Learning-based Phase-aware Multi-core CPU Workload Forecasting

ERIKA S. ALCORTA and ANDREAS GERSTLAUER, The University of Texas at Austin, USA

Predicting workload behavior during workload execution is essential for dynamic resource optimization in multi-processor systems. Recent studies have proposed advanced machine learning techniques for dynamic workload prediction. Workload prediction can be cast as a time series forecasting problem. However, traditional forecasting models struggle to predict abrupt workload changes. These changes occur because workloads are known to go through phases. Prior work has investigated machine learning-based approaches for phase detection and prediction, but such approaches have not been studied in the context of dynamic workload forecasting. In this paper, we propose phase-aware CPU workload forecasting as a novel approach that applies long-term phase prediction to improve the accuracy of short-term workload forecasting. Phase-aware forecasting requires machine learning models for phase classification, phase prediction, and phase-based forecasting that have not been explored in this combination before. Furthermore, existing prediction approaches have only been studied in single-core settings. This work explores phase-aware workload forecasting with multi-threaded workloads running on multi-core systems. We propose different multi-core settings differentiated by the number of cores they access and whether they produce specialized or global outputs per core. We study various advanced machine learning models for phase classification, phase prediction, and phase-based forecasting in isolation and different combinations for each setting.

We apply our approach to forecasting of multi-threaded Parsec and SPEC workloads running on an 8-core Intel Core-i9 platform. Our results show that combining GMM clustering with LSTMs for phase prediction and phase-based forecasting yields the best phase-aware forecasting results. An approach that uses specialized models per core achieves an average error of 23% with up to 22% improvement in prediction accuracy compared to a phase-unaware setup.

Additional Key Words and Phrases: phase classification, multi-core workload forecasting, phase prediction, hardware counters

ACM Reference Format:

Erika S. Alcorta and Andreas Gerstlauer. 2022. Learning-based Phase-aware Multi-core CPU Workload Forecasting. 1, 1 (October 2022), 27 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Predicting future dynamic workload behavior is essential in optimizing hardware resources at runtime. For example, anticipating an application's memory-intensive periods can be leveraged for power management to reduce core frequency and voltage [29, 37]. Prediction of workload metrics such as CPI has also been exploited in a variety of other applications, including reduction of task interference in multi-tenant systems [27], task migration and scheduling [34], defending against side-channel attacks [31], and cache reconfiguration [50]. Predictions allow systems to behave proactively instead of reactively. It has been previously shown that proactive decisions can yield better results for a variety of optimization tasks [1]. However, they are challenging because they require predicting the future.

Looking at the past is often a reliable way of estimating the future. Program applications present variable workload behaviors throughout their execution, and many exhibit periodic trends or patterns. Workload prediction techniques exploit these characteristics to estimate future behaviors. Previous work in dynamic workload forecasting ranges

Authors' address: Erika S. Alcorta, esalcort@utexas.edu; Andreas Gerstlauer, gerstl@utexas.edu, The University of Texas at Austin, Austin, Texas, USA.

Manuscript submitted to ACM

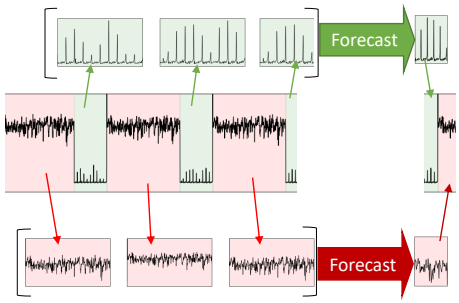


Fig. 1. Example of forecasting executions of two different phases of *nab*

from basic methods such as exponential averaging and history tables [14], to more advanced machine learning-based approaches such as recurrent neural networks (RNNs) [27]. Their objective is to minimize the forecasting error of periodically measured CPU workload metrics obtained at runtime using hardware counters, such as CPI. This periodic collection of metrics forms a time series. Hence, runtime workload behavior forecasting is formally a time series forecasting problem [11, 14, 37, 38, 48].

Time series forecasting techniques struggle to predict abrupt workload changes. Such changes occur because workloads are known to go through phases. Such phases typically also show repetitive patterns. Researchers have proposed methods to first detect and classify workloads into phases and then learn their long-term behavior to predict and anticipate phase changes [12, 22, 23, 44]. However, existing approaches for classification and prediction of phase patterns have only been applied in isolation or in limited combinations for specific optimization contexts, such as task mapping or thermal modeling [10, 21, 24, 34, 51]. Furthermore, only simple learning-based techniques such as decision trees have been explored.

In this paper, we study advanced learning-based methods that combine phase classification and phase prediction with workload forecasting. We propose a novel phase-aware workload forecasting approach that applies long-term phase classification and phase prediction to improve the accuracy of short-term workload forecasting. Figure 1 shows the intuition behind our approach. It shows a simple example of a hardware counter trace in the center of the figure. This workload is going through two phases, highlighted with green and red backgrounds. We partition the trace into two sub-traces based on its phases. The sub-traces are formed by concatenating consecutive counter samples of the same phase and represented by square brackets in the figure. We train a predictor that outputs a forecast of future counter samples specific to each phase, represented by the large green and red arrows. The forecasts belonging to a phase are thus only dependent on the history of that phase, and phase-based predictors are specialized to each phase to increase accuracy. Finally, with knowledge of future phase behavior, phase-specific forecasts are concatenated and assembled to reconstruct the forecast for the overall time series, depicted on the right side of Figure 1.

In our previous work [3], we introduced the concept of phase-aware forecasting using an oracle phase model. However, the definition of phases can significantly impact the accuracy of phase-aware workload predictors. In more recent work [2], we evaluated different combinations of phase classification and phase prediction models using advanced machine learning methods. Nonetheless, they have not been studied in combination with phase-aware forecasting. In this paper, we perform a comprehensive study of phase classification, phase prediction and phase-based forecasting in isolation and in different phase-aware forecasting combinations.

Moreover, previous approaches evaluated classification, prediction and forecasting models for single-threaded workloads running on a single core only. In this work, we further extend our contributions by studying multi-threaded workloads on multi-core platforms. Phase classification, phase prediction and workload forecasting pose various modeling challenges in multi-core systems. Phases of different threads executing on different cores can be distinct or shared. Furthermore, threads executing concurrently may be competing for the same resources, which can lead to interference that can affect hardware metrics such as CPI. We propose different model architectures and study various advanced machine learning techniques to learn such complex multi-threaded workload patterns. We study each architecture for different phase-aware forecasting tasks both in isolation and in different combinations.

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

(1) We propose three different modeling architectures for hardware counter-based runtime phase classification, phase prediction, and workload forecasting in multi-core systems, and we evaluate different advanced machine learning models for each architecture and task.
(2) We perform a comparative study to explore the best modeling architecture and combination of phase classification, phase prediction, and phase-based forecasting for phase-aware multi-core workload forecasting.
(3) We perform our study and evaluate our approach on multi-threaded workloads from SPEC and PARSEC benchmark suites running on a state-of-the-art 8-core Intel Core i9-9900k workstation. Results show that phase-aware forecasting using a local per-core definition of phases with LSTM-based long-term phase predictors and short-term forecasters improves prediction accuracy by up to 22% compared to a phase-unaware setup.

The remainder of this paper is organized as follows. We review the related work in the next section. Section 3 summarizes the phase-aware workload forecasting formulation. Section 4 describes the different modeling settings that we use for a multi-core environment and their implications on phase classification, phase prediction, and short-term forecasting. We present our experimental results in Section 5. Finally, Section 6 summarizes our work and discusses future work directions.

2 RELATED WORK

Predicting workload behaviors at runtime has been studied for a variety of hardware optimization objectives. In [28], [29], and [30], the authors study proactive dynamic voltage/frequency management (DVFS) and report improvements in the energy-delay product of multi-threaded applications when compared to reactive DVFS. They forecast CPI and the number of instructions of the next sampling period to select the lowest core frequency that satisfies performance loss constraints. In [37], instead of forecasting CPI, they directly forecast the power consumption at different voltage-frequency levels. The study in [11] proposed proactive thread migration to reduce temperature hotspots in multiprocessor SoCs. They use per-core temperature forecasts to migrate threads from cores that are anticipated to be hot to cool ones. Another proactive optimization consists of anticipating unused cache ways and shutting them down to reduce power consumption. W. Zhang et al. [50] use phase prediction and show that more accurate prediction results in lower average cache sizes. Preventing side channel attacks is another application of runtime workload prediction. The study in [31] proposes to monitor shared resources such as caches and floating point units with hardware counters, and schedule tasks predicted to have similar resource usage into separate cores to prevent side channel attacks.

In the rest of this section, we discuss prior work related to phase detection, phase prediction and workload forecasting models.

The set of inputs and granularity that researchers have used to define phases is very diverse. Some researchers aim at finding similar blocks of code [6, 8, 13, 20, 25, 35, 39–41, 50], while others focus on execution time workload metrics [10, 17, 23, 44]. Most prior work in phase classification for parallel applications focused on finding similar blocks of code [6, 35, 39] because such features are not affected by executions of other threads. However, finding similar blocks of code typically requires either access to the source code [8, 41, 50] or augmenting the hardware [13, 20, 25]. For workload forecasting at runtime, we aim to rely on methods that only use standard hardware counters as inputs for phase classification.

In addition to exploring different inputs, various clustering methods have been explored to define phases. Some studies have looked at methods that can perform clustering in an incremental and online fashion [6, 13, 20, 35, 39, 40, 50] while others propose iterative clustering methods [8, 10, 23, 41]. In [17], a pre-defined static set of value ranges are used to bin workloads based on their memory boundedness. The ranges are relevant to dynamic power management (DPM) as their target use case and require a static definition. A hierarchical, multi-level phase approach is presented in [50] and [23]. In [50], two separate table-based classifiers are used with different granularities to define fine and coarse phases. In [23], k-means is used to generate one sub-phase per sample, and the sub-phases are then grouped into fixed-size windows to define phases with a second instance of k-means. In our prior work [2], we explored multiple classification methods for single-threaded applications running on one core, where we adapted phase classification methods previously used with other inputs to support hardware counters and compared them to existing counter-based solutions both in isolation and in combination with multiple phase predictors. In this paper, we focus on the two best-performing clustering methods. Furthermore, existing approaches for phase classification and prediction have been mostly studied in single-threaded or single-core contexts only. The work in [39] extends the classifier from [40] to study and compare phase behavior in serial (SPEC) and parallel (Parsec) workloads. Similarly, [35] extends the classifier from [20] with a per-thread table of instruction type vectors (ITVs), while [6] samples the instruction pointer on a per-thread basis. All these approaches apply clustering and prediction methods separately and independently to each thread, which ignores interactions among threads in multi-core systems. In this paper, we extend our prior work from [2] to explore various multi-core scenarios that can account for interference and interaction among multi-threaded workloads.

Classified phases are used by phase prediction methods to learn patterns between them and foretell future phase behavior. Prior research has followed multiple strategies to predict upcoming phases at runtime. Traditionally, table-based predictors like global history tables (GHTs) [17] and Markov chain tables [6, 25, 50] were used. Most of these predictors foretell the phase of the following interval. Many researchers have noticed that the same phase may last multiple intervals. Consequently, previous work has also proposed either predicting the duration of phases [6, 42, 50] and the phase ID of the next change [6, 50], or predicting multiple future intervals instead of just one [8]. Predicting the exact duration of a phase is a challenging task. Chang et al. [6] approximated the duration of phases into ranges, limiting the precision of a prediction of when a change will happen. Srinivasan et al. [42] proposed to use a linear adaptive filter that learns and predicts the duration of phases online. Other researchers have proposed predicting the phase of a window of upcoming sample intervals instead. For example, in [8] a decision tree predictor is trained offline to predict the phase labels of the following 30 intervals with high accuracy. Each of these phase predictors has been evaluated with only a single classification scheme. In our prior work [2], we evaluated classifiers and predictors in multiple combinations and showed that a predictor's accuracy is highly dependent on the quality of the classified phases. Additionally, we investigated other more advanced machine learning-based prediction techniques that were not evaluated for these tasks in the past. Existing work, however, did not apply phase classification and prediction in combination with workload forecasting. This paper expands on our prior studies and evaluates different phase classifiers

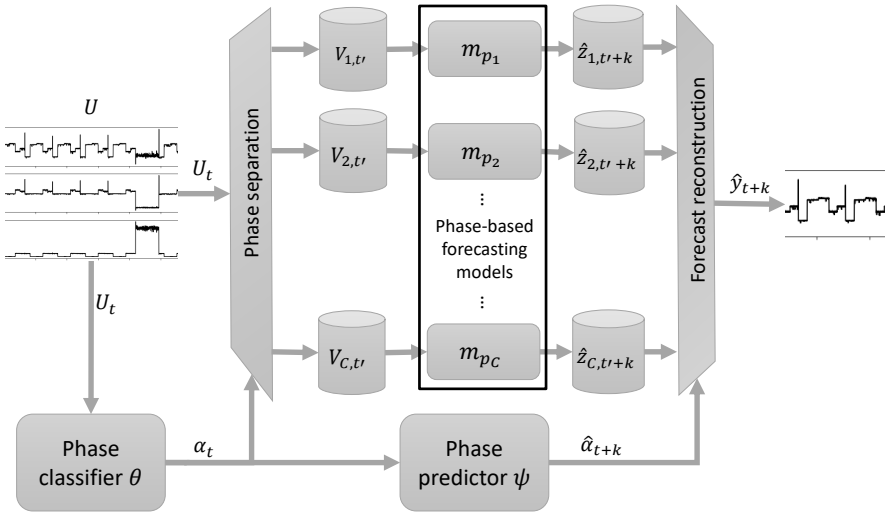


Fig. 2. Phase-aware workload forecasting overview.

and predictors for use with phase-aware forecasting. Our goal is to find phases whose interactions are easy to predict while also capturing intra-phase behaviors that specialized phase-based workload forecasters can in turn easily learn.

Early studies in forecasting dynamic workload metrics proposed basic statistical and table-based predictors. Duesterwald et al. [14] compared a last-value predictor with exponentially weighted moving average (EWMA) and history predictors to forecast instructions per cycle (IPC), and L1D cache misses. The history table predictor resulted in the lowest mean absolute error (MAE). Another study [38] evaluated linear regressors to forecast IPC and showed that they have a lower MAE than the last-value predictor. Kalman filters have been recently used in the context of CPU workload prediction [28] by predicting cycles per instruction (CPI) to optimize dynamic energy management. Advanced machine learning techniques have been more accurate than traditional predictors. Zaman et al. [48] found that an SVM regressor results in the lowest MAE when forecasting various performance counters. They compared an SVM against last-value, history table, and ARMA predictors. With the recent popularity of RNNs, a later study [27] investigated the design space of LSTMs to forecast IPC and other CPU metrics of workloads when they are co-allocated with other tasks in data centers. They compared LSTMs against linear regression and MLPs, concluding that LSTMs result in the highest coefficient of determination (R^2) scores. None of the existing works has considered the impact of phases on workload forecasting, however. In [3], we demonstrated that short-term workload forecasting models struggle to learn long-term patterns and proposed phase-aware workload forecasting to address this problem. We evaluated four representative time series forecasting techniques for single-threaded workloads in their phase-unaware and phase-aware settings, where the phase-aware setting used an oracle for phase classification and prediction. In this paper, we study the full phase-aware forecasting problem by replacing the oracles with learning-based phase classification and prediction models. Furthermore, we extend our prior work to different multi-core forecasting settings, which have not been explored before. Some prior approaches have studied interference prediction in multi-core platforms [27, 36], but their focus is on predicting co-location effects and not on forecasting of future workload behavior.

3 OVERVIEW

In this section, we summarize the formal definition of phase-aware workload forecasting. Figure 2 shows an overview of our approach.

The input is a multivariate time series, $U \in \mathbb{R}^{T \times M}$, composed out of T observations U_t of M sampled system variables. In our case, M is the number of hardware counters. The output is a forecast of future workload samples $\hat{y}_{t+k}$. We are interested in predicting one of the M variables, $y \in U$. The observation of this variable at time t, $1 \leq t \leq T$, is denoted as y_t. We predict k future values of y at time t, $(\hat{y}_{t+1}, ..., \hat{y}_{t+k})$, using only past observations $U_i, i \leq t$, where k is the forecast horizon. The rest of the figure shows our proposed phase-aware forecasting process. It internally consists of three high-level stages: (1) phase classification and separation, (2) phase prediction, and (3) phase-based forecasting and forecast reconstruction. In the following, we formalize each step in detail.

3.1 Phase Classification and Separation

A phase classifier, Θ, maps each sample, U_t, to a phase $\alpha_t \in 1, 2, ..., C$:

$$\alpha_t = \Theta(U_t). \tag{1}$$

The samples of U that share the same phase, c, are concatenated into a single vector. In total, there are C disjoint time series V_c, defined as follows:

$$V_c = \big(U_t | \alpha_t = c\big), \alpha_t \in 1, 2, ..., C \tag{2}$$

with observations $V_{c,t'}$, where t' represents the mapping of original observations U_t into a new time dimension t' for each series. Note that the following conditions must be true:

$$U = \bigcup_{c=1}^{C} V_c, \text{and } \forall i \neq j : V_i \bigcap V_j = \emptyset \tag{3}$$

3.2 Phase Prediction

A phase predictor, ψ, further takes outputs α_t from the phase classifier to predict k future phases at time t based on the history of d previous phases. In other words:

$$(\hat{\alpha}_{t+1}, ..., \hat{\alpha}_{t+k}) = \psi\big((\alpha_{t-d+1}, ..., \alpha_t)\big). \tag{4}$$

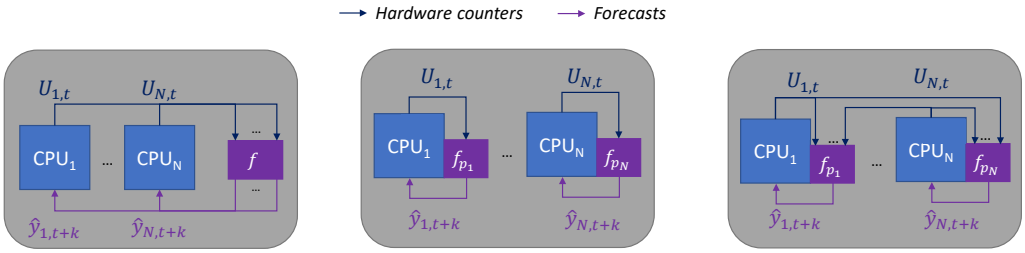
3.3 Phase-Based Forecasting

Each new time series, V_c is processed by a different model m_{p_c}, where p_c corresponds to a set of trained parameters. Note that all models use the same model architecture and predictor type, i.e. they differ only in trained parameters. Each model uses h samples to forecast k values of a predicted variable $z_c \in V_c$ as follows:

$$(\hat{z}_{c,t'+1}, ..., \hat{z}_{c,t'+k}) = m_{p_c}\big((V_{c,t'-h+1}, ..., V_{c,t'})\big). \tag{5}$$

Finally, since the forecasting models m_{p_c} are unaware of their interactions and relationships to the original time series, we use the outputs $\hat{\alpha}_t$ from the phase predictor to select those values of $\hat{z}_{c,t'}$ that should be output as overall forecast $\hat{y}_t$. Formally:

$$(\hat{y}_{t+1}, ..., \hat{y}_{t+k}) = (\hat{z}_{\hat{\alpha}_{t+1},t'+1}, ..., \hat{z}_{\hat{\alpha}_{t+k},t'+k}). \tag{6}$$



(a) Global (G) (b) Local (L) (c) Distributed (D)

Fig. 3. Multicore workload forecasting system settings.

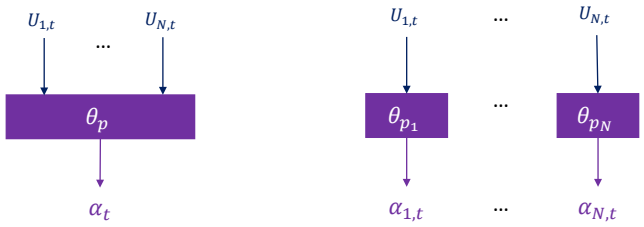
4 MULTI-CORE WORKLOAD FORECASTING

There are numerous ways to design the architecture of per-core workload forecasting predictors in a multi-core system. It can be designed with one dedicated predictor per core or a single global predictor for all cores. Furthermore, the inputs of per-core predictors may either remain local, i.e., hardware counters data from a single core; or be distributed, i.e., use data from multiple cores. In this work, we investigate three multi-core workload forecasting architectures, shown in Fig. 3, where N represents the number of cores in the system, f is a predictor that learns from the history of hardware counters sampled during runtime and maps it to workload forecasts, and p_i is a set of trainable parameters of f. The global architecture (G), on the left side of the figure, requires access to data from all cores and simultaneously outputs the forecasts of each core. The local architecture (L), shown in Fig. 3b, consists of multiple predictors that target one core and access input data from that core only. On the right of the figure, the distributed architecture (D) is designed to have one predictor per core, where each predictor accesses data from all cores while producing forecasts specialized for one core. We also study two variants of L and D: local shared (L-S) and distributed shared (D-S). These variants enable sharing of knowledge between predictors by sharing the same set of trainable parameters. In other words, for L-S and D-S, all models are trained with the same training set that consists of data and labels from all cores, and there is a single set of trainable parameters, ρ, such that $p_i = \rho \forall i \in 1, ..., N$. The implementation of these variants requires considering how to share and update the trainable parameters during runtime.

We assume that predictors specific to each workload are trained at runtime. Since applications and their inputs are generally not known at design time, models will need to learn application-specific behavior during runtime. As such, the efficiency of training overhead as well as data and parameter sharing must be considered to deploy an online learning approach. Note that if the set of applications is restricted and known at design time, e.g. in an embedded context, models can potentially be trained offline on as subset of inputs to predict application behavior for arbitrary inputs at runtime. In addition to training, each architecture has different implications from the modeling and implementation perspectives in terms of inference overhead. In the remainder of this paper, however, we primarily focus on the machine learning and accuracy aspects for each stage presented in Section 3.

4.1 Phase Classification

Phase classification is the first stage of our phase-aware workload forecasting approach. It maps a trace of hardware counters to a series of discrete labels, i.e., phases. We observe that the definition of phases in a multi-core system can be either global or local as shown in Figure 4. Note that phase classification settings have a direct relationship with



(a) Global (G) (b) Local (L)

Fig. 4. Multi-core phase classification settings.

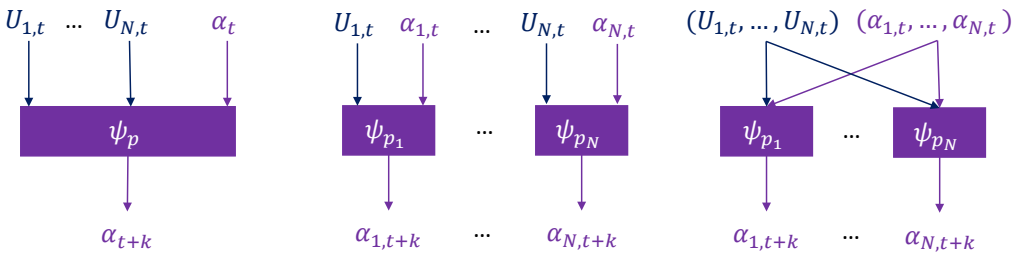
the multi-core settings shown in Figure 3, and that there is no distributed (D) definition of phase classification. This is because workload phase classification is an unsupervised learning problem. Therefore, a D setting would fundamentally produce the same results as G with N identical copies instead of one.

In a multi-core setting, the G classifier produces one phase label per time step. Therefore, it results in a single-dimensional series, α_t. By contrast, a local (L) definition of phases results in one series of phases per core, or a two-dimensional series, $\alpha_{n,t}$, where $n \in 1, ..., N$ is the core ID. Additionally, as mentioned earlier, the L setting has a variant, L-S, which shares trainable parameters across all cores. In other words, the L-S phases are defined as $\forall n \in 1, ..., N : \alpha_{n,t} = \Theta_\rho(U_{n,t})$, where ρ is the single set of trainable parameters for the classifier Θ.

The different definitions of phases that we propose (G, L, and L-S), may be suited for different types of workloads. Therefore, they have distinct impacts on the overall accuracy of phase-aware forecasting. The G definition is better suited for workloads whose behaviors across cores are mostly synchronized. It is also suited for a broad, system definition of phases, e.g., to define which cores have a memory-intensive period. This global definition of phases implies that phase transitions may be triggered even when the workload behavior of one or more cores did not change. In such scenarios, a local definition of phases may be more accurate. L is better suited for workloads whose phase behavior across cores is independent and asynchronous. This classifier must learn local phase behaviors without explicit information about the workload in other cores. It is also the most specialized classifier per core, but it may be prone to overfitting, i.e., finding more phases than necessary. Finally, L-S enables sharing knowledge of phases across cores. It is better suited for workloads whose threads run the same or remarkably similar operations on different cores. This definition of phases allows cores to classify phases that they have not encountered before when at least one other core has.

In our most recent work [2], we studied different machine learning-based phase classification and prediction models in isolation and in different combinations on a single core and compared them against a simple table-based approach that uses a leader-follower clustering algorithm. In this work, we selected the two best performing learning-based classification methods from our prior work to study in a multi-core setting.

Two-level k-means. This method is based on [23]. The workloads are classified into sub-phases and phases. The sub-phases are defined per sample using k-means clustering. Then, the phases are defined by a second instance of k-means that clusters frequency vectors of sub-phases. The frequency vectors are generated by counting the number of times a sub-phase is encountered in fixed size windows with size W. We use the same maximum number of sub-phases as in [23], $N_1 = 10$. We select the hyperparameters W and C per benchmark by optimizing for the average duration of



(a) Global (G) (b) Local (L) (c) Distributed (D)

Fig. 5. Multi-core phase prediction settings.

phases and for the average corrected coefficient of variation (CCoV), as defined in [40]. In the rest of this paper, we refer to this classifier as *2kmeans*. When we evaluated *2kmeans* in isolation in [2], its performance in terms of CCoV was poor compared to other classification methods. However, when evaluated in combination with different phase predictors, it outperformed all other methods in thirteen out of fifteen phase predictors. The most remarkable difference between *2kmeans* and all other classifiers is its two-level clustering approach, which defines phases using a window of samples instead of a single sample. The sub-phases acted as a filter for the main definition of phases.

Gaussian Mixture Model. This clustering method was studied for program phase definition in the past [8], taking as inputs annotated labels for program branches. In [2], we adopted the clustering method but used hardware counter data for its inputs instead. GMM is an iterative clustering algorithm that finds its clusters in an analogous way as k-means. It assumes that combining different Gaussian models creates the data points in a cluster. Therefore, in addition to considering the cluster means, it considers their variances when assigning samples to clusters. When we evaluated this classification method in isolation [2], it was the best performing in terms of CCoV. Additionally, it outperformed the other classification methods for two of the phase predictors. In this work, we evaluate this method with an additional modification: we filter the data before clustering, giving the clustering method a further advantage similar to *2kmeans*, which also performs 2-level clustering by filtering data before defining phases. Like *2kmeans*, we select the filter size and number of phases per benchmark by optimizing for the average duration of phases and the average CCoV. We refer to this classification method as *fgmm* in the rest of this paper.

4.2 Phase Prediction

Phase prediction is the second stage in our phase-aware workload forecasting approach. A phase predictor takes the history of phase labels at the output of the phase classifier or a sub-trace of past hardware counter samples to predict the phase label of one or more future samples. We present the different phase prediction settings of a multi-core system in Figure 5. In a multi-core system, the choice of phase prediction setting depends on the multi-core phase definition. The G phase predictor is a single model dedicated to predicting G phases. Therefore, its output is a single label per timestep for the entire system, α_{t+k}, $k \geq 1$. By contrast, L and D phase predictors consist of different models, Ψ_{p_n}, each dedicated to learning different phase patterns $\alpha_{n,t+k}$ per core n. L relies solely on the data of one core to learn phase

patterns, whereas D allows explicit interference-aware predictions by accessing data from all the cores. Additionally, phase prediction supports both variants L-S and D-S, where trainable parameters are shared across all cores.

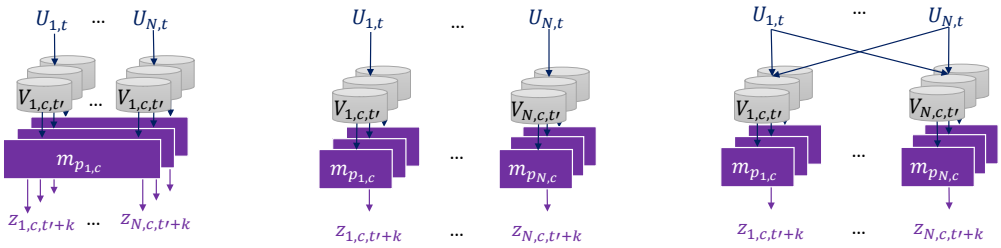
We study two types of phase predictors, window-based prediction and phase-change prediction [2], including how they adapt to each multi-core setting.

4.2.1 Window-Based Phase Prediction. Window-based prediction aims to accurately determine the phase label of the immediately upcoming sample window of size k. These predictors can either look at the history of phases or the history of pre-classification data, in our case, hardware counters. Some workload phases last for very long periods, and using the history of phase labels per sample may not be helpful information to the predictor. Therefore, all our window predictors use the trace of hardware counters as inputs. The assumption is that the predictors will learn the relationship between the counters' short-term variations and the upcoming phases. Note that while window-based predictions do not directly utilize the output of phase classifiers, phase classification ground truth is needed to train the predictors. Formally, the window-based phase prediction problem is a supervised learning problem with inputs $(U_{n,t-d+1}, ..., U_{n,t})$, and outputs $(\alpha_{t+1}, ..., \alpha_{t+k})$ for a global (G) window-based predictor or $(\alpha_{n,t+1}, ..., \alpha_{n,t+k})$ for local (L) and distributed (D) predictors, where d is the input window size, k is the output window size, and n is the core ID.

In [2], we studied three different models for window-based predictors: decision trees (DT), support vector machines (SVM), and long short-term memory (LSTM). We found remarkably similar accuracies, with an LSTM delivering the best accuracy and a DT exhibiting the lowest computational inference complexity. Therefore, we investigate all window-based multi-core prediction settings with LSTM and DT models in this work. The inputs to the DT models consist of a single vector containing the values of the hardware counters at different time steps. On the other hand, the inputs to the LSTM models are given as a sequence of d hardware counter vectors. The LSTM encodes the sequence into a single vector that is then used by a fully connected output layer to produce the phase predictions.

4.2.2 Phase Change Prediction. Phases typically last multiple consecutive intervals. Rather than predicting phase labels at each interval, phase change prediction focuses on learning and predicting phase transitions. To do so, a series of phases α_t is transformed using run-length encoding to generate a new series, Q_j. The new series consists of pairs $Q_j = (\alpha_j, r_j)$ of phase labels α_j and their duration r_j. At each detected phase transition, j, we use a next phase prediction model to determine the label of the next phase, $\hat{\alpha}_{j+1}$, and a phase duration prediction model to estimate how far into the future the transition to this next phase will happen, i.e., the duration of the current phase α_j, $\hat{r}_j$. Note that for an L or L-S definition of phases, the phase transitions in each core may be asynchronous. As such, each core n has its own set of transitions $Q_{n,j} = (\alpha_{n,j}, r_{n,j})$. We support all three multi-core settings, G, L, and D, for phase change prediction. A G phase change predictor is a combination of G next phase prediction and G phase duration prediction, which look at the history of G phase transitions. This applies to the L predictors as well, where each predictor accesses its local history. However, while we support D next phase prediction, we find no value in a duration predictor accessing a history of phase duration in other cores. Instead, a D phase change predictor consists of a D next phase predictor and an L phase duration predictor. Furthermore, note that phase change predictors do not consider counter histories $U_{n,t}$ as inputs.

Next Phase Prediction. Next phase predictors look at the history of distinct phases as a sequence α_j to predict the label of the next phase $\hat{\alpha}_{j+1}$. In [2], we showed that table-based predictors could trivially solve this problem for workloads with uniform phase patterns that repeat without variations. However, more robust predictors that can generalize the input data can yield better results when the pattern is not entirely uniform. In our previous work [2], we performed an extensive analysis of existing work in phase classification and prediction, and compared it against learning-based



(a) Global (G) (b) Local (L) (c) Distributed (D)

Fig. 6. Multi-core phase-aware short-term forecasting modeling settings

methods such as SVMs, MLPs, and LSTMs. The best performing combination of phase classification and phase prediction included an SVM as next phase prediction. Therefore, in this work, we investigate all multi-core next phase prediction settings with an SVM model. The SVM is configured as a classifier whose input is a vector of one-hot encoded values corresponding to a fixed-size history of phase transition labels.

Phase Duration Prediction. Phase duration prediction aims at accurately determining how long a phase will last. In the past, phase duration has been framed as either a categorical problem, i.e., predicting whether the duration is within a pre-defined set of ranges, or a regression problem, i.e., predicting the exact duration of a phase. The former loses precision when estimating phase changes. Therefore, we frame phase duration prediction as a regression problem. We have considered two types of inputs for this task. The first is to use the duration history of past phases with the same phase label, α_j, $(r_i | i < j, \alpha_i = \alpha_j)$. The second is using the history of phase labels and their duration, $Q_i, i < j$. The best choice depends on the predictor and the workload. We generally find that the relationships between durations of the same phase are linear. Therefore, linear predictors are better suited for these inputs. Some workloads may have relationships between the duration of different phases, and in general, these inter-phase relationships may not be linear. Our previous study's best-performing phase classification and prediction combination included an SVM as the phase duration predictor. Therefore, we investigate all multi-core phase duration settings with an SVM model in this work. This SVM model is configured as a regressor. The duration history inputs, r_i, are encoded as integer values, while the history of phase transition labels are one-hot encoded.

4.3 Phase-based Forecasting

Phase-based forecasting is the last stage in our phase-aware approach. It dedicates one forecasting model per phase to map the history of hardware counter data within a single phase to a fixed-size window of workload forecasts specialized in that phase. We investigate all three multi-core settings, global (G), local (L), and distributed (D), for phase-based forecasting (Figure 6). The choice of phase-based forecasting setting depends on the definition of phases. A G phase-based forecasting architecture is tied to a G definition of phases. It consists of C models that forecast N values per time step. By contrast, L and L-S definition of phases can only be combined with L and D forecasters. They consist of up to $C \times N$ models, each dedicated to learning workload patterns per phase in each core. L models are designed to access hardware counter data from a single core, while D models are explicitly interference-aware and use data from all cores as inputs. We also study shared variants L-S and D-S. They share trainable parameters across cores. As such, they conceptually have C different models, which are instantiated in and replicated per core. In other words, for L-S and

D-S, there exist C sets of trainable parameters ρ_c such that $p_{n,c} = \rho_c \forall c \in 1, ..., C; \forall n \in 1, ..., N$. These variants are better suited for the L-S classification, where all cores share the same definition of phases. By contrast, L and D forecasters are better suited for an L phase classification.

Each phase-based forecasting model has a different definition of the set of disjoint time series, V, as defined in Equation 2. For the G models, the resulting number of disjoint time series is C, and each includes data from all cores: $V_c = \big((U_{1,t}, ..., U_{N,t}) | \alpha_t = c\big)$. The L and D models have up to $N \times C$ disjoint series, $V_{n,c}$, with different definitions. For the L models, the series are defined as using data from a single core: $V_{n,c} = (U_{n,t} | \alpha_{n,t} = c)$. On the other hand, the distributed models access data from all cores, i.e., $V_{n,c} = \big((U_{1,t}, ..., U_{N,t}) | \alpha_{n,t} = c\big)$.

In general, per-core workload forecasting is a multi-dimensional regression estimation problem, where the dependent variable is a vector, $y_t \in \mathbb{R}^N$. L and D divide this problem into multiple one-dimensional problems. Therefore, they cannot learn the inter-dependent relationships of their outputs. By contrast, G forecasting models are designed to learn such relationships by minimizing a multi-dimensional output function. For example, consider an artificial neural network (ANN) for our per-core workload forecasting problem. In this scenario, the output layer of L and D models is a single neuron, whereas the output layer of G models has N neurons. The hidden layers of G models are updated with gradients estimated from all outputs.

In [3], we explored four short-term workload forecasting techniques in their phase-aware and phase-unaware settings for a single core. We showed that the phase-aware forecasting models yield lower forecasting error for workloads that go through distinct phases, measured with the mean absolute percentage error (MAPE) metric. The models that we explored were support vector machines (SVMs), dynamic linear models (DLMs), long short-term memory (LSTM), and matrix profile (MP). The best performing phase-aware and -unaware models were LSTMs and SVMs, respectively. Therefore, we use these two models to study short-term multi-core forecasting in this work. Both LSTMs and SVMs are configured as regressors. The input to the SVM is a single vector in which all the hardware counter values of different time steps are concatenated. The inputs to the LSTM models consist of a sequence of h vectors of concatenated hardware counter values. The LSTM encodes this sequence into a vector that a fully connected layer uses to compute the forecasts.

5 EXPERIMENTS AND RESULTS

This section presents the evaluation of each component of our phase-aware approach and its various settings. We study multiple combinations of G, L, and D models and their shared variants for phase classification, phase prediction, and phase-based forecasting. When evaluated together, we use the notation (C, P, F), where C is the phase classification, P is the phase prediction, and F is the phase-based forecasting setting. We study global (G) phase classification, phase prediction and phase-based forecasting in a (G, G, G) combination. For local (L) phase classification, we assess all combinations of L and D phase prediction and phase-based forecasting, i.e., (L, L, L), (L, L, D), (L, D, L), and (L, D, D). Finally, for local phase classification with shared parameters (L-S), we evaluate it in all combinations with L-S and D-S phase prediction and phase-based forecasting, i.e., (L-S, L-S, L-S), (L-S, L-S, D-S), (L-S, D-S, L-S), and (L-S, D-S, D-S).

To generate the workload traces, we executed multi-threaded workloads from the PARSEC-3.0 and SPEC CPU 2017 suites. The PARSEC applications run with their *native* input set and the SPEC applications use the reference input set. We selected a subset of workloads whose traces are long enough to contain sufficient data for training and validation. The workloads that we study from each suite and their respective number of samples are shown in Table 1.

Our target platform is a desktop-class Intel Core i9-9900K running Debian 9.13 with the *ondemand* Linux governor enabled. We execute the workloads on four cores and collect hardware counters from the performance monitoring unit

Table 1. Benchmarks.

SPEC 2017		Parsec-3.0	
Benchmark	Samples	Benchmark	Samples
cactuBSSN	47,411	blackscholes	7,577
fotonik3d	55,100	bodytrack	8,744
imagick	120,658	canneal	10,870
nab	71,769	facesim	18,607
wrf	91,523	ferret	17,769
xz	29,163	fluidanimate	18,701
		freqmine	26,268
		raytrace	10,016
		streamcluster	29,424
		swaptions	12,520
		vips	7,315

Table 2. Hardware counters.

Fixed counters	Variable counters
INST_RETIRED.ANY	L2_RQSTS.MISS
CPU_CLK_UNHALTED.THREAD	OFFCORE_REQUESTS.DEMAND_DATA_RD
	BR_MISP_RETIRED.ALL_BRANCHES
	FP_ARITH_INST_RETIRED.SCALAR_DOUBLE

(PMU) every 10ms using Intel's EMON tool. The Intel platform can sample 4 fixed counters and up to 4 variable counters simultaneously per core. We select four variable counters to characterize the workload by its memory boundedness (L2 misses and main memory read accesses), control flow predictability (mispredicted branch instructions), and operation mix (floating-point operations). The exact names of hardware counters used are listed and summarized in Table 2. Normalizing the PMU counters to the number of instructions yielded more accurate forecasting results. We also found that reducing dimensionality with principal component analysis (PCA) improves the performance of phase-based forecasting and window-based predictors. We use standard scaling to map the values of each counter to a range between 0 and 1, and then select the two components with the highest variance using PCA.

We evaluate the ability of our models to generalize to unseen samples and perform an offline study where we split each time series into 70% for training and 30% for testing and train separate models with data for each benchmark.

CPI is used as the variable of interest, y, to compare our models. We use mean absolute percentage error (MAPE) to measure accuracy of phase-based and phase-aware forecasting. We set a fixed forecast horizon of all models of $k = 20$. With this, we compute a separate $MAPE_i$ for every step $1 \leq i \leq k$ in the forecast horizon as follows:

$$MAPE_i = \frac{100\%}{n} \sum_{t=1}^{n-k} \left(\frac{|y_{t+i} - \hat{y}_{t+i}|}{y_{t+i}} \right) \tag{7}$$

When comparing different models, we look at the average MAPE ($AMAPE$) of all $k = 20$ predictions: $AMAPE = \frac{1}{k} \sum_{i=1}^{k} MAPE_i$. Similarly, we use average accuracy of all $k = 20$ predictions to evaluate window-based phase prediction. Note that in this work, we focus our exploration on a generic and application-independent notion of forecast error. The impact of and tolerance to this error depends on the targeted runtime optimization (e.g., maximizing performance with DVFS or reducing interference with task migration). We leave such investigations for future work.

In addition to forecast error, we measured the inference time corresponding to one prediction step with the purpose of comparing model complexities. Forecasting models are implemented and trained using *Keras* [9] for all LSTM models, *scikit-learn* [32] for k-means and GMM clustering as well as phase prediction SVM and DT, and *msvr* [5] for multi-output SVM regressors. We use Python's *time* standard library to measure inference times on our target platform. We also profiled the memory requirements and runtime of our offline training setup with a fixed number of samples. We used *mprof* [15] for memory profiling. Note that while inference time is measured per sample, training times are reported for the whole training set.

5.1 Phase Classification

We explore two different clustering techniques, 2kmeans based on [23] and our proposed fgmm. Both methods require two hyperparameters as inputs, the number of phases and the filter size. Our goal for phase classification is to find long-term workload phases whose intra-phase patterns can be learned by short-term phase-based forecasting. We employ a hyperparameter selection methodology with two optimization objectives: (1) minimize the corrected coefficient of variation (CCoV) and (2) maximize average phase duration. The CCoV metric was introduced in previous work [40] as an extension of the coefficient of variation (CoV), computed as variance divided by mean, to penalize phases that last a single interval. To calculate the CCoV, samples whose phase labels are different from their neighbors' phases are assignd a "virtual phase" label. Then, the CoV of the whole program is computed (as if the whole program was a single phase), and this value is assigned to the virtual phases when computing the per-phase CoV. In addition to minimizing phase variations, we are interested in finding stable long-term patterns. We use the second objective to prevent classification overfitting, where the classifier defines more phases than necessary.

For hyperparameter tuning, we execute the phase clustering techniques with multiple combinations of phase count and filter size. We then obtain the Pareto-optimal data points that maximize the average phase duration and minimize the CCoV. Next, we sort the Pareto-optimal data points in descending order of duration and apply an elbow method to select the optimized parameters, i.e. we iteratively go through each hyperparameter's data points until the CCoV is no longer decreased by 5% or more. We follow this process for each clustering technique (2kmeans and fgmm) and each multi-core phase classification setting (G, L, and L-S). The outcomes are six different definitions of phases per benchmark. Table 3 shows the resulting number of unique phases, average duration, and average CCoV for each definition.

When comparing the average CCoV across benchmarks for each multi-core setting and classifier, fgmm yields lower values than 2kmeans for all three settings, with G providing the lowest CCoV. We observe that *cactuBSSN* and *xz* have significantly higher CCoV results in some settings when compared to the other benchmarks. The phases that 2kmeans defines for *cactuBSSN* have very high CCoV because this benchmark has short regions where the CPI is two orders of magnitude higher than the rest of the trace. The 2kmeans filter is unable to account for this difference in magnitude and misses these transitions. In case of *xz*, the traces in all cores have very high-frequency CPI variations. Additionally, there is a region with an evident change in CPI behavior for all cores. However, most classifiers failed to detect this change because the difference in hardware counter values was not as significant in all cores.

In terms of phase duration, 2kmeans finds, on average, longer phases than fgmm for G and L. However, L-S has the longest duration of phases on average for both fgmm and 2kmeans, with fgmm finding the longest phases. Both classifiers tend to find very long phases in the PARSEC benchmark suite. They also find very long phases in *imagick*, *fotonik3d*, *wrf*, and *xz* for some settings. Therefore, there is a limited number of phase transitions in these multi-threaded workloads, especially compared to their single-threaded variants that we previously studied.

Table 3. Phase Classification Results

		G			L			L-S		
		CCoV	C	dur.	CCoV	C	dur.	CCoV	C	dur.
cactuBSSN	2kmeans	6.4	6	74	9.2	2	130	7.2	5	38
	fgmm	0.5	5	16	0.8	3	28	0.9	3	34
fotonik3d	2kmeans	0.6	2	27549	0.6	2	17218	0.7	2	44
	fgmm	0.5	3	44	0.4	3	16	0.6	3	17218
imagick	2kmeans	0.2	5	4160	0.2	4	9896	0.1	5	4841
	fgmm	0.1	6	2741	0.1	5	3086	0.3	3	8658
nab	2kmeans	1.1	5	98	1.2	2	160	1.1	6	92
	fgmm	1.0	4	71	0.6	5	71	1.7	2	771
wrf	2kmeans	0.3	2	854	0.3	2	11588	0.3	2	188
	fgmm	0.3	3	29	0.3	2	201	0.3	3	37
xz	2kmeans	2.6	5	149	4.3	6	54	8.3	2	12758
	fgmm	2.4	2	9720	6.6	3	447	8.2	2	10935
blackscholes	2kmeans	0.3	6	686	0.2	6	223	0.4	4	1353
	fgmm	0.2	6	755	0.2	3	1133	0.2	5	1021
bodytrack	2kmeans	0.9	2	1248	0.9	2	775	0.8	2	203
	fgmm	1.0	3	672	1.0	2	1109	1.0	2	1179
canneal	2kmeans	0.1	5	1358	0.2	3	1954	0.2	3	3849
	fgmm	0.1	6	1552	0.1	3	1864	0.1	6	1554
facesim	2kmeans	0.2	6	89	0.2	4	375	0.2	6	2931
	fgmm	0.1	5	96	0.2	3	1709	0.3	2	12404
ferret	2kmeans	0.6	4	203	0.7	2	264	0.6	3	1100
	fgmm	0.6	2	5922	0.6	2	1630	0.7	2	14806
fluidanimate	2kmeans	0.3	6	2077	0.3	3	3134	0.3	3	4804
	fgmm	0.4	4	3739	0.2	3	2108	0.5	2	15583
freqmine	2kmeans	0.3	6	772	0.2	4	184	0.3	3	2725
	fgmm	0.4	3	4377	0.2	3	2643	0.5	2	6785
raytrace	2kmeans	0.6	2	1112	0.2	3	1354	0.6	2	4798
	fgmm	0.2	6	667	0.2	3	1232	0.5	2	8346
streamcluster	2kmeans	0.2	4	1730	0.5	2	9632	0.5	2	13135
	fgmm	0.3	3	2451	0.1	3	2396	0.1	4	2941
swaptions	2kmeans	0.1	3	131	0.1	5	98	0.2	2	12519
	fgmm	0.1	2	4172	0.1	2	2758	0.1	2	10432
vips	2kmeans	0.3	6	187	0.4	4	318	0.4	5	484
	fgmm	0.3	6	260	0.2	4	288	0.5	2	5746

Overall, the isolated evaluation of phase classification shows that fgmm with global (G) phases is the best classifier in terms of CCoV, and fgmm with an L-S setting is the best in terms of average duration.

With regards to runtimes, the fastest phase classifier is fgmm with the L setting, with an inference time of 2.14ms per sample, followed by the G-type fgmm classifier with 2.37ms. 2kmeans inference times are 3.81ms and 4ms for the L and G settings, respectively. As can be seen, the choice of multicore setting has less impact on the inference time than the choice of clustering model, where inference times of 2kmeans are approximately 1.7x slower than fgmm.

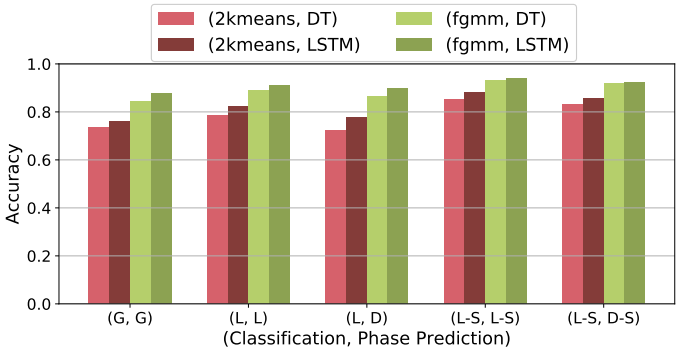


Fig. 7. Average window-based phase prediction accuracy using DT and LSTM models with different multi-core settings and classifiers.

5.2 Phase Prediction

Next, we evaluate the different multi-core phase prediction model settings introduced in Section 4.2. The quality of phases impacts the accuracy of phase prediction. Therefore, we compare each phase prediction setting against its corresponding definitions of phases.

5.2.1 Window-based Prediction. We evaluate two models for window-based phase prediction, DT and LSTM. We follow the hyperparameter selection from [2] and use models with an input window size of $d = 20$ steps. Additionally, we select a tree depth of 8 for DT, and a single-layer LSTM with 128 neurons. The output layer of the LSTM model has $C \times k$ neurons and computes $k = 20$ soft-max activation functions (for each group of C neurons).

Figure 7 shows the average accuracy of each model across all benchmarks with different multi-core settings and the two classifiers, 2kmeans and fgmm. We observe that DT and LSTM models have consistently higher accuracy with fgmm phases regardless of the multi-core setting. When comparing DT and LSTM, LSTM is better than DT, even if only marginally. To better analyze this trend, we compare the accuracy of DT and LSTM predictors with the fgmm classifier across benchmarks and settings in Figure 8. We observe that an LSTM is better for most benchmarks and classification combinations, with very few exceptions. When comparing the accuracy of different multi-core settings, the average trend shows that L-S and D-S phase prediction settings are the best. Figure 8 also shows a general trend of L-S and D-S having the highest accuracy, with a few exceptions, specifically *wrf*, *xz*, and *canneal*. Additionally, D and D-S have a lower accuracy than their L counterparts. They work with L and L-S definitions of phases, respectively. These phase classification settings use data from a single core to classify phases. We explored distributed (D) settings for phase prediction to account for activity of different cores interfering with each other, where information from other cores may help to anticipate phases. Such additional inputs helped in a few cases, such as *cactuBSSN* and *wrf*. However, the local predictors were generally capable of learning without the explicit activity information from other cores.

Overall, the best window-based phase predictor in terms of accuracy is an LSTM with fgmm classification and a (L-S, L-S) multi-core setting.

In terms of model runtimes and complexities, the measured inference times for DT are 1.23ms for G and D models and 1.22ms for L models. By contrast, LSTMs run in 2.94ms, 3.12ms, and 3.13ms per sample for G, L, and D settings, respectively. Similar to phase classification, the inference time of window-based predictors is impacted more by the choice of model (DT vs. LSTM) than by the choice of setting (G, L, or D).

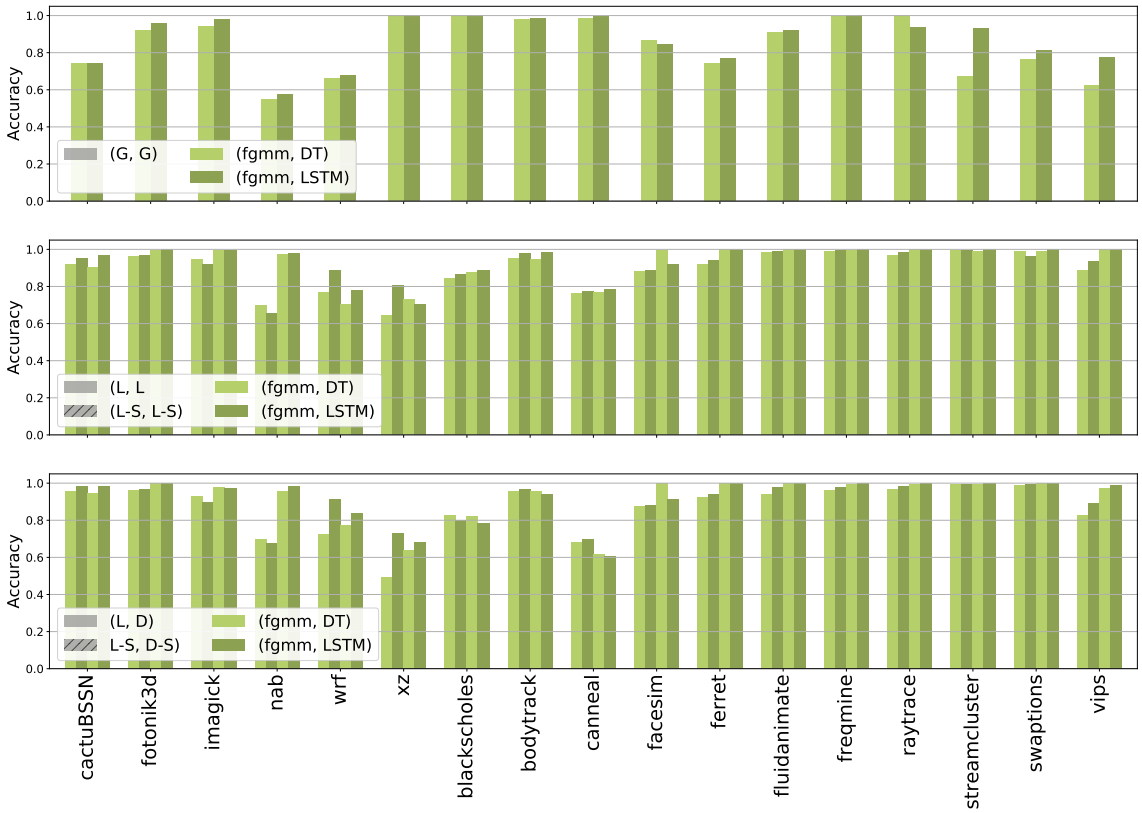
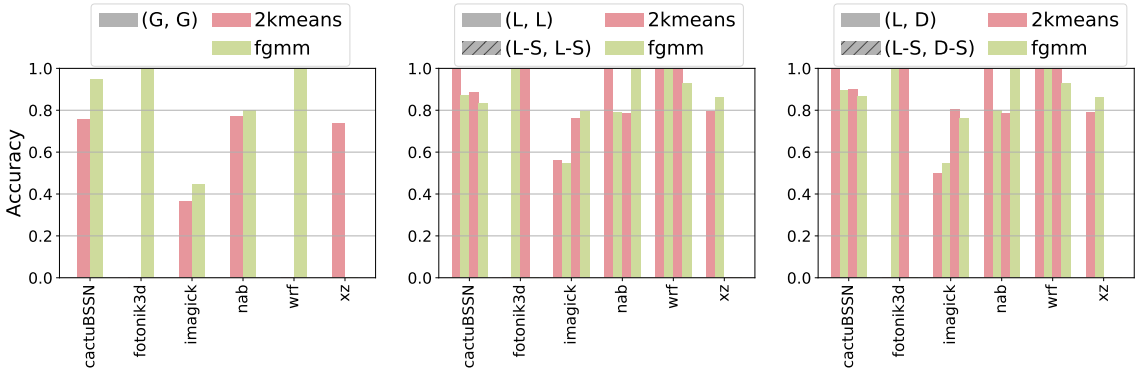


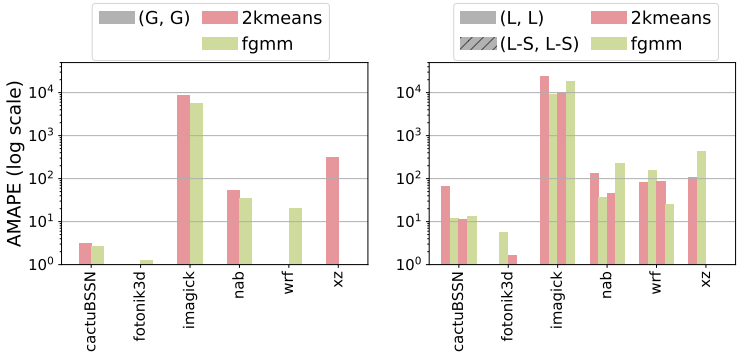
Fig. 8. Window-based phase prediction accuracy using DT and LSTM models with a fgmm classifier under different multi-core (classification, phase prediction) settings.

5.2.2 Phase Change Prediction. We chose SVM models for both next phase and phase duration predictions. Following the hyperparameter exploration in [2], we use an input window size of 10 phase transitions for next phase prediction and the duration and label of the 5 previous phases of any type and the duration of the prior execution of the target phase for phase duration models. Furthermore, we selected a polynomial and linear kernel for next phase and phase duration SVMs, respectively.

As opposed to window-based predictors, which make predictions at every sample, phase change predictors are activated on phase transitions only. Since, as mentioned earlier, many benchmarks have a limited number of phase transitions throughout their executions, such benchmarks have insufficient data points to train phase change predictors. In particular, PARSEC benchmarks run only for relatively short amounts of time while also having long phases. By contrast, the majority of SPEC benchmarks have sufficient data points to train phase change predictors. Note, however, that different combinations of multi-core settings and classifiers yield a different definition of phases per benchmark. As such, we show phase change predictor accuracy only for SPEC benchmarks in combinations with enough data. Results are shown in Figure 9. Figure 9a shows the accuracy of next phase prediction and Figure 9b shows the mean absolute error (MAE) of phase duration prediction in units of samples. The missing data points in the figures correspond to



(a) Next phase prediction accuracy



(b) Phase duration prediction error

Fig. 9. Phase change prediction accuracy using SVM models with 2kmeans and fgmm classifiers for different multi-core settings.

definitions of phases that did not have sufficient data samples in the training set. Note that, as discussed earlier, we only support D multi-core settings for next phase prediction and we combine them with L duration predictors.

Next phase prediction is faced with some trivial cases when the classifier finds two phases only, which results in perfect accuracy. These cases are *cactuBSSN* and *nab* with the 2kmeans classifier and *wrf* with both classifiers for the L setting, and *fotonik3d* and *wrf* with 2kmeans and *nab* with fgmm for the L-S setting. The G phase predictor exhibits higher accuracy with the fgmm classifier for the benchmarks where the comparison between classifiers is available. However, the rest of the phase predictors do not show a clear trend. When comparing the accuracy of the different predictor settings, L and D predictors have either similar or better accuracy than G. For phase duration prediction, the G setting with the fgmm classifier results in the lowest error for all benchmarks except *xz*, where 2kmeans with L is better.

To explore the combined accuracy of next phase and phase duration prediction, we evaluate phase change prediction in conjunction with phase-based forecasting as part of finding the best complete phase-aware combination in Section 5.4 (see Figure 12).

The measured inference times of next phase prediction are 0.08ms for G and L settings and 0.07ms for D settings. For phase duration, the measured times are 1.59ms and 1.56ms for G and L settings, respectively. Next phase and phase

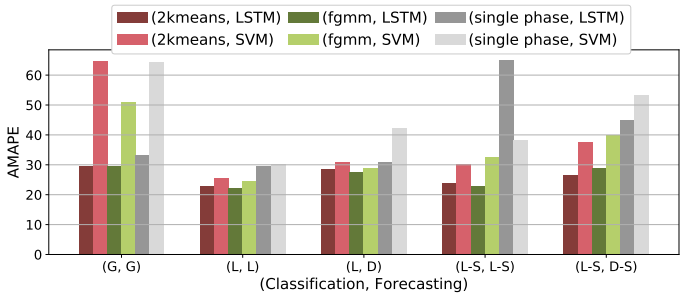


Fig. 10. Average AMAPE of phase-based forecasting with different classifiers and multi-core settings.

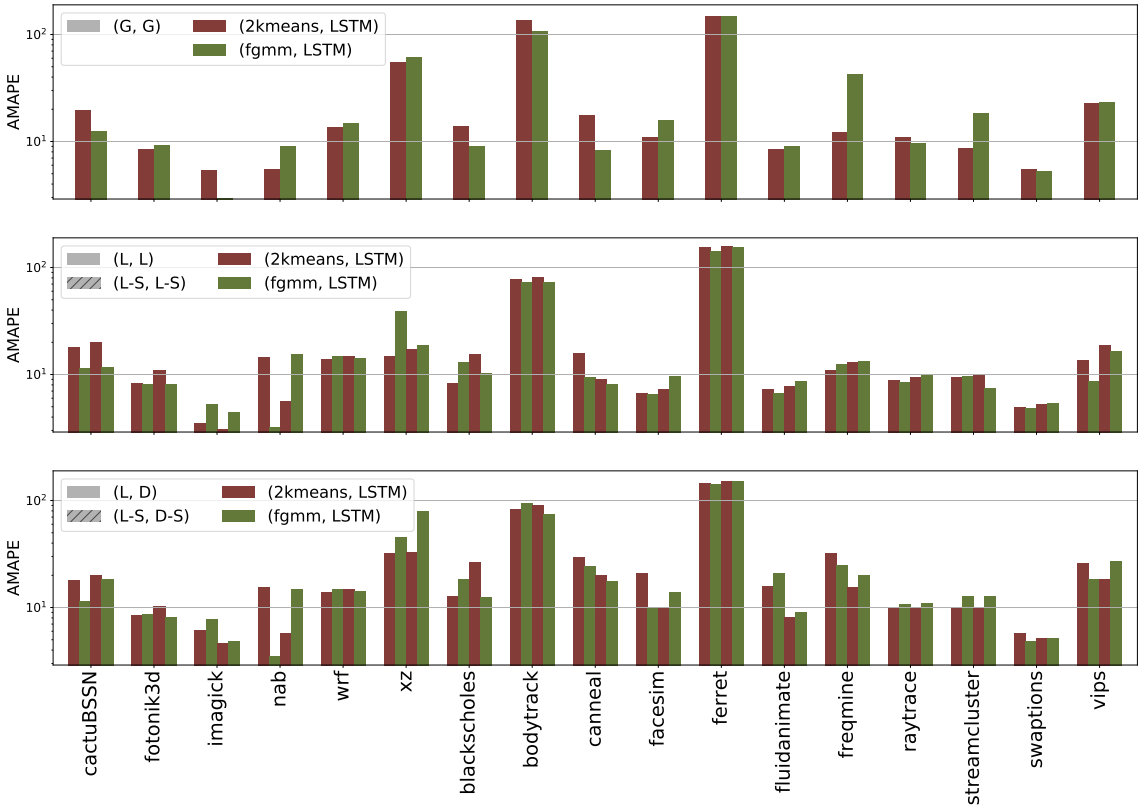


Fig. 11. AMAPE of phase-based forecasting with different classifiers and multi-core (classification, forecasting) settings.

duration prediction tasks can be performed in parallel since they do not have any dependencies. As such, inference time measurements show that predicting phase duration would be the bottleneck of phase change prediction. By contrast, the choice of multi-core setting does not have a major impact on the inference time of these models.

5.3 Phase-based Forecasting

We investigated the accuracy of LSTMs and SVMs in phase-based forecasting with different multi-core settings. Following the hyperparameter tuning and model selection exploration from [3], we employ an LSTM architecture that is comprised of a single-layer LSTM using 128 cells and one fully connected layer of $k = 20$ neurons with linear activation to output the forecasts. The LSTM layer has an input window of $h = 100$ inputs. The SVM uses an input window size of $h = 50$ and a radial-basis function (RBF) kernel to improve the forecasting accuracy compared to a linear SVM. For the G setting, we use a multi-dimensional support vector regressor (MSVR) as presented in [5, 33] to differentiate from D models with only a single output.

Figure 10 shows their average AMAPE across all benchmarks. Note that this accuracy evaluation is independent of phase prediction. We evaluate the average error over all phase-specialized forecasts independent of which model is ultimately selected for forecast reconstruction. The figure shows the AMAPE of each phase-based setting using 2kmeans- and fgmm-classified phases compared to the AMAPE of phase-unaware models using only a single phase.

Results show the benefits of specializing models to phases. Phase-based models (using 2kmeans and fgmm phases) consistently show lower forecasting error than phase-unaware ones, with the exception of a global (G) SVM model using 2kmeans. In general, the G setting performs poorly with SVM models. This figure also shows that an LSTM yields lower forecasting error than an SVM for all phase-based forecasting settings and classifications, while single-phase forecasting has only one case when an SVM is better, L-S.

The difference in forecasting error between fgmm and 2kmeans is not very significant. The lowest forecasting error overall is given by (L, L) using fgmm phases with an LSTM. However, when looking at each setting individually, the best-performing classifier is not as clear as it was when we analyzed window-based phase prediction. We show the comparison of both classifiers with LSTM models in Figure 11. The best classifier for phase-based forecasting varies across benchmarks and multi-core settings. Therefore, we explore the combined accuracy of phase prediction and phase-based forecasting to find the best phase-aware combination in the next section.

Single-phase LSTM models struggle to learn workload patterns when sharing trainable parameters. This trend is persistent in various PARSEC benchmarks, whose CPI traces exhibit different behaviors across different cores. D-S alleviates some of the issues for LSTM models. For example, the single-phase L-S LSTM has its AMAPE of over 150% for *blackscholes* reduced to less than 40% with D-S. Similarly, the AMAPE of *streamcluster* with L-S is 105% while it is 18% with D-S. We observe similar trends between single-phase L-S and D-S LSTMs for *facesim* and *swaptions*.

The high AMAPE difference between L-S and D-S in single-phase LSTM models is not observed for phase-based models. However, when comparing L and D models, the forecasting error increases from L to D. During training, the D forecasters sometimes fail to identify the target core from their inputs and rely heavily on other core inputs to compute the target core's outputs. Therefore, the D forecasts are negatively impacted if other cores go through different phases later in their execution. We observed this behavior in *facesim*, *fluidanimate*, *freqmine*, and *vips*. The same benchmarks do not show such behavior comparing L-S and D-S variants. D-S models share data from all cores during training, forcing the models to identify the target core for forecasting. However, we found that *xz* exhibited a particular case with an LSTM and fgmm phases. One of *xz*'s phases occurs only for a short amount of time at the beginning of execution, and when it is is reencountered, the D-S model has not yet learned to identify the target core. Instead, another core highly influences its predictions.

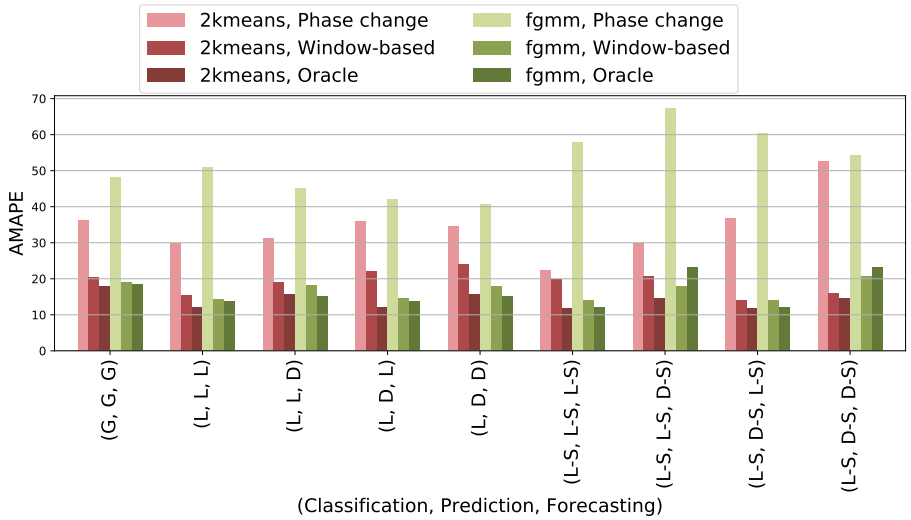


Fig. 12. Average AMAPE of each phase-aware multi-core forecasting settings for SPEC benchmarks.

On average, the best phase-based forecasting result in terms of accuracy is given by the L multi-core setting with an fgmm classifier and LSTM models. Similarly, the best phase-unaware result in terms of forecasting error is given by an L multi-core setting with LSTM models.

In terms of runtimes, as opposed to phase classification and phase prediction, the choice of multi-core setting can strongly impact the inference time of phase-based forecasting models. Particularly, the L setting of the SVM models is up to 2.4x faster than the other settings, with 1.19ms, 0.52ms, and 1.25ms for G, L, and D settings, respectively. A similar behavior is not observed with LSTMs, whose inference times are slower than SVM for any setting, with 4.31ms, 4.19ms, and 4,28ms for G, L, and D settings, respectively.

5.4 Combined Phase-aware Workload Forecasting

We finally study the combined accuracy of phase classification, phase prediction and phase-based forecasting for all the multi-core settings. Results of different settings averaged over benchmarks are shown in Figure 12 and Figure 13. Both figures include the results of using an oracle phase predictor with 100% phase prediction accuracy. We found that a fgmm phase classifier was better than 2kmeans for window-based phase prediction, but not necessarily for phase change prediction and phase-based forecasting. Therefore, we compare average results using both classifiers. Figure 12 shows averaged results only for SPEC benchmarks for which both window-based and phase change predictors are available. By contrast, Figure 13 shows results averaged over all benchmarks using window-based phase predictors. Phase change predictors use SVM models. Window-based phase prediction and phase-based forecasting use LSTMs as their models since those resulted in the best accuracy when evaluated separately.

As seen in Figure 12, a classification with 2kmeans yields better results than fgmm across different multi-core settings when using phase change prediction, where the best phase change predictor in combination with phase-based forecasting is (L-S, L-S, L-S) with a 2kmeans classifier. However, an opposite trend is observed for window-based predictions, where fgmm yields the lowest error with the exception of (L-S, D-S, D-S). This generally confirms that

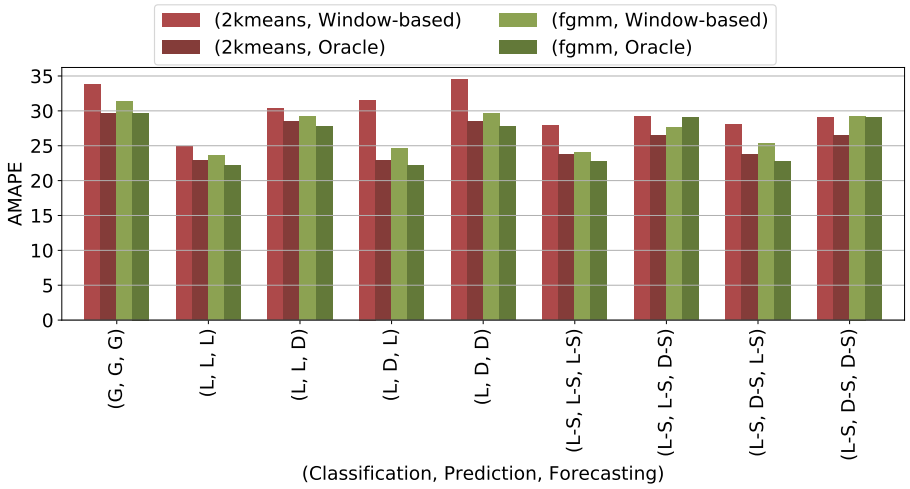


Fig. 13. Average AMAPE accuracy of each phase-aware multi-core forecasting settings using window-based phase predictors.

isolated phase prediction trends from Section 5.2 hold when combined with phase-based forecasting. Overall, window-based predictors consistently exhibit the lowest forecasting error across all multi-core settings. Therefore, we focus on window-based phase prediction in the rest of this evaluation.

Figure 13 shows that when averaging results using window-based phase prediction over all benchmarks, an fgmm phase classification achieves consistently better results than 2kmeans, even for the (L-S, D-S, D-S) setting. When comparing the error difference between oracle and window-based prediction, 2kmeans exhibits a larger gap than fgmm. This is consistent with our results in Section 5.2. On average, the best and worst phase-aware combinations are (L, L, L) and (G, G, G), respectively. Distributed phased-based forecasting variants (combinations with D and D-S as the last forecasting element) generally have worse errors than their local counterparts. This trend was already observed in Section 5.3. By contrast, there is no clear advantage of distributed versus local phase predictors in average accuracy.

To further analyze different multi-core settings per benchmark, Figure 14 compares phase-aware combinations using an fgmm classifier and window-based phase prediction against corresponding phase-unaware settings using only a single phase. We again include results of using an oracle phase predictor. We observe that in global settings (top graph), phase awareness does not consistently improve forecasting error. Phases sometimes underfit the trace, where per-core changes remain undetected. As discussed earlier, when we analyzed phase classification data, *xz* suffered from these undetected changes in CPI behavior, leading to high CCoV values. We find similar issues in *bodytrack*, *freqmine* and *streamcluster*, even though this did not show up during the phase classification evaluation.

By contrast, local settings (second graph from the top), show in some cases significant benefits of phase-aware forecasting, with shared or non-shared variants having the lowest forecasting error for most benchmarks. In particular, *blackscholes* exhibits the most benefits, where phase-awareness reduces the error by over 22%, followed by *canneal* with an error reduction of 13%. There are only two cases where phase specialization was not able to improve on phase-unaware forecasting, *ferret* and *bodytrack*. All models struggle to learn their workload patterns. In the case of *bodytrack*, the forecasting error had some minor improvements with phase-based forecasting, but the error introduced by phase prediction negated the benefits. In the *ferret* case, the hardware counter data is comprised of very high-frequency

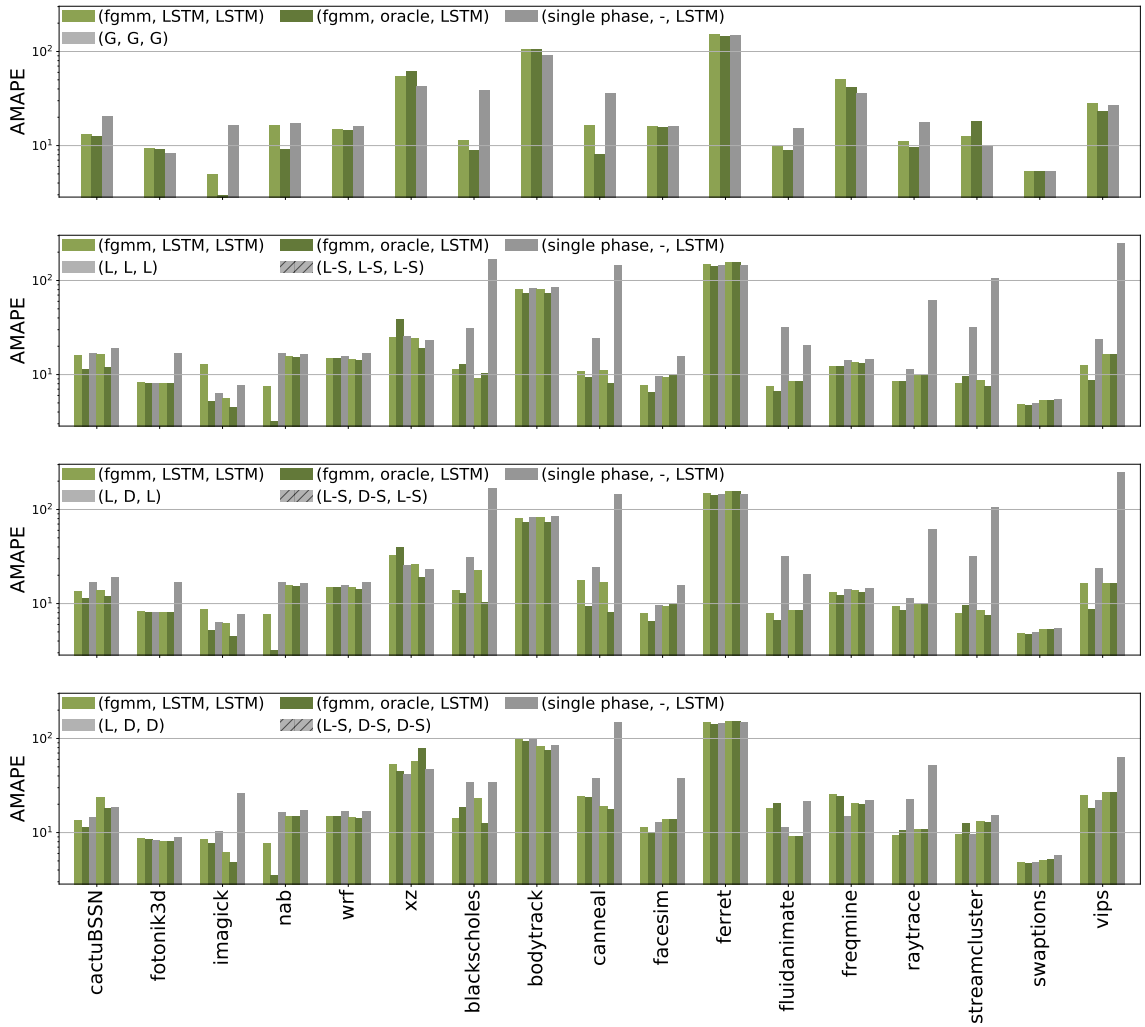


Fig. 14. Accuracy of best phase-aware and -unaware models for different multi-core settings.

components, and all models struggled to find patterns. Additionally, the inability of phase specialization to improve forecasting error is an indication of poor phase classification. As mentioned before, some benchmarks have phases with changes in CPI behavior that remain undetected by the phase classifiers. Interestingly, for *xz*, the phase predictor in some cases recognized these changes and improved the overall phase-aware forecasting accuracy. This can be observed by comparing the oracle phase prediction to the LSTM of the (L, L, L) setting. However, since the phase classifier dictates which phase-based forecaster should be activated, the forecasts from the phase corresponding to the phase predictor's output are not updated, limiting the accuracy improvements. In some benchmarks, the error gap between the oracle and the window-based phase prediction LSTM is high. This can be observed in *imagick, nab* and *vips* with the (L, L, L) setting. Note that this gap is not as high with the (L-S, L-S, L-S) variant. This is consistent with the results in Section 5.2, where the shared variant exhibited higher phase prediction accuracy for these benchmarks.

Table 4. Model complexities for each phase-aware task.

Task		Model	Inference time (ms)		
			G	L(-S)	D(-S)
Phase Classification		2kmeans	4.00	3.81	-
		fgmm	2.37	2.14	-
Phase Prediction	Window-based	DT	1.23	1.22	1.23
		LSTM	2.94	3.12	3.13
	Phase Change	Next phase SVM	0.08	0.08	0.07
		Phase duration SVM	1.59	1.56	-
Phase-based Forecsting		LSTM	4.31	4.19	4.28
		SVM	1.19	0.52	1.25

Finally, the bottom graphs show the distributed (L, D, L) and (L, D, D) settings in their regular and shared variants. As mentioned above, phase-aware (L, D, D) variants (bottom graph) generally have higher forecasting error than their (L, D, L) counterparts (second to last graph). By contrast, this trend is not seen in phase-unaware models. When compared to the (L, L, L) setting, distributed results are very similar, with a few exceptions. For both phase-aware and -unaware forecasting, a distributed setup can improve the accuracy of *cactuBSSN* and *imagick*. However, it increases the error of other benchmarks, including *blackscholes* and *canneal*.

Table 4 compares the complexities of all models and multi-core settings of each phase-aware task in terms of inference times for a basic software implementation. For phase classification, the best model in terms of accuracy, fgmm, is also the one with the least overhead. However, this was not the case for phase prediction and phase-based forecasting, where the most accurate models, LSTMs, are also the slowest. Phase prediction and phase-based forecasting tasks are independent from each other and can therefore be computed in parallel. As such, in the best phase-aware model combination in terms of accuracy, (L, L, L) and (fgmm, LSTM, LSTM), the main bottleneck is the phase-based forecasting task. Note that these numbers are only meant to provide an indication of relative model complexity rather than an actual implementation proposal. Our user-level software implementations have not been optimized for deployment as workload forecasting tasks. Optimized hardware or software implementations of models such as LSTMs have been studied extensively using methods such as pruning [7, 45] or hardware acceleration [4, 43, 45, 47, 49]. These methods demonstrate inference time acceleration of 43x when compared to a CPU implementation [49] and power consumption reduction by more than 50% with pruning and quantization without compromising accuracy of LSTM models [7]. In our own prior work [3], we also performed an exploration of the hyperparameter space of such models (number of LSTM cells and number of input steps, h) to study tradeoffs between model complexity and accuracy. Results showed that the smallest LSTM can reduce inference times down to 0.5ms per prediction albeit at a cost of 9% higher average error.

In addition to inference complexity of phase-aware models, their training overhead must ultimately also be considered when deploying an online learning approach. Training time is dependent on multiple factors such as the optimization algorithm, e.g. whether batching is utilized, and the number of samples used. As an example, we measured the time and memory space taken to update the weights of our phase-based forecasting LSTM with 5,000 samples consecutively and a minibatch size of 10. This requires up to 48MB of memory, and runs in faster than real-time requiring 4.8ms per sample and 24 seconds total. Reducing the minibatch to size 1 reduces the memory space to 24MB, but increases the training time to 230 seconds total. Using the same training set size, the training time of the phase-based forecasting SVM is in the order of hundreds of milliseconds, but requires up to 84MB of memory. The window-based phase prediction DT is the most

memory-efficient task using only up to 7MB of space and up to 8 seconds. Overall, a standard offline setup is not the best fit for training during runtime. Instead, optimized online training methods that accelerate gradient computations [26], intelligently train on a fraction of the samples [18], or reduce the model search space [16, 52] can be applied. In particular, continual learning is an active field of research [16] with many studies focusing on reducing training overhead, including approaches such as few-shot learning [46] and importance sampling [18, 19], that can potentially enable low-overhead re-training of our models at runtime in the future. For example, importance sampling [18] can reduce training times by at least 20% and up to 2x for various deep learning models without compromising accuracy.

6 SUMMARY AND CONCLUSIONS

In this paper, we studied learning-based methods for phase-aware workload forecasting in multi-core systems using performance monitoring counters. Our approach combines workload phase classification and phase prediction to separate time series into phases and train a separate, specialized prediction models for each phase. We proposed three main multi-core forecasting settings and performed a comparative study of hardware counter-based phase classification, phase prediction and phase-based forecasting techniques in isolation and under different combinations.

Our isolated studies concluded that a global definition of phases is best at minimizing the average variation per phase. However, when evaluating the combined solution, a phase-aware LSTM with per-core phase definitions was the best predictor with 23% average MAPE across benchmarks and up to 22% improvement over a phase-unaware approach. Results from SPEC 2017 and PARSEC-3.0 benchmarks running on a state-of-the-art workstation show that phase-aware forecasting reduces MAPE by 6% on average across different benchmarks when compared to the best-performing phase-unaware approach. Compared to our prior work using oracle phase classification and prediction [3], effectiveness of phase-aware forecasting is limited by phase classification results. A poor quality of phases will negatively affect both phase prediction and phased-based forecasting accuracy.

Future work includes investigating better multi-core phase classifiers as well as phase-aware forecasting for a wider range of workloads, approaches for online training of predictors, efficient hardware or software deployment of predictors, and application of phase-aware workload forecasting to various use cases such as power management or system scheduling.

ACKNOWLEDGMENTS

This work was supported by NSF grant CCF-1763848 and the Texas Advanced Computing Center (TACC).

REFERENCES

[1] Cristinel Ababei and Milad Ghorbani Moghaddam. 2018. A Survey of Prediction and Classification Techniques in Multicore Processor Systems. IEEE Transactions on Parallel and Distributed Systems (TPDS) 30, 5 (2018), 1184–1200. https://doi.org/10.1109/TPDS.2018.2878699

[2] Erika S. Alcorta and Andreas Gerstlauer. 2021. Learning-Based Workload Phase Classification and Prediction Using Performance Monitoring Counters. In Workshop on Machine Learning for CAD (MLCAD). https://doi.org/10.1109/MLCAD52597.2021.9531161

[3] Erika S. Alcorta, Pranav Rama, Aswin Ramachandran, and Andreas Gerstlauer. 2021. Phase-Aware CPU Workload Forecasting. In International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS).

[4] Erfan Bank-Tavakoli, Seyed Abolfazl Ghasemzadeh, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2020. POLAR: A Pipelined/Overlapped FPGA-Based LSTM Accelerator. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 28, 3 (2020), 838–842. https://doi.org/10.1109/TVLSI.2019.2947639

[5] Yukun Bao, Tao Xiong, and Zhongyi Hu. 2014. Multi-step-ahead time series prediction using multiple-output support vector regression. Neurocomputing 129 (2014), 482–493.

[6] Chin-Hao Chang, Pangfeng Liu, and Jan-Jan Wu. 2013. Sampling-Based Phase Classification and Prediction for Multi-threaded Program Execution on Multi-core Architectures. In International Conference on Parallel Processing (ICPP). https://doi.org/10.1109/ICPP.2013.44

[7] Zhe Chen, Hugh T. Blair, and Jason Cong. 2022. Energy-Efficient LSTM Inference Accelerator for Real-Time Causal Prediction. ACM Transactions on Design Automation of Electronic Systems (TODAES) 27, 5, Article 44 (jun 2022), 19 pages. https://doi.org/10.1145/3495006

[8] Meng Chieh Chiu and Eliot Moss. 2018. Run-time program-specific phase prediction for python programs. In International Conference on Managed Languages & Runtimes (ManLang). https://doi.org/10.1145/3237009.3237011

[9] François Chollet et al. 2015. Keras. https://keras.io.

[10] Ryan Cochran and Sherief Reda. 2010. Consistent runtime thermal prediction and control through workload phase detection. In Design Automation Conference (DAC). https://doi.org/10.1145/1837274.1837292

[11] Ayse Kivilcim Coskun, Tajana Simunic Rosing, and Kenny C. Gross. 2009. Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 28, 10 (2009), 1503–1516. https://doi.org/10.1109/TCAD.2009.2026357

[12] Keeley Criswell and Tosiron Adegbija. 2019. A Survey of Phase Classification Techniques for Characterizing Variable Application Behavior. IEEE Transactions on Parallel and Distributed Systems (TPDS) 31, 1 (2019), 224–236.

[13] Miguel Tairum Cruz, Pedro Tomás, and Nuno Roma. 2016. Unsupervised Variable-Grained Online Phase Clustering for Heterogeneous/Morphable Processors. In International Conference on High Performance Computing and Simulation (HPCS). https://doi.org/10.1109/HPCSim.2016.7568424

[14] Evelyn Duesterwald, Calin Caşcaval, and Sandhya Dwarkadas. 2003. Characterizing and predicting program behavior and its variability. In International Conference on Parallel Architectures and Compilation Techniques (PACT). https://doi.org/10.1109/PACT.2003.1238018

[15] Philippe Gervais Fabian Pedregosa et al. 2021. Memory Profiler. https://pypi.org/project/memory-profiler/.

[16] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. 2020. Embracing change: Continual learning in deep neural networks. Trends in cognitive sciences 24, 12 (2020), 1028–1040.

[17] Canturk Isci, Gilberto Contreras, and Margaret Martonosi. 2006. Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management. In International Symposium on Microarchitecture (MICRO). https://doi.org/10.1109/MICRO.2006.30

[18] Tyler B Johnson and Carlos Guestrin. 2018. Training deep models faster with robust, approximate importance sampling. Advances in Neural Information Processing Systems 31 (2018).

[19] Angelos Katharopoulos and François Fleuret. 2018. Not all samples are created equal: Deep learning with importance sampling. In International conference on machine learning. PMLR, 2525–2534.

[20] Omer Khan and Sandip Kundu. 2011. Microvisor: A runtime architecture for thermal management in chip multiprocessors. In Lecture Notes in Computer Science (LNCS). https://doi.org/10.1007/978-3-642-24568-8_5

[21] Rahul Khanna, Jaiber John, and Thanunathan Rangarajan. 2012. Phase-aware predictive thermal modeling for proactive load-balancing of compute clusters. In International Conference on Energy Aware Computing (ICEAC).

[22] Saman Khoshbakht and Nikitas Dimopoulos. 2017. Execution Phase Prediction Based on Phase Precursors and Locality. In International Workshop on Energy Efficient Supercomputing (E2SC). https://doi.org/10.1145/3149412.3149415

[23] Saman Khoshbakht and Nikitas Dimopoulos. 2017. A new approach to detecting execution phases using performance monitoring counters. In Lecture Notes in Computer Science (LNCS). https://doi.org/10.1007/978-3-319-54999-6_7

[24] Yeseong Kim, Pietro Mercati, Ankit More, Emily Shriver, and Tajana Rosing. 2017. P^4: Phase-Based Power/Performance Prediction of Heterogeneous Systems via Neural Networks. In International Conference on Computer-Aided Design (ICCAD). https://doi.org/10.1109/ICCAD.2017.8203843

[25] Jeremy Lau, Stefan Schoenmackers, and Brad Calder. 2005. Transition Phase Classification and Prediction. In International Symposium on High-Performance Computer Architecture (HPCA). https://doi.org/10.1109/HPCA.2005.39

[26] Qiang Liu, Jia Liu, Ruoyu Sang, Jiajun Li, Tao Zhang, and Qijun Zhang. 2018. Fast Neural Network Training on FPGA Using Quasi-Newton Optimization Method. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 26, 8 (2018), 1575–1579. https://doi.org/10.1109/TVLSI.2018.2820016

[27] Dimosthenis Masouros, Sotirios Xydis, and Dimitrios Soudris. 2021. Rusty: Runtime Interference-Aware Predictive Monitoring for Modern Multi-Tenant Systems. IEEE Transactions on Parallel and Distributed Systems (TPDS) 32, 1 (2021), 184–198. Issue 1. https://doi.org/10.1109/TPDS.2020.3013948

[28] Milad Ghorbani Moghaddam and Cristinel Ababei. 2017. Dynamic Energy management for Chip Multi-Processors under Performance Constraints. Microprocessors and Microsystems 54 (2017), 1–13.

[29] Milad Ghorbani Moghaddam, Wenkai Guan, and Cristinel Ababei. 2017. Investigation of LSTM based prediction for dynamic energy management in chip multiprocessors. In International Green and Sustainable Computing Conference (IGSC). 1–8. https://doi.org/10.1109/IGCC.2017.8323597

[30] Milad Ghorbani Moghaddam, Wenkai Guan, and Cristinel Ababei. 2018. Dynamic Energy Optimization in Chip Multiprocessors using Deep Neural Networks. IEEE TMSCS 4, 4 (2018), 649–661.

[31] Junaid Nomani and Jakub Szefer. 2015. Predicting Program Phases and Defending against Side-Channel Attacks Using Hardware Performance Counters. In Workshop on Hardware and Architectural Support for Security and Privacy (HASP). https://doi.org/10.1145/2768566.2768575

[32] F. Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. JMLR 12 (2011), 2825–2830.

[33] Fernando Pérez-Cruz, Gustavo Camps-Valls, Emilio Soria-Olivas, Juan José Pérez-Ruixo, Aníbal R Figueiras-Vidal, and Antonio Artés-Rodríguez. 2002. Multi-Dimensional Function Approximation and Regression Estimation. In International Conference on Artificial Neural Networks. Springer, 757–762.

[34] Martin Rapp, Anuj Pathania, Tulika Mitra, and Jörg Henkel. 2021. Neural Network-based Performance Prediction for Task Migration on S-NUCA Many-Cores. IEEE Transactions on Computers (TC) 70, 10 (2021), 1691 – 1704. https://doi.org/10.1109/TC.2020.3023022

[35] Rance Rodrigues, Arunachalam Annamalai, Israel Koren, and Sandip Kundu. 2013. Improving Performance per Watt of Asymmetric Multi-Core Processors via Online Program Phase Classification and Adaptive Core Morphing. ACM Transactions on Design Automation of Electronic Systems (TODAES) 18, 1, Article 5 (jan 2013), 23 pages. https://doi.org/10.1145/2390191.2390196

[36] Ahsan Saeed, Daniel Mueller-Gritschneder, Falk Rehm, Arne Hamann, Dirk Ziegenbein, Ulf Schlichtmann, and Andreas Gerstlauer. 2021. Learning based Memory Interference Prediction for Co-running Applications on Multi-Cores. In Workshop on Machine Learning for CAD (MLCAD). https://doi.org/10.1109/MLCAD52597.2021.9531245

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

[38] Ruhi Sarikaya and Alper Buyuktosunoglu. 2007. Predicting program behavior based on objective function minimization. In International Symposium on Workload Characterization (IISWC). https://doi.org/10.1109/IISWC.2007.4362178

[39] Andreas Sembrant, David Black-Schaffer, and Erik Hagersten. 2012. Phase behavior in serial and parallel applications. In IEEE International Symposium on Workload Characterization (IISWC). 47–58. https://doi.org/10.1109/IISWC.2012.6402900

[40] Andreas Sembrant, David Eklov, and Erik Hagersten. 2011. Efficient Software-Based Online Phase Classification. In International Symposium on Workload Characterization (IISWC). https://doi.org/10.1109/IISWC.2011.6114207

[41] Tim Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. 2002. Automatically Characterizing Large Scale Program Behavior. In International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). https://doi.org/10.1145/605397.605403

[42] Sudarshan Srinivasan, Raghavan Kumar, and Sandip Kundu. 2013. Program Phase Duration Prediction and Its Application to Fine-Grain Power Management. In Symposium on VLSI (ISVLSI). https://doi.org/10.1109/ISVLSI.2013.6654634

[43] Kriti Suneja, Aniket Chaudhary, Arun Kumar, and Ayush Srivastava. 2022. Recent Advancements in FPGA-based LSTM Accelerator. In 2022 International Conference for Advancement in Technology (ICONAT). 1–5. https://doi.org/10.1109/ICONAT53423.2022.9726002

[44] Karl Taht, James Greensky, and Rajeev Balasubramonian. 2019. The POP Detector: A Lightweight Online Program Phase Detection Framework. In International Symposium on Performance Analysis of Systems and Software (ISPASS). https://doi.org/10.1109/ISPASS.2019.00013

[45] Meiqi Wang, Zhisheng Wang, Jinming Lu, Jun Lin, and Zhongfeng Wang. 2019. E-LSTM: An Efficient Hardware Architecture for Long Short-Term Memory. IEEE Journal on Emerging and Selected Topics in Circuits and Systems 9, 2 (2019), 280–291. https://doi.org/10.1109/JETCAS.2019.2911739

[46] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. ACM computing surveys (csur) 53, 3 (2020), 1–34.

[47] Reza Yazdani, Olatunji Ruwase, Minjia Zhang, Yuxiong He, José-María Arnau, and Antonio González. 2019. LSTM-Sharp: An Adaptable, Energy-Efficient Hardware Accelerator for Long Short-Term Memory. CoRR abs/1911.01258 (2019). arXiv:1911.01258 http://arxiv.org/abs/1911.01258

[48] Monir Zaman, Ali Ahmadi, and Yiorgos Makris. 2015. Workload characterization and prediction: A pathway to reliable multi-core systems. In International On-Line Testing Symposium (IOLTS). https://doi.org/10.1109/IOLTS.2015.7229843

[49] Weifeng Zhang, Fen Ge, Chenchen Cui, Ying Yang, Fang Zhou, and Ning Wu. 2020. Design and Implementation of LSTM Accelerator Based on FPGA. In IEEE International Conference on Communication Technology (ICCT). 1675–1679. https://doi.org/10.1109/ICCT50939.2020.9295665

[50] Weihua Zhang, Jiaxin Li, Yi Li, and Haibo Chen. 2015. Multilevel phase analysis. ACM Transactions on Embedded Computing Systems (TECS) 14, 2 (2015), 31:1–31:29. Issue 2. https://doi.org/10.1145/2629594

[51] Xinnian Zheng, Lizy K. John, and Andreas Gerstlauer. 2016. Accurate Phase-Level Cross-Platform Power and Performance Estimation. In Design Automation Conference (DAC). https://doi.org/10.1145/2897937.2897977

[52] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2021. A Comprehensive Survey on Transfer Learning. Proc. IEEE 109, 1 (2021), 43–76. https://doi.org/10.1109/JPROC.2020.3004555