STAFF: Online Learning with Stabilized Adaptive Forgetting Factor and Feature Selection Algorithm

Ujjwal Gupta¹, Manoj Babu¹, Raid Ayoub²,
Michael Kishinevsky², Francesco Paterna², Umit Y. Ogras¹
¹Arizona State University, ²Intel Corporation

ABSTRACT

Dynamic resource management techniques rely on power consumption and performance models to optimize the operating frequency and utilization of processing elements, such as CPU and GPU. Despite the importance of these decisions, many existing approaches rely on fixed power and performance models that are learned offline. However, offline models cannot guarantee accuracy when workloads differ significantly from the training available at design time. This paper presents an online learning framework (STAFF) that constructs adaptive run-time models for stationary and non-stationary workloads. STAFF is the first framework that (1) *guarantees stability while quickly adapting to workload changes*, (2) *performs online feature selection with linear complexity*, and (3) *adapts to new model coefficients by employing adaptively varying forgetting factor, all at the same time*. Experiments on an Intel® Core™ i5 6th generation platform demonstrate up to 6× improvement in the performance prediction accuracy compared to existing techniques.

1 INTRODUCTION

Computing systems ranging from mobile platforms to servers run millions of applications, such as games, navigation and browsers. The number and types of these applications are expected to increase further [21]. Sophisticated runtime techniques schedule these applications to the hardware resources and perform dynamic thermal and power management (DTPM) decisions [2, 9, 12, 17, 19]. These techniques need to assess the impact of control variables, such as the operating frequency, to optimize power, performance, energy or other metrics. Therefore, the accuracy of the models used at runtime is critical to meet stringent optimization objectives.

Typically, runtime models are trained offline by employing supervised machine learning, such as batch linear regression [11]. Data used for training involves a set of known application scenarios. There are three main problems with offline approaches. *First*, offline models can only be trained on a limited set of applications available at design time. Hence, these models cannot guarantee a reliable operation for a wide set of applications. *Second*, workloads can be non-stationary, i.e., the statistics like mean and variance of data can change at runtime for different applications [3]. Figure 1(a) depicts a non-stationary GPU workload, where both the frame processing time and its frequency sensitivity vary with time. Similarly,

Figure 1: A non-stationary GPU workload (a) example and (b) analysis using autocorrelation function (ACF).

Figure 1(b) shows that the autocorrelation function (ACF) decreases slowly for the entire workload from 0-30 seconds, while ACF for individual phases of the workload ([0,10], [10,20], [20,30] seconds) decays faster. Standard Kwiatkowski-Phillips-Schmidt-Shin (KPSS) and augmented Dickey-Fuller (ADF) tests [8] also confirm that individual regions are stationary, while the entire workload is non-stationary. Consequently, offline models are unsuitable for learning non-stationary workloads as they have fixed coefficients that cannot change at runtime. *Finally*, the optimum set of features that describe the underlying metric change dynamically with the workload. For example, Intel Skylake GPU has 35 hardware counters that can be used as feature inputs to a model [10]. Our experiments show that only a dynamically varying subset of such counters is a good indicator for building a model. Hence, features should be selected at runtime, in contrast to offline models, which rely on a set of features determined at design time.

We present an online learning framework, STAFF, to estimate model coefficients without offline training. STAFF combines *guaranteed stability*, *online feature selection* and *adaptive forgetting factor* into a *single computationally efficient runtime framework*. Online algorithms, such as Recursive Least Squares (RLS), employ exponential forgetting factor to discard old data points and make room for learning from new data points [18]. However, this can result in instability during non-persistently exciting inputs, such as idle period between two workloads [13]. We *guarantee stability* by bounding the correlation matrix of the proposed online algorithm. Moreover, the set of most useful features may change over time, while using all the features leads to overfitting and increases computational cost. Thus, our framework performs *runtime feature selection* by dynamically computing the correlation of each feature with the output. Finally, the forgetting factor should be adaptive, since old data points need to be forgotten faster during phase transitions, as illustrated in Figure 1(a). Our framework also dynamically adapts

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DAC '18, San Francisco, CA, USA

© 2018 ACM. 978-1-4503-5700-5/18/06...\$15.00

DOI: 10.1145/3195970.3196122

the forgetting factor to the workload. In summary, STAFF improves the accuracy of runtime models, and reduces their deployment cost on new systems by eliminating offline characterization effort.

In summary, the main contributions of this work are as follows:

- Online feature selection with linear complexity,
- Dynamic forgetting factor methodology that employs a novel entropy-based workload change detection,
- Guaranteed stability while quickly adapting to workload,
- Empirical evaluation on an Intel® Core™ i5 6th generation platform using 17 graphics benchmark scenarios. We achieve up to 6× better accuracy compared to existing techniques.

The remainder of the paper is organized as follows: §2 presents an overview of related research. §3 presents the methodology to develop the proposed STAFF framework. §4 summarizes the STAFF framework and shows the complexity analysis. §5 discusses the experimental results, and §6 presents the conclusion.

2 RELATED RESEARCH

Online learning for power and performance models is relatively new in the power management field. In online learning, model coefficients are estimated at runtime using standard algorithms, such as batch linear regression [17], least mean squares (LMS) [5], recursive least squares [7, 9, 15], and an array form of RLS algorithm using QR decomposition [19]. The standard versions of these algorithms are well-known in literature and have severe limitations [18]. In particular, the batch linear regression approach uses costly matrix inversions and large memory in each step, thus providing poor scalability. The standard LMS technique has slower convergence than RLS. Therefore, RLS has turned up as a popular choice for online learning of power and performance models.

RLS is a type of Kalman Filter with fast tracking capabilities for sequential data [18]. Recently, RLS algorithm *without forgetting factor* has been adopted for learning performance models for CPU and GPU [7, 19]. This works well for single workload scenarios, but incorporating variable forgetting factor is crucial for practical applications which switch between different workloads and idle periods [6]. A constant exponential forgetting RLS has been employed for power and performance predictions for CPUs [9, 15]. It is known that all methods that employ exponential forgetting are prone to instability during non-persistently exciting inputs [16]. Kreisselmeier proposed a general class of exponential forgetting RLS algorithm with stability [13]. A special case of Kreisselmeier's algorithm has been shown to guarantee stability under dynamically varying forgetting factor by Milek [16]. However, these algorithms still cannot select features at runtime. A recent approach performs online feature selection by combining all the features into a single feature [20]. However, this method uses a fixed forgetting factor without stability guarantees. Similarly, Fortescue et al. proposed a variable forgetting factor RLS [6], but their solution can neither guarantee stability nor perform online feature selection. In contrast to these approaches [6, 13, 16, 20], STAFF combines adaptive forgetting factor, guaranteed stability and runtime feature selection into a single online learning framework.

An online learning framework also requires a runtime technique to track large changes in workloads, such as transitioning from one application to another. A recent proposal employs Kullback-Leibler (KL) divergence test for this purpose [4]. The authors use only one

input variable (the CPU cycles) due to runtime complexity of the KL test. In contrast, we develop a novel low-cost information theoretic approach that utilizes multiple hardware counters.

3 STAFF ONLINE LEARNING FRAMEWORK

This section introduces first the model templates used for performance modeling. Then, it presents the standard form of RLS with constant forgetting and stabilization. Finally, it presents the online feature selection and adaptive forgetting factor mechanisms.

3.1 Model Template

Consider a general linear model with input features $\mathbf{h}_k = [h_{0,k}, h_{1,k}, \dots, h_{M-1,k}]^T$ and output y_k at time k ,

$$y_k = \mathbf{a}^T \mathbf{h}_k \quad (1)$$

where $\mathbf{a} = [a_0, a_1, \dots, a_{M-1}]^T$ are model coefficients, M is the number of features, and $(\cdot)^T$ is the transpose operator. Such linear models are employed by a number of approaches for power and performance modeling of the CPU and GPU [7, 15, 22]. For example, a CPU performance model includes predicting cycles per instruction [22]. Similarly, a GPU performance model includes predicting frame processing time [7]. For instance, one can express the change in frame time $\Delta t_{F,k}$ as a function of the GPU frequency f_k and hardware counters $x_{i,k}$ as:

$$\Delta t_{F,k} = a_0 t_{F,k-1} \left(\frac{f_{k-1}}{f_k} - 1 \right) + \sum_{i=1}^{M-1} a_i \Delta x_{i,k} \quad (2)$$

where $t_{F,k}$ is the frame time at time k , $\mathbf{a}$ are the model coefficients and $[t_{F,k-1}(\frac{f_{k-1}}{f_k} - 1), \Delta x_{1,k}, \Delta x_{2,k}, \dots, \Delta x_{M-1,k}]^T = \mathbf{h}_k$ are the features. DTPM and scheduling algorithms can use the model coefficients to make runtime decisions [1, 2]. For example, a_0 denotes the frequency sensitivity of frame time, i.e., it quantifies how frequency affects the frame time. Note that the GPU performance modeling technique proposed in [7] cannot perform online feature selection, and uses a constant forgetting factor equal to one. Hence, it is not suitable for tracking non-stationary workloads.

3.2 Stability under Exponential Forgetting

The information form of RLS with stabilization and exponential forgetting can estimate the model coefficients $\mathbf{a}$ as follows [13]:

$$\mathbf{R}_k = \lambda \mathbf{R}_{k-1} + \mathbf{h}_k \mathbf{h}_k^T + (1 - \lambda) \alpha \mathbf{I}, \quad \lambda \in (0, 1] \text{ and } \alpha \geq 0 \quad (3)$$

$$e_k = y_k - \mathbf{a}_{k-1}^T \mathbf{h}_k \quad (4)$$

$$\mathbf{a}_k = \mathbf{a}_{k-1} + \mathbf{R}_k^{-1} \mathbf{h}_k e_k \quad (5)$$

The correlation matrix $\mathbf{R}_k$ is recursively updated in each iteration. The forgetting factor λ and stabilization factor α are usually set to one and zero, respectively, for simplicity. The RLS algorithm without forgetting ($\lambda = 1$) has been employed previously for CPU and GPU performance predictions [7, 19]. However, this leads to infinite memory as it accounts for complete history. In turn, this makes RLS unable to adapt to non-stationary workloads. Hence, it is crucial to make $\lambda < 1$. The RLS algorithm with constant $\lambda < 1$ has been employed for CPU performance and power predictions [9, 15]. Doing so leads to another problem related to exponential decrease in $\mathbf{R}_k$ or blow-up of $\mathbf{R}_k^{-1}$ under non-persistently exciting inputs, such as no change in input during idle periods in workloads [13, 14]. For

Figure 2: The y-axis is in $\log_{10}$ scale. ℓ_1 norm of the inverse of the correlation matrix R shows unstable behavior for $\alpha = 0$ and stability for $\alpha = 10^{-5}$, respectively.

example, when $\mathbf{h}_k \rightarrow 0$, $\alpha = 0$ and $\lambda < 1$, the recursion in Equation 3 leads to a large unbounded increases in $\mathbf{R}_k^{-1}$, as illustrated in Figure 2. This can make RLS unstable and sensitive to noise based on the coefficient update in Equation 5.

The update of the correlation matrix with $\alpha > 0$ has been theoretically proven to guarantee stability [16]. The stabilizing factor α puts an upper bound on $\mathbf{R}_k^{-1}$. The RLS minimizes an ℓ_2 regularized cost function with regularization parameter μ that is determined using cross-validation [18]. We set $\alpha = \mu$ because the stabilization factor has a similar effect as the regularization parameter. The correlation matrix and weights are initialized as $\mathbf{R}_0 = \mu \mathbf{I}$ and $\mathbf{a}_0 = 0$, respectively. Figure 2 contrasts the blow-up of the inverse of the correlation matrix without stability ($\alpha = 0$) and our result. The STAFF framework with $\alpha = 10^{-5}$ adds an upper bound on the inverse of the correlation matrix, which avoids instability.

3.3 Online Feature Selection

State-of-the-art platforms provide a large number of hardware counters that can be used as input features to the power and performance models. For example, the Intel Skylake GPU has 35 hardware counters that can be used as feature inputs to a model [10]. Using all the features leads to large computation overhead due to the inversion of the correlation matrix $\mathbf{R}_k$ in Equation 5.

A recently proposed mechanism for runtime feature selection [20] combines all the features into a single combined feature. However, one might need to preserve a certain term, such as the frequency sensitivity term a_0 used by DTPM algorithms. Therefore, instead of using the model $\mathbf{a}^T \mathbf{h}_k$ in Equation 1, we estimate an auxiliary model given as follows:

$$y_k = a_0 h_{0,k} + a_c h_{c,k} \quad (6)$$

The feature $h_{0,k}$ and coefficient a_0 still represent the frequency term and sensitivity, respectively. The combined feature $h_{c,k}$ is a function of the features $h_{i,k} \forall i \in [1, M-1]$, which captures the workload changes. Hence, our revised model *successfully decouples* the impact of frequency (first term) and workload (second term).

Computing the Combined Feature $h_{c,k}$: A feature that is highly correlated to the output will also be more likely to predict the output compared to the features that are less correlated. Thus, we start with a number of single-input affine functions between the output y_k and each of the features $h_{i,k}$,

$$y_k = b_i + c_i h_{i,k} \quad \forall i \in [1, M-1] \quad (7)$$

where the model coefficients b_i and c_i are estimated by applying two-input RLS described by Equations 3-5. Then, we compute the Pearson correlation coefficients, $\rho_{i,k}$ between the output y_k and each feature $h_{i,k}$ as follows:

$$\rho_{i,k} = \frac{\sigma_{h_{i,k}}}{\sigma_{y_k}} c_{i,k}, \text{ where } -1 \leq \rho_{i,k} \leq 1 \quad \forall i \in [1, M-1] \quad (8)$$

The standard deviations $\sigma_{h_{i,k}}$ and σ_{y_k} are computed recursively by employing forgetting factors. Note that each model can employ a different forgetting factor λ ; therefore, the standard deviations σ_{i,y_k} may not be same for each model output. Then, we combine the correlation coefficients to find the likelihood $\pi_{i,k}$ of a feature $h_{i,k}$ to predict the output y_k as follows:

$$\pi_{i,k} = \frac{\rho_{i,k}^2}{\sum_{i=1}^{M-1} \rho_{i,k}^2}, \text{ where } 0 \leq \pi_{i,k} \leq 1 \quad \forall i \in [1, M-1] \quad (9)$$

Note that the likelihoods $\pi_{i,k}$ sum up to one. Finally, the combined feature is expressed as the weighted sum of original features $h_{i,k}$:

$$h_{c,k} = \sum_{i=1}^{M-1} \pi_{i,k} h_{i,k} \quad \forall i \in [1, M-1] \quad (10)$$

Now that $h_{0,k}$ and $h_{c,k}$ are known, we estimate the model coefficients a_0 and a_c in Equation 6 by applying the two-input RLS described by Equations 3-5.

Figure 3 shows the comparison between the offline feature selection algorithm and the STAFF framework. STAFF tracks the reference value a_0 without the erroneous troughs by changing the features dynamically at runtime. The offline selection algorithm gives poorer results due to overfitting. There are two additional advantages of STAFF: First, the implementation is very low cost (see §4). Second, it provides the flexibility to select desired features, such as $h_{0,k}$, while merging other features, such as $h_{i,k} \forall i \in [1, M-1]$, to reduce the runtime complexity.

Figure 3: Online STAFF framework has superior tracking performance to offline feature selection.

3.4 Adaptive Forgetting Factor

A constant forgetting factor is not desirable, because the amount of history that needs to be forgotten changes dynamically. Therefore, we employ an auto-tuning technique to adapt λ to a wide variety of workloads [6]. This technique preserves the information content in each sample of RLS by adapting the forgetting factor as a function of the inputs $\mathbf{h}_k$, inverse of correlation matrix $\mathbf{R}_{k-1}^{-1}$, RLS error e_{k-1} and an initial estimate of the information content $\Sigma_0 = \sigma^2 N_0$. In these expressions, σ^2 is the process noise estimate and $N_0 = \frac{1}{1-\lambda_0}$ is the initial asymptotic memory length.

$$\lambda_k = 1 - \frac{(1 - \mathbf{h}_k \mathbf{R}_{k-1}^{-1} \mathbf{h}_k^T) e_{k-1}^2}{\Sigma_0} \quad (11)$$

λ_k is bounded using a lower bound λ_{LB} and an upper bound λ_{UB} such that $0 < \lambda_{LB} < \lambda_k < \lambda_{UB} \leq 1$. However, Equation 11 *cannot track fast changes*, such as switches from one application to another. In these cases, information stored in the correlation matrix need

to be forgotten faster. Therefore, we propose a novel technique for workload change detection based on Shannon's entropy.

Workload Change Detection: A large change in workload, such as a new type of frame, can vary the model output y_k and parameters of Equation 6. Information about the type of frame is embedded in the features $h_{i,k} \forall i \in [1, M-1]$ of the hardware counters, as mentioned in §3.3. We use the likelihoods $\pi_{i,k}$, of obtaining y_k from $h_{i,k}$, to compute the entropy H_k as follows:

$$H_k = - \sum_{i=1}^{M-1} \pi_{i,k} \log_b(\pi_{i,k}) \quad (12)$$

where the logarithm base $b = M-1$, such that the entropy is normalized. When the likelihoods $\pi_{i,k}$ are all equal, the hardware counters have the most disorder (highest entropy), i.e., no one feature is more important than the other. Conversely, when the likelihoods are not equal, then the entropy is smaller. Figure 4 illustrates the change in entropy for two features. In region 1, the likelihoods of feature 1 and feature 2 are $\pi_1 = 0.8$ and $\pi_2 = 0.2$, respectively. Then, the workload changes from region 1 to region 2 where $\pi_1 = 0.1$ and $\pi_2 = 0.9$. Due to the recursive computations involved in finding $\pi_{i,k}$, the likelihoods do not instantly jump from small to large values or vice versa. Therefore, the likelihoods change slowly and become equal ($\pi_1 = 0.5$ and $\pi_2 = 0.5$) during the transitions. Consequently, the entropy H_k approaches to one whenever the workload changes significantly. In this condition, we set the lower bound $\lambda_{LB} = 0.001$, which is still guaranteed to be stable. If the adaptive forgetting mechanism chooses the lower bound λ_{LB} , a resetting effect on the correlation matrix is achieved, leading to very fast tracking of workloads. Based on our experiments, we find this method is suitable for capturing all workload changes. Figure 5 shows that variable forgetting performed by STAFF is more desirable than constant forgetting ($\lambda = 0.99$ and $\lambda = 0.9$) algorithms that lead to poor tracking of the frequency sensitivity a_0 .

Of note, the proposed approach has a low computational overhead, since the logarithm function in Equation 12 can be implemented using a fast binary algorithm [23].

Figure 4: Illustration of the entropy-based change detection. The solid-line shows the entropy, while the and dashed-lines show the likelihoods of feature 1 and feature 2, respectively.

4 SUMMARY AND COMPLEXITY ANALYSIS

Figure 6 summarizes one iteration of the STAFF framework. We first achieve stability by employing the term $\alpha > 0$ in Equation 3. Next, we perform online feature selection by combining features at runtime based on their correlation coefficient to the output using Equations 9 and 10. This requires running a total of $M-1$ two-input RLS for estimating the composite feature in Equation 6. Then, we dynamically compute the forgetting factor using Equations 11

Figure 5: The STAFF framework adapts much faster to the new workload compared to $\lambda = 0.99$. In addition, it does not possess the local erroneous tracking of a_0 caused by $\lambda = 0.9$.

and 12. Subsequently, we learn the model coefficients in 7 using one more two-input RLS. Finally, we compute the frequency sensitivity by using a moving average with forgetting on the estimated value of a_0 in Equation 6. This is done to remove edge effect overshoots caused by sharp tracking during workload changes.

Figure 6: Summary of the STAFF algorithm.

Complexity Analysis: Model estimation algorithms must incur a low overhead as they are part of DTPM algorithms. Table 1 presents the number of scalar operations for each iteration of the proposed STAFF algorithm and a competitive baseline algorithm that employs stabilized exponential variable forgetting (SEF) with offline feature selection [16]. The SEF algorithm requires inverting the $\mathbf{R}$ matrix in Equation 5, whose computational complexity is $O(M^3)$. In contrast, the STAFF approach requires only M simple 2×2 matrix inversions of $\mathbf{R}$, which increases the complexity linearly. Therefore, the STAFF algorithm provides better scalability than the SEF algorithm. Of note, the unstabilized version of the standard RLS algorithm has a computational complexity of $O(M^2)$, since it is possible to apply the matrix inversion lemma [18].

In practice, the SEF algorithm is used in conjunction with offline feature selection (Offline_FS) methods that reduce the number of input features. The complexity of the SEF algorithm changes with the number of features M . In contrast, the STAFF framework uses all the features ($M = 17$) while keeping a constant complexity, as shown in Figure 7. We observe that the complexity of SEF is $6.5 \times$ larger than STAFF for $M = 17$. At this complexity, SEF leads to $3.2 \times$ larger mean absolute error in predicting frequency sensitivity compared to STAFF. Even though SEF has lower complexity with $M = 3$ and $M = 5$, this leads to $2.2 \times$ and $1.9 \times$ higher error, respectively. Thus, STAFF achieves higher accuracy by using online feature selection with only linear complexity.

Table 1: The number of algebraic operations in the SEF and STAFF algorithms in each iteration.

Alg.	Mul	Add/Sub	Div	Big O
SEF	$M^3 + 4M^2 + 7M + 5$	$M^3 + 4M^2 + 2M + 3$	$2M^2 - M$	M^3
STAFF	$60M - 6$	$55M - 12$	$2M + 2$	M

Figure 7: STAFF framework has $3.2\times$ lower error (right axis) in frequency sensitivity, and $6.5\times$ lower complexity compared to the SEF algorithm for $M = 17$. The errors are computed using the non-stationary workload employed in §5.2

5 EXPERIMENTS

5.1 Experimental Setup

In our experiments, we utilize the Intel® Core™ i5 6th generation platform with Microsoft Windows 10 OS. The kernel drivers read the performance counters, frame time, and GPU frequency with the help of our custom user-space routines in C. To evaluate our approach, we employ a representative set of graphic applications, such as, 3DMark, Angry Birds, Angry Bots, Jungle Run, Shark Dash, and Fruit Ninja. By utilizing the AutoHotKey tool, 17 different mixes of application scenarios are generated from these applications. The noise variance of frame time (σ_0) in our data set is found to be 0.1 ms^2 . The source code for the STAFF framework is available at <http://elab.engineering.asu.edu/public-release/>.

Baseline Algorithms: We compare our approach to several baseline algorithms, shown in Table 2. One approach employs RLS with offline feature selection (Offline_FS) [7]. We report results with two versions of this algorithm: One with constant $\lambda = 1$, and the other with $\lambda = 0.99$ and $\lambda = 0.96$. The third baseline algorithm is SEF [13, 16], which guarantees stability. The first version of this algorithm uses all the available hardware counters as features (SEF+All features), while the second one uses a subset of features selected offline (SEF+Offline_FS*).

5.2 Evaluating the GPU Performance Model

We conjoin the individual applications to compose a real world scenario that transitions between different phases. For example, Figure 8 shows a workload that includes three distinct frequency sensitivity patterns. First, three applications with high sensitivity ($a_0 \approx 1$) run consecutively during the first 2-minute duration. Then, three applications with low sensitivity ($a_0 \approx 0$) run during the next 2 minutes. Finally, three applications with high sensitivity run in the last 2-minute interval. Clearly, this workload is non-stationary because the reference frequency sensitivity (model coefficient a_0) is time varying.

Benefits of Adaptive Forgetting: Figure 8 compares STAFF against the baseline RLS algorithms. A basic RLS approach with Offline_FS ($\lambda = 1$) quickly learns the workload during the first 2-minute interval. However, it cannot track the sensitivity after the workload changes ($t = 2 \text{ min}$), since the history learned from 0 to 2 minutes is not forgotten. Reducing the forgetting factor to $\lambda = 0.99$ helps, but it is still very slow; it takes about 1 minute to go from high to low frequency sensitivity. Lowering the forgetting factor further ($\lambda = 0.96$) improves the tracking performance. However, the algorithm becomes too sensitive to the application transitions during the first two minutes (e.g., $t = 0.5$), since it starts forgetting

Table 2: Summary of the baseline algorithms and the proposed STAFF framework.

Algorithm	# Features	Var- λ & $\alpha > 0$	Complexity
Offline_FS ($\lambda = 1$)	3	–	$O(M^2)$
Offline_FS ($\lambda < 1$)	3	–	$O(M^2)$
SEF+All features	17	✓	$O(M^3)$
SEF+Offline_FS*	5	✓	$O(M^3)$
STAFF	17	✓	$O(M)$

individual applications. Therefore, an adaptive forgetting scheme is better, as it can forget slowly during the high sensitivity regions and faster during transitions. Indeed, STAFF tracks the sensitivity accurately throughout the workload, as illustrated in Figure 8. The transition interval from high to low sensitivity is about 3 seconds, which is $14\times$ faster than $\lambda = 0.99$. This happens, since the forgetting factor becomes low ($\lambda = 0.001$) when a workload change is detected. Otherwise, λ varies between the bounds $\lambda_{LB} = 0.96$ to $\lambda_{UB} = 0.99$.

Benefits of Online Feature Selection: Figure 9 shows the comparison of the proposed STAFF framework with SEF baseline algorithms that can dynamically change the forgetting factor with stabilization, but without online feature selection. The SEF algorithm that uses all features fail to provide good accuracy over the entire workload. In particular, we find that using all the features leads to overfitting and poor estimates of the frequency sensitivity during the 0-2 minute and 4-6 minute regions. Similarly, we find that SEF algorithm that uses the best set of features selected offline (Offline_FS*) performs better than using all the features. However, it has local erroneous peaks, and it converges significantly slower than the STAFF framework during the application scenario changes. To provide a deeper insight, Table 3 shows the index of the three most frequently used features throughout the lifetime of the workload. Clearly, the different features become more important as the workload changes. Therefore, algorithms that use a static set of features, such as SEF, cannot track the frequency sensitivity with high accuracy. In contrast, STAFF shows superior performance in the presence of dynamic variations.

Table 3: Top three features used by the STAFF in different time regions of the workload in Figure 9.

Time in Fig. 9	0-2 min	2-4 min	4-6 min	All (0-6 min)
Feature #1	h_1	h_{14}	h_1	h_1
Feature #2	h_2	h_7	h_2	h_2
Feature #3	h_4	h_6	h_4	h_6

Error Analysis: Figure 10 shows the relative errors in the frequency sensitivity estimate with respect to the best baseline (SEF+Offline_FS*) algorithm during different time intervals. This figure excludes the baseline Offline_FS with $\lambda = 0.99$, since it is clearly worse than Offline_FS with $\lambda = 0.96$. STAFF achieves the largest improvement during the transition period of 2-3 minutes due to our entropy based workload change detection mechanism. The Offline_FS ($\lambda = 1$) algorithm leads up to $11.9\times$ higher error during the low frequency sensitivity regions of 2-4 minutes. Similarly, the Offline_FS ($\lambda = 0.96$) and SEF algorithms with all features lead up to $6.5\times$ and $8.5\times$ higher error during the high frequency sensitivity region of 5-6 minutes. The error in STAFF is marginally higher than these two algorithms in the period 3-4 minutes. However, it is still

Figure 8: Comparison of the STAFF framework against constant forgetting factor approaches.

Figure 9: Comparison of the STAFF framework against adaptive forgetting factor approaches.

Figure 10: Normalized RMS error in the frequency sensitivity estimates of the algorithms in different workload regions. Errors are normalized with respect to the best baseline approach (SEF+Offline_FS*).

within one standard deviation of the frequency sensitivity reference value. STAFF outperforms the baseline algorithms during all other intervals. In summary, for the entire 6-minute duration, the STAFF framework provides 6 $\times$, 2.2 $\times$, 3.2 $\times$, and 1.9 $\times$ better accuracy than the Offline_FS ($\lambda = 1$), Offline_FS ($\lambda = 0.96$), SEF with all features, and SEF with Offline_FS* algorithms, respectively.

6 CONCLUSION

In this paper, we present the STAFF framework that guarantees stability, performs online feature selection with linear complexity, and dynamically changes the forgetting factor. We evaluate STAFF by predicting the frequency sensitivity of a graphics unit in a commercial Intel platform. The proposed framework provides fast tracking with up to 6 $\times$ improvement in the prediction accuracy compared to existing state-of-the-art techniques.

Acknowledgements: This work was supported partially by Semiconductor Research Corporation (SRC) task 2721.001 and National Science Foundation (NSF) grant CNS-1526562.

REFERENCES

- [1] R. Z. Ayoub *et al.*, "OS-level Power Minimization under Tight Performance Constraints in General Purpose Systems," in *Proc. of the Intl. Symp. on Low-power Elec. and Design*, 2011, pp. 321–326.
- [2] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras, "Algorithmic Optimization of Thermal and Power Management for Heterogeneous Mobile Platforms," *IEEE*

- Trans. on Very Large Scale Integration Syst.*, vol. 26, no. 3, pp. 544–557, 2018.
- [3] P. Bogdan and R. Marculescu, "Non-stationary Traffic Analysis and Its Implications on Multicore Platform Design," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Syst.*, vol. 30, no. 4, pp. 508–519, 2011.
- [4] A. Das, G. V. Merrett, M. Tribastone, and B. M. Al-Hashimi, "Workload Change Point Detection for Runtime Thermal Management of Embedded Systems," *Trans. on Comp.-Aided Des. of Int. Circuits and Syst.*, vol. 35, no. 8, pp. 1358–1371, 2016.
- [5] B. Dietrich *et al.*, "LMS-based Low-complexity Game Workload Prediction for DVFS," in *Proc. of the Intl. Conf. on Comp. Design*, 2010, pp. 417–424.
- [6] T. Fortescue, L. S. Kershenbaum, and B. E. Ydstie, "Implementation of Self-Tuning Regulators with Variable Forgetting Factors," *Automatica*, vol. 17, no. 6, pp. 831–835, 1981.
- [7] U. Gupta *et al.*, "Adaptive Performance Prediction for Integrated GPUs," in *Proc. of the Intl. Conf. on Comp.-Aided Design*, 2016, p. 61.
- [8] J. D. Hamilton, *Time Series Analysis*. Princeton University Press, 1994, vol. 2.
- [9] V. Hanumaiah, D. Desai, B. Gaudette, C.-J. Wu, and S. Vrudhula, "STEAM: a Smart Temperature and Energy Aware Multicore Controller," *Trans. on Embedded Comp. Sys.*, vol. 13, no. 5s, p. 151, 2014.
- [10] Intel Corp. PRM for Open Source HD Graphics. <https://01.org/sites/default/files/documentation/intel-gfx-prm-osrc-skl-vol14-observability.pdf>.
- [11] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*. Springer, 2013, vol. 6.
- [12] R. G. Kim *et al.*, "Imitation Learning for Dynamic VFI Control in Large-Scale Manycore Systems," *IEEE Trans. on Very Large Scale Integration Syst.*, vol. 25, no. 9, pp. 2458–2471, 2017.
- [13] G. Kreisselmeier, "Stabilized Least-Squares Type Adaptive Identifiers," *Trans. on Automatic Control*, vol. 35, no. 3, pp. 306–310, 1990.
- [14] L. Ljung, "Characterization of the Concept of 'Persistently Exciting' in the Frequency Domain," *Report TFRT*, vol. 3038, 1971.
- [15] K. Ma, X. Wang, and Y. Wang, "DPPC: Dynamic Power Partitioning and Control for Improved Chip Multiprocessor Performance," *IEEE Trans. on Comp.*, vol. 63, no. 7, pp. 1736–1750, 2014.
- [16] J. Milek, *Stabilized Adaptive Forgetting in Recursive Parameter Estimation*. vdf Hochschulverlag AG, 1995, no. 4.
- [17] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra, "Power-Performance Modelling of Mobile Gaming Workloads on Heterogeneous MPSoCs," in *Proc. of the Design Autom. Conf.*, 2015, pp. 201:1–201:6.
- [18] A. H. Sayed, *Fundamentals of Adaptive Filtering*. John Wiley & Sons, 2003.
- [19] T. Song, D. Lo, and G. E. Suh, "Prediction-Guided Performance-Energy Trade-off with Continuous Run-Time Adaptation," in *Proc. of the Intl. Symp. on Low Power Elec. and Design*, 2016, pp. 224–229.
- [20] F. Souza and R. Araújo, "An Online Variable Selection Method Using Recursive Least Squares," in *Proc. of the Emerging Tech. & Factory Autom.*, 2012, pp. 1–8.
- [21] Statista. Worldwide mobile app revenues in 2015, 2016 and 2020. <https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/>.
- [22] B. Su *et al.*, "PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration," in *Proc. Intl. Symp. on Microarch.*, 2014, pp. 445–457.
- [23] C. S. Turner, "A Fast Binary Logarithm Algorithm," *Streamlining Digital Signal Processing: A Tricks of the Trade Guidebook*, pp. 281–283, 2012.