learning method is widely used in the energy efficiency optimization of mobile devices. Ren et al. used machine learning method to predict the optimal CPU configuration and adjust the CPU frequency based on Web content and network status to reduce energy consumption for the mobile devices [19,20]. Yuan et al. used neural networks to dynamically schedule the CPU resources for the mobile Web interaction [21]. Aiming at the energy efficiency of mobile streaming media, Zhao et al. collected user experience and used machine learning method to configure the optimal CPU frequency and video bit rate based on the state of mobile devices and video task complexity, thus improving the user experience and reducing the energy consumption [22]. Chen et al. used a machine learning model to predict the optimal image coding rate based on the complexity of multi-artifact reduction (MAR) image and mobile device network status for mobile augmented reality applications, thus reducing MAR response delay and improving device energy efficiency [23]. Fan et al. used the machine learning method to predict the optimal Graphics Processing Unit (GPU) kernel and CPU frequency configuration for the input OpenCL kernel [24].

3 Our Approach

This paper designs and implements Benchmark according to OP-TEE, and uses performance analysis tools to analyze program features to build data sets, selects lightweight model for optimal governor prediction, and finally realizes governor scheduling. The experimental process is shown in Fig. 2.

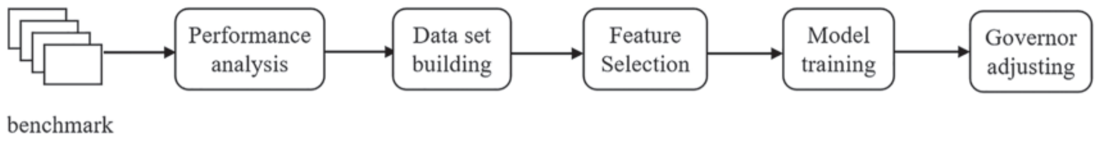

Figure 2: Overview of our approach

Combining the advantages of lightweight model and high accuracy model, a high energy efficiency frequency scheduling strategy based on trusted environment is proposed for different indicators. The strategy aims to reduce reasoning time and energy consumption as much as possible while ensuring accuracy. By deploying a lightweight and low overhead composite model on the mobile side to provide low latency response. The model flow is shown in the Fig. 3.

(1) Mobile terminal calls trusted applications.
(2) Extract typical features representing program type and complexity, input into the regression model.
(3) The output program of the regression model predicts the results of the corresponding optimization indexes, and outputs the optimal Governor according to the predicted results.
(4) The mobile terminal adjusts the CPU frequency according to the output results.

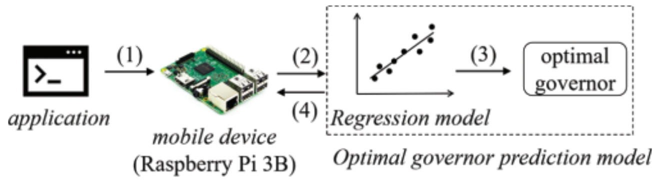

Figure 3: Efficient frequency scheduling policy process based on OP-TEE

In this section, the benchmark designed for OP-TEE is introduced, respectively, and the characteristics of the benchmark test program are collected by Perf [25] for feature screening. The classical regression algorithm is used to predict the experimental optimization indexes, and the performance of different models is compared.

3.1 The Optimization Goals

In this paper, the program execution time, energy consumption and energy-delay product (EDP) were taken as targets for optimization [26]. EDP refers to the product of energy and time consumed to complete a computing task, an evaluation index adopted when considering both high performance and low power consumption.

3.2 Benchmark

Since the mainstream benchmark Unixbench [27], Sysbench [28] is only applicable to REE, TEE has not yet been implemented. This paper designed and implemented a benchmark for TEE based on Unixbench and Sysbench, which was divided into CPU computation-intensive applications and I/O intensive applications. CPU computation-intensive applications are characterized by the need to make full use of CPU resources for calculation and logical judgment. I/O intensive applications are characterized by the majority of program execution in which the CPU is waiting for I/O (hard disk/memory) read/write operations. Our benchmark includes the basic operations of embedded applications, which can fully reflect the execution characteristics of practical applications, as shown in Table 6.

Table 6: Benchmark applications

Item	Description	Goal
Prime	Computes all prime numbers in a specified range of natural numbers.	Measure CPU integer performance.
String	Count the number of characters and invert the string with the length of 2048 bytes, etc.	Measure CPU character manipulation performance.

(Continued)

Table 6: Continued

Item	Description	Goal
Hanoi	20 disks of increasing size are transferred through 3 towers.	Measure calling stack performance.
Process communication	Two processes are piped and processed a specified number of times by integer decrement.	Measure the efficiency of data exchange between processes.
Mutex	All threads are simulated to run concurrently at the same time, and shared variables are protected by mutex. The operation of shared variables is realized in TA. Set the number of concurrent threads to be tested to 1000, and randomly assign mutexes to each thread.	Measure OP-TEE's performance in allocating mutex resources under high thread concurrency.
File	Write, read, and delete files of specified size.	Measure the performance of file operations.

3.3 Features Selection

Perf was used to measure the features of 31 measurement program types in the experiment. Fig. 4 shows the results of Pearson correlation coefficient.

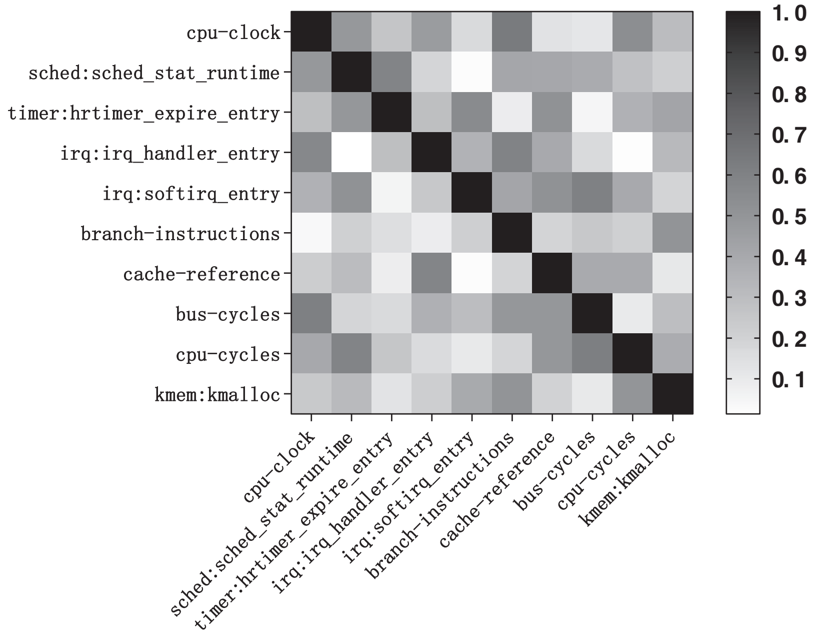

Figure 4: Pearson correlation coefficient heat map between ten features

After Pearson correlation coefficient was calculated with the optimization target, 10 independent features were screened and retained as model features, as shown in Table 7.

Table 7: Model features

Feature	Description
cpu-clock	CPU clock frequency.
sched:sched_stat_runtime	CPU scheduler event that quantifies scheduler delay. The program is in running time.
timer:hrtimer_expire_entry	Kernel high precision timer expiration time.
irq:irq_handler_entry	Number of times the interrupt handler processed the request.
irq:softirq_entry	Number of soft interruption calls.
branch-instructions	Branch instruction number.
cache-reference	Cache reference size.
bus-cycles	The bus cycle.
cpu-cycles	Number of processor cycles consumed.
kmem:kmalloc	Number of kernel memory allocations.

4 Experiment

4.1 Platform

Hardware platform. This paper uses Raspberry Pi 3B embedded platform, which is equipped with Broadcom BCM2837 system on chip (1GB RAM, quad-core ARM Cortex A53 running at 1.2 GHz).

Measurement tools. The test was conducted by connecting the POWER-ZKM001 device to the USB Power supply and the Raspberry Pi device and using the POWER-Z desktop software to control and measure and record the energy consumption data of the application execution.

Machine learning models. Considering the resource constraints of the hardware platform, the regression model selected should consider two factors: low reasoning time and high accuracy. In this paper, six classic regression algorithms were selected, including Line Regression, KN), Decision Tree, Gradient Boosting, SVR and Random Forest.

4.2 Single Model

4.2.1 Time Optimization Model

Fig. 5 shows the prediction accuracy of program execution time under the six regression models where the black bar indicates accuracy and corresponds to the left y-axis. The gray bar indicates inference time and corresponds to the right y-axis. As can be seen from the figure, Random Forest and KNN have the highest accuracy, which is 93.627% and 91.774%, respectively.

4.2.2 Energy Optimization Model

Fig. 6 shows the prediction accuracy of the six models for program energy consumption. As can be seen from the figure, KNN and SVR have the highest accuracy, which is 91.376% and 91.886%, respectively.

4.2.3 EDP Optimization Model

Fig. 7 shows the accuracy of six models in predicting EDP index. According to the figure, Decision Tree and KNN have the highest accuracy, which is 92.259% and 90.62%, respectively.

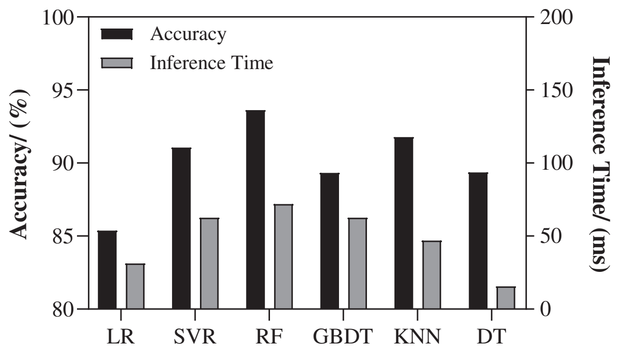

Figure 5: Comparison of time optimization models

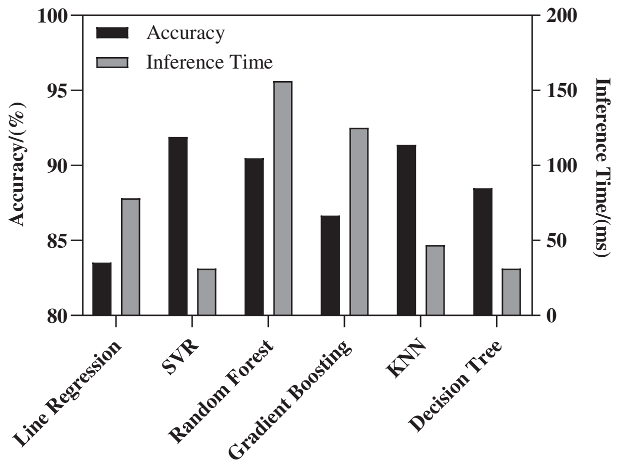

Figure 6: Comparison of energy optimization models

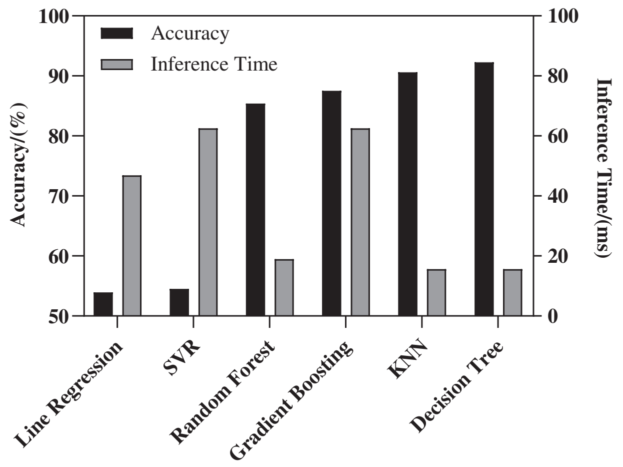

Figure 7: Comparison of EDP optimization models

4.3 Combined Model

In order to further improve the prediction accuracy of the single model used in this article, this article combines multiple models and uses a single fully connected layer to fuse the results of a single model to obtain a combined model. After considering the weakness of the mobile platform's weak computing power, this article uses two basic models to combine, and the modeling process is shown in Fig. 8, Among them, input X of the combined model is model 1, model 2, ..., model n output. In addition, $w_1, w_2, \dots, w_n$ represent the weight of the models, and $w_1 + w_2 + \dots + w_n = 1$.

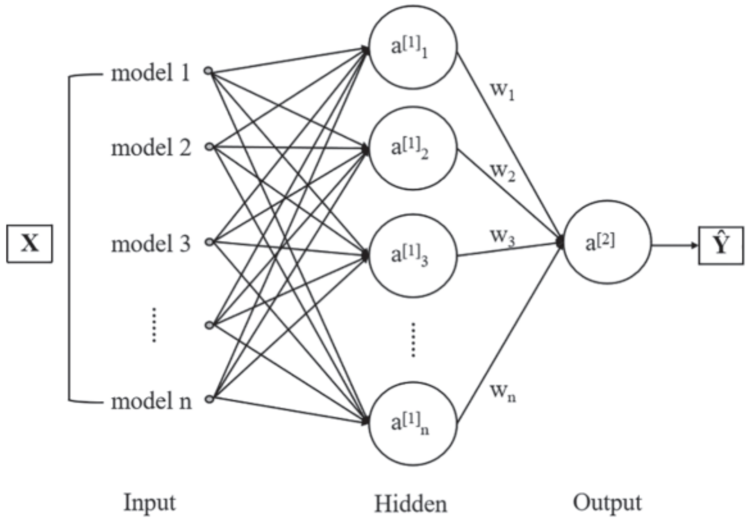

Figure 8: Combined model modeling process

Aiming at time, energy consumption and EDP, this paper uses the combined model for modeling with high accuracy and low inference time as the objectives respectively, and evaluates the optimization effect of the combined model.

4.3.1 Low Delay Combined Model

a) High accuracy: According to the comparison of the prediction accuracy in Fig. 5, the two models with the highest accuracy, Random Forest and KNN. The prediction accuracy of the combined model is shown in Fig. 9. The accuracy is 97.267%, and the reasoning time is about 103.13 ms, which can optimize the average execution time of the program by 34.138%.
b) Low inference time: Through the steps shown in Fig. 7, the model prediction performance after combining the two models (Line Regression: 31.25 ms, Decision Tree: 15.63 ms) with the least inference time in Fig. 2 is shown in Fig. 10, and the model accuracy rate is about 91.699%, The inference time is about 46.875 ms. Compared with Fig. 6 high-accuracy combined model, the model reasoning time can be saved by 76.923%. Compared with the test set data, the program execution time can be optimized by 21.181% on average.

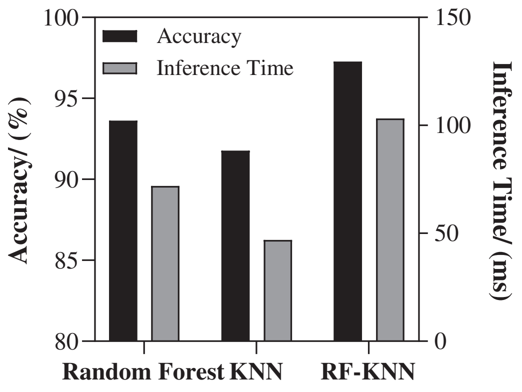

Figure 9: Time combination model for high accuracy

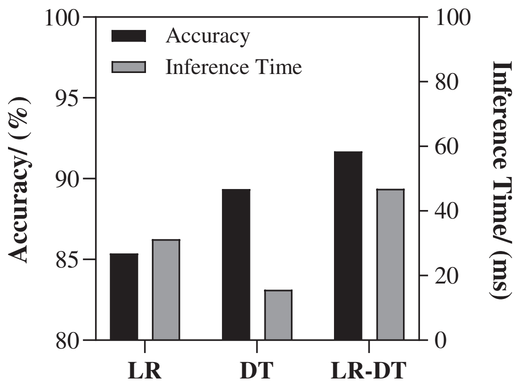

Figure 10: Time combination model for low inference time

4.3.2 Low Energy Consumption Combined Model

a) High accuracy: It can be seen from the prediction accuracy of the model in Fig. 5 that KNN and SVR have the highest prediction accuracy. The prediction accuracy and inference time of the combined model are shown in Fig. 11. The accuracy of the model is about 95.087%, and the inference time is about 62.5 ms. Compared with the test data, the energy consumption of the program can be optimized by 31.116% on average.
b) Low inference time: By comparing the inference time of the models in Fig. 5, the two models with the lowest inference time are Decision Tree and SVR, both of which are 31.25 ms. As shown in Fig. 12, the accuracy of energy consumption prediction of the combined model and the inference time of the model is about 92.589%, and the inference time is about 46.875 ms. By comparing the inference time of the model, it can be seen that the inference time of this model is 25% less than that of the KNN+SVR in Fig. 10. Compared with the test set, the combined model can optimize the program energy consumption by 22.003% on average.

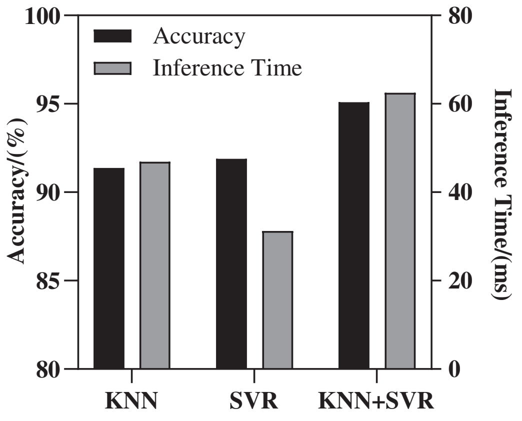

Figure 11: Energy combination model for high accuracy

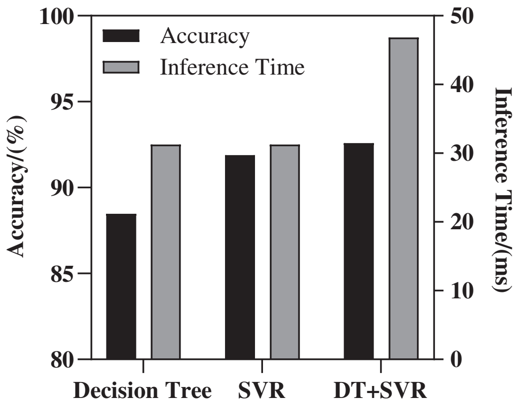

Figure 12: Energy combination model for low inference time

4.3.3 Low EDP Combined Model

a) High accuracy: The comparison of EDP prediction accuracy results of each single model in the Fig. 6 shows that the Decision Tree and KNN models have the highest accuracy. After the combination, the prediction performance of the combined model is shown in Fig. 13, and the accuracy of the combined model is about 96.377%. By comparing the prediction results of the combined model with the results of the test set, the model can optimize the program EDP by 33.973% on average.

b) Low inference time: By comparing the inference time of various models in Fig. 6, it can be seen that the inference time of KNN and Decision Tree models is the lowest, both 15.625 ms, while the inference time of the combined model is 46.86 ms, as shown in Fig. 12.

Since machine learning methods have a low CPU and memory overhead and have a higher prediction accuracy, the machine learning methods achieve good performance in energy efficiency optimization based on program characteristics for TEE. And for our constructed small-scale benchmark, our proposed machine learning methods perform better than the system default CPU frequency scheduling strategy.

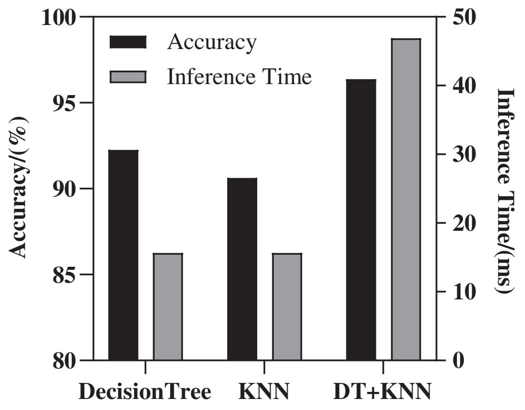

Figure 13: Combined models for EDP

5 Conclusion

In this paper, Raspberry Pi 3B hardware platform is used to study the application performance in the op-tee environment. By changing the computation amount of tasks under different CPU frequency scheduling strategies, it is found that the optimal scheduling mode is not fixed for time, energy consumption and EDP. Therefore, the machine learning model can be used to dynamically change the CPU frequency scheduling strategy according to the application characteristics to reduce program energy consumption.

In this paper, regression models are used to combine the models with high accuracy and low reasoning time as the combination goals respectively, and the parallel combination mode is used to combine the models. Experimental verification results show that the proposed scheduling method has better energy efficiency than the default scheduling strategy. Among them, the shortest delay scheduling model with high accuracy as the combined target Random Forest-KNN can save 34.14% of the time on average. The combination of Line Regression-Decision Tree with low inference time as the target can save 21.18% on average; The combination of KNN-SVR with high accuracy can save 31.12% on average, and the combination of Decision Tree-SVR with low reasoning time can save 22% on average. Decision Tree-KNN, the lowest EDP prediction model, combined with the advantages of high accuracy and low reasoning time, can be optimized by 33.97% on average. In addition, the combined model proposed in this paper is of low time consuming and universal applicability.

Funding Statement: This work is funded by National Key Research and Development Program of China under Grant No. 2019YFC1520904 from January 2020 to April 2023. This work is funded by Shaanxi Innovation Program under Grant 2023-CX-TD-04 January 2023 to December 2025.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] 2015, ARM. “ARM security technology-building a secure system using TrustZone technology [EB/OL],” https://developer.arm.com/document-ation/genc009492/c

[2] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi *et al.,* “Innovative instructions and software model for isolated execution,” in *Proc. of HASP@ Isca*, Tel-Aviv, Israel, 10, 2013.

[3] J. Luo, C. H. Jiang and X. Yan, "Design and implementation of security OS based on TrustZone," in *Proc. of Int. Conf. on Electronic Measurement & Instruments*, Harbin, China, pp. 1027–1032, 2013.

[4] J. E. Ekberg, K. Kostiainen and N. Asokan, "Trusted execution environments on mobile devices," in *Proc. of ACM SIGSAC Conf. on Computer & Communications Security*, Berlin, Germany, pp. 1497–1498, 2013.

[5] X. Y. Zheng, W. Li and D. Meng, "Analysis and research on TrustZone technology," *Chinese Journal of Computers*, vol. 39, no. 9, pp. 1912–1928, 2016.

[6] H. Wang, L. Cai, X. Hao, J. Ren and Y. H. Ma, "ETS-TEE: An energy-efficient task scheduling strategy in a mobile trusted computing environment," *Tsinghua Science and Technology*, vol. 28, no. 1, pp. 105–116, 2022.

[7] S. R. Huang, "Research and implementation of data security encryption method based on TrustZone technology," M.S. Dissertation, Beijing University of Posts and Telecommunications, China, 2019.

[8] J. W. Wang, Y. Jiang, Q. Li and Y. Yang. "Survey of research on SGX technology application," *Journal of Network New Media*, vol. 6, no. 5, pp. 3–9, 2017.

[9] G. Beniamini. "Exploring qualcomm's secure execution environment," 2016. [Online]. Available: http://bits-please.blogspot.gr/2016/04/exploring-qualc-omms-secure-execution.html (accessed on 4 January 2021).

[10] K. Xiao, "A solution to make trusted execution environment more trustworthy," *International Journal of Performability Engineering*, vol. 14, no. 9, pp. 2127, 2018.

[11] D. Kohlbrenner, S. Shinde, D. Lee, K. Asanovic and D. Song, "Building open trusted execution environments," *IEEE Security & Privacy*, vol. 18, no. 5, pp. 47–56, 2020.

[12] C. Shepherd, G. Arfaoui, I. Gurulian, R. P. Lee, K. Markantonakis *et al.,* "Secure and trusted execution: Past, present, and future-a critical review in the context of the internet of things and cyber-physical systems," *IEEE Trustcom/BigDataSE/ISPA*, Tianjin, China, pp. 168–177, 2016.

[13] 2021, Linaro. "OP-TEE documentation-about OP-TEE," https://optee.readthedocs.io/en/latest/general/about.html

[14] M. Anway, M. Tanmaya, C. Thidapat, F. Nathan and G. Ryan, "Optimized trusted execution for hard real-time applications on COTS processors," in *Proc. of RTNS*, Toulouse, France, pp. 50–60, 2019.

[15] J. Amacher and V. Schiavoni, "On the performance of arm trustzone," in *Proc. of DAIS*, Kongens Lyngby, Denmark, pp. 133–151, 2019.

[16] Y. Liu, K. An and E. Tilevich, "RT-Trust: Automated refactoring for different trusted execution environments under real-time constraints," *Journal of Computer Languages*, vol. 56, pp. 100939, 2019.

[17] B. X. Yang, P. Dong, L. J. Zhang and Y. Ding, "Performance optimization of secure application based on TrustZone," *Computer Engineering & Science*, vol. 42, no. 12, pp. 2141–2150, 2020.

[18] M. Bailleu, D. Dragoti, P. Bhatotia and C. Fetzer, "TEE-Perf: A profiler for trusted execution environments," in *Proc. of DSN*, Portland, OR, USA, pp. 414–421, 2019.

[19] J. Ren, X. Wang and J. Fang, "Proteus: Network-aware web browsing on heterogeneous mobile systems," in *Proc. of CoNEXT*, New York, USA, pp. 379–392, 2018.

[20] J. Ren, L. Yuan, P. Nurmi, X. M. Wang, M. Ma *et al.,* "Camel: Smart, adaptive energy optimization for mobile web interactions," in *Proc. of INFOCOM*, Toronto, ON, Canada, pp. 119–128, 2020.

[21] L. Yuan, J. Ren, L. Gao, Z. Y. Tang and Z. Wang, "Using machine learning to optimize web interactions on heterogeneous mobile systems," *IEEE Access*, vol. 7, pp. 139394–139408, 2019.

[22] Z. Zhao, L. Gao, J. Ren, L. Yuan, C. G. Qin *et al.,* "Optimization for mobile streaming media based on deep Q-learning," in *Proc. of CBD*, Suzhou, China, pp. 285–290, 2019.

[23] L. Chen, L. Gao, J. Ren, X. Dang, Y. H. Wang *et al.* "Adaptive bitrate streaming for energy-efficiency mobile augmented reality," *Computer Science*, vol. 49, no. 1, pp. 194–203, 2022.

[24] K. Fan, B. Cosenza and B. Juurlink. "Accurate energy and performance prediction for frequency-scaled GPU kernels," *Computation*, vol. 8, no. 2, pp. 37, 2020.

[25] H. Q. Zhang and H. Cheng. "Research on evaluation of linux performance monitoring tools," *Computer Technology and Development*, vol. 28, no. 9, pp. 88–93, 2018.

[26] J. Ren, L. Gao, J. L. Yu and L. Yuan. "Energy-efficient deep learning task scheduling strategy for edge device," *Chinese Journal of Computers*, vol. 43, no. 3, pp. 440–452, 2020.

[27] M. Raho, A. Spyridakis, M. Paolino and D. Raho, “KVM, Xen and docker: A performance analysis for ARM based NFV and cloud computing,” in *Proc.of AIEEE*, Riga, Latvia, pp. 1–8, 2015.
[28] J. W. Krogh, “Benchmarking with Sysbench,” in *MySQL 8 Query Performance Tuning*, Berkeley, CA: Apress, pp. 19–53, 2020.