Pós-Graduação em Ciência da Computação

Sidartha Azevedo Lobo de Carvalho

Gerenciamento autônomo de energia em dispositivos móveis utilizando aprendizagem por reforço para economia de energia

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

Sidartha Azevedo Lobo de Carvalho

Gerenciamento autônomo de energia em dispositivos móveis utilizando aprendizagem por reforço para economia de energia

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Sistemas Embarcados
Orientador: Abel Guilhermino da Silva Filho

Recife
2019

Catalogação na fonte
Bibliotecária Mariana de Souza Alves CRB4-2106

C331g Carvalho, Sidartha Azevedo Lobo de.
Gerenciamento autônomo de energia em dispositivos móveis utilizando aprendizagem por reforço para economia de energia/ Sidartha Azevedo Lobo de Carvalho. – 2019.
147 f.: il., fig., tab.

Orientador: Abel Guilhermino da Silva Filho.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da computação. Recife, 2019.
Inclui referências.

1. Sistemas Embarcados. 2. Gerenciamento de energia. 3. Aprendizado de máquina. 4. Redução de energia. I. Silva Filho, Abel Guilhermino da (orientador). II. Título.

005.256 CDD (22. ed.) UFPE-MEI 2019-158

Sidartha Azevedo Lobo de Carvalho

“Gerenciamento autônomo de energia em dispositivos móveis utilizando aprendizagem por reforço para economia de energia”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Aprovado em: 22/08/2019.

_____

Orientador: Prof. Dr. Abel Guilhermino da Silva Filho

BANCA EXAMINADORA

_____

Profa. Dra. Edna Natividade da Silva Barros
Centro de Informática/UFPE

_____

Prof. Dr. Alexandre Cabral Mota
Centro de Informática/UFPE

_____

Prof. Dr. Daniel Carvalho da Cunha
Centro de Informática/UFPE

_____

Prof. Dr. Elmar Uwe Kurt Melcher
Centro de Engenharia Elétrica e Informática/ UFCG

_____

Prof. Dr. Wellington Pinheiro dos Santos
Departamento de Engenharia Biomédica/UFPE

Eu acredito que às vezes, são as pessoas de quem ninguém espera nada que fazem as coisas que ninguém consegue imaginar (HODGES, 1983).

RESUMO

Sistemas embarcados e móveis executam diferentes tipos de aplicações que estimulam o hardware de maneira distinta, gerando cargas de processamento variáveis com o tempo. Uma redução no consumo de energia pode ser alcançada usando uma frequência de *Unidade Central de Processamento ou Central Processing Unit* (CPU) específica para cada tipo de carga de processamento. É necessário que a abordagem seja capaz de reduzir o consumo de energia a partir da adaptação às variações da carga de processamento, mesmo em um ambiente desconhecido. Por este problema, propomos um novo método para predizer a carga de processamento da CPU para dispositivos móveis com o diferencial da funcionalidade de detecção de mudanças na carga de processamento de forma autônoma, chamado de AEWMA-MSE. Além disso, um novo modelo de predição de potência, baseado no *k-Nearest Neighbor* (k-NN) para regressão, foi proposto e validado demonstrando um melhor balanceamento entre tempo de execução e precisão quando comparado à rede neural e modelos de regressão lineares. Após isso, o AEWMA-MSE e o modelo de predição de potência são integrados em um novo algoritmo para gerenciamento de energia, baseado em aprendizagem por reforço (*Q-learning*), que seleciona a frequência de CPU que minimiza o consumo de energia. A abordagem proposta foi validada utilizando simulação e medições reais com dois *smartphones* comerciais. A abordagem proposta demonstrou um melhoramento na função de custo do *Q-learning* que conseguiu atingir uma redução do consumo de energia, alcançando até 42% de economia, a depender da abordagem e *benchmark* em comparação. A abordagem proposta demonstrou cumprir as restrições de tempo e utilização de recursos necessários para os dispositivos móveis, além disso, provendo níveis significantes de economia energética.

Palavras-chaves: Gerenciamento de energia. Aprendizado de máquina. Redução de energia. Otimização de energia. Android.

ABSTRACT

Embedded and mobile systems execute applications that exercise hardware differently depending on the computation task, generating time-varying workloads. Energy minimization can be reached by using the low-power CPU frequency for each workload. Identify an approach capable of reducing energy consumption from adaptation to workload variations, even in an unknown environment is necessary. We proposed a new method to predict the CPU workload called AEWMA-MSE and added new functionality to detect workload changes. Also, a new power model for mobile devices based on k-NN algorithm for regression was proposed and validated proving to have a better trade-off between execution time and precision than neural networks and linear regression-based models. AEWMA-MSE and the proposed power model are integrated into a novel algorithm for energy management based on reinforcement learning (*Q-learning*) that suitably selects the appropriate CPU frequency based on workload predictions to minimize energy consumption. The proposed approach is validated through simulation and real measurement by using two commercial smartphones. Our proposal proved to have an improvement in the Q-learning cost function and can effectively minimize the energy consumption by up to 42% when compared to the already existing approaches. Our approach has demonstrated to have the restrictions of time and resources utilization required for mobile devices, besides that, providing significant levels of energy savings.

Keywords: Power management. Machine learning. Mobile low power. Energy optimization. Android

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

f_{LP}	*Low-power frequency*
k-NN	*k-Nearest Neighbor*
2G	Tecnologia de Celular de 2ª Geração - *Second Generation Cellular Technology*
3G	Tecnologia de Celular de 3ª Geração - *Third Generation Cellular Technology*
AA	Atualizar Algoritmos
ADB	*Android Debug Bridge*
AEWMA	*Adaptive Exponential Weighted Moving Average*
AEWMA-MSE	*Adaptive Exponential Weighted Moving Average MSE*
ANN	*Artificial Neural Network*
AP	Abordagem Proposta
API	*Application Program Interface*
ARM	*Advanced RISC Machine*
ART	*Android Runtime*
CH	Google Chrome
CL	*Context Logger*
CMOS	*Complementary Metal–Oxide–Semiconductor*
CPU	*Unidade Central de Processamento ou Central Processing Unit*
CS	*Conservative*
DPM	*Gerenciamento Dinâmico de Potência - Dynamic Power Management*
DTM	Gerenciamento Dinâmico de Temperatura - *Dynamic Thermal Management*
DVFS	*Escalonamento Dinâmico de Tensão e Frequência - Dynamic Voltage and Frequency Scaling*
EMA	Escolher Melhor Ação
EWMA	*Exponential Weighted Moving Average*
FB	Facebook
FNN	*Fast Nearest Neighbor Search Algorithms and Applications*

GPU	Unidade de Processamento Gráfico - *Graphics Processing Unit*
HD	Alta Definição - *High Definition*
HS	Shen et al. (2013)
I/O	*Input/Output*
IoT	Internet das Coisas - *Internet of Things*
IT	*Interactive*
JS	JavaScript
KL	*Kullback–Leibler*
LM	*Linear Model*
MAE	Erro Médio Absoluto - *Mean Absolute Error*
MAPE	Erro Percentual Médio e Absoluto - *Mean Absolute Percentage Error*
MN	Maeda-Nunez (2016)
MOSFET	*Metal-Oxide-Semiconductor Field Effect Transistor*
MSE	Erro Médio Quadrado - *Mean Squared Error*
OD	*Ondemand*
PE	*Performance*
PG	*Power Gating*
PQ	*PegasusQ*
PS	*Powersave*
PVD	Predição de Variáveis de Desempenho
RL	Aprendizado por Reforço - *Reinforcement Learning*
RTM	*Run-Time Manager*
SARSA	*State-Action-Reward-State-Action*
SK	Shafik et al. (2016)
SO	Sistema Operacional
SoC	*System on a Chip*
US	*Userspace*
UTC	*Coordinated Universal Time*
UX	Experiência do Usuário - *User Experience*
WCET	*Worst Case Execution Time*
WL	Carga de Processamento - *Workload*

YT	YouTube
ZT	Tian et al. (2018)

SUMÁRIO

1 INTRODUÇÃO

Diante do crescente mercado de dispositivos móveis nos últimos anos, esses têm se tornado ubíquos em nossa rotina diária. De acordo com Gartner (2016), aproximadamente 344 milhões de *smartphones* foram vendidos em todo o planeta no segundo quadriênio de 2016, representando 4,3% de aumento comparado ao mesmo período de 2015.

Em adição, esse crescimento continua para os anos de 2017 e 2018. No segundo quadriênio de 2018, o total de *smartphones* vendidos foi de aproximadamente 374 milhões, um crescimento de 2,10% comparado ao mesmo período de 2017. Neste cenário de crescimento, o Sistema Operacional (SO) Android manteve a liderança com 88% da fatia de mercado, representando 329 milhões de *smartphones* vendidos (GARTNER, 2018). O segundo mais vendido foi o iOS com 11,9% da fatia de mercado.

O aumento da presença dos *smartphones* tende ao aumento do uso desses dispositivos em tarefas diárias, tanto em cenários pessoais quanto profissionais. O maior envolvimento dos dispositivos em tarefas do cotidiano estimulou investimentos da indústria no desenvolvimento de dispositivos com especificações mais robustas, com maior poder computacional.

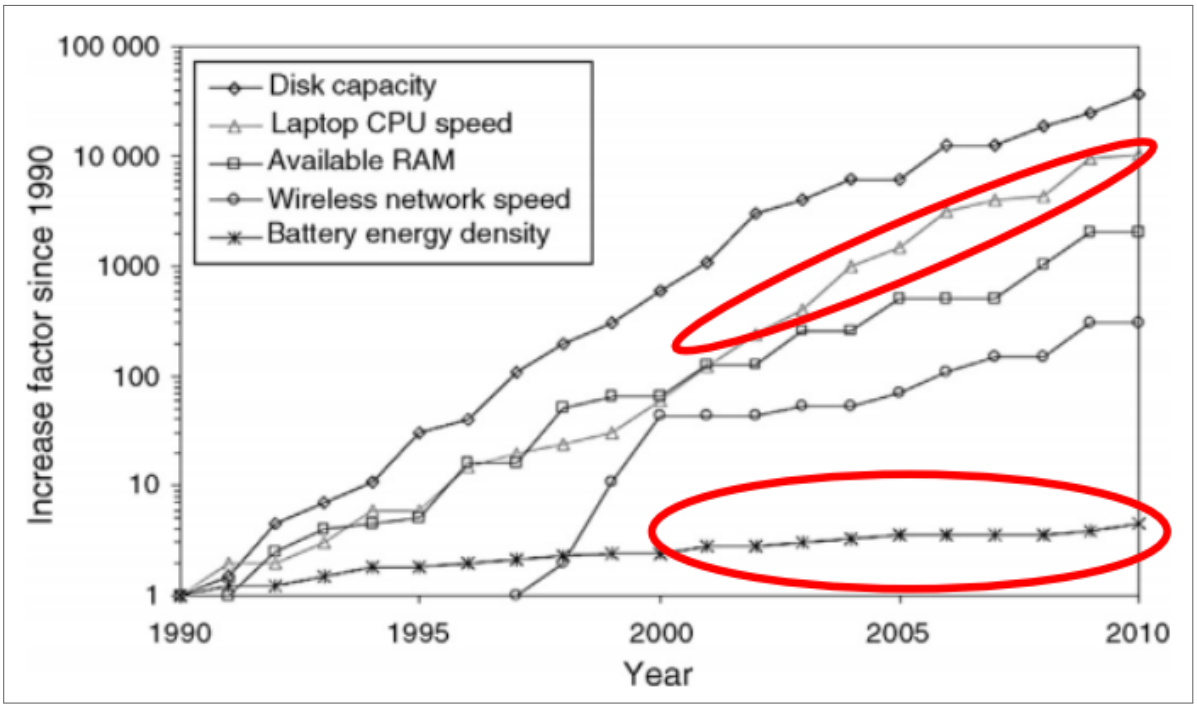
Dispositivos móveis tendem a serem fabricados com um número crescente de funcionalidades, alto desempenho e em um espaço físico relativamente reduzido. Essas funcionalidades incluem tecnologias de comunicação, entretenimento e multimídia, sensores de localização e multiprocessadores, resultando em plataformas com alto consumo de energia.

A mobilidade e portabilidade exigem que sejam alimentados por uma fonte restrita de energia, como uma bateria. Enquanto um Nokia 3210, desenvolvido em 1999, tem energia para funcionar por uma semana, um *smartphone* moderno, como o Samsung Galaxy S6 ou um iPhone 6, geralmente não duram um dia de uso (SEEKER, 2017).

A Figura 1 ilustra a evolução de diversas tecnologias, por exemplo, capacidade dos discos de armazenamento, velocidade da *Unidade Central de Processamento ou Central Processing Unit* (CPU) e densidade das baterias. É perceptível que a evolução das baterias não está acompanhando as demais tecnologias, implicando em dispositivos com maior poder de processamento e armazenamento, mas com baixa densidade de baterias.

Em adição, a popularização de dispositivos inteligentes destacou a Internet das Coisas - *Internet of Things* (IoT), também conhecida como internet dos objetos. A IoT é caracterizada pela comunicação entre objetos que estão presentes no cotidiano. A IoT é formada por dispositivos (físicos ou virtuais) que são conectados por canais sem fio e possuem acesso à internet. Cada dispositivo ligado em uma rede de IoT possui um identificar único, permitindo que enviem e recebam dados de outros dispositivos. A IoT possui algumas características: natureza dinâmica, auto-adaptação e auto-configuração. A maior limitação dos dispositivos que compõem a IoT é a fonte de energia destes dispositivos.

Figura 1 – Evolução das tecnologias.



Fonte: Adaptado de Niroomand e Foroughi (2016).

Dado que há comunicação entre estes dispositivos, grande quantidade de energia é consumida para que eles operem e se comuniquem. Esta característica torna o baixo consumo de energia um fator importante para estes dispositivos (GARG; GARG, 2017). Em complemento, Popli, Jha e Jain (2018) destacam que a maior dificuldade dos componentes da IoT é a disponibilidade de recursos, podendo ser: energia, computação e processamento limitado.

Uma categoria de uso de IoT é as cidades inteligentes (*smart cities*). As cidades inteligentes fazem uso da IoT para observar o ambiente, por exemplo, observar os recursos de uma cidade. Estes recursos podem ser: sistemas de transporte, casas inteligentes, agricultura inteligente e estruturas de saúde. Fazer uso correto de IoT pode permitir melhorias na administração dos recursos de uma cidade inteligente. Com os sensores da IoT, é possível observar e digitalizar o mundo real, possibilitando otimizações no uso dos recursos das cidades e melhoria da qualidade de vida das pessoas. Porém, estes dispositivos são dependentes de baterias para operar e se comunicar, tornando a fonte de energia um recurso essencial para o funcionamento (AKAN et al., 2018).

Equipamentos dependentes de bateria e com diversas funcionalidades têm destacado a importância do desenvolvimento de técnicas para reduzir o consumo de energia. As baterias não conseguem atingir a mesma evolução que os demais componentes dos *smartphones*, resultando em plataformas que consomem mais energia do que as baterias podem prover. A capacidade limitada da bateria destaca uma preocupação importante no desenvolvimento dos dispositivos móveis, sendo preferível plataformas e técnicas que promovam

a extensão do tempo de uso da bateria ao invés de construir baterias maiores. Por exemplo, os *smartphones*, além das funcionalidades de receber e realizar chamadas, também podem executar e gravar vídeos em Alta Definição - *High Definition* (HD), navegar na Internet, executar jogos, conectar nas mídias sociais, transferir arquivos através da rede de dados, fornecer serviços de localização, tirar fotos, dentre outros.

Além dos módulos das antenas e comunicação sem fio, e das telas maiores, a CPU é o componente que mais contribui para o consumo de energia (CARROLL; HEISER, 2010; CARROLL; HEISER, 2013; CHEN et al., 2013; TORCHIANO et al., 2013; TARKOMA et al., 2014), sendo as cargas de processamento excessivas a principal fonte do consumo em dispositivos móveis (MAHESRI; VARDHAN, 2005).

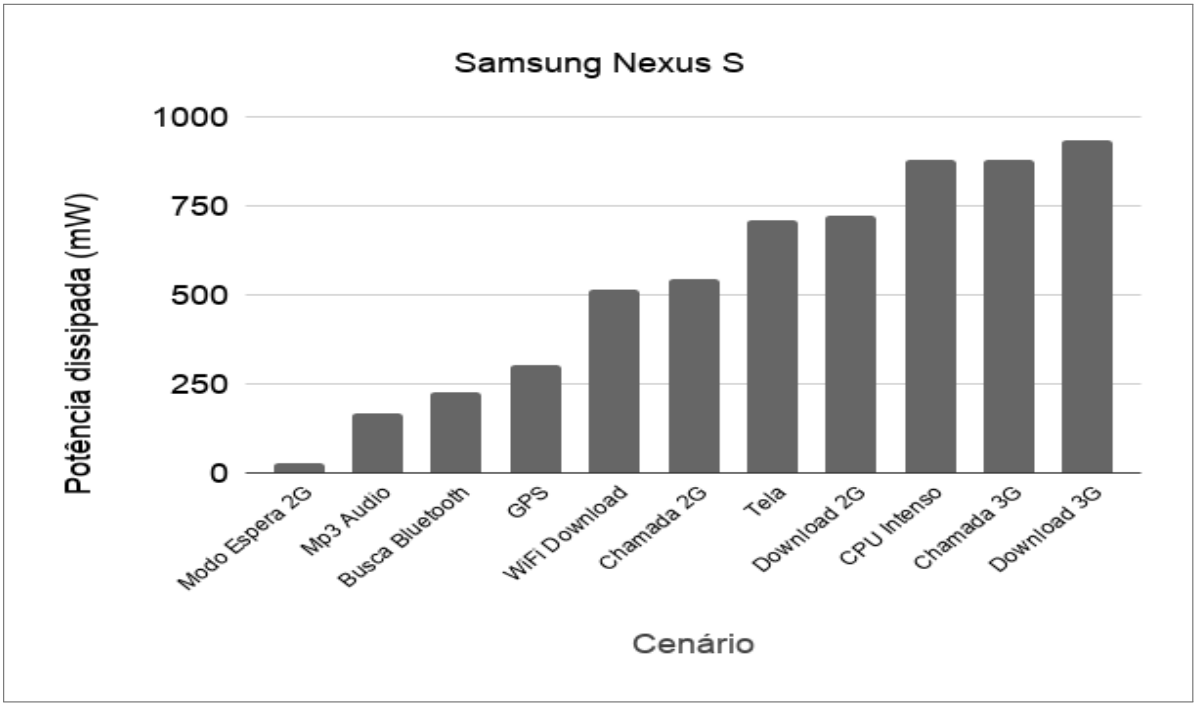
A Figura 2, adaptada de Torchiano et al. (2013), ilustra a potência dissipada por módulo do *smartphone* Samsung Nexus S de acordo com o cenário de uso, cada cenário foi construído de forma a fazer uso intenso de um módulo em específico. Os módulos de redes celulares (Tecnologia de Celular de 3ª Geração - *Third Generation Cellular Technology* (3G), Tecnologia de Celular de 2ª Geração - *Second Generation Cellular Technology* (2G)) são responsáveis por fazerem chamadas telefônicas e estabelecer uma conexão com a Internet. A figura também representa os módulos de CPU, áudio, tela, dentre outros. De acordo com os dados experimentais coletados, os módulos 3G e 2G são os maiores consumidores de energia, bem próximos da CPU e tela. Os dados foram capturados de diversos cenários, cada cenário utiliza aplicações que fazem mais uso de um determinado recurso do dispositivo, por exemplo, CPU, *download* de dados usando redes móveis, etc (TORCHIANO et al., 2013).

Diante do problema do consumo de energia, componentes dos sistemas móveis têm sido aprimorados. A eficiência das baterias tem sido otimizada, provendo mais capacidade de carga e maior tempo de vida útil também, porém, esses avanços não acompanham a rápida evolução dos microprocessadores. Algumas tecnologias foram desenvolvidas em semicondutores para otimizar o consumo de energia: algumas incluem a otimização da tecnologia de fabricação e implementação de técnicas de baixa potência nos circuitos, memórias energeticamente eficientes, redução da tensão de alimentação e frequência de operação da CPU.

As duas categorias de técnicas mais importantes para reduzir o consumo de energia da CPU são: usar vários estados de dormência enquanto o dispositivo está inativo com *Gerenciamento Dinâmico de Potência - Dynamic Power Management* (DPM); e *Escalonamento Dinâmico de Tensão e Frequência - Dynamic Voltage and Frequency Scaling* (DVFS) enquanto está ativo, ou seja, realizando processamento.

DVFS é uma técnica amplamente usada para poupar energia da CPU, prolongando o tempo de uso da bateria. Um algoritmo DVFS reduz o consumo de energia reduzindo a tensão de alimentação da CPU necessária para ativar os elementos lógicos nos transistores. Entretanto, reduzir a tensão da CPU incrementa o atraso no circuito e o processador

Figura 2 – Potência elétrica por módulo no Galaxy Nexus S em diferentes cenários. Cada barra representa um cenário de uso que faz uso intenso de um módulo específico.



Fonte: Adaptado de Seeker (2017).

não pode continuar operando na mesma frequência de *clock*. Reduzir a tensão requer a redução da frequência de operação, sendo que reduções podem levar a atrasos no tempo de processamento do sistema.

A intensidade de uso (maior carga de processamento) da CPU é um fator impactante no consumo de energia (VOGELEER et al., 2014), bem como a forma como a energia é drenada da bateria. Nos processadores de propósito geral, como é o caso dos *smartphones*, não é possível conhecer antecipadamente a carga de processamento ou tempo de execução da tarefa. Conhecer a carga de processamento permite chavear a frequência de CPU que melhor se adapta à carga, podendo economizar energia. A partir da premissa de que não é possível prever com exatidão a carga de processamento e que há diversas aplicações que usam o módulos do dispositivo de forma distinta, realizar a caracterização *offline* não é recomendada, dado o tempo de coleta dos diversos cenários. Diante disto, uma estimação da carga de processamento é mais adequada.

Uma forma promissora de modelar a dinâmica da carga de processamento é utilizando Aprendizado por Reforço - *Reinforcement Learning* (RL) (DAS et al., 2016; SHEN et al., 2013; SHAFIK et al., 2016). Abordagens que usam RL não exigem qualquer conhecimento prévio do ambiente e permitem obter conhecimento a partir da interação com o dispositivo. É possível identificar qual frequência de CPU é mais indicada para um determinado cenário usando RL, porém, é necessário que haja uma função de custo adequada para guiar o

aprendizado do agente.

Além disso, a RL permite a modelagem da relação da carga de processamento, frequência de CPU e energia, mas não limitada a essas características. Essa modelagem propicia o conhecimento para selecionar frequências de CPU que reduzem o consumo de energia e também o controle de variáveis de restrição, como temperatura e tempo de execução.

A função de custo necessária para o funcionamento dos algoritmos de RL pode ser feita de forma estática, a partir de premissas como: a relação direta do aumento da frequência de CPU com o consumo de energia (SHEN et al., 2013) ou de forma dinâmica, fazendo uso de preditores de potência elétrica com métodos lineares (CHEN et al., 2015; TUTOR, 2015; WALKER et al., 2017). Além disso, Carvalho, Cunha e Silva-Filho (2016) demonstraram que existe uma acurácia maior na predição de potência do dispositivo com o uso de métodos não-lineares quando comparado aos lineares.

Dado o exposto, nessa tese, iremos adotar a notação DVFS e DVFS de forma similar, ambas são usadas para se referir ao chaveamento da frequência de CPU, sendo a tensão alterada, quando necessário, pelas rotinas do sistema operacional. Como objetivo principal desta tese, iremos explorar o uso de técnicas de RL e DVFS aplicadas à redução do consumo de energia em *smartphones*, consequentemente, prolongando o tempo de uso.

Nossa principal pergunta de pesquisa é:

- Como descobrir as frequências de CPU que reduzem o consumo de energia global do dispositivo móvel para diferentes tipos de processamento?

Essa pergunta deve ser respondida levando em consideração a grande quantidade de cenários distintos que um dispositivo móvel pode enfrentar, resultado de muitos aplicativos que fazem uso da CPU com diversas intensidades de processamento. A partir disto, propomos uma abordagem autônoma, capaz de adaptar-se aos diferentes cenários em tempo de execução do dispositivo, sem precisar ter conhecido os cenários.

1.1 EXEMPLO MOTIVACIONAL

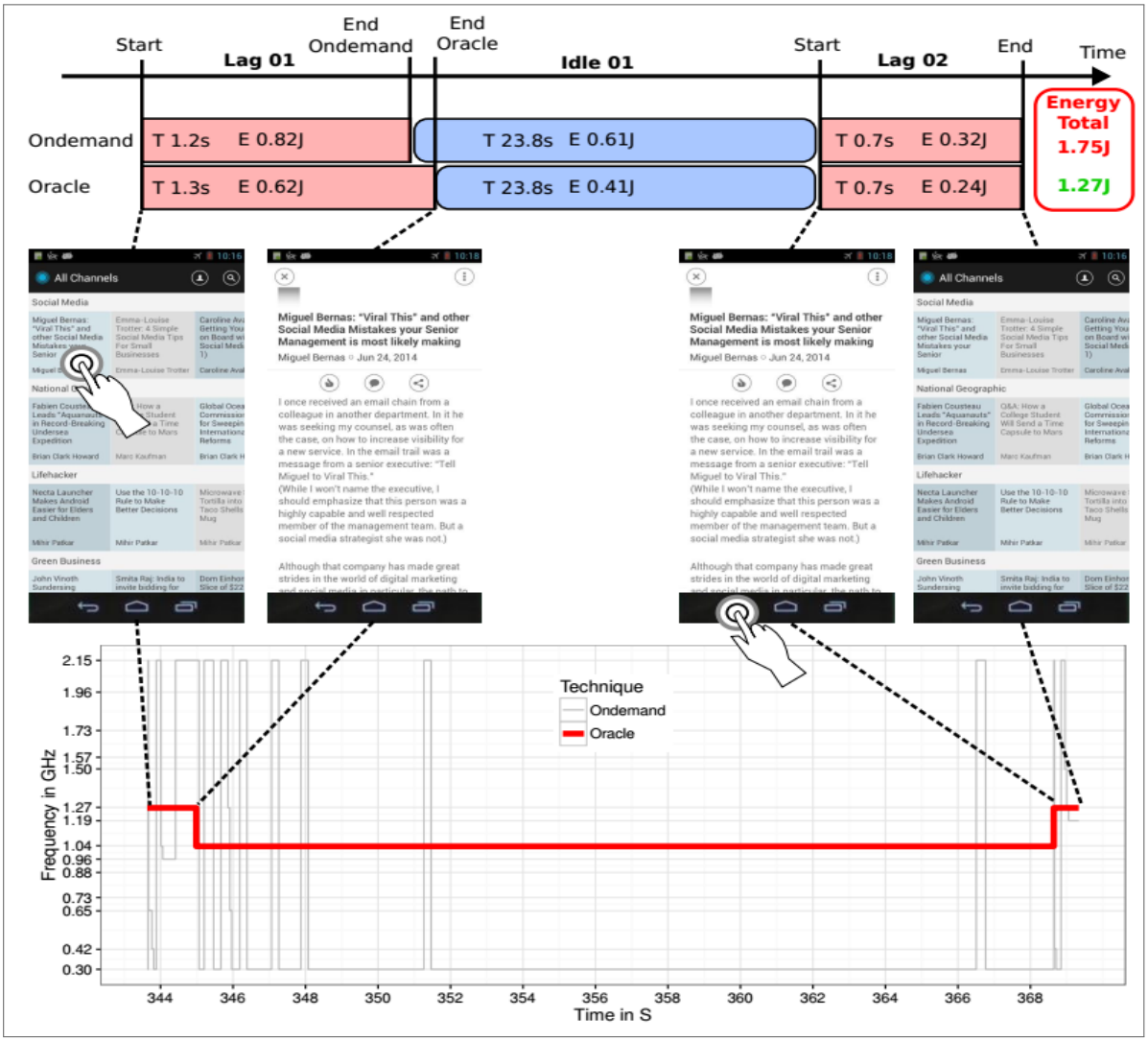
Uma das afirmações que guia esta pesquisa é que realizar DVFS em um ambiente *mobile* interativo tem o potencial de economizar energia, se o algoritmo de escalonamento de frequência da CPU considerar o ponto de vista do usuário, realizando chaveamentos de frequência somente quando necessário.

Em um exemplo retirado de Seeker (2017), a Figura 3 ilustra as escolhas feitas pelo algoritmo de chaveamento padrão em alguns dispositivos móveis, chamado de *Ondemand* (OD) *governor* no Linux/Android e o Oráculo, algoritmo que sempre seleciona as frequências que resultam em economia máxima de energia sem reduzir o desempenho (ou percepção do usuário), quando comparado à execução na frequência máxima.

O exemplo empregado é um leitor de notícias chamado de *Pulse news*. O dedo na figura indica onde houve interação pelo toque com o dispositivo. A primeira captura de

tela ilustra a seleção de um artigo no aplicativo de notícias. A tela é tocada no primeiro artigo, no canto superior esquerdo, abrindo o texto correspondente. Após uma animação de abertura entre o menu e o texto do artigo, a segunda captura de tela corresponde ao texto do artigo, disponível para leitura.

Figura 3 – Comparação do *governor* OD e Oráculo para um exemplo interativo. As capturas de tela no centro mostram duas interações seguidas: seleção de um artigo de notícias e pressionando o botão de voltar enquanto o artigo é exibido para retornar a listagem dos artigos. O gráfico do canto inferior ilustra a seleção de frequência de CPU para os *governors*. Os retângulos e linha do tempo no canto superior indicam valores de tempo de execução e energia.



Fonte: (SEEKER, 2017).

Do ponto de vista do usuário do *smartphone*, a leitura do texto indica o fim do primeiro bloco de interação. Após a finalização da leitura, o usuário apertou o botão de voltar no canto inferior esquerdo da tela. Essa é a segunda interação, resultando na aplicação voltando à listagem dos artigos disponíveis para leitura, acabando quando todos os artigos estão totalmente visíveis.

A ilustração na parte inferior da Figura 3, logo abaixo às capturas de tela, representa as frequências de operação escolhidas pelo *governor* OD (linha cinza) e Oráculo (linha vermelha). O OD aumenta a frequência logo que a primeira interação é detectada, alternando entre a mais alta e a mais baixa enquanto o *smartphone* está abrindo a notícia.

Durante a fase de leitura da notícia, chamada na figura de *Idle 01*, alguns chaveamentos para a frequência máxima são identificados no *governor* OD, próximo à primeira interação com o *smartphone*. Quando a segunda interação começa, o OD escalona para a frequência mais alta novamente para processar a interação.

O *governor* OD pode escolher qualquer frequência disponível na CPU alvo, mas geralmente escalona entre a maior e a menor. Neste exemplo, o *governor* Oráculo somente usa duas frequências distintas: 1,27 GHz durante os intervalos que há interação e 1,04 GHz durante a fase de leitura da notícia, alcançando a economia máxima de energia.

A parte superior da Figura 3 ilustra a sequência temporal da interação, destacando início e fim das interações. O fim é marcado de acordo com a percepção do usuário, ou seja, quando ele sente que o sistema completou o processamento e está disponível. Os retângulos vermelhos indicam a duração dos intervalos, enquanto os azuis representam o tempo necessário para leitura da notícia pelo usuário. Dentro dos retângulos, as marcações com T correspondem à duração em segundos, enquanto E corresponde à energia consumida no período. A caixa no canto superior direito indica a energia total consumida para cada abordagem, OD e Oráculo. Mesmo a diferença no tempo de execução sendo mínima, o *governor* OD usa 27% mais energia que a abordagem Oráculo.

1.2 OBJETIVOS

O trabalho proposto tem como objetivo principal desenvolver uma técnica para redução do consumo de energia em *smartphones* utilizando RL e DVFS com foco no sistema operacional Android, mas genérica o suficiente para ser adaptada para outros sistemas operacionais e plataformas de *hardware*.

Para alcançar o objetivo principal deste trabalho, alguns objetivos específicos foram elencados a seguir:

1. Desenvolver um algoritmo de predição de carga de processamento da CPU para que seja possível aplicar a frequência de CPU mais indicada antes da carga entrar em execução. Além disso, o algoritmo deve detectar, de forma autônoma, mudanças significativas na carga de processamento.

2. Desenvolver e disponibilizar um aplicativo para o sistema operacional Android que colete variáveis de contexto e desempenho do dispositivo para construção de bases de dados de testes.

3. Avaliar a eficácia de métodos não-lineares para predição de potência em *smartphones* com diferentes tipos e intensidades da carga de processamento.

4. Propor e avaliar uma nova função de custo para algoritmos de RL adaptada para dispositivos móveis, objetivando redução do consumo de energia.

5. Integrar o algoritmo de predição da carga de processamento e potência da CPU com a função do algoritmo de RL.

6. Implementar a abordagem proposta no sistema operacional Android.

7. Avaliar a implementação da abordagem proposta utilizando diferentes tipos de processamento e intensidades. Além disso, avaliar os principais algoritmos de escalonamento de frequência de CPU da literatura e indústria em pelo menos duas plataformas distintas de *hardware*.

1.3 MÉTODO DE PESQUISA

Esta seção apresenta a metodologia utilizada na tese, descrevendo uma visão geral dos procedimentos utilizados para a execução da pesquisa. A metodologia apresentada nesta tese pode ser facilmente estendida para outros tipos de trabalhos que envolvam análises energéticas em dispositivos móveis.

A metodologia utilizada no desenvolvimento de um projeto define a forma como as atividades são realizadas, o ambiente experimental e a forma de interação entre as etapas de desenvolvimento. O entendimento da metodologia aplicada ajuda a perceber como o problema foi solucionado, permitindo a adaptação da solução desenvolvida para aplicação em problemas similares. A estratégia utilizada na metodologia também é capaz de definir a qualidade do projeto, destacando as características da implementação da solução desenvolvida. A metodologia aplicada nesse trabalho envolve o fluxo de desenvolvimento, descrição das técnicas implementadas, métricas de análise e construção do ambiente experimental e bases de dados.

A metodologia utilizada é composta de: levantamento bibliográfico para entendimento dos trabalhos mais atuais; construção do ambiente experimental para avaliar os métodos da literatura e testar as hipóteses de pesquisa envolvendo a análise energética; avaliar a acurácia dos métodos e recursos utilizados pelos algoritmos.

O ambiente experimental serviu de base para a condução de experimentos preliminares que nos guiaram para a produção do método proposto. Além disso, foram construídas diversas bases de dados com informações energéticas e de comportamento do dispositivo, que servem para validar outras pesquisas.

Após isso, foram desenvolvidos alguns protótipos para testar as premissas de economia de energia. Com esses protótipos, foi possível comparar as abordagens da literatura e a

proposta neste trabalho. Uma versão mais detalhada das etapas de validação é encontrada no Capítulo 5. A seguir, apresentamos um resumo das principais etapas:

- Etapa 1: avaliação da acurácia dos métodos de predição da carga de processamento utilizando diversas bases de dados que representam a carga de processamento do dispositivo móvel.
- Etapa 2: avaliação dos algoritmos de regressão para predição da potência utilizada pelo dispositivo a partir de variáveis de contexto do dispositivo móvel.
- Etapa 3: avaliação das funções de custo para guiar o processo de aprendizado do algoritmo de escolha da frequência de CPU que minimiza o consumo de energia do dispositivo. São analisados os recursos utilizados pelos algoritmos e a acurácia de predição da frequência de CPU.
- Etapa 4: avaliação do tempo de execução das diversas etapas da abordagem proposta para certificar que a abordagem é viável para implementação.
- Etapa 5: avaliação do consumo de energia de diversos algoritmos para escalonamento de frequência de CPU utilizando diferentes *benchmarks*.

1.4 ORGANIZAÇÃO DO DOCUMENTO

Este documento, que contém a Tese de Doutorado, encontra-se estruturado da seguinte forma:

O **Capítulo 2** apresenta os conceitos básicos sobre consumo de energia em microprocessadores, abordagens que utilizam chaveamento de frequência para redução do consumo de energia e conceitos básicos sobre RL. Além disso, é discutida a importância da carga de processamento na CPU e os algoritmos para predição da carga, usando abordagens reativas e proativas. O objetivo deste capítulo é familiarizar o leitor no contexto do trabalho e facilitar o entendimento do mesmo.

O **Capítulo 3** apresenta e discute os trabalhos relacionados ao proposto. O objetivo é apresentar as principais contribuições realizadas nos últimos anos no contexto de economia de energia em dispositivos móveis com foco no processador. Além disso, são referenciados os principais trabalhos que serviram como base para o desenvolvimento do trabalho proposto.

O **Capítulo 4** contém a descrição da abordagem proposta, mostrando a estratégia utilizada, o fluxo geral da abordagem e o detalhamento de cada etapa. As etapas incluem predição da carga de processamento; escolha da frequência de CPU, que minimiza o consumo de energia; estratégias para estimação e coleta da potência do dispositivo; e as etapas de implementação da abordagem proposta no SO Android.

O **Capítulo 5** apresenta os resultados obtidos, utilizando o método proposto e as técnicas do estado da arte. Esse capítulo exibe o resultado das métricas utilizadas para os diferentes algoritmos avaliados. Cada etapa da abordagem proposta é avaliada com métricas específicas e, por fim, todas as etapas são integradas em uma abordagem que é comparada com as demais da literatura, identificando a energia relativa consumida pelos métodos.

O **Capítulo 6** apresenta as considerações finais sobre os principais tópicos abordados neste trabalho, incluindo as contribuições alcançadas e indicações para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo apresenta alguns dos conceitos necessário para o entendimento do restante desta tese. Mais especificamente, conceitos relacionados à: DVFS aplicado à eficiência energética para sistemas móveis e interativos; e algoritmos para predição da carga de processamento da CPU e RL para aprender, em tempo de execução, a frequência de CPU que proporciona o menor consumo de energia para o cenário do dispositivo utilizando uma abordagem de tentativa e erro.

2.1 CONSUMO DE ENERGIA EM MICROPROCESSADORES

Nesta seção, explicamos de forma resumida os motivadores do aumento no consumo de energia e como ocorre o consumo de energia nos microprocessadores com tecnologia *Complementary Metal–Oxide–Semiconductor* (CMOS), descrevendo os principais componentes dissipadores de potência elétrica.

É notável que, com o rápido crescimento e propagação dos dispositivos móveis, as últimas décadas foram marcadas como a era da comunicação pessoal e social. O cenário da computação móvel está em constante mudança. Por exemplo, nos *smartphones*, dispositivos com mais funcionalidades são lançados e, consequentemente, com maior consumo de potência elétrica. Entender o consumo de potência nos dispositivos móveis é um desafio essencial para a proposição de técnicas que possibilitem um consumo menor, prolongando o tempo de uso do dispositivo.

DVFS é uma técnica para reduzir o consumo de energia da CPU enquanto o processador está realizando processamento. A abordagem geral reduz a frequência de operação da CPU durante execuções não críticas, reduzindo o consumo de energia. As duas grandes questões dessa abordagem são: identificar execuções críticas e não críticas em uma determinada carga de processamento e prever como as mudanças na frequência da CPU impactam o desempenho do sistema enquanto executa a carga de processamento.

Adicionalmente, consumo de potência em circuitos CMOS é um aspecto importante na concepção de sistemas computacionais, especialmente os sistemas embarcados. Por este motivo, técnicas foram desenvolvidas para reduzir o consumo de energia nos dispositivos móveis. Duas técnicas para gerenciamento de energia bastante difundidas na literatura são: *Power Gating* (PG) e DVFS. A primeira técnica consiste em cortar a tensão que alimenta o circuito após um período de ociosidade, enquanto a segunda diminui a tensão de alimentação, possibilitando reduzir a frequência de operação. A redução da frequência de operação implica na redução da potência dissipada, diminuindo também a temperatura.

CPUs modernas são parte integrante de um *System on a Chip* (SoC); este inclui outros componentes que consomem energia. A potência dissipada advém de duas fontes:

Potência Dinâmica (P_{dyn}) e Estática (P_{stat}) (ou *Leakage*). A potência dinâmica P_{dyn} é caracterizada pelo chaveamento dos transistores *Metal-Oxide-Semiconductor Field Effect Transistor* (MOSFET) dentro do circuito CMOS, permitindo que a corrente flua pelo circuito. A potência estática P_{stat} não depende da frequência de chaveamentos nos transistores, é dependente da tecnologia de fabricação empregada no circuito. A potência total dissipada em um circuito CMOS é a soma da potência dinâmica P_{dyn} e da potência estática P_{stat}, definida como

$$P_{total} = P_{dyn} + P_{stat} \tag{2.1}$$

em que a potência dinâmica P_{dyn} é formada por

$$P_{dyn} = P_{troca} + P_{interna} \tag{2.2}$$

onde

$$P_{troca} = \alpha \; C \; V_{DD}^2 \; f \tag{2.3}$$

Na Equação (2.3), a potência consumida é proporcional a capacitância (C), tensão (V_{DD}^2) e frequência (f) empregada no circuito, mostrando que reduzir a frequência de operação leva a uma redução da potência dissipada pelo circuito. A potência dinâmica interna $P_{interna}$ possui valores muito baixos relativos ao valor total da potência dinâmica P_{dyn} em circuitos com tecnologia nanométrica, sendo assim, pode ser considerada desprezível (WESTE; HARRIS, 2011).

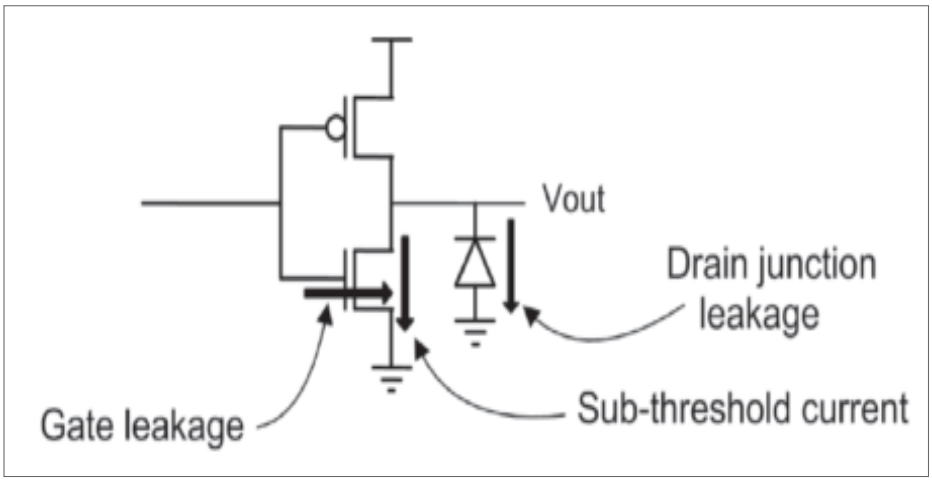
A potência estática P_{stat} é composta por:

- *Sub-threshold Leakage*: corrente que flui do dreno para a fonte. Esta é considerada o componente que causa maior dissipação de potência (WESTE; HARRIS, 2011).
- *Gate Leakage*: corrente produzida pelo efeito de tunelamento.
- *Gate Induced Drain Leakage*: corrente produzida pelo alto campo no dreno, fluindo do dreno para o substrato.
- *Reverse Bias Junction Leakage*: corrente produzida pela criação de pares portadores nas regiões de depleção.

A Figura 4 ilustra três das quatro principais fontes de dissipação de potência estática em circuitos CMOS, e são elas: *Sub-threshold Leakage*, *Gate Leakage* e *Gate Induced Drain Leakage*, sendo essa última ilustrada na figura por *drain junction leakage*.

De agora em diante, será considerada a potência dinâmica P_{dyn} como sendo composta somente pela potência dinâmica de troca P_{troca}. Além disso, a potência estática P_{stat} será considerada como composta somente pela P_{stl} (do inglês, *Sub-threshold Leakage*).

Figura 4 – Exemplificação das fontes de dissipação de potência estática em um circuito CMOS.



Fonte: (KEATING et al., 2008).

A P_{stl} aumenta exponencialmente com o aumento da temperatura. Mais detalhes sobre o consumo de potência em circuitos CMOS podem ser consultados no livro de Weste e Harris (WESTE; HARRIS, 2011).

Nos últimos anos, várias estratégias foram desenvolvidas para economizar energia nos processadores, sendo dentro das seguintes áreas de pesquisa: DVFS, DPM e gerenciamento de temperatura. Essa última objetiva reduzir a temperatura do processador para economizar energia.

Na sequência, é necessário esclarecer os conceitos de Energia e Potência. Energia é a potência dissipada por um componente em uma unidade de tempo, enquanto a energia consumida é dada pela equação:

$$E = \int_0^T P(t)dt \tag{2.4}$$

em que o valor da integral da $P(t)$ (potência instantânea) durante um intervalo de tempo T é o valor da energia consumida. A potência média é dada por

$$P_{média} = \frac{E}{T} = \frac{1}{T} \int_0^T P(t)dt \tag{2.5}$$

Em circuitos eletrônicos, a potência instantânea é o produto da $I(t)$ (corrente instantânea) e a $V(t)$ (tensão instantânea), como em

$$P(t) = I(t)\ V(t) \tag{2.6}$$

A $P_{total}(t)$ (potência instantânea total) é proporcional à soma das $I_{total}(t)$ (correntes dos transistores), dada uma tensão constante V_{DD}, tal que

$$P_{total}(t) = V_{DD}\ I_{total}(t). \tag{2.7}$$

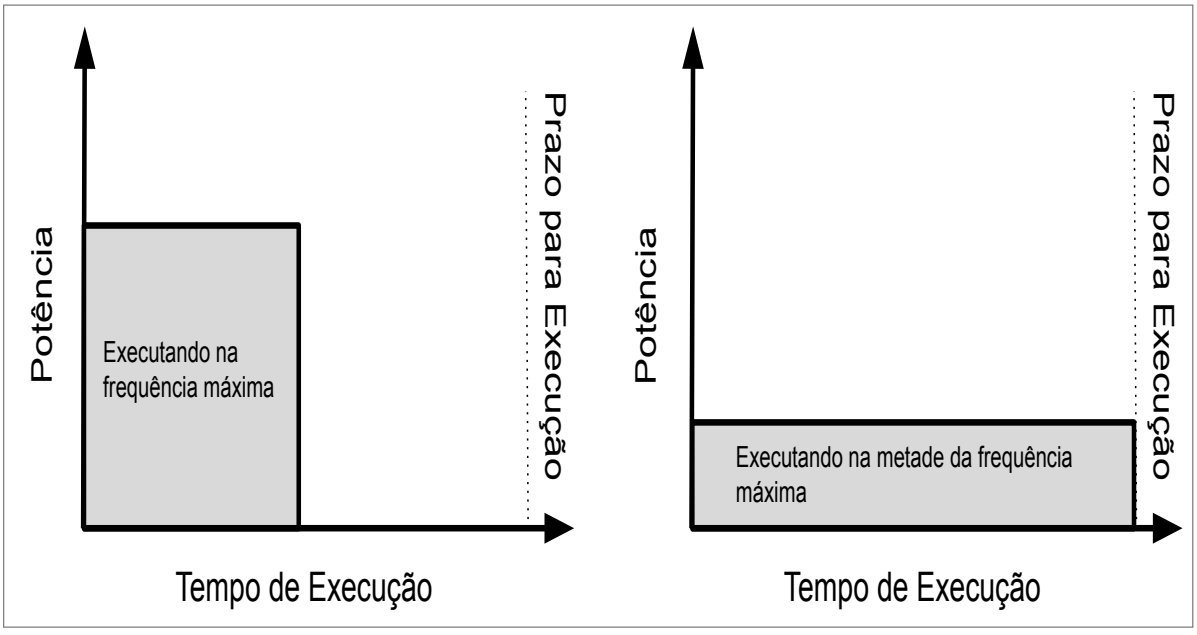
Diante do exposto, foi demonstrado o impacto da potência estática em microprocessadores modernos, tal como é perceptível que a maior dissipação de potência implica em um maior aquecimento do processador, podendo o fator temperatura ser também controlado utilizando DVFS para economizar energia.

2.2 GERENCIAMENTO DE ENERGIA

Nesta seção, é demonstrado como pode ser realizado o gerenciamento de energia em dispositivos móveis com foco no processador e as principais técnicas para redução do consumo de energia. Além disso, exemplos de algoritmos de gerenciamento de frequência da CPU são demonstrados.

A ideia principal presente na abordagem DVFS está ilustrada na Figura 5. Partimos da observação, em termos de energia, se é mais eficiente terminar a execução da tarefa o mais rápido possível e, então, pôr a CPU em algum estado de dormência ou postergar a execução em uma frequência menor, até o prazo máximo de execução. No lado esquerdo da figura, a tarefa é executada na frequência máxima da CPU, terminando antes do prazo máximo da tarefa. No lado direito, a frequência de CPU é reduzida, possibilitando uma redução na tensão fornecida e, então, redução na energia total consumida pela execução da tarefa.

Figura 5 – A mesma tarefa é executada em dois esquemas diferentes de DVFS. O lado esquerdo da figura ilustra a tarefa sendo executada na frequência máxima da CPU, terminando a execução antes do prazo máximo de execução. O lado direito ilustra a execução da tarefa usando metade da frequência máxima, terminando no prazo máximo. Dada a relação entre a frequência da CPU e a potência dinâmica, o consumo de energia pode ser reduzido.



Fonte: O autor (2019).

Essa demonstração da abordagem DVFS é simplificada. Os efeitos do escalonamento da frequência da CPU no desempenho e consumo de energia do sistema não podem ser facilmente determinados, muitas complexidades do mundo real precisam ser consideradas. Por exemplo, um grande problema é a natureza imprevisível das cargas de processamento em ambientes interativos. É difícil estimar quais tarefas irão aparecer e em qual intervalo de tempo, além das características dessa tarefa. Sistemas do mundo real mostram um número de casos não determinísticos e anomalias na relação: frequência de CPU, tempo de execução e no consumo de energia da tarefa (VENKATACHALAM; FRANZ, 2005). Além disso, algumas características que podem impactar os sistemas são: consumo de energia total do sistema, temperatura, tempo de execução e trocas de contexto da CPU.

O valor da dissipação de potência de um microprocessador tem uma relação quadrática com a tensão de alimentação do circuito, porém, essa afirmação não é verdadeira para o consumo do sistema inteiro. Uma tarefa pode usar um módulo fora da CPU, como as antenas celulares. Se essa tarefa for postergada enquanto a CPU opera em uma frequência menor, a CPU pode usar menos energia, mas manter o módulo das antenas ativo por mais tempo pode levar a um aumento significativo da energia total do sistema. Também não há uma métrica determinística para saber se terminar a execução da tarefa o mais rápido possível e entrar em estado de hibernação irá consumir menos energia. A seguir, iremos falar brevemente de algumas abordagens envolvendo DVFS.

2.2.1 Abordagens com DVFS

Um dos primeiros trabalhos feitos usando DVFS data de 1994, desenvolvido por Weiser et al. (1994), no qual ele realizou experimentos envolvendo várias técnicas de escalonamento, objetivando reduzir a frequência de operação da CPU e acabar a tarefa mais próximo do prazo final de execução. Foram reportados ganhos entre 50% e 70% após analisar as simulações.

Alguns trabalhos que exploram a abordagem DVFS focam em sistemas de tempo real, sendo as tarefas e os tempos de execução máximos conhecidos antecipadamente, chamado de *Worst Case Execution Time* (WCET) (XIAN; LU; LI, 2007). Alguns dos algoritmos de DVFS desenvolvidos ao longo dos anos fazem uso de um modelo de testes usando microcargas de processamento para estimar a perda de desempenho ao aplicar redução de frequência na CPU, por exemplo, Hsu e Feng (2004), Freeh et al. (2007), Dhiman e Rosing (2009), Saez et al. (2010), Shafik et al. (2016), Das et al. (2016).

Um fator indicativo para perda de desempenho é o tempo para finalização de uma tarefa guiada principalmente por um componente, como memória (*memory-bound*) ou CPU (*CPU-bound*). Se uma tarefa tem intervalos de tempo excessivos, esperando por um acesso à memória, significa que a frequência de CPU pode ser reduzida sem impacto significativo no tempo de execução da tarefa. Do ponto de vista da CPU, entende-se que há uma dependência linear entre a frequência de CPU e o tempo de execução para tarefas que

usam principalmente CPU (*CPU-bound*). Se uma tarefa possui uma combinação de uso de memória e CPU, é necessário encontrar um balanceamento entre essas características. Um dos maiores desafios nesse cenário de pesquisa é detectar, em tempo de execução, se uma tarefa não conhecida previamente é guiada por uso de memória ou CPU.

No trabalho de Venkatachalam e Franz (2005), foi proposta uma divisão dos algoritmos que usam DVFS em três grandes categorias: intervalo, *inter-task* e *intra-task*.

1. Na abordagem baseada em **intervalo**, é medido o tempo em que o processador está ocupado em um intervalo de tempo, ou seja, realizando processamento. Baseado em estatísticas e na medição atual, uma frequência adequada de CPU é selecionada. A maioria das técnicas de escalonamento de frequência de CPU para o sistema operacional Linux utiliza a abordagem de intervalo, como o algoritmo *Ondemand* (PALLIPADI; STARIKOVSKIY, 2006).

2. A abordagem ***inter-task*** opera a nível de tarefas, alocando uma frequência de CPU para determinada tarefa, ou seja, cada tarefa pré-determinada possui uma frequência de CPU que otimiza seu desempenho ou reduz o consumo de energia. A cada mudança de contexto, uma nova frequência é alocada de acordo com a tabela de relação tarefas-frequências (FLAUTNER; REINHARDT; MUDGE, 2001).

3. A abordagem ***intra-task*** aumenta a granularidade da análise, considerando a tarefa e as mudanças que podem ocorrer na carga de processamento dessa tarefa, podendo a carga ser mais intensa em um determinado intervalo de tempo dentro da mesma tarefa. Pesquisadores dividiram uma única tarefa em várias regiões, alocando uma frequência de CPU para cada região (YUAN; NAHRSTEDT, 2003).

Os algoritmos que usam DVFS foram aprimorados para realizar o chaveamento da frequência em um nível mais preciso, permitindo que módulos de memória operem em velocidade diferente do processador, até mesmo que, no mesmo processador, cada núcleo opere em uma frequência diferente. Processadores com múltiplos domínios de frequência foram introduzidos em 2002 (SEMERARO et al., 2002), permitindo que somente os componentes sob alta carga de processamento operassem em uma frequência mais alta. Vários estudos foram conduzidos para aprimorar os métodos DVFS, inicialmente, com processadores de núcleo único (SEMERARO et al., 2002; MARCULESCU, 2004) e, posteriormente, com vários núcleos (TALPES; MARCULESCU, 2005; ISCI et al., 2006; HERBERT; MARCULESCU, 2007; HERBERT; MARCULESCU, 2009; CARROLL; HEISER, 2014).

Adicionalmente, também foi proposta a arquitetura *Big-Little* (ARM, 2013), onde são usados pelo menos dois processadores: um para baixa carga de processamento e outro para cargas mais intensas. Isso permite que o processador de alto desempenho, que consome mais energia, seja desligado em momentos de baixa carga de processamento. Essa arquite-

tura parte do princípio que manter o processador de alto desempenho ativo, mesmo com a frequência mínima de operação, consome mais energia.

Para escolher a frequência de CPU adequada sem comprometer o desempenho do sistema, é fundamental saber informações da carga de processamento que é executada no momento da análise. O trabalho de Dhiman e Rosing (2007) sugere uma divisão de técnicas de DVFS em três áreas: 1) assume-se que os prazos de entrega do processamento, tempos de chegada e carga de processamento são conhecidas antecipadamente; 2) outras técnicas precisam de algum suporte do compilador onde se pode caracterizar seções do código do programa, permitindo que uma determinada configuração de DVFS seja aplicada ao alcançar essa seção; e 3) nenhuma informação a nível de aplicação é conhecida, o gerenciador de energia precisa se adaptar ao fluxo de entrada da carga de processamento e modelar o comportamento da atividade. Assim, nosso trabalho se propõe a atuar na última categoria (3), utilizando algoritmos de predição de carga de processamento e RL para se adaptar às diferentes cargas de processamento.

As abordagens DVFS podem ser classificadas em dois tipos principais: *offline* e *online*. A abordagem *offline* usa conhecimento específico da aplicação, armazenando a melhor frequência de CPU para um determinado intervalo de carga de processamento. A informação armazenada é usada em tempo de execução para descobrir a frequência de CPU mais indicada de acordo com a carga de processamento e aplicação executando (SHAFIK et al., 2016). A abordagem *online* pode ser classificada em **reativa** ou **proativa**, sendo a abordagem reativa caracterizada pela medição da carga de processamento e, em seguida, utilizada uma política de DVFS apropriada. Na abordagem proativa, a carga de processamento é predita e então a política de DVFS apropriada é aplicada.

Em relação à abordagem *offline*, um conjunto de métricas de desempenho é mapeado para uma determinada frequência de CPU que economiza energia. No trabalho de Moeng e Melhem (2010), alguns contadores de desempenho são mapeados para métricas de desempenho da CPU e a relação entre o gasto de energia pela operação de CPU são estabelecidos. AbouGhazaleh et al. (2007) apresentam uma modelagem usando aprendizado supervisionado com a carga de processamento para gerar a política de chaveamento; DVFS é utilizado no chaveamento de frequências da CPU e memória cache. Duas fases são utilizadas: a primeira de treinamento, na qual os contadores de desempenho são mapeados para um estado do sistema (combinação de diferentes contadores de desempenho); e a segunda fase, em que é feito o mapeamento dos estados para a frequência mais adequada.

Na abordagem *online*, Dhiman e Rosing (2007) propõe um algoritmo para o chaveamento de frequência baseado na carga de processamento da CPU. Contadores de desempenho da CPU foram usados para determinar a intensidade da carga de processamento, indicando que altas cargas possuem mais operações de CPU, enquanto baixas cargas são indicativos de operações de acesso à memória, permitindo que a CPU execute em frequências mais baixas.

Para testes, foi utilizada uma placa protótipo XScale PXA270, permitindo a obtenção de quatro contadores de desempenho da CPU simultaneamente. A partir das métricas coletadas, a porcentagem de uso da CPU, representada por μ e pertencente ao intervalo [0,1], onde valores mais próximos de 1 indicam mais instruções orientadas à CPU, enquanto valores próximos de 0 indicam mais instruções de acesso à memória.

O algoritmo de Dhiman e Rosing (2007) se adapta, penalizando frequências que acarretam em maior tempo de execução ou consumo de energia, atualizando os pesos e a probabilidade dessa frequência ser escolhida novamente para esse determinado contexto.

O fator de importância do desempenho ou da economia de energia é indicado pelo usuário, sendo uma das principais contribuições do trabalho a combinação de diversas métricas de desempenho da CPU, resultando na porcentagem de uso, facilitando a caracterização das cargas de processamento. Outra observação importante feita por Dhiman e Rosing (2006) relativo ao que é estimado pela função de custo é o impacto real no ambiente analisado, estabelecendo que um comportamento ruim para a função de custo pode não representar uma perda na execução real do dispositivo, a função de custo é uma estimação.

Na próxima seção, iremos explanar o gerenciamento de energia em uma abordagem dentro do SO Linux.

2.2.2 Abordagem no Linux

O SO Linux, usado amplamente em sistemas embarcados, também é a base do SO Android. O Linux possui um sistema de gerenciamento de energia que tenta aumentar o tempo de uso da bateria. Os algoritmos de gerenciamento de energia são chamados de *governors*, os quais são módulos que executam em nível de *kernel* (MAUERER, 2010; SALZMAN, 2007; KROAH-HARTMAN, 2006). O módulo de escalonamento de frequência da CPU é chamado de *cpufreq*. Os parâmetros podem ser passados para o *kernel* utilizando a interface /sys, no endereço /sys/devices/system/cpu/.

Os *governors* podem ser estáticos ou dinâmicos, sendo os estáticos caracterizados pelo uso de frequência fixa na CPU, enquanto o dinâmico realiza diversos chaveamentos durante a execução da aplicação. Exemplos de técnicas estáticas são: *Powersave* (PS) e *Performance* (PE), sendo o PS configurado para utilizar a menor frequência disponível, enquanto o PE utiliza a mais alta.

Além disso, os *governors* dinâmicos realizam chaveamentos de frequência em tempo de execução, permitindo adaptação à carga de processamento corrente. Os *governors* dinâmicos mais comuns são: OD, *Conservative* (CS), *PegasusQ* (PQ) e *Interactive* (IT). Os *governors* calculam a porcentagem de uso da CPU em períodos de tempo e, se o valor estiver fora de um limite, a frequência da CPU é ajustada de acordo com o algoritmo (*governors Linux, por exemplo*). Também há o *governor Userspace* (US), este algoritmo

permite que qualquer frequência da CPU ser utilizada a nível da camada de aplicação. Os *governors* mais comuns são descritos a seguir:

1. *Powersave*: é escolhida a frequência mínima disponível na CPU.

2. *Performance*: é escolhida a frequência máxima disponível na CPU.

3. *Userspace*: permite que um aplicativo escolha qualquer frequência de CPU disponível. Esse *governor* permite que algoritmos a nível de aplicação possam realizar chaveamentos de frequência.

4. *Conservative*: é escolhida a frequência de acordo com a carga de processamento na CPU. Quando a carga aumenta, a frequência aumenta gradativamente de acordo com o tempo em que a carga alta é mantida, diminuindo quando a carga diminui. Essa abordagem foi desenvolvida com o intuito de economizar energia, porém, também aumenta o tempo de resposta do sistema.

5. *Ondemand*: esse *governor* se comporta de forma parecida com o CS, porém, quando uma carga alta de processamento é identificada, é escolhida a frequência de CPU máxima, retornando à mínima, quando a carga não é mais identificada como alta. OD é usado por padrão em alguns dispositivos dependentes do Linux. O algoritmo original foi proposto por Pallipadi e Starikovskiy (2006) e está resumido no Algoritmo 1.

6. *PegasusQ*: é um *governor* proprietário da Samsung. É basicamente a implementação do OD com suporte a *hotplugging*, permitindo adicionar ou remover processadores/-cores sem precisar reiniciar o SO.

7. *Interactive*: assim como o OD e PQ, esse *governor* realiza o chaveamento de acordo com a carga de processamento na CPU. A diferença entre o OD está na resposta mais rápida ao chaveamento de frequência, enquanto a mantém por mais tempo (CHAN, 2015). Quando o usuário iniciar a interação com o dispositivo, o *governor* OD monitora a carga de processamento por alguns milissegundos antes de alterar a frequência da CPU. IT consegue detectar a mudança de contexto da CPU e utiliza um tempo menor de checagem, em torno de 10 ms, enquanto o OD consome entre 100-300 ms. Essa abordagem pode levar a um maior consumo de energia, dado que vai permanecer em frequências mais altas por mais tempo, mas pode reduzir o tempo de resposta do sistema, melhorando a interação do usuário.

Algoritmo 1: Algoritmo do *governor Ondemand*.

```
1 Repete (para cada iteração do sistema):
2     A cada X milissegundos
3         Obtém a utilização de CPU (CPU_load) desde a última checagem
4         if CPU_load > limite-superior-carga-CPU then
5             Escolhe a frequência máxima disponível na CPU
6     A cada Y milissegundos
7         Obtém CPU_load desde a última checagem
8         if CPU_load < limite-inferior-carga-CPU then
9             Reduz em 20% a frequência de CPU atual
```

Fonte: Adaptado de Pallipadi e Starikovskiy (2006).

Os *governors* OD e CS são algoritmos reativos: assumem que no próximo intervalo de tempo o valor da carga de processamento será a mesma. Esses algoritmos são dependentes da taxa de checagem da carga de CPU e dos chaveamentos, contudo, cada chaveamento possui um custo associado. Realizar vários chaveamentos de frequência pode prejudicar a economia de energia, enquanto realizar poucos pode prejudicar a resposta do sistema. Os algoritmos não possuem ciência do que acontece na camada de aplicação, sendo alheios a qual aplicação está executando ou ao nível de satisfação do usuário sobre a resposta do sistema.

Em resumo, a abordagem *online* proativa têm demonstrado ser melhor em sistemas embarcados do que a abordagem reativa, provendo maiores porcentagens de economia de energia e também sendo mais adaptável a dispositivos distintos. Alguns trabalhos comparam a abordagem reativa OD *governor* (PALLIPADI; STARIKOVSKIY, 2006)) com as proativas (SHEN et al., 2013; SHAFIK et al., 2016; DAS et al., 2016), obtendo economia de energia e melhorias no desempenho. Trabalhos atuais têm usado aprendizado de máquina, especificamente a RL, para aprender a política de DVFS de forma *online*, utilizando o comportamento do sistema (SHAFIK et al., 2016; DAS et al., 2016).

2.3 TIPOS DE APRENDIZAGEM DE MÁQUINA

Nesta seção, serão descritos alguns dos principais tipos de aprendizagem de máquina. No entanto, como essa é uma área bastante ampla, serão citados apenas alguns conceitos necessários para entender onde a RL se enquadra na aprendizagem de máquina. Sendo assim, as categorias de aprendizado de máquina são tipicamente classificadas de acordo com a sua natureza, sendo os tipos de aprendizagem:

1. Aprendizado supervisionado: na aprendizagem supervisionada, o agente inteligente aprende e constrói um mapeamento (funcionalidade de rastreamento) entre entradas (atributos de classe) e saídas (categoria de dados rotulados). Essas saídas são previamente conhecidas e mostram ao agente o que fazer e melhorar o mapeamento. O objetivo é construir uma regra geral que mapeia as entradas para as saídas de modo a obter previsões corretas. Há duas maneiras de resolver a questão do mapeamento, em aprendizado supervisionado: classificação e regressão. A diferença entre elas é de acordo com as classes das instâncias, ou seja, na classificação os rótulos de classes são discretas e na regressão são contínuos. Neste tipo de aprendizado existe algum tipo de informação sobre as relações de causa e efeito entre as ações do sistema (saída) e o estado do ambiente (entrada). Em um sistema de aprendizado supervisionado por um instrutor, existe um conjunto de treinamento contendo pares de estados do ambiente e ações correspondentes, fornecidos pelo instrutor. Desta forma, o algoritmo de aprendizado recebe um conjunto de exemplos de treinamento para os quais os rótulos da classe associadas são conhecidos. Cada exemplo (instância ou padrão) é descrito por um vetor de valores (atributos) e pelo rótulo da classe associada (CHAPELLE; SCHLKOPF; ZIEN, 2009; ZHU; GOLDBERG, 2009).

2. Não-supervisionado: no aprendizado não supervisionado, os dados apresentados ao algoritmo não são rotulados. Em um aprendizado não-supervisionado, ele ocorre apenas através das observações dos estados do ambiente, sem conhecimento prévio. Nessa forma de aprendizado, o sistema reconhece padrões nos dados de entrada mesmo sem nenhuma informação se a saída está correta ou não (RUSSELL; NORVIG, 2003).

3. Semi-supervisionado: é fornecido algum tipo de informação supervisionada, mas não necessariamente para todos os exemplos. Esses algoritmos podem ser uma adaptação de um algoritmo supervisionado, que vai levar em conta exemplos não rotulados durante o processo de aprendizado. (CHAPELLE; SCHLKOPF; ZIEN, 2009).

4. Aprendizagem por Reforço (RL): é um ramo de pesquisa dentro da aprendizagem de máquina que difere da aprendizagem supervisionada e da não-supervisionada. A ideia principal na RL é a habilidade de um agente tomar decisões que afetem o ambiente em que está inserido, obtendo alguma informação útil no aprendizado de como agir nesse ambiente, tomando ações que ensinem a escolher melhores ações no futuro. Na RL se aprende o que fazer a partir da interação com o ambiente (SUTTON; BARTO, 2012).

2.3.1 Aprendizagem por reforço (RL)

Na RL, o agente não é ensinado como agir, diferente das outras formas na aprendizagem de máquina. O agente deve descobrir quais ações lhe retornam uma maior recompensa ou

menor custo. Em alguns casos, as ações podem afetar não somente a recompensa imediata, mas também a situação seguinte e, assim, todas as ações subsequentes.

A ideia principal é capturar os aspectos mais importantes do problema pela interação do agente com o ambiente para atingir um ou vários objetivos. O agente deve ter a habilidade de sentir o ambiente e tomar ações que modifiquem o estado do ambiente. Os conceitos utilizados sobre RL foram retirados do livro de Sutton e Barto (2012).

Em contraste com a RL, na aprendizagem supervisionada, o conhecimento é adquirido a partir dos exemplos da base de dados, não há aprendizado a partir da interação com o ambiente. Em ambientes de interação, geralmente não é possível obter exemplos de determinadas situações que permitam o agente entender o ambiente de forma completa. Em um ambiente desconhecido, o agente deve ser capaz de aprender a partir da própria experiência.

Podemos citar alguns exemplos em alto nível de RL: na natureza, quando um bebê pássaro aprende como mover as asas para voar ou quando humanos praticam a coordenação motora ao tentar colocar leite no cereal. RL também pode ser usada em robótica, onde um braço robô é treinado para pegar objetos. O robô aprende o valor correto da tensão enviada aos motores para mover o braço com precisão. Também pode-se aplicar RL para aprender as regras de jogos de tabuleiro, como *Backgammon*, *Go* ou *Space Invaders* (SEEKER, 2017).

Os principais componentes usados em RL são:

1. Agente: é o responsável por aprender como agir e tomar as ações.

2. Ambiente: é tudo que não está considerado no agente. Agente e ambiente interagem continuamente: o agente escolhendo e executando ações no ambiente, enquanto o ambiente responde às ações executadas e apresenta novas situações ao agente. O ambiente provê uma recompensa ao agente de acordo com a ação executada.

3. Política: a política do agente descreve quais ações estão disponíveis para serem executadas, dependendo do estado atual do agente. Essa política muda durante o tempo para maximizar a recompensa final para as ações executadas.

4. Função de Recompensa/Custo: a função de recompensa descreve o objetivo desejado pelo sistema, objetivando maximizar o valor obtido, enquanto a função de custo objetiva minimizar o valor. Essa função mapeia as ações tomadas e os resultados obtidos a partir dos estados alcançados para um único número. Esse número representa quão bem as execuções de ações foram bem sucedidas. Neste trabalho, usaremos função de custo para descrever custo ou recompensa, exceto quando explicitado o contrário.

5. Função de Valor: enquanto a função de custo indica o bom comportamento da última ação executada pelo agente, a função de valor indica o bom comportamento a longo prazo. É considerado o potencial de futuras recompensas melhores que a atual e é

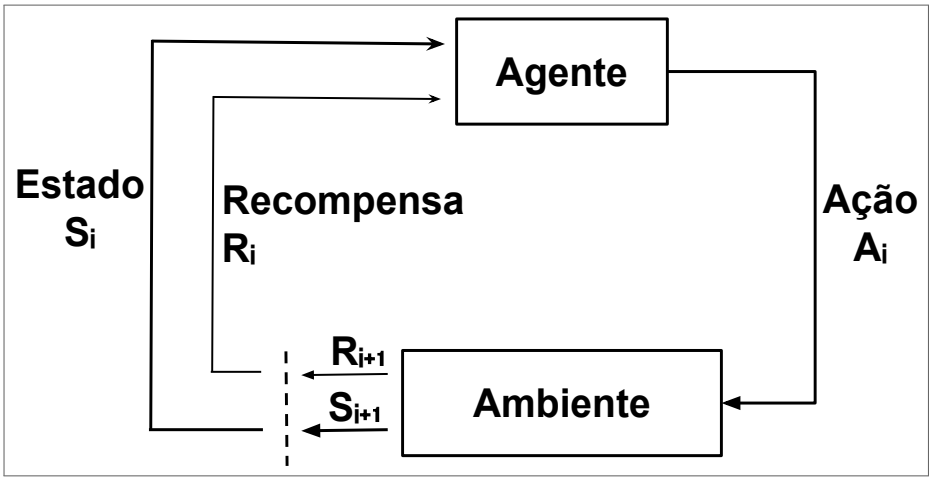
usada para planejamento de ações futuras. Uma política gulosa (*greedy policy*) deve sempre selecionar a ação que leva a uma maior recompensa imediata, essa ação pode levar a mínimos locais, estagnando e não encontrando o ótimo global.

6. Modelo do Ambiente: em alguns sistemas de aprendizado, um modelo pode ser usado para planejar futuras ações. O modelo de ambiente simula o comportamento do ambiente para as ações executadas e pode ser usado para prever ações que minimizem o custo.

Um dos desafios que surge na RL e não é destacado em outros tipos de aprendizagem é o balanceamento entre exploração e explotação. Para obter uma recompensa maior, o agente deve executar ações que ele já testou e sabe que são boas, porém, é necessário descobrir novos caminhos, novas ações, e por isso o agente deve escolher ações que não foram usadas no passado. Exploração é a ação de descobrir novos caminhos, enquanto explotação é utilizar conhecimento prévio do ambiente para tomar decisões que maximizem a recompensa obtida.

De uma maneira mais formal, o agente e o ambiente interagem em uma sequência discreta de tempo, cada unidade da sequência é chamada de época. Na época i o agente recebe uma representação do ambiente chamada estado S_i onde S é a sequência de possíveis estados. O agente executa uma ação A_i no estado S_i onde A é o conjunto de ações possíveis. $A(S_i)$ é o conjunto das possíveis ações no estado S_i. Após a execução de A_i em S_i, o agente recebe uma recompensa numérica (R_{i+1}) onde $R_{i+1} \in \mathbb{R}$ e se encontra em um novo estado (S_{i+1}). Toda a interação é ilustrada na Figura 6.

Figura 6 – Interação do agente com o ambiente.



Fonte: Adaptado de Sutton e Barto (2012).

O agente implementa um mapeamento dos estados com as ações para cada época. Esse mapeamento é chamado de política do agente (π), onde $\pi_i(a|s)$ é a probabilidade de $A_i = a$ se $S_i = s$. Os métodos utilizados em RL especificam como o agente muda sua

política baseado no resultado da execução das ações no ambiente. O objetivo do agente é minimizar o valor do custo que recebe durante toda a execução.

RL é designada para ser implementada em um ambiente livre de modelo, podendo se adaptar sem conhecimento anterior do ambiente. Visando à redução da complexidade dos algoritmos, os estados podem ser discretos e finitos, porém, há casos em que uma função de aproximação é necessária dada a grande quantidade de estados, também é possível o mapeamento de um grande conjunto de estados em um conjunto menor.

2.3.2 Algoritmos independentes de modelo

Um dos mais populares algoritmos em RL é o *Q-learning* (WATKINS, 1989). Ele tem por objetivo buscar a política ótima usando as recompensas passadas. Um requisito importante para a convergência do algoritmo é possuir ou assumir ser um processo Markoviano, onde os estados atuais são independentes dos estados passados, dependendo somente do estado atual e da ação tomada nesse estado. Essa propriedade permite que o algoritmo seja compacto, encorajando o uso em sistemas com restrições de recursos, como os sistemas embarcados.

O algoritmo *Q-learning* observa o ambiente s, executa uma ação a e alcança um novo estado. O novo estado retornará uma recompensa positiva ou negativa, dependendo se a ação executada foi boa ou ruim, de acordo com a função de custo/recompensa. O conhecimento do algoritmo é armazenado em uma tabela chamada *Q-table*. $Q^{\pi}(s, a)$ representa a recompensa esperada ao longo da execução do algoritmo seguindo a política π. A forma procedural do *Q-learning* está descrita no Algoritmo 2.

Algoritmo 2: Algoritmo do *Q-learning*.

Inicia aleatoriamente $Q(s, a)$, onde $\forall s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ e $Q(S_{final}, \cdot) = 0$
Repete (para cada época):
    Inicializa S para o estado inicial
    Repete (para cada iteração da época):
        Escolhe A de S usando a política derivada de Q
        Executa ação A, observa R e S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$;
    Até que $S = S_{final}$

Fonte: Tradução livre de Sutton e Barto (2012).

Segundo o algoritmo do *Q-learning*, primeiramente os valores da *Q-table* são inicializados, sendo os passos seguintes repetidos até a condição de parada ($Q(S_{final}, \cdot) = 0$).

Para cada época, o ambiente é mapeado para um estado S. Uma ação A, disponível a partir do estado S é executada. Após a execução, o ambiente é mapeado para o estado futuro S' e a recompensa R referente à ação A tomada no estado S é calculada. O valor da *Q-table* é atualizado e o estado S se torna o estado futuro S'. O resumo da notação utilizada neste trabalho está na Tabela 1.

Tabela 1 – Termos utilizados nos algoritmos de aprendizagem por reforço neste trabalho.

Termo	Descrição
$\mathcal{S}$ e $\mathcal{A}$	Conjunto de estados e ações possíveis
S e S'	Estado atual e próximo estado
A e A'	Ação atual e próxima ação
s e a	Estado s e ação a qualquer
$A(s)$	Ação a tomada no estado s
$Q(s, a)$	Valor da *Q-table* no estado s e ação a
$Q(S_{final}, \cdot)$	Estado terminal, encerramento da época
$\max_a Q(S', a)$	Valor máximo da *Q-table* dentre todas as ações a partir do estado S'
S_i, A_i e R_i	Estado s, ação a e recompensa r no tempo i
α	Taxa de aprendizagem
γ	Fator de desconto
R	Recompensa

Fonte: O autor (2019).

Os termos S e S' são utilizados para representar os estados presente e futuro, respectivamente. A notação S e S' se refere ao mesmo que S_i e S_{i+1} quando se está no tempo i. No algoritmo, foi mantida a notação utilizada por Sutton e Barto (2012), sendo melhor compreendido o algoritmo quando utilizando termos genéricos e não utilizando referências a tempo.

A equação de atualização do *Q-learning* para um passo é dado por:

$$Q(S_i, A_i) \leftarrow Q(S_i, A_i) + \alpha[R_{i+1} + \gamma \max_a Q(S_{i+1}, a) - Q(S_i, A_i)] \tag{2.8}$$

O valor atualizado altera o valor da *Q-table* referente à ação tomada no estado S_i, atualizando o par $Q(S_i, A_i)$ com a recompensa R_{i+1}. O algoritmo *Q-learning* irá convergir para a política ótima, independente da política de exploração seguida, partindo do pressuposto que cada par estado-ação é visitado um número infinito de vezes e o parâmetro de aprendizagem γ é decrementado de forma incremental (WIERING; OTTERLO, 2012; WATKINS, 1989).

Outro algoritmo bastante difundido na literatura é o *State-Action-Reward-State-Action* (SARSA). A execução é similar à do *Q-learning*, divergindo na equação de atualização do

valor da *Q-table*:

$$Q(S_i, A_i) \leftarrow Q(S_i, A_i) + \alpha[R_{i+1} + \gamma Q(S_{i+1}, A_{i+1}) - Q(S_i, A_i)] \tag{2.9}$$

onde a ação A_{i+1} é a ação executada no estado S_{i+1}. Note que o termo $\max_a Q(S_{i+1}, a)$ do *Q-learning* é substituído pela estimação do valor do estado futuro de acordo com a política vigente. Após a ação ter sido executada, o ambiente é observado e a recompensa R calculada, mostrando se a ação tomada foi boa ou ruim. O valor Q da ação a executada no estado s é atualizado usando a equação de atualização (2.8 ou 2.9). Na equação, o α ($0 \leq \alpha \leq 1$) determina quanto a recompensa afeta o valor atual da *Q-table* para $Q(s, a)$. O fator de desconto γ ($0 \leq \gamma \leq 1$) introduz um viés na política a ser seguida.

A forma procedural do algoritmo SARSA está no Algoritmo 3. Usando o algoritmo SARSA, o agente começa no estado S_i, executa a ação A_i e recebe uma recompensa R_i. Agora no estado S_{i+1}, executa a ação A_{i+1} e recebe a recompensa R_{i+1}, antes de atualizar o valor da *Q-table* para o estado S_i e ação A_i. No *Q-learning* o agente começa no estado S_i, executa a ação A_i e recebe a recompensa R_i, avalia qual o valor máximo de recompensa a ser alcançado para uma ação no estado S_{i+1} e usa para atualizar a *Q-table* referente ao estado S_i e a ação A_i.

Algoritmo 3: Algoritmo *SARSA*.

```
Inicia aleatoriamente Q(s, a), onde para ∀s ∈ S, a ∈ A(s) e Q(S_final, ·) = 0
Repete (para cada época):
    Inicializa S
    Escolhe A de S usando a política derivada de Q
    Repete (para cada iteração da época):
        Executa ação A, observa R e S'
        Escolhe A' de S' usando a política derivada de Q
        Q(S, A) ← Q(S, A) + α[R + γQ(S', A') − Q(S, A)]
        S ← S'; A ← A';
    Até S ser um estado final
```

Fonte: Tradução livre de Sutton e Barto (2012).

Resumindo, o algoritmo SARSA escolhe a ação A' e estado S' e então atualiza a *Q-table*. No *Q-learning*, primeiro atualiza-se a *Q-table* e, então, na iteração seguinte, a próxima ação é selecionada, usando o novo conhecimento da *Q-table*. A nova ação escolhida não é obrigatoriamente igual à ação A' selecionada para atualizar a *Q-table*. Vale destacar que o algoritmo SARSA é um algoritmo *on-policy*, ou seja, ele segue a política que está

aprendendo, enquanto o *Q-learning* é *off-policy*, podendo seguir qualquer política que garanta os requisitos de convergência.

A *Q-table* pode ser inicializada com valores aleatórios ou com conhecimento prévio do sistema. Se o conhecimento anterior existe, pode ser usado na inicialização do algoritmo, economizando iterações. A cada escolha da ação A, algumas opções podem ser empregadas. Uma solução simples é chamada de ϵ-*greedy*, exposta no Algoritmo 4. Há duas fases distintas, chamadas de exploração e explotação, em que a primeira é responsável pela exploração de novas ações para adquirir conhecimento do sistema e a segunda é responsável por utilizar o conhecimento adquirido para escolha da melhor ação.

Algoritmo 4: Estratégia ϵ-greedy.

```
1  função Escolhe_AÇÃO(ε, s) :
2      Gera x aleatório
3      if x < ε then
4          A ← max_a Q(S, a)
5      else
6          A ← ação aleatória
7      end
8      if ε < 1 then
9          incrementa ε
10     end
11     retorna A, ε
```

Fonte: Adaptado de Sutton e Barto (2012).

Para escolha da ação utilizando o algoritmo ϵ-*greedy*, um valor de exploração ϵ deve ser definido; a partir de um x aleatório, é decidido se a ação será de explotação (Algoritmo 4, linha 4) ou exploração (Algoritmo 4, linha 6). Se a ação for de explotação, a melhor ação de acordo com os valores da *Q-table* é executada, caso contrário, uma ação aleatória dentro do conjunto possível de ações pertencentes a estado do sistema é executada.

No próximo tópico, iremos demonstrar a execução do algoritmo *Q-learning*, utilizando um exemplo prático envolvendo a locomoção de um robô dentro de um ambiente finito.

2.3.3 Exemplo prático usando *Q-learning*

Nesse exemplo prático usando o *Q-learning*, vamos demonstrar, utilizando um robô que está em um tabuleiro, como atingir o objetivo, utilizando uma função de recompensa fixa. O objetivo é alcançar o quarto da posição (5) no tabuleiro. De acordo com a Figura 8, o robô está em algum dos quartos (0 até 5) e somente pode se locomover onde há passagens.

Por exemplo, se está no quarto (1), somente pode se mover para o quarto (3) ou (5). O exemplo foi adaptado de McCullock (2013). Para esse exemplo, definimos os valores $\alpha = 1$ e $\gamma = 0.8$ por motivos de ilustração.

Na Figura 8, é ilustrada a localização do robô no mapa do lado esquerdo e no lado direito da figura estão os valores da *Q-table*, usada para escolher as ações do agente. A Figura 7 ilustra os valores usados para a função de recompensa. Neste exemplo, o objetivo é maximizar os valores, ou seja, receber 100 é melhor que receber 0.

Figura 7 – Valores de recompensas da *Q-table* para nosso exemplo.

$R=$

	A_0	A_1	A_2	A_3	A_4	A_5
S_0	-1	-1	-1	-1	0	-1
S_1	-1	-1	-1	0	-1	100
S_2	-1	-1	-1	0	-1	-1
S_3	-1	0	0	-1	0	-1
S_4	0	-1	-1	0	-1	100
S_5	-1	0	-1	-1	0	100

Fonte: Adaptado de Mnemstudio.org (2018).

No passo (1) da Figura 8, o robô começa no quarto de número três e ainda não há conhecimento dentro do agente, por isso, todos os valores da *Q-table* são 0. No passo (2), o agente moveu para o quarto 1 e do lado direito é a *Q-table* atualizada. Como não houve nenhuma recompensa ganha, dado que somente ao chegar no quarto 5 que haverá recompensa, podemos notar que não houve alteração na *Q-table*, mas o valor da *Q-table*(3,1), referente ao valor da *Q-table* do estado 3 e ação 1, foi atualizado. A equação (2.10), utilizando a equação de atualização do *Q-learning* (2.8), demonstra o cálculo realizado.

$$\begin{aligned} Q(S_i, A_i) &\leftarrow Q(S_i, A_i) + \alpha[R_{i+1} + \gamma \max_a Q(S_{i+1}, a) - Q(S_i, A_i)] \\ &= Q(3,1) + 1\{R(3,1) + 0.8 \max_a [Q(1,3), Q(1,5)] - Q(3,1)\} \\ &= 0 + 1\{0 + 0.8 \max_a [0,0] - 0\} \\ &= 0 \end{aligned} \tag{2.10}$$

Dando continuidade ao exemplo, no passo (3), o agente move-se do quarto 1 para o quarto 5, onde há recompensa, de acordo com a tabela R de recompensas. No lado direito, há a *Q-table* atualizada após o agente mover-se para o quarto cinco. Pode-se notar que o valor da posição da *Q-table*(1,5) foi atualizado para 100. A equação (2.11) indica o cálculo

realizado.

$$\begin{aligned} Q(S_i, A_i) &\leftarrow Q(S_i, A_i) + \alpha[R_{i+1} + \gamma \max_a Q(S_{i+1}, a) - Q(S_i, A_i)] \\ &= Q(1,5) + 1\{R(1,5) + 0.8 \max_a [Q(5,1), Q(5,4), Q(5,5)] - Q(1,5)\} \\ &= 0 + 1\{100 + 0.8 \max_a [0,0,0] - 0\} \\ &= 100 \end{aligned} \tag{2.11}$$

Em uma nova época, quando o robô estiver novamente no quarto 3, será atualizado de acordo com a equação (2.12). Nessa iteração, será usado o valor de Q(1,5) atualizado na iteração anterior. Podemos notar que a recompensa não é máxima, mas é um caminho plausível para se chegar ao objetivo e esse conhecimento ficará guardado na *Q-table*.

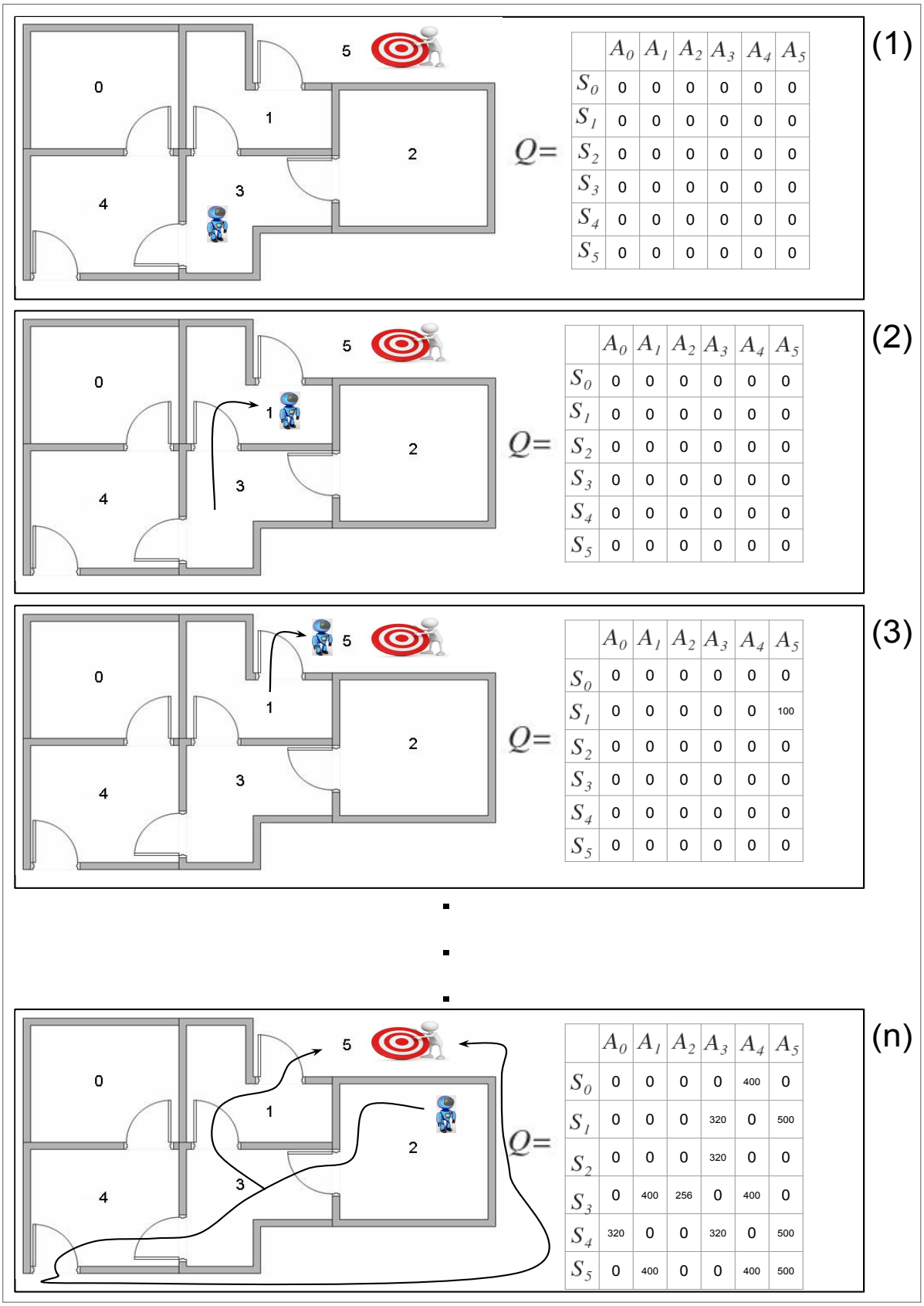
$$\begin{aligned} Q(S_i, A_i) &\leftarrow Q(S_i, A_i) + \alpha[R_{i+1} + \gamma \max_a Q(S_{i+1}, a) - Q(S_i, A_i)] \\ &= Q(3,1) + 1\{R(3,1) + 0.8 \max_a [Q(1,3), Q(1,5)] - Q(3,1)\} \\ &= 0 + 1\{0 + 0.8 \max_a [0,100] - 0\} \\ &= 80 \end{aligned} \tag{2.12}$$

Após n iterações, podemos ver na parte inferior da Figura 8 que o agente está localizado no quarto 2 e a *Q-table* já possui vários valores, adquiridos durante o aprendizado. De acordo com o conhecimento adquirido, o melhor caminho para alcançar o objetivo (quarto 5) dado que o agente está no quarto 2, é mover-se para o quarto 3 e então ir para o quarto 1 e então para o 5, sendo os dois caminhos iguais do ponto de vista de recompensas.

Na Figura 9, é possível analisar outra perspectiva, em forma de grafo, dos caminhos possíveis e valores de recompensa de acordo com a *Q-table*. As setas indicam caminhos possíveis entre os estados e os valores indicam recompensas em forma de porcentagem. Dado que o robô está no quarto 2, as setas mais escuras ilustram o caminho que provê maior recompensa até o objetivo.

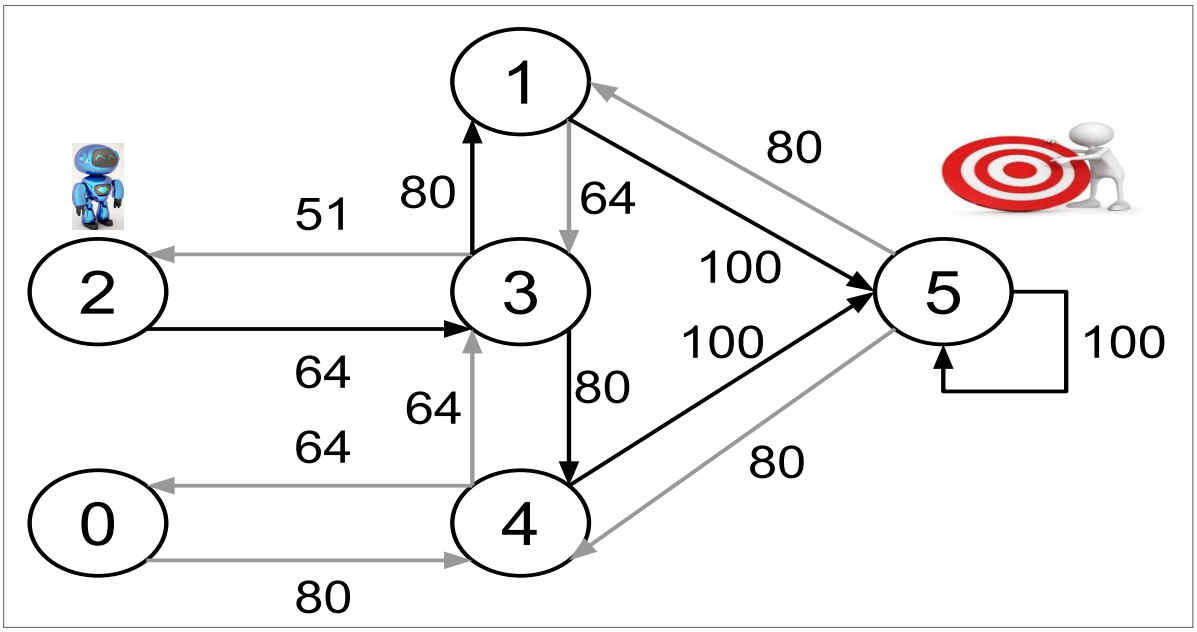
Na próxima seção será descrito o processo de predição de carga de processamento e os algoritmos utilizados para realizar a predição.

Figura 8 – Exemplo de aprendizado usando *Q-learning*.



Fonte: Adaptado de Mnemstudio.org (2018) .

Figura 9 – Ilustração com grafos do aprendizado do agente.



Fonte: Adaptado de Mnemstudio.org (2018).

2.4 PREDIÇÃO DA CARGA DE PROCESSAMENTO

Nesta seção, é destacada a importância da predição da carga de processamento e como isso pode ser utilizado para prover eficiência energética nos dispositivos móveis. Além disso, é exemplificado como realizar a predição da carga de processamento da CPU.

Nos processadores de propósito geral, como é o caso dos *smartphones*, não é possível conhecer antecipadamente a carga de processamento ou tempo de execução da tarefa com precisão. Há duas soluções para compreender as cargas de processamento: é possível criar perfis *offline* de todas as tarefas que irão ser executadas, obtendo as características da carga de processamento em cada momento da execução, resultando em um perfil completo da tarefa. Outra solução é realizar estimações da carga de processamento durante o tempo de execução da tarefa.

Uma implicação da primeira técnica, criar perfis de todas as tarefas, é justamente que há dependência de conhecer antecipadamente todas as tarefas que serão executadas, além da quantidade de memória disponível para armazenar essas informações. Além disso, a complexidade do processo da criação dos perfis é aumentada em cenários reais de uso do dispositivo. Irá haver atrasos do escalonador de tarefas, SO e outros processos que estarão executando em segundo plano. Essas são características que inviabilizam o emprego dessa técnica em *smartphones*, com restrições de processamento e memória. Dadas as circunstâncias, realizar predições das cargas de processamento é o mais indicado.

Deve-se achar um balanceamento entre custo computacional e acurácia do algoritmo de predição, de forma que não afete significativamente os recursos do *smartphone*, sendo a primeira característica mais restritiva que a acurácia de predição.

Geralmente, há uma quantidade de frequências de operação disponíveis na CPU dos *smartphones* (valor abaixo de 10 frequências). Isso permite que mesmo com um erro de predição, de 5% por exemplo, seja alocada a frequência de CPU correta para a carga de processamento vigente.

O algoritmo *Exponential Weighted Moving Average* (EWMA) é comumente usado na literatura para resolução do problema de predição da carga de processamento (MAEDA-NUNEZ, 2016; DAS et al., 2016; SHAFIK et al., 2016). No trabalho de Maeda-Nunez (2016), é implementada uma abordagem inteligente usando o algoritmo EWMA e RL no contexto de decodificação de quadros de vídeo, sendo que a abordagem inteligente tem somente um atraso de, aproximadamente, $0,6\%$ do tempo total gasto para decodificar o quadro. No trabalho de Fathabadi et al. (2015), a abordagem proposta usando EWMA e incluindo os chaveamentos de frequência utiliza, em média, $41,7$ms, equivalendo a $1,51\%$ do total de tempo usado para a decodificação de um quadro operando a 300 MHz.

O trabalho de Shen et al. (2013) demonstrou que a abordagem proposta teve um custo de aproximadamente $2,5$ms, enquanto a abordagem padrão OD no Android é $1,5$ms e a abordagem de Shafik et al. (2016) é aproximadamente $2,5$ms. As abordagens de Shen et al. (2013) e OD possuem um atraso menor, dado que são algoritmos mais simples, necessitando de um número menor de decisões de controle e menos variáveis envolvidas. É necessário encontrar o balanceamento correto entre complexidade e atrasos de execução, dado que algoritmos mais sofisticados possuem um atraso maior. Diante do exposto, ficou claro que a abordagem utilizando EWMA é viável para ser implementada em *smartphones*.

Dando continuidade à explicação do algoritmo EWMA, de maneira mais formal, assumindo que S_i representa o estado do sistema no instante de tempo atual, o valor predito para o instante de tempo $(i+1)$ é dado por:

$$W_{i+1}^{pred} = \lambda W_i + \sum_{\ell=t-D}^{i} (1-\lambda) W_\ell \quad , \tag{2.13}$$

onde W_i é a carga de processamento mensurada para o instante de tempo i, λ é o coeficiente de média variável e D é o tamanho da janela de valores de observações passadas. Altos valores de λ priorizam valores mais recentes, enquanto baixos valores priorizam a média dos valores da janela no histórico.

Uma versão mais sofisticada do EWMA foi proposta por Capizzi e Masarotto (2003), chamada de *Adaptive Exponential Weighted Moving Average* (AEWMA), onde o parâmetro λ que controla a relevância dos valores antigos é modificado em tempo de execução para valorizar menos o histórico quando uma mudança de carga de processamento for detectada, permitindo uma adaptação mais rápida a mudanças. Quando uma mudança é detectada, o parâmetro λ é incrementado para tornar os valores recentes mais relevantes na predição. Após algumas iterações, λ é ajustado para o seu valor inicial usando uma função de decaimento exponencial.

Diante do exposto, é possível idear sobre a utilização de técnicas que envolvem previsões em séries temporais para o problema proposto da predição da carga de processamento, mas isto foge ao escopo deste trabalho.

Na próxima Seção, mais trabalhos relacionados ao proposto são analisados, destacando características importantes para o gerenciamento energético dos dispositivos móveis e economia de energia.

3 TRABALHOS RELACIONADOS

Este capítulo tem como objetivo apresentar o estado da arte em relação as principais técnicas para alcançar eficiência energética em dispositivos móveis. Além disso, as principais técnicas dos últimos anos são descritas e comparadas entre si, destacando a importância da abordagem proposta por este trabalho.

3.1 ABORDAGENS DVFS

Sistemas computacionais podem ser representados como um sistema de três camadas: a camada de aplicação, o SO e o *hardware*. O propósito dos sistemas é prover funcionalidades para o usuário enquanto executa aplicações que fazem uso das outras camadas mais abaixo. Além disso, a camada do SO é responsável pelo gerenciamento das aplicações, alocando tempo de execução e gerenciando os recursos de *hardware*, incluindo o controle do gerenciamento de energia.

Apesar de o gerenciamento de energia ser responsabilidade do SO, algoritmos de gerenciamento podem estar em qualquer uma das três camadas, ativando núcleos da CPU e gerenciando a frequência de operação. Exemplos de algoritmos implementados na camada de *hardware* são: Sinha et al. (2011) e Prabha e Monie (2007). Em Sinha et al. (2011), os autores usam o algoritmo SARSA, um dos algoritmos mais conhecidos em RL, sendo aplicado para controle do tempo de dormência do periférico.

Na camada do SO Linux, o gerenciamento de energia é feito a nível de *kernel*, utilizando os *governors*. Por outro lado, o SO também oferece interfaces para o controle do gerenciamento de energia a nível de aplicação, reduzindo a complexidade da implementação (BHATTI; BELLEUDY; AUGUIN, 2010) e aumentando o tempo de resposta do sistema (HOLMBACKA et al., 2016). Assim como no Linux, o Android também utiliza a mesma nomenclatura, dado que é Linux.

Em resumo, independente de onde é implementado, o algoritmo de gerenciamento de energia, também chamado de *Run-Time Manager* (RTM) (MAEDA-NUNEZ, 2016), é responsável por tomar as decisões, priorizando a economia de energia, desempenho, redução da temperatura ou outra variável de restrição. As decisões de escolha da frequência da CPU são executadas usando DVFS e baseadas em restrições como: altas temperaturas, tempo de execução excessivo, alto consumo de energia, dentre outras. Para tomar decisões eficientes, é necessário observar o ambiente constantemente, contabilizando métricas de desempenho, consumo de energia e temperatura.

Trabalhos que utilizam aprendizagem de máquina treinam um modelo para predizer a melhor frequência de CPU para minimizar o consumo de energia e manter ou maximizar a métrica de desempenho. Alguns dos trabalhos desenvolvidos usando DVFS com apren-

dizagem de máquina objetivam economia de energia, sendo que sua maioria usa como ambiente experimental computadores *desktop* e servidores. Alguns desses trabalhos são: Jung e Pedram (2010), Moeng e Melhem (2010), Rountree et al. (2011), Kana, Chanb e Tsea (2012), Shen et al. (2013), Islam e Lin (2015), Wang et al. (2017).

A principal estratégia, tanto para *desktop*, servidores e embarcados, é coletar variáveis da arquitetura como contadores de desempenho ou utilização da CPU e acessos à memória (JUNG; PEDRAM, 2010; MOENG; MELHEM, 2010; ROUNTREE et al., 2011; CHANG; LIANG, 2011; KANA; CHANB; TSEA, 2012; MIFTAKHUTDINOV; EBRAHIMI; PATT, 2012; CARROLL; HEISER, 2014; DAS et al., 2016; SHAFIK et al., 2016; CARVALHO; CUNHA; SILVA-FILHO, 2016; CARVALHO; CUNHA; SILVA-FILHO, 2017).

As características capturadas do ambiente são usadas para criar modelos estáticos de predição, que são aplicados durante a execução do algoritmo. Diante do amadurecimento e evolução dos sistemas embarcados e *smartphones*, alguns trabalhos tem considerado mais os sistemas embarcados: Chang e Liang (2011), Li et al. (2013), Carroll e Heiser (2014), Das et al. (2016), Shafik et al. (2016), Carvalho, Cunha e Silva-Filho (2016), Carvalho, Cunha e Silva-Filho (2017).

Neste trabalho, iremos caracterizar as abordagens de acordo com a taxonomia construída pelo autor, considerando os trabalhos analisados na seguinte estrutura:

Estratégia de aprendizado utilizada:

- *Offline*: É caracterizada pelo uso de algum procedimento *offline* para caracterizar os componentes, por exemplo, a relação entre utilização de CPU e as frequências de CPU disponíveis. Após a caracterização, o conhecimento (heurísticas) é usado durante a execução do algoritmo, sendo que não há novo aprendizado.
- *Online*: É caracterizada pela aprendizagem durante a execução do algoritmo, mesmo que tenha tido aprendizado *offline*, esse conhecimento é refinado durante a execução.

Forma de aplicar a política aprendida:

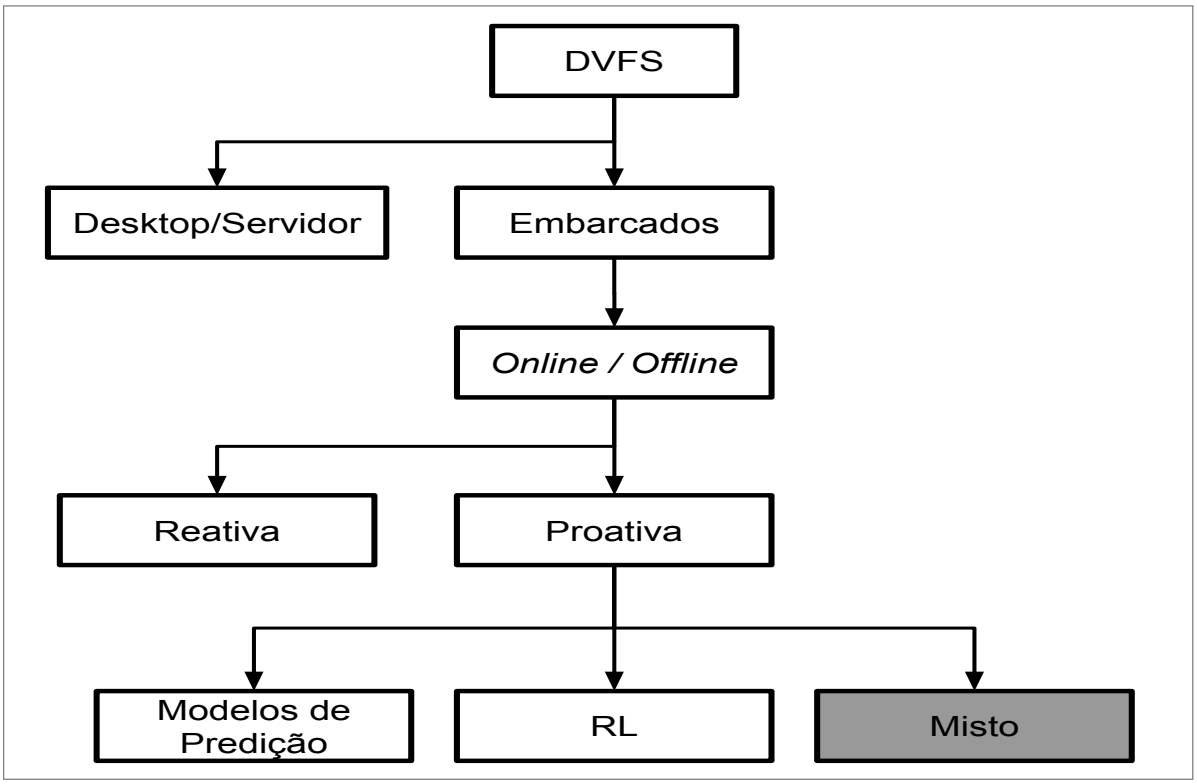
- Reativa: é realizada medição de algum evento ou variável de desempenho seguida pela aplicação da política de escalonamento de frequência de CPU.
- Proativa: realiza alguma predição (por exemplo, da carga de processamento) e aplica a frequência de CPU para um tempo futuro, antes da carga de processamento chegar para processamento. Essa abordagem também pode fazer leitura de alguma variável de desempenho ou evento, mas é utilizada a predição da carga de processamento em um instante futuro para aplicar a política de escalonamento de frequência de CPU.

Além disso, na abordagem proativa, podemos criar mais três categorias, sendo:

- os trabalhos que utilizam somente modelos de predição, como utilização de CPU ou frequência de CPU;
- os trabalhos que utilizam RL, mas não utilizam modelos de predição; e
- os trabalhos com abordagem mista, que utilizam tanto os modelos quanto RL.

Um resumo das abordagens está ilustrado na taxonomia da Figura 10. Para criar a classificação apresentada na Figura 10, foram analisados os trabalhos relacionados e consideramos somente os trabalhos que usam dispositivos móveis/embarcados como ambiente experimental. Dado o exposto, nosso trabalho se enquadra na abordagem mista, utilizando predição e RL.

Figura 10 – Taxonomia DVFS dos trabalhos consultados.



Fonte: O autor (2019).

Na seção 3.3, estão os trabalhos que utilizam a abordagem *online*. Na próxima seção, iremos abordar os trabalhos com abordagens *offline*.

3.2 ABORDAGENS *OFFLINE*

Alguns exemplos de trabalhos dessa categoria foram citados no Capítulo 2: AbouGhazaleh et al. (2007) e Moeng e Melhem (2010). Além deles, Carroll e Heiser (2014) fazem uso de DPM e DVFS, medindo o consumo de potência de dois *smartphones* e criando um modelo que é incorporado a um *governor* para o Linux, chamado de Medusa. O Medusa

foi implementado a nível de *kernel* no Android e comparado com os *governors* padrões disponíveis no SO Android dos *smartphones* testados. Foi demonstrado que é melhor para a redução do consumo de energia, ativar mais núcleos do processador ao invés de aumentar a frequência da CPU acima de um determinado ponto de corte. Este trabalho não aborda a mudança de carga de processamento ou predição dessa carga. Medusa é um RTM estático que proporcionou economia de energia de até 26% quando comparado aos *governors* padrões do Android.

Os *smartphones* possuem diversas aplicações, onde cada aplicação exercita módulos distintos, como CPU, localização, transferência de dados, dentre outros. Os módulos podem ser dependentes, ou seja, um modulo pode fazer uso de outro. Ao realizar DVFS, diminuindo a frequência de operação da CPU, acaba-se prolongando o tempo de execução, mantendo o módulo dependente ativo por mais tempo e, consequentemente, consumindo mais energia. As abordagens *offline* consultadas avaliam o consumo de energia apenas do componente, não avaliando o comportamento geral do sistema, incluindo o consumo de energia. Avaliar o comportamento energético de todo o sistema é uma abordagem que se apresenta mais correta da perspectiva do usuário, pois reduzir o consumo de energia de um componente que causa dependência de outros módulos, necessitando deixar os módulos dependentes ativos, pode não ser uma abordagem que efetivamente economiza energia para o usuário final do dispositivo.

As abordagens *offline* são boas para os casos de teste em que são avaliadas, embora possam ser executadas em outras cargas de processamento, as abordagens *offline* não se adaptam às especificidades das cargas de processamento não conhecidas e nem se permitem reaprender qual frequência de CPU é melhor para esse novo cenário. Dado isso, as abordagens *online* são mais indicadas, pois podem se adaptar a novos cenários.

3.3 ABORDAGENS *ONLINE*

Guo e Potkonjak (2016) utilizam uma abordagem DVFS a nível de aplicação. Usando aprendizagem de máquina, é construído um modelo de predição da carga de processamento usando a carga de processamento estimada e o progresso da decodificação como entradas do modelo. No trabalho de Guo e Potkonjak (2016), somente é analisado decodificações de vídeo. Foi demonstrado que, mesmo com uma taxa de chaveamento de frequência relativamente alta (um segundo) quando comparado aos *governors* a nível de *kernel* (10-300ms), foi possível conseguir economia de energia.

Uma taxa de chaveamento muito alta pode levar a um consumo excessivo de energia, pois cada chaveamento gera picos de potência que corroboram para o aumento do consumo de energia. Diante disso, o mais indicado é encontrar o balanceamento entre as restrições impostas pela abordagem, como desempenho, por exemplo. Foi alcançada uma economia média de energia de 40,1%, quando comparado à execução normal. É usada uma etapa *offline* para caracterização da carga de processamento e treinamento do modelo, e uma

online, onde é feita a predição da carga de processamento em tempo de execução do algoritmo.

Nos esquemas convencionais de Gerenciamento Dinâmico de Temperatura - *Dynamic Thermal Management* (DTM) usando dispositivos móveis, a temperatura é monitorada a partir dos sensores. Ao alcançar uma temperatura limite superior, o DTM é invocado para reduzir a temperatura. Uma abordagem comum em DTM é usar DVFS para controlar a temperatura. O uso convencional de DTM pode resultar em diminuição de desempenho por conta da redução da frequência da CPU.

De acordo com Kim, Kim e Chung (2015), os *smartphones* atuais usam processadores de alto desempenho para facilitar a execução de várias funcionalidades, porém, como efeito colateral, maior desempenho inevitavelmente implica em maior consumo de energia, eventualmente levando a problemas no controle da temperatura. Para aliviar os problemas de aquecimento, Kim, Kim e Chung (2015) sugerem o uso de DVFS aplicado a DTM. O esquema pode ser usado em duas maneiras distintas: otimização de energia e otimização de desempenho. Quando ativado o modo de economia de energia, foi alcançada uma economia de 12,7% em média, sem perda de desempenho. Ao ativar a opção de otimização de desempenho, esse foi melhorado em 6,3% enquanto o consumo de energia teve uma redução de 6,7% em média. A abordagem é mesclada com o *governor* OD, sendo que o método proposto é ativado quando a temperatura atinge uma temperatura crítica e o OD ativado quando a temperatura é controlada para um nível seguro.

RL é uma técnica promissora para modelar a dinâmica das cargas de processamento, uma abordagem que não necessita nenhum conhecimento prévio do ambiente e permite adquirir a frequência de CPU mais indicada para um determinado cenário usando a interação com o ambiente. RL difere da aprendizagem supervisionada e da não supervisionada. Aprendizagem supervisionada depende de exemplos externos ao algoritmo, quando usada sozinha, não é indicada para aprendizado a partir da interação (SUTTON; BARTO, 2012).

Em um ambiente desconhecido, onde o agente deve ser habilitado a aprender a partir da própria experiência, é frequentemente inalcançável obter exemplos do comportamento desejado que são corretos e representativos para todos os ambientes que o agente deve atuar (SUTTON; BARTO, 2012). Na aprendizagem não supervisionada, como na supervisionada, é necessário ter dados estruturados com entradas e saídas bem definidas. Entretanto, para os *smartphones*, é enfrentado um ambiente dinâmico que pode tornar os modelos criados a partir das amostras coletadas inadequados para especificar o ambiente.

RL utiliza uma função de custo para guiar o processo de aprendizagem, relacionando a carga de processamento, frequência de CPU e o consumo de energia. Essa relação é aprendida durante a execução do algoritmo e permite escolher a frequência de CPU que minimiza o consumo de energia (DAS et al., 2016; SHEN et al., 2013; SHAFIK et al., 2016). A função de custo na RL pode ser construída com conhecimento *offline*, usando a premissa que altas frequências de CPU implicam em mais consumo de energia, por exemplo, Shen

et al. (2013) ou usando estimadores lineares para predição de potência, como em Chen et al. (2015), Tutor (2015), Walker et al. (2017).

No próximo tópico, as abordagens que utilizam aprendizagem por reforço serão discutidas.

3.3.1 Trabalhos com aprendizagem por reforço

Diante da necessidade de uma abordagem que se adapte a ambientes dinâmicos, o uso da RL se mostra promissor. Os trabalhos que utilizam RL aprendem conforme a execução do algoritmo e se tornam mais robustos com o passar do tempo, adquirindo conhecimento e refinando as predições.

Shen et al. (2013) utilizam uma abordagem mista que envolve o uso de DPM e DVFS para o gerenciamento de energia do dispositivo. Além disso, propõem uma abordagem *online*, usando o algoritmo *Q-learning*, obtendo economia de energia e redução no tempo de execução da tarefa.

Shen et al. (2013) não consideram mudanças na carga de processamento ou nos requisitos de desempenho da aplicação, por exemplo, taxa de exibição de quadros em decodificadores de vídeo ou tempo de carregamento para páginas Web. Os autores assumem que todas as frequências de CPU que proporcionam o menor consumo de potência para cada intervalo discreto de ciclos de CPU são conhecidas antes da execução do RTM. Essa suposição exige que para cada sistema embarcado, a relação entre carga de processamento e frequência de CPU que proporciona menor consumo de potência seja conhecida e armazenada. Essa abordagem é bastante complexa de ser generalizada para o uso comum dos *smartphones*, onde há uma diversidade de plataformas de *hardware*, aplicações e cargas de processamento. Para evitar um mecanismo de coleta de potência sofisticado, foi usada uma abordagem mais simples, onde os autores assumiram que o consumo de energia no tempo t ($C_{e,t}$) é proporcional à frequência de CPU utilizada, definindo:

$$C_{e,t} = \frac{|f_t - f_e \times \mu|}{f_{max} - f_{min}} \tag{3.1}$$

onde f_e é a frequência de CPU que alcança o menor valor de consumo de energia para uma determinada carga de processamento μ. f_t e μ_t são frequência de CPU e carga de processamento (ciclos de CPU) respectivamente, durante o período t, f_{max} e f_{min} são os valores de frequência de CPU máximo e mínimo suportado pelo dispositivo. O μ deve ser definido antes da execução do RTM, em uma etapa *offline*. Nossa proposta oferece uma abordagem autônoma, onde o μ e f_e são adquiridos durante a execução do RTM, usando o modelo de predição não-linear proposto.

O trabalho de Tian et al. (2018) propôs uma abordagem para gerenciamento de energia colaborativo em múltiplos dispositivos. Realizando a combinação do conhecimento de diversos dispositivos, espera-se uma seleção de frequência de CPU que economiza energia,

podendo ser completada em um tempo reduzido, quando comparado ao aprendizado individual do dispositivo. As políticas geradas entre os dispositivos são formadas de forma local e compartilhadas de forma global, entre todos os colaboradores. A política ideal de DVFS é escolhida usando o conhecimento local e global.

O conhecimento de qual frequência de CPU utilizar, dependendo da carga de processamento, é compartilhado periodicamente entre os dispositivos, utilizando uma abordagem em nuvem. O gerenciamento colaborativo de energia consiste, principalmente, do compartilhamento do conhecimento de DVFS entre os dispositivos.

Dentro da abordagem de Tian et al. (2018), é utilizado RL, mais especificamente o algoritmo *Q-learning*, onde o mapeamento da carga de processamento para os estados do *Q-learning* são discretizados em oito intervalos. Para guiar o processo de aprendizagem, é utilizada uma abordagem estática, similar ao trabalho de Shen et al. (2013). A função de recompensa do *Q-learning* é o consumo de energia do dispositivo no tempo t, calculado de acordo com:

$$C_t = \frac{f_{max} - f_t}{f_{max} - f_{min}} \tag{3.2}$$

onde f_{max} e f_{min} são os valores de frequência de CPU máximo e mínimo suportado pelo dispositivo. f_t é a frequência de CPU executando no tempo t.

A abordagem proposta por Tian et al. (2018) foi implementada em um ambiente simulado, usando o simulador JADE (MAEDA et al., 2016). Para calcular o impacto energético, foi utilizado o modelo de predição baseado no McPAT (LI et al., 2009). Além disso, foram utilizados *microbenchmarks* da suíte COSMIC. Resultados experimentais mostraram que a abordagem colaborativa pode alcançar, em média, 10% de economia de energia quando comparada a abordagens do estado da arte.

A abordagem falha em considerar que o conhecimento global, adquirido de diversos dispositivos, pode ser usado em dispositivos distintos sem adaptação. Cada *smartphone* possui um perfil de consumo distinto e isso deve ser considerado na adaptação do conhecimento. Usar políticas de dispositivos distintos pode ser prejudicial para a economia de energia do *smartphone* alvo.

Além disso, o simulador utilizado avalia o consumo de energia do processador e dos chaveamentos de frequência, não levando em consideração variáveis que impactam o consumo de energia do *smartphone*, como temperatura e outros módulos que precisam estar ativos para o processamento desejado. O simulador opera a nível de processador e não considera o consumo total do sistema, mesmo sendo esse o que mais impacta no tempo de uso do dispositivo pelo usuário.

Pasricha, Donohoo e Ohlsen (2015) propuseram o arcabouço AURA para otimizações energéticas para dispositivos móveis. A ideia principal é explotar o intervalo entre a percepção e interação do usuário com o dispositivo móvel para reduzir a frequência da CPU,

além disso, há uma vertente para redução do brilho da tela sem impactar a percepção do usuário.

A partir das métricas de tempo de interação do usuário com o sistema, foi desenvolvido um classificador bayesiano para categorizar o tipo de interação dentro de um intervalo predefinido de intensidades, variando de interação muito lenta até interação muito rápida, contendo sete níveis de intensidade. Foram realizados testes com usuários reais, onde cada usuário foi instruído a usar o *smartphone* enquanto seu tempo de reação era coletada para a criação do perfil de interação. Para coletar as variáveis de tempo de interação foi criado um aplicativo executando como um Android *service*, esse é ativado a partir dos eventos globais do SO Android.

A validação da proposta foi realizada, tanto em ambiente simulado quanto em padrões de usuários reais, foi alcançada economia energética de até 29%, variando de acordo com o dispositivo móvel, perfil do usuário e aplicação em uso. Além disso, se um aplicativo ainda não está classificado, a classificação é feita em tempo de execução, exigindo um tempo de treinamento para coletar os tempos de interação do usuário na aplicação.

Um possível melhoramento para o trabalho de Pasricha, Donohoo e Ohlsen (2015) é aumentar a quantidade de usuários analisados e o tempo de interação, permitindo a criação de padrões de interação mais precisos. Além disso, não foram analisados os modelos de predição de potência para ambientes desconhecidos, implicando em uma possível falha na predição correta de potência para novos cenários, afetando todo o aprendizado posterior, incluindo a RL e o chaveamento correto da frequência de CPU. Adicionalmente, os *smartphones* utilizados executam a versão do Android 2.2 e 2.3.3, sendo consideradas obsoletas, visto que a versão 8.1 foi lançada no final de 2017. Após a versão 5.0 foram implantadas melhorias significativas no consumo de energia dos dispositivos que utilizam o SO, alterando o mecanismo Dalvik pelo *Android Runtime* (ART) (DEVELOPER, 2015).

Em outro exemplo, Shafik et al. (2016) usam o algoritmo *Q-learning* para aprender durante a execução da aplicação a melhor frequência de CPU para as diferentes cargas de processamento, usando uma comunicação direta com a camada de aplicação. A aplicação avisa ao RTM qual aplicação está executando e os requisitos de desempenho necessários. Esta abordagem mostrou ser melhor quando comparada ao *governor Ondemand*, nativo do SO Linux, e a abordagem com DVFS de Shen et al. (2013). O trabalho de Shafik et al. (2016) não contempla a abordagem DPM, controle de temperatura ou predição de potência e, mesmo assim, atingiram uma redução considerável de até 33% no consumo de energia.

No trabalho de Das et al. (2016), a mudança de carga de processamento é uma das principais preocupações. Os autores propõem uma abordagem para detecção da mudança na carga de processamento de forma autônoma, usando divergência de *Kullback–Leibler* (KL). O algoritmo *Q-learning* é usado para melhorar o controle de temperatura da CPU, reduzindo a temperatura média, temperatura de pico e ciclos térmicos, quando comparado

às abordagens da literatura. Essa abordagem não explora a predição de potência ou DPM.

Por fim, Wang et al. (2017) abordam a frequência de menor consumo no contexto de processadores multi-núcleos, explorando processadores com até 64 núcleos usando o simulador JADE (MAEDA et al., 2016). Os custos de potência e energia também são simulados, usando a ferramenta McPAT (LI et al., 2009). Este trabalho propôs um algoritmo usando aprendizagem por reforço modular, usando o *Q-learning* modular para predizer a frequência global que minimiza energia para cada núcleo, individualmente. O impacto da temperatura não é abordado no trabalho de Wang et al. (2017). Nos resultados de Wang et al. (2017), altas frequências de CPU são indicadas como as que proporcionam menor consumo, porém, a potência estática não é analisada, mesmo sendo um fator relevante no impacto do consumo de energia e diretamente dependente da temperatura. Foi obtida uma economia de até 28% no consumo de energia quando comparado à versão individual do *Q-learning*.

Por fim, as abordagens *offline* se mostraram inadequadas para o ambiente dinâmico que são os *smartphones*, sendo as abordagens *online* mais adequadas para o cenário de estudo. Dentro das abordagens *online*, as proativas mostraram consumir menos energia que as abordagens reativas. Em conclusão, nas abordagens proativas, abordagens que fazem uso misto, tanto de modelos de predição de potência como de RL tem potencial para proporcionarem mais economia de energia que suas antecessoras, sendo objeto de estudo deste trabalho.

3.4 RESUMO DOS TRABALHOS

A Tabela 2 exibe um resumo dos principais trabalhos analisados nesta tese. A coluna aprendizado demonstra o tipo de técnica utilizada no aprendizado da política de escalonamento de frequência, podendo ser *online* ou *offline*. Na coluna política é exibido o tipo de política usada para aplicar o escalonamento de frequência, podendo ser reativa ou proativa. Os *governors* padrões no sistema Android são reativos, enquanto alguns trabalhos recentes demonstraram que os algoritmos que utilizam a abordagem proativa são mais eficientes.

Predição de carga (Pred. Carga) representa qual técnica é utilizada para predição da carga de processamento. A coluna ajuste λ ilustra como o λ é refinado para uma versão adaptativa do algoritmo de predição AEWMA. A coluna detecção de mudança (Det. Mudança) contém a técnica utilizada para detecção da mudança da carga de processamento.

A coluna predição de potência (Pred. Pot.) mostra se alguma estratégia é usada para prever a potência do dispositivo, informando se o método de predição é linear ou não-linear, adicionalmente, a coluna ilustra se a abordagem suporta o uso de sensor de potência (Pot. Real (sensor)) para alimentar o algoritmo de aprendizagem. Esse sensor está se tornando um componente comum nos *smartphones* atuais, utilizar o sensor de potência

para ensinar ao algoritmo inibe os erros de predição presente nos modelos de predição de potência.

Ademais, o ambiente experimental utilizado é descrito na coluna Amb. Exp. onde é destacado o tipo de ambiente utilizado nos experimentos, como *smartphones* ou protótipos como infraestrutura de validação ou se foram utilizados computadores pessoas ou servidores. Além disso, a coluna economia de energia (Eco. Energia) exibe a porcentagem de economia de energia alcançada pelas abordagens. Em sequência, a coluna Ano exibe o ano de publicação do trabalho analisado.

Tabela 2 – Comparativo entre os trabalhos relacionados.

	Aprendizado	Política	Pred. Carga	Ajuste λ	Det. Mudança	Pred. Pot.	Pot. Real (sensor)	Amb. Exp.	Eco. Energia	Ano
Tian et al. (2018)	*Online*	Reativa	n.d.	n.d.	n.d.	Linear	Não	Móvel	Até 10%	2018
Wang et al. (2017)	*Online*	Reativa	n.d.	n.d.	n.d.	n.d.	Não	Servidor	Até 28%	2017
Das et al. (2016)	*Online*	Proativa	EWMA	n.d.	KL	n.d.	Não	Móvel	n.d.	2016
Shafik et al. (2016)	*Online*	Proativa	AEWMA	Decai. exp.	APP $\leftrightarrow$ *Governor*	n.d.	Não	Móvel	Até 33%	2016
Pasricha, Donohoo e Ohlsen (2015)	*Online*	Reativa	n.d.	n.d.	n.d.	Linear	Não	Móvel	Até 29%	2015
Carroll e Heiser (2014)	*Offline*	Reativa	n.d.	n.d.	n.d.	n.d.	Não	Móvel	Até 26%	2014
Shen et al. (2013)	*Online*	Reativa	n.d.	n.d.	n.d.	Linear	Não	Pessoal	n.d.	2013

Fonte: O autor (2019).

É notável que a maioria dos trabalhos consultados utilizam abordagens com aprendizado *online*, pois é sabido que as abordagens *offline* precisam de procedimentos custosos para aprender a se adaptar aos diferentes contextos, tornando-se um tipo de abordagem menos relevante para o cenário atual dos dispositivos móveis. Além disso, poucas abordagens utilizam predição da carga de processamento, porém, essa estratégia se mostrou eficiente em alguns dos trabalhos mais recentes expostos nesta tese.

Ademais, nenhum dos trabalhos consultados utiliza um sensor de medição de potência elétrica integrado ao algoritmo de escalonamento de frequência. Essa oportunidade é explorada por este trabalho. Utilizamos abordagens com predição da potência e também utilizando o sensor de potência nativo do *smartphone*. Walker et al. (2017) demonstra que alguns dos principais simuladores utilizados por alguns dos trabalhos consultados possuem uma margem de erro considerável na predição da potência, julgamos essa margem de erro ineficiente para avaliar o trabalho proposto. Para contornar essa restrição, utilizamos a medição a partir do sensor de potência em um *smartphone* real, evidenciando que as otimizações energéticas alcançadas são reais para o usuário do dispositivo móvel.

Por fim, apresentamos uma adaptação da Tabela 2 com as características do trabalho proposto nesta tese ao final da Seção de resultados.

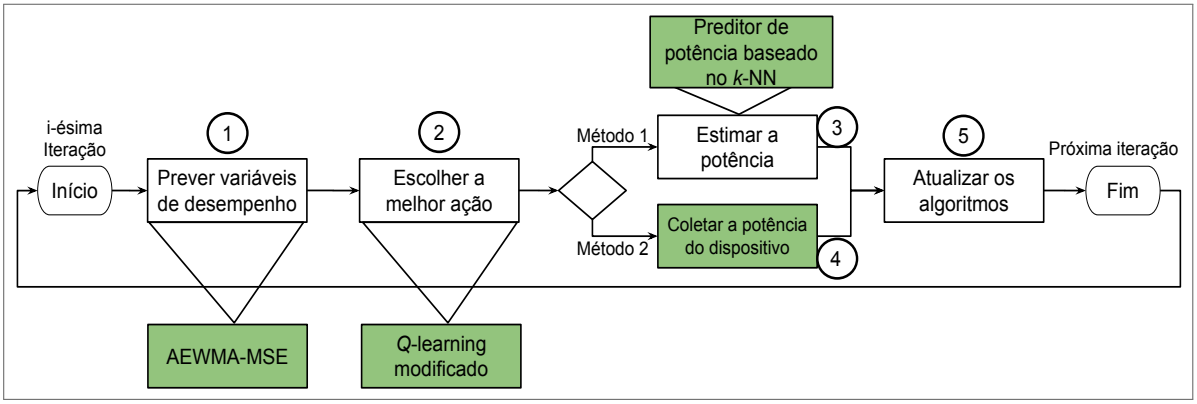
4 TRABALHO PROPOSTO

O objetivo principal deste trabalho é desenvolver uma técnica para redução do consumo de energia em dispositivos móveis, com foco nos *smartphones*. Para alcançar o objetivo, aplicamos uma versão modificada do algoritmo *Q-learning*, fazendo uso de aprendizagem por reforço, chaveamento de frequência da CPU e predição da carga de processamento e potência do dispositivo móvel.

A abordagem é implementada como um serviço na camada de aplicação, em segundo plano, do Android e executa de forma indefinida. Este tipo de abordagem é similar a um algoritmo de escalonamento de frequência de CPU que executa a nível de *kernel*. Para testar a implementação, utilizamos dois *smartphones* com sistema operacional Android: um Motorola, modelo XT1033 (conhecido como moto G) e um Xiaomi, modelo Redmi Note 4.

A Figura 11 ilustra, de forma simplificada, as etapas executadas pela abordagem proposta e suas principais contribuições para a literatura. As marcações em verde simbolizam as inovações propostas por este trabalho, adicionando conhecimento à literatura. De forma complementar, a Figura 12 ilustra de forma mais detalhada as etapas e as variáveis passadas entre estas etapas.

Figura 11 – Fluxo geral simplificado da abordagem proposta.



Fonte: O autor (2019).

Na etapa (1) da Figura 11, é feita a previsão das variáveis de desempenho (utilização da CPU, ciclos de CPU, temperatura, dentre outras), concretizando a abordagem proposta como proativa. Os algoritmos de predição EWMA e uma implementação própria do AEWMA, chamado de *Adaptive Exponential Weighted Moving Average MSE* (AEWMA-MSE), são avaliados.

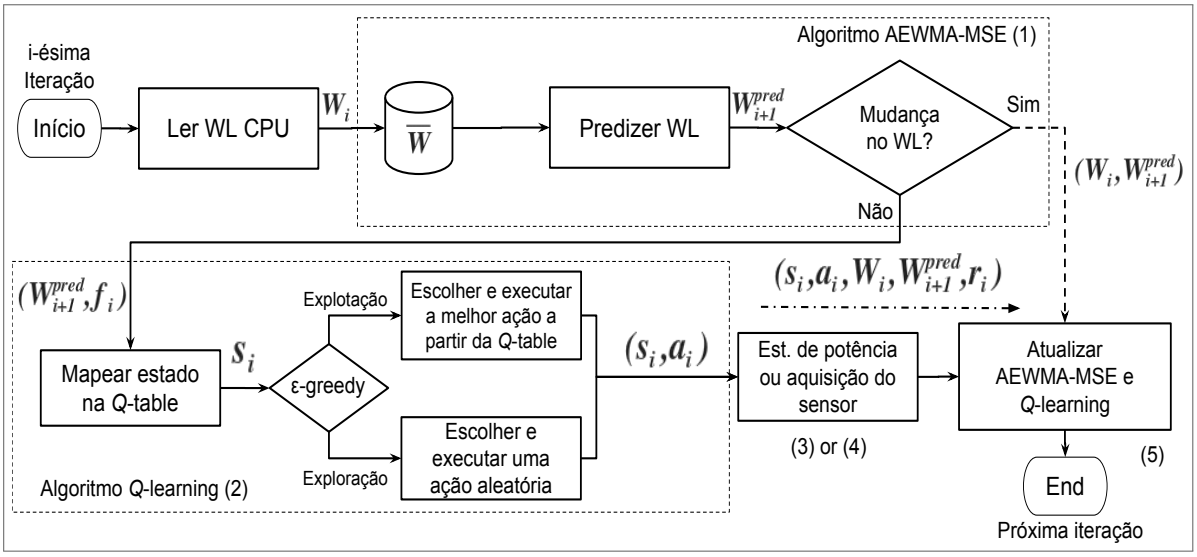
Na etapa (2), a abordagem com RL permite o aprendizado em tempo de execução de qual frequência de CPU deve ser utilizada para um determinado contexto, visando

à redução do consumo de energia pelo dispositivo. É utilizado o algoritmo *Q-learning* e algumas variações de seus parâmetros aplicado ao problema deste trabalho.

As etapas (3) e (4) representam a aquisição do valor de potência que representa o contexto do dispositivo no momento da execução. Essa aquisição da potência pode ser por predição ou por coleta de um sensor. Durante a implementação da abordagem, somente um dos métodos é utilizado por execução, considerando restrições do dispositivo. Na etapa (3), analisamos alguns métodos de predição, avaliando a acurácia e custo computacional de cada método. Além disso, os métodos de predição envolvem abordagens lineares e não-lineares.

Na etapa (4), a metodologia de coleta de potência utilizando sensores, nativos de alguns dispositivo atuais, é demonstrada. Na etapa (5), os algoritmos são atualizados utilizando a resposta do ambiente. Por conseguinte, na Figura 12, há mais detalhes da abordagem proposta, demonstrando como ocorre uma única iteração, chamada de i-ésima iteração.

Figura 12 – Fluxo geral detalhado da abordagem proposta.



Fonte: O autor (2019).

Primeiramente, o aplicativo é iniciado no dispositivo móvel e algumas configurações são feitas para permitir o chaveamento da frequência de CPU a nível de aplicação, além disso, os arquivos para armazenar o conhecimento (*Q-table* do *Q-learning*) e permitir a execução da abordagem são criados. Mais detalhes quanto à implementação serão descritos na Seção 4.5.

Após feita as cessões de algumas permissões do SO para o aplicativo e todos os ajustes necessários para o início do procedimento de aprendizagem, a abordagem proposta pode executar as etapas seguintes.

Ao início da execução da implementação da abordagem proposta, é feita a leitura da carga de processamento Carga de Processamento - *Workload* (WL) da CPU, chamada

de W_i. Esta etapa é o início do algoritmo AEWMA-MSE. Todos os valores de W serão armazenados em um banco de dados para calcular a média desses valores para ser utilizada no processo de predição.

Após aplicar o algoritmo de predição AEWMA-MSE utilizando o valor de WL atual (W_i) e o valor médio das últimas n iterações ($\overline{W}$), é obtido o valor predito do WL para o instante $i + 1$, chamado de W_{i+1}^{pred}. Diferentemente das outras versões do algoritmo AEWMA, nossa abordagem utiliza um recurso para detecção da mudança no WL, permitindo identificar quando houve uma mudança significativa na carga de processamento e possibilitando ajustar a frequência da CPU de forma adequada.

Após a etapa de detecção da mudança, finalizamos o algoritmo AEWMA-MSE e iniciamos a etapa de aprendizagem com RL, utilizando o algoritmo *Q-learning*. Após o início do *Q-learning*, o algoritmo recebe o W_{i+1}^{pred} e o f_i para realizar o mapeamento dessas variáveis em um estado pré-definido e utilizado para guiar o processo de economia de energia.

Após isso, o processo de escolha da melhor ação a ser utilizada é executado utilizando a abordagem ϵ-*greedy*. O estado mapeado no *Q-learning* (s_i) e a ação a ser executada (a_i) são passados para o algoritmo de predição da potência ou é utilizado o sensor embutido no dispositivo móvel para obtenção do valor de potência elétrica. O valor de potência (função de custo no *Q-learning*) é utilizado como parâmetro para guiar o processo de quais frequência de CPU são melhores para economizar energia.

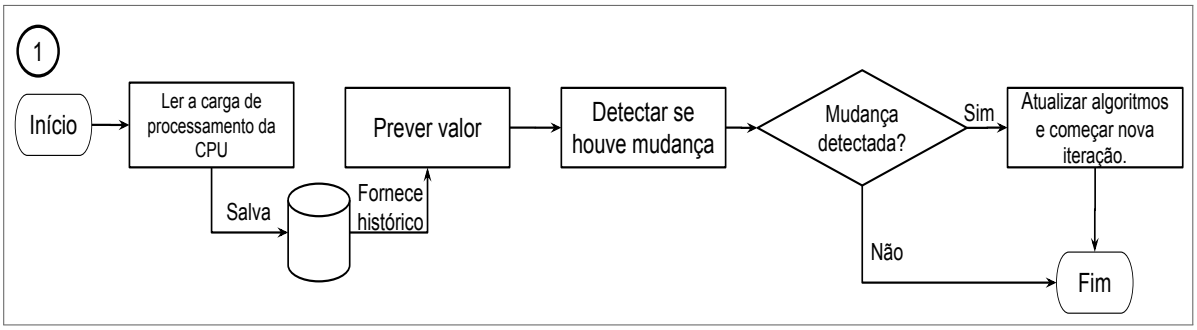
Por fim, os parâmetros estado atual (s_i), ação tomada (a_i), carga de processamento da iteração atual (W_i), a carga de processamento predita (W_{i+1}^{pred}) e a potência que representa o estado atual do dispositivo (r_i) são utilizados para atualizar os algoritmos AEWMA-MSE e *Q-learning*, fixando o conhecimento adquirido na iteração. Nas seções seguintes, cada etapa é detalhada, começando com a etapa de predição de variáveis.

4.1 PREDIÇÃO DE VARIÁVEIS

A Figura 13 ilustra a sequência de execução do processo de predição da variável de desempenho. A abordagem proposta é expansível a ser utilizada com várias variáveis de desempenho (ex.: temperatura e ciclos de CPU), mas para nossa abordagem, somente utilizamos uma variável de desempenho: a carga de processamento da CPU. A variável de desempenho é lida durante a execução do RTM e sua média é calculada a partir dos dados históricos armazenados. Após a predição, é realizado o procedimento de detecção de mudança utilizando o valor predito. Se uma mudança tiver sido detectada, o processo de predição é reiniciado, com a base de dados atualizada; caso contrário, o fluxo continua.

Na literatura, há duas abordagens distintas para a detecção de mudança na carga de processamento. A primeira opção é realizar um canal de comunicação da aplicação que está executando no dispositivo com o RTM (SHAFIK et al., 2016; MAEDA-NUNEZ, 2016). Por outro lado, a segunda opção é fazer uso de abordagens autônomas, onde, em tempo de execução, é realizada a detecção da mudança.

Figura 13 – Fluxo do processo de predição da variável de desempenho.



Fonte: O autor (2019).

Os dispositivos móveis possuem um ambiente desconhecido, onde as cargas de processamento não são mapeadas, exigindo uma modificação em toda a estrutura dos aplicativos que serão utilizados pelo usuário para permitir o canal de comunicação com o RTM. Obviamente, modificações em tantos aplicativos distintos e com diversos fornecedores não é uma escolha razoável para uma abordagem genérica. Diante disto, optamos pelo uso da abordagem autônoma, sendo esta mais genérica e expansível a qualquer dispositivo.

Há poucos trabalhos que utilizam a abordagem autônoma, dentre estes, temos o trabalho de Das et al. (2016), que utiliza divergência de KL aplicada ao cálculo da distância estatística entre janelas de amostras. Objetivando simplicidade e cumprimento das restrições do dispositivo móvel, propomos uma abordagem para detecção de mudança que usa poucos recursos e pode operar em dispositivos móveis.

Propomos um método chamado AEWMA-MSE, algoritmo usado para realizar a predição e detecção de mudança na carga de processamento. O algoritmo AEWMA-MSE, ilustrado no Algoritmo 5, incrementa o algoritmo de predição EWMA, adicionando a funcionalidade de detecção de mudança utilizando a métrica Erro Médio Quadrado - *Mean Squared Error* (MSE).

Algoritmo 5: Descrição do algoritmo AEWMA-MSE.

Defina $\lambda_{in} = 0,6 \mid \beta = 0,1 \mid EM = 0,2$
for W_i **do**
  Calcula $W_{i+1}^{pred} = \lambda W_i + \sum_{\ell=t-D}^{i}(1-\lambda)W_\ell \quad (2.13)$
  $MSE_i \leftarrow (W_i^{pred} - W_i)^2$
  $MSE_i^{hist} \leftarrow \dfrac{MSE_i + MSE_i^{hist}}{2}$
  if $MSE_i > MSE_i^{hist} \times (1 + EM)$ **then**
    $\lambda = 1$
  else
    if $\lambda > \lambda_{in}$ **then**
      $\lambda \leftarrow \lambda$ - β
    end
  end
end

Fonte: O autor (2019).

No primeiro passo do Algoritmo 5, a inicialização dos parâmetros é realizada para $\lambda_{in} = 0,6$, $\beta = 0,1$ e $EM = 0,2$, que são, respectivamente, o coeficiente móvel de médias, a velocidade de adaptação e a margem de erro. Construímos uma base de dados para armazenar os valores de porcentagem de uso da CPU (carga de processamento) e potência instantânea, para alguns *benchmarks* reais, com foco nos usuários, envolvendo navegação Web, mídias sociais e *streaming* de vídeo. Este banco de dados foi utilizado para testar quais valores possuem o menor erro MSE (acurácia aumentada) para λ_{in} e EM, utilizados na versão final do AEWMA-MSE.

Antes da leitura do valor da carga de processamento atual W_i, assumimos que D valores de carga de processamento (dados históricos) foram obtidos previamente e armazenados em um banco de dados para serem computados, obtendo-se a média dos valores de carga de processamento $\overline{W}$. Em outras palavras, consideramos que a i-éssima iteração é maior que D. Dito isso, o RTM lê a carga de processamento da CPU W_i para adquirir W_i^{pred} usando 2.13.

Na iteração atual, a carga de processamento predita W_i^{pred}, obtida na iteração $(i-1)$, é utilizada para detectar a mudança na carga de processamento utilizando os dados históricos. Se uma mudança na carga de processamento for detectada, a carga de processamento atual W_i é utilizada para atualizar a média $\overline{W}$, além disso, W_{i+1}^{pred} é armazenada para ser utilizada na próxima iteração.

Para verificar a existência de uma mudança na carga de processamento, utilizamos o valor MSE entre a carga de processamento predita W_i^{pred} e a carga de processamento

atual W_i, descrita como MSE_i. A métrica de erro é empregada para atualizar o MSE_i^{hist}, erro médio das D iterações anteriores (linha 5 do Algoritmo 5). Quando MSE_i exceder MSE_i^{hist} em mais que EM, uma mudança na carga de processamento é detectada ($\lambda = 1$).

Em adição ao que foi exposto, o algoritmo AEWMA-MSE é atualizado e o *Q-learning* não é considerado até a próxima iteração. De outra maneira, se nenhuma mudança na carga de processamento for detectada, λ é decrementado em β ate que seja alcançado o valor de melhor custo-benefício ($\lambda = \lambda_{in}$).

No tópico seguinte, iremos descrever o procedimento de cálculo da porcentagem de uso da CPU utilizado como métrica em nossa abordagem.

4.1.1 Cálculo da porcentagem de uso da CPU

De forma simplificada, a porcentagem de uso da CPU é calculada dividindo o tempo em que a CPU está no estado *idle* e o tempo total em que o processador está em execução.

Os valores de uso da CPU são obtidos a partir do arquivo de sistema que armazena diversas estatísticas referentes ao *kernel* Linux, localizadas em */proc/stat*. Para realizar a leitura desse arquivo, é utilizado o comando: *cat /proc/stat* na *shell* do sistema operacional. Além disso, para nosso cálculo, é considerado o tempo de execução de todos os processadores e seus núcleos em uma única medida. O cálculo do tempo total de execução da CPU é como segue:

$$T_i^{total} = T_i^{user} + T_i^{nice} + T_i^{system} + T_i^{idle} + T_i^{io\ wait} + T_i^{irq} + T_i^{soft\ irq} + T_i^{steal} + T_i^{guest} + T_i^{guest\ nice}, \tag{4.1}$$

onde T_i^{total} é o tempo total de uso da CPU para o tempo i; T_i^{user} é o tempo de uso da CPU para execução normal dos processos em modo de usuário e assim por diante. A Tabela 3 descreve as demais variáveis de acordo com a documentação do *kernel* Linux (KERNEL, 2009).

Em seguida, T_i^{total} é utilizado para calcular a porcentagem de uso da CPU (W_i) para o tempo i, juntamente com o tempo T_i^{idle}, relacionados como segue:

$$W_i = \frac{\Delta T_i^{idle}}{\Delta T_i^{total}} = \frac{T_i^{idle} - T_{i-1}^{idle}}{T_i^{total} - T_{i-1}^{total}}, \tag{4.2}$$

Assumimos que $T_i^{idle} > T_{i-1}^{idle}$, bem como, $T_i^{total} > T_{i-1}^{total}$. Esta assertiva é garantida pelo sistema operacional, onde há uma contagem crescente de tempo, utilizando a notação *Unix time* que conta o número de segundos que se passaram desde as 00:00:00h *Coordinated Universal Time* (UTC), de 1 de janeiro de 1970.

Além disso, a diferença (Δ) de T_{i-1}^{total} e T_i^{total} e também a de T_{i-1}^{idle} e T_i^{idle} são utilizadas para o cálculo. Em outras palavras, é calculado o tempo em que há utilização da CPU para o intervalo de tempo entre a iteração atual i e a iteração anterior $i - 1$. Em nossos

Tabela 3 – Descrição das variáveis adquiridas do arquivo */proc/stat*.

Nome da coluna	Descrição
user	Tempo de CPU utilizado para execução normal dos processos em modo de usuário
nice	Tempo de CPU utilizado para execução de processos prioritário (processos iniciados com o comando *nice* do Linux)
system	Tempo de CPU utilizado para execução dos processos em modo de *kernel*
idle	Tempo de CPU quando em modo de espera (*idle*)
io wait	Tempo de CPU utilizado enquanto aguardando completar as requisições de entrada e saída (*Input/Output* (I/O))
irq	Tempo de CPU utilizado por interrupções
soft irq	Tempo de CPU utilizado por interrupções de software
steal	Tempo de CPU utilizado por espera involuntária
guest	Tempo de CPU utilizado enquanto executando processos normais de visitantes
guest nice	Tempo de CPU utilizado enquanto executando processos prioritários de visitantes

Fonte: O autor (2019).

experimentos, utilizamos o valor de 300 *ms* para esse intervalo. Esse valor de 300 *ms* é utilizado tanto em abordagens padrões do *kernel* Linux, como o *governor* OD como também em trabalhos semelhantes ao nosso, como o de Carroll e Heiser (2014).

No próximo tópico, iremos detalhar o passo seguinte: a predição da variável de desempenho, o procedimento de escolha da frequência de CPU que minimiza o consumo de energia, utilizando a abordagem com o algoritmo *Q-learning* da aprendizagem por reforço.

4.2 ESCOLHA DA MELHOR AÇÃO

O processo de escolha da melhor ação inclui as etapas de inserção e extração de conhecimento no agente inteligente. Após o processo de escolha da melhor ação, é possível a seleção de uma frequência de CPU mais apropriada para o contexto do dispositivo, priorizando a redução energética.

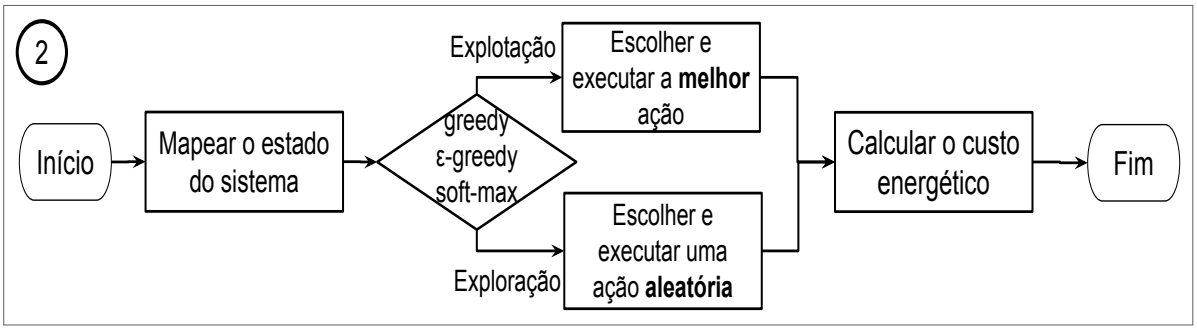
Além disso, iremos utilizar uma versão modificada do algoritmo *Q-learning*, da aprendizagem por reforço. O *Q-learning* permite a construção de um modelo do sistema em tempo de execução, sem nenhum conhecimento prévio do ambiente, sendo seu principal objetivo aprender a fazer decisões melhores com o passar das iterações. Ademais, o *Q-learning* permite ser utilizado em um ambiente desconhecido que sofre variações na carga

de processamento, temperatura, dentre outras alterações no contexto do dispositivo.

A Figura 14 mostra a sequência temporal de execução do processo de escolha da melhor ação. Após o mapeamento do contexto do dispositivo para um estado pré-definido do *Q-learning*, é aplicada a política de exploração e explotação. A abordagem ϵ-*greedy* é a mais utilizada nos trabalhos consultados, onde ϵ ações serão de exploração e $(1 - \epsilon)$ para explotação. Neste trabalho, utilizaremos a abordagem ϵ-*greedy*, porém, o impacto de uma versão adaptativa desta abordagem deve ser pesquisada em trabalhos futuros. Além disso, nossa abordagem é flexível para permitir o uso de outras abordagens, por exemplo, *greedy* e *soft-max* (BISHOP, 2006; SUTTON; BARTO, 2012).

Adicionalmente, o cálculo da função de custo (também chamada de recompensa) é uma etapa essencial deste processo, guiando o aprendizado do agente e impactando diretamente a acurácia de predição do algoritmo na obtenção da frequência de CPU que minimiza o consumo de energia. Diante disso, é proposta uma nova função de custo que utiliza a potência consumida por todo o dispositivo como parâmetro desta função.

Figura 14 – Fluxo do processo de escolha da melhor ação.



Fonte: O autor (2019).

Para cada época de execução, tanto a carga de processamento predita W_{i+1}^{pred} e a frequência de CPU atual f_i são utilizadas como parâmetros de entrada no algoritmo *Q-learning* para estimar a frequência de CPU que melhor representa a carga de processamento, objetivando a economia de energia. Ademais, o intervalo entre t_i e t_{i-1} é considerado uma época $(t_{i-1} \rightarrow t_i)$.

A versão modificada do algoritmo *Q-learning* é apresentada no Algoritmo 6. Uma das variáveis de atualização do algoritmo *Q-learning* é a γ, também chamada de fator de desconto, pertencendo ao intervalo $0 \leq \gamma < 1$ (SUTTON; BARTO, 2012). A variável γ representa a importância de recompensas futuras na execução do *Q-learning*, ou seja, recompensas obtidas posteriormente possuem um desconto maior que recompensas obtidas mais cedo no tempo. Em outras palavras, as recompensas recebidas são descontadas de acordo com quão distante elas estão no tempo.

Assumindo que em nosso problema um estado futuro não depende dos estados passados, recompensas futuras são irrelevantes, resultando em $(\gamma = 0)$. Segundo Sutton e

Barto (2012), sistemas que possuem ($\gamma = 0$) são considerados míopes, ou seja, são sistemas que só estão interessados em recompensas imediatas. Por conta disso, o termo $\max_{a_{i+1}} Q(s_{i+1}, a_{i+1})$ em (2.8) foi removido, resultando em:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha[r_{i+1} - Q(s_i, a_i)] \ . \tag{4.3}$$

Algoritmo 6: Versão modificada do *Q-learning*.

```
1 Inicializa a Q-table.
2 for época in Épocas do
3 |   Mapeia (W_{i+1}^{pred}, f_i) para o estado s_i
4 |   Escolhe a ação a_i para o estado s_i usando a política ϵ-greedy
5 |   Executa a ação a_i
6 |   Obtém r_i a partir da função de custo
7 |   Atualiza a Q-table usando (4.3)
8 end
```

Fonte: O autor (2019).

Os valores calculados utilizando a equação 4.3 são as recompensas acumuladas das ações tomadas em cada estado. Esses valores são armazenados na *Q-table* e representam o conhecimento do algoritmo com relação ao ambiente. Na implementação proposta, foram utilizadas duas estruturas para a arquitetura do *Q-learning*: uma abordagem com sete ações possíveis e três faixas de carga de processamento; a outra abordagem sete ações possíveis e cinco faixas de carga de processamento, em ambos os casos, os valores são inicializados com zeros.

Em complemento, possuir uma *Q-table* com muitos estados e ações não é recomendado, pois exigirá uma quantidade maior de armazenamento e processamento para ler e gravar na matriz. Os valores utilizados neste trabalho foram definidos, baseados em trabalhos anteriores e testados empiricamente para possibilitar que o uso de memória e processamento no manuseio da *Q-table* seja reduzido, permitindo acurácia suficiente nas predições para economizar energia quando comparado aos trabalhos relacionados.

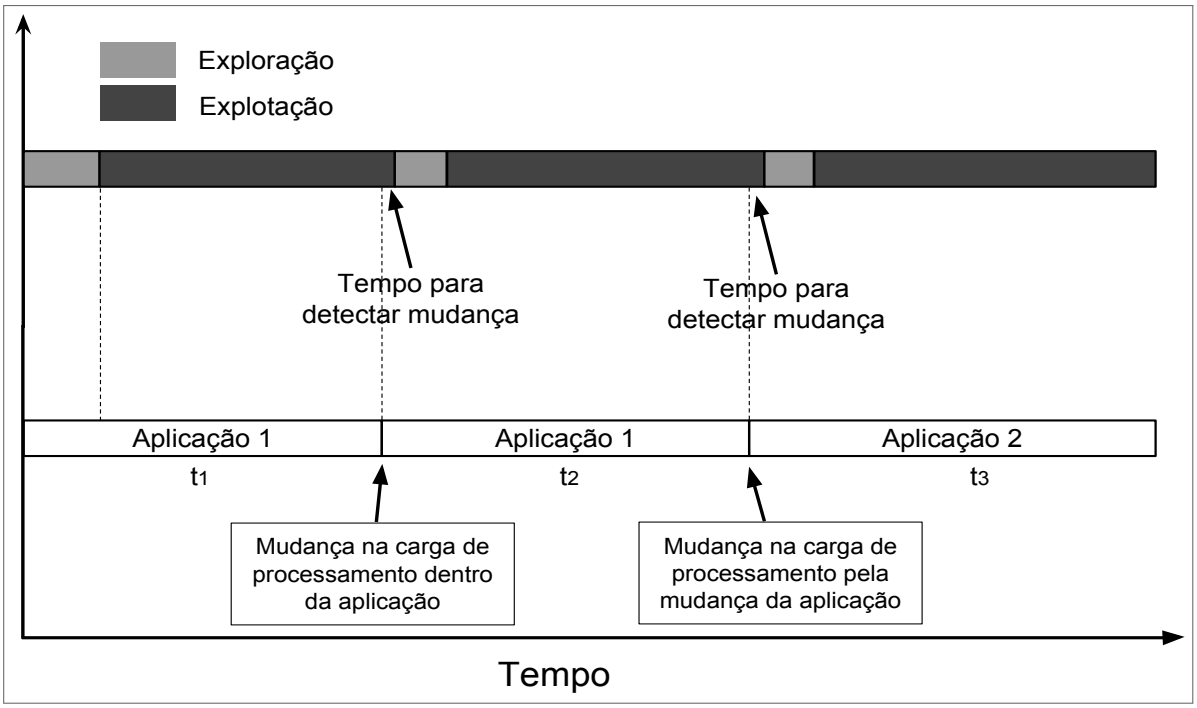
Dando continuidade, no início da execução da abordagem, não há conhecimento prévio do contexto do dispositivo, a política de chaveamento de frequências é construída durante a execução. A estratégia *ϵ-greedy*, apresentada no Algoritmo 4, provê uma forma de construir essa política, alternando entre situações de exploração e explotação.

Após algumas etapas de exploração, é possível obter a melhor ação a partir da *Q-table*, iniciando a etapa de explotação. Durante a exploração, boas ações são recompensadas e más ações são penalizadas. Todo esse conhecimento é armazenado na *Q-table*.

Para exemplificar como a abordagem proposta pode ser benéfica para o problema explorado, a Figura 15 ilustra a execução de duas aplicações distintas no dispositivo móvel e como acontecem as fases de exploração e explotação com dados reais coletados por experimentação.

Dando continuidade, podemos perceber na Figura 15 que uma mudança na carga de processamento ocorre durante a execução da Aplicação 1 (do tempo t_1 para t_2) e outra quando há a mudança para a Aplicação 2 (do tempo t_2 para t_3). O *Q-learning* e o AEWMA-MSE trabalham juntos e interagem entre si. O *Q-learning* é avisado para reinicializar a *Q-table* quando uma mudança significativa na carga de processamento é detectada, recomeçando o processo de exploração.

Figura 15 – Fase de exploração e explotação do *Q-learning*.



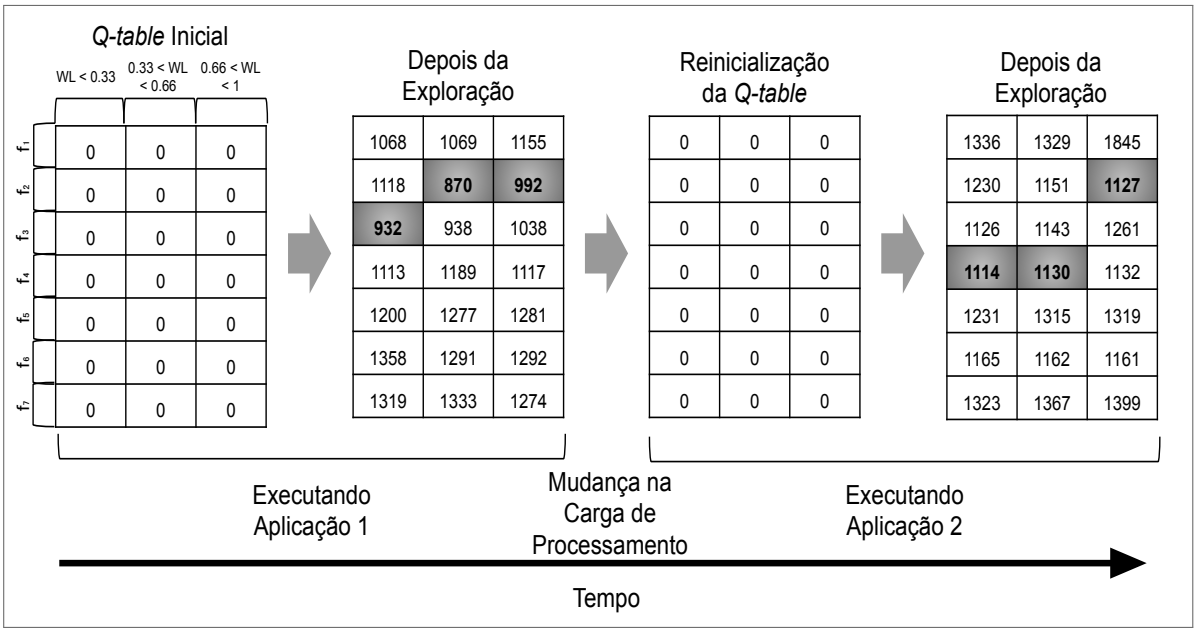
Fonte: O autor (2019).

Para exemplificação, utilizamos o exemplo mais simples, usando três intervalos para a carga de processamento e sete frequências de CPU (ao total o smartphone Motorola XT1033 possui somente sete frequência de CPU distintas). A Figura 16 ilustra o comportamento da *Q-table* ilustrada com dados reais dos nossos experimentos.

É perceptível que durante a execução da Aplicação 1, há uma mudança na carga de processamento, após isso, a Aplicação 2 começa a executar. Quando uma mudança significativa na carga de processamento é detectada, todos os valores da *Q-table* são definidos para zero, iniciando uma nova etapa de exploração. Após algumas iterações, o processo de explotação é reiniciado.

Ademais, analisando os valores da *Q-table* para a Aplicação 1, a frequência de CPU mais adequada para uma carga de processamento abaixo de 33% de utilização de CPU é a

Figura 16 – Exemplo de *Q-table* com dados reais do trabalho proposto.



Fonte: O autor (2019).

frequência f_3 (600MHz) e para uma carga de processamento superior a 33% a frequência f_2 (384MHz) é a mais adequada. Por fim, para a Aplicação 2, para uma carga de processamento até 66%, a frequência f_4 é a mais indicada, enquanto para cargas de processamento maiores que 66%, é indicada a frequência f_2 (384MHz).

No próximo tópico, iremos detalhar a etapa de aquisição dos valores de potência, utiliza na função de custo do *Q-learning*. Na seção 4.3, descrevemos o processo de estimação da potência utilizando métodos lineares e não-lineares e a coleta da potência diretamente do sensor de potência embutido em alguns dispositivos móveis.

4.3 ESTIMAÇÃO E COLETA DE POTÊNCIA

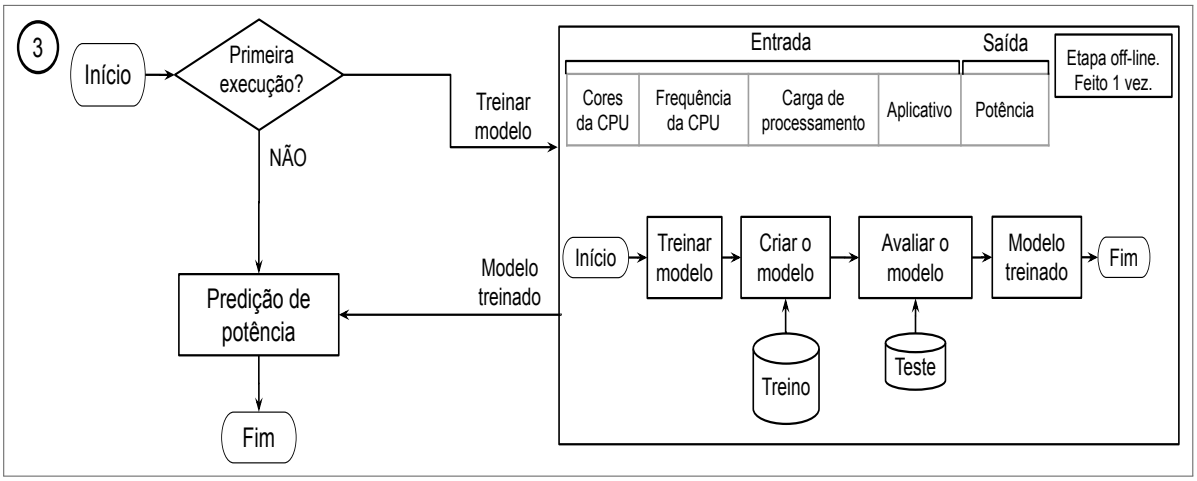
Tanto a estimação quanto a coleta da potência são abordagens viáveis para serem utilizadas em conjunto com nossa proposta. Os valores de potência guiam o aprendizado do escalonador de frequência de CPU proposto. Além disso, o sensor de potência está presente em alguns *smartphones* e *tablets* atuais, sendo possível perceber uma tendência de que, cada vez mais, os dispositivos mais novos possuem o sensor.

A potência é usada no cálculo da recompensa das ações tomadas pelo algoritmo de escolha da melhor ação. Nossa abordagem difere das abordagens da literatura na utilização de mecanismos para predição de potência baseados em modelos de predição não-lineares, enquanto os demais usam regressão linear. O uso de métodos não-lineares nos possibilitou um aumento na acurácia de predição quando comparado aos métodos lineares, consequentemente, auxiliando na redução do consumo de energia do dispositivo.

Nossa abordagem utiliza a porcentagem de uso da CPU, mas é genérica o suficiente

para utilizar outras métricas, como: ciclos de CPU, acessos à memória, falhas de acesso à memória, dentre outros. A Figura 17 ilustra nossa sequência de execução proposta para estimação de potência usando modelos não-lineares.

Figura 17 – Fluxo do processo de estimar potência.



Fonte: O autor (2019).

Além disso, diante do que a literatura aponta, nenhum trabalho usou algum algoritmo de RL (*Q-learning*, SARSA ou outro) com a função de custo sendo alimentada por um sensor de potência durante a execução. Alguns trabalhos usam somente a temperatura (DAS et al., 2016), outros usam heurísticas para guiar o processo de aprendizado (SHEN et al., 2013; TIAN et al., 2018), mas nossa abordagem é mais genérica e autônoma que os trabalhos citados anteriormente, focando em redução energética, modelos não-lineares e aquisição de potência diretamente por um sensor.

A abordagem proposta permite a implementação da técnica tanto em dispositivos não tão modernos, através do modelo de predição, como também nos atuais, que possuem o sensor de potência nativo.

Os *smartphones* Google Nexus 6 (lançado em 2014), Google Nexus 9 (lançado em 2014) e Xiaomi Redmi Note 4 (lançado em 2017 e utilizado neste trabalho) são exemplos de aparelhos que possuem os sensores de potência, permitindo a aquisição de potência gasta pelo dispositivo através da *Application Program Interface* (API) Android ou fazendo a leitura de arquivos do sistema Linux.

Os *smartphones* fabricados pelo Google, apresentados anteriormente, fornecem um período de atualização do valor de potência de $175,8$ ms, sendo que não é uma medição instantânea, mas uma atualização com a potência média durante os $175,8$ ms de execução (DEVELOPER, 2017). Além disso, o *smartphone* Xiaomi utilizado precisa somente de $5,32$ ms para acessar o dado do sensor, porém, o sensor demora aproximadamente 1000 ms para atualizar o valor da potência. Essas informações sobre o *smartphone* Xiaomi foram obtidos por experimentação, dada a ausência de informações nos canais oficiais.

Para realizar a coleta do valor de potência a partir do sensor no dispositivo, há um resistor *shunt*, embarcado no projeto de alguns *smartphones*, que permite obter a corrente que está sendo drenada pelo dispositivo, juntamente com o valor de tensão, é possível calcular a potência instantânea.

4.3.1 Metodologia utilizada na estimação de potência

Na geração do modelo e estimação da potência há dois procedimentos distintos, o treinamento e teste do modelo; e a predição do valor de potência, feita em tempo de execução, usando o modelo treinado. O procedimento *offline* é composto de um estágio de treinamento e outro de teste.

A Tabela 4 ilustra uma amostra da base de dados usada no estágio de treinamento da etapa *offline*. Na primeira coluna, temos o número de núcleos ativos da CPU. A segunda coluna representa a frequência em uso da CPU quando as variáveis de contexto são coletadas e é medida em kHz.

A taxa de utilização da CPU, representada na terceira coluna, indica quanto tempo a CPU esteve realizando trabalho. A quarta coluna representa a aplicação que estava executando no momento da coleta dos dados. Cada aplicativo que executa no Android possui uma identificação única, que é convertida para um inteiro, por exemplo: aplicativo Facebook é representado por 1, Google Chrome por 2, e assim por diante. Por fim, a variável de saída é o valor de potência estimado $P_f(\cdot)$, medido em mW.

Tabela 4 – Amostra das características da base de dados.

Entradas				Saída
Núcleos ativos	Freq. da CPU	Carga de uso	APP ID	Potência
3	300 MHz	0,016	1	892,80 mW
1	998 MHz	0,025	4	909,69 mW
2	1190 MHz	0,008	2	1106,42 mW
4	600 MHz	0,168	3	1308,65 mW

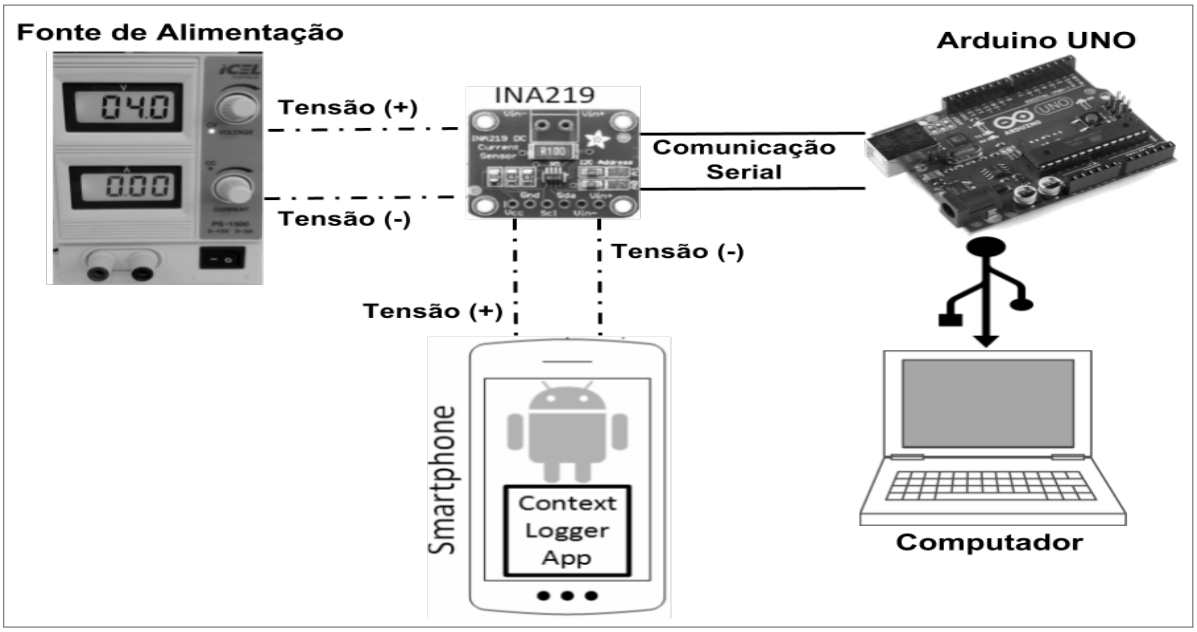
Fonte: O autor

A Figura 18 ilustra o ambiente experimental, utilizado para as medições de potência no dispositivo. Uma placa de aquisição de dados de potência foi desenvolvida para medir valores de tensão e corrente do dispositivo que não possui sensor de potência integrado, possuindo uma taxa amostral de 300 amostras por segundo. Uma fonte de alimentação modelo PS-1500 é usada para fornecer energia para o dispositivo.

Para coleta dos valores de corrente e tensão, é utilizado um microprocessador Arduino UNO (ARDUINO, 2015), que comunica com o sensor de corrente INA219 (INDUSTRIES, 2015) e então envia os dados para o computador. A partir dos dados de tensão obtidos da fonte de alimentação e dos valores de corrente, obtidos do INA219, é possível calcular

a potência instantânea. Além disso, o dispositivo utilizado é um *smartphone* Motorola XT1033 com Android versão 4.4.4 KitKat, as características são listadas na Tabela 5.

Figura 18 – Diagrama do ambiente experimental usado para medir corrente e tensão do dispositivo.



Fonte: O autor (2019).

Tabela 5 – Características do *smartphone* Motorola XT1033

	XT1033
SoC	Snapdragon 400
Processador	ARM Cortex A7
Frequências (MHz)	300 - 1190
Número de Núcleos	4
Versão do Android	4.4.4
Capacidade da bateria (mAh)	2070

Fonte: O autor (2019).

Desenvolvemos uma aplicação Android para capturar as variáveis de contexto do dispositivo, chamada de *Context Logger* (CL). Essa aplicação executa como um serviço, capturando 65 atributos do sistema, como:

- Informações do processador;
- Tempo e data;
- Estado da bateria;

- Informações da rede e de sensores;
- Aplicações executando em primeiro e segundo plano;
- Outros.

Todas as variáveis são coletadas a cada $0,25$ s, por restrições de desempenho do sistema Android e do dispositivo em análise. Dentre as variáveis coletadas, definimos que as variáveis que melhor representam o consumo de energia do dispositivo em termos de CPU são: número de núcleos ativos no processador, a frequência da CPU, a carga total de uso da CPU e a aplicação que está executando em primeiro plano. Os experimentos são realizados em conjunto com a aplicação CL e o circuito de medição de potência.

Por conseguinte, três tipos distintos de processamento foram executados: Google Chrome, Facebook (FB) e YouTube (YT). Cada tipo de processamento é realizado percorrendo todas as frequências de CPU disponíveis no dispositivo. Ações comuns de usuário do dispositivo foram simuladas através de comandos *Android Debug Bridge* (ADB) no Android. O ADB fornece uma linha de comunicação com a shell do Android, permitindo interação com o dispositivo, como: copiar arquivos, instalar ou desinstalar aplicativos, executar comandos da *shell* Linux, simular toques na tela, dentre outros.

Além disso, foram coletadas amostras para cada tipo de processamento, utilizando cada frequência de CPU (300, 384, 600, 787, 998, 1094, e 1190 MHz). Em média, 1.200 amostras de variáveis de contexto e 90.000 amostras de potência para cada frequência de CPU, resultando em 8.400 amostras de variáveis de contexto e 630.000 amostras de potência no total.

Sendo assim, as amostras coletadas pelo CL foram unificadas com as amostras de potência, formando uma base de dados única. Como as taxas de amostragem da aquisição das variáveis de contexto e de potência foram diferentes (300 e 4 amostras por segundo, respectivamente), foi necessário fazer um ajuste, sendo para cada amostra de contexto, a média de 75 amostras de potência.

Para avaliar a metodologia proposta para geração do modelo de potência, foram usados os métodos de regressão não-linear baseados em: *Artificial Neural Network* (ANN) e *k-Nearest Neighbor* (k-NN) e o método de regressão linear. No método linear, foram usadas quatro variáveis como entrada do modelo e uma variável de saída:

$$y_{LM} = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 \,, \tag{4.4}$$

em que x_1 representa o número de núcleos ativos no processador, x_2 a frequência de CPU em uso, x_3 a porcentagem de uso da CPU, x_4 o valor correspondente ao nome da aplicação em primeiro plano, α é o ponto de interseção e y_{LM} é o valor de potência predito.

Para os modelos não-lineares, usamos uma ANN com retroalimentação, sendo os pesos inicializados de forma aleatória. A ANN é treinada usando o pacote *H2O*, disponível

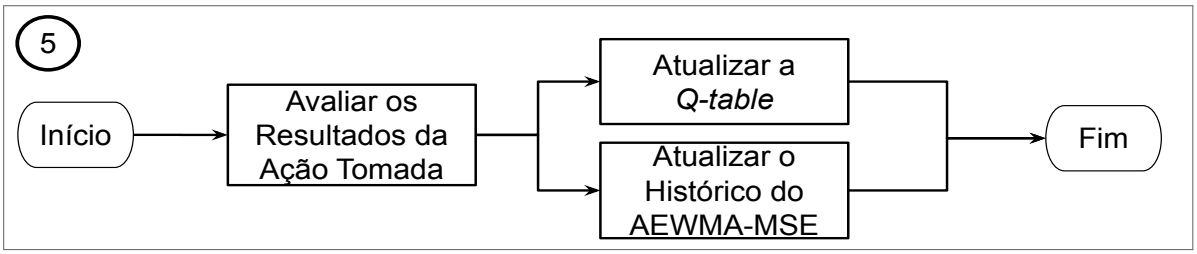
para a linguagem R[1], e o impacto do número de neurônios na acurácia e tempo de treinamento analisados. O segundo método não-linear é o k-NN, implementado pelo pacote *Fast Nearest Neighbor Search Algorithms and Applications* (FNN), também disponível para a linguagem R. Nesse caso, o impacto do número de vizinhos (k) é analisado.

Para avaliar a proposta, foi utilizado um método de validação similar ao empregado por Kuhn e Johnson (2013). Além disso, foi utilizado a técnica de validação conhecida por validação cruzada usando K-*fold*, um método usado para avaliar a acurácia de modelos de predição. A técnica K-*fold* é caracterizada pela reamostragem de dados de treinamento e teste onde as amostras foram divididas aleatoriamente em K conjuntos de aproximadamente o mesmo tamanho para serem usadas como treinamento e teste. Um valor comum adotado para o K é 10 para um banco de dados pequeno ou mediano (YADAV; SHUKLA, 2016), como o utilizado neste trabalho. Adicionalmente, os dados foram particionados em subconjuntos com 80% e 20% para treinamento e teste (KUHN; JOHNSON, 2013; YADAV; SHUKLA, 2016), respectivamente.

4.4 ATUALIZAÇÃO DOS ALGORITMOS

Nesta etapa, é feita a atualização dos algoritmos de predição e escolha da melhor ação. Desse modo, os algoritmos tentam melhorar a acurácia de predição a cada iteração. A ação tomada é avaliada de acordo com seu comportamento no ambiente utilizando a função de custo, indicando quão bom foi o comportamento da ação tomada anteriormente. Após isso, o retorno do comportamento do sistema é refletido na atualização da Q-*table* e na média do histórico da variável de desempenho. A Figura 19 mostra a sequência de atualização dos algoritmos.

Figura 19 – Fluxo do processo de atualização dos algoritmos.



Fonte: O autor (2019).

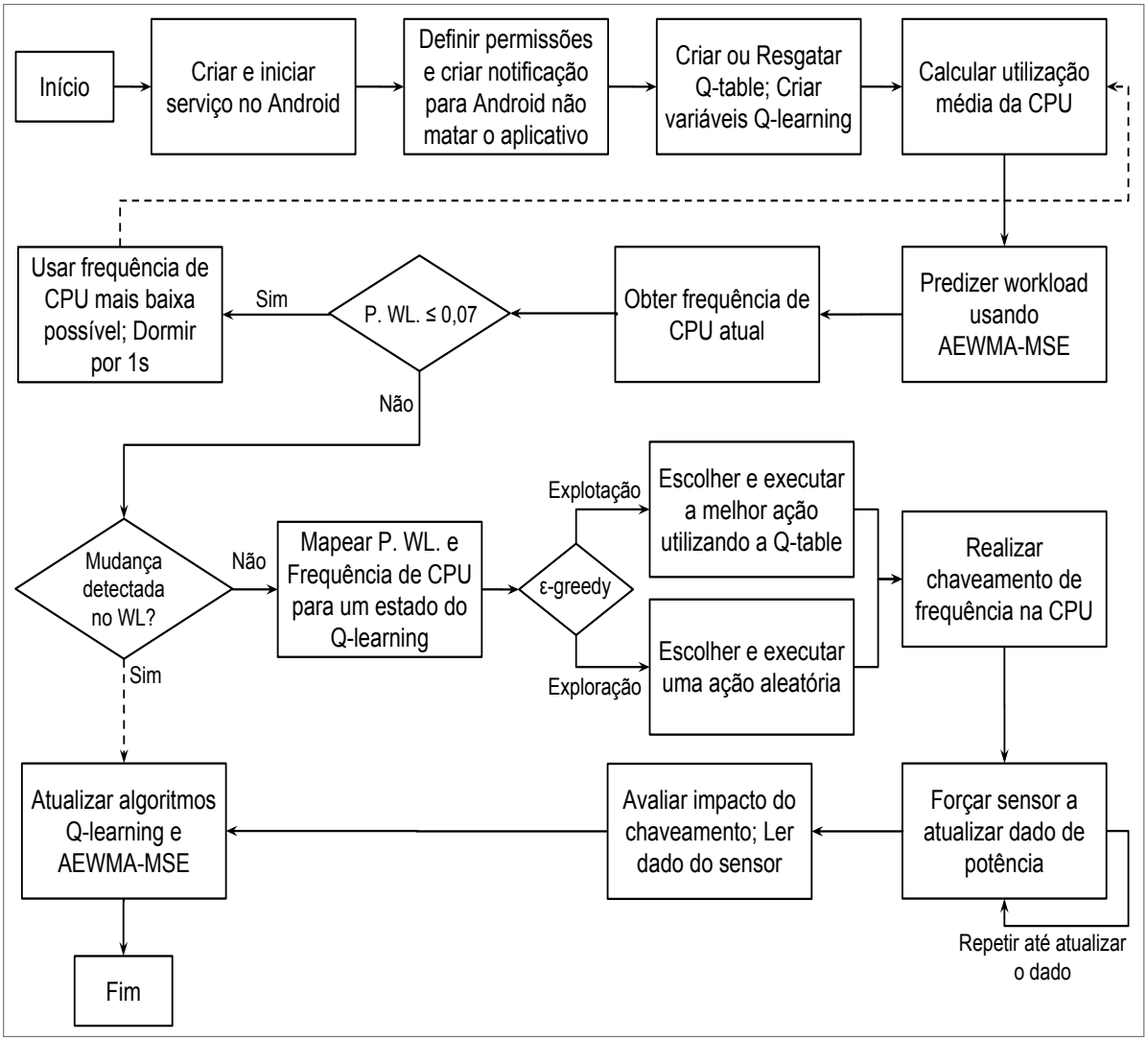
Primeiramente, é feita a avaliação do ambiente para mensurar o impacto da ação tomada. Após isso, é atualizada a Q-*table*; em simultâneo, o algoritmo AEWMA-MSE, que armazena as métricas para predição da carga de processamento também é atualizado.

[1] R é uma linguagem de programação usada para análise estatística e visualização de grandes bancos de dados.

4.5 IMPLEMENTAÇÃO ANDROID

Para realizar a implementação da abordagem proposta, foi necessário fazer algumas modificações pontuais por conta de restrições da plataforma Android e para melhorar o desempenho da execução. A Figura 20 ilustra o fluxo geral de execução da proposta na visão da implementação.

Figura 20 – Fluxo geral da abordagem: visão da implementação.



Fonte: O autor (2019).

Primeiramente, o serviço é criado utilizando *INTENTS*, uma abstração do Android utilizada para descrever uma ação a ser realizada. Após a inicialização do serviço, é preciso criar uma notificação permanente na barra de notificações para que o Android não mate a aplicação tentando economizar recursos. Além disso, o serviço deve ser registrado no arquivo *AndroidManifest.XML*. Esse arquivo é responsável por armazenar informações essenciais para a execução da aplicação Android.

O restante das permissões é definido para permitir o chaveamento da frequência de

CPU por programas na camada de aplicação e com acesso *root*. Além disso, temos um componente chamado *mpdecision*, presente nos *chipsets Qualcomm* e responsável por sobrescrever as políticas de chaveamento de frequência e ativação de núcleos. Dito isso, é necessário desativar o *mpdecision* para realizar o controle da CPU por nossa abordagem de forma precisa. Após isso, definimos o algoritmo de chaveamento de frequência da CPU para *userspace*, permitindo que nosso aplicativo Android possa realizar os chaveamentos sem interferências.

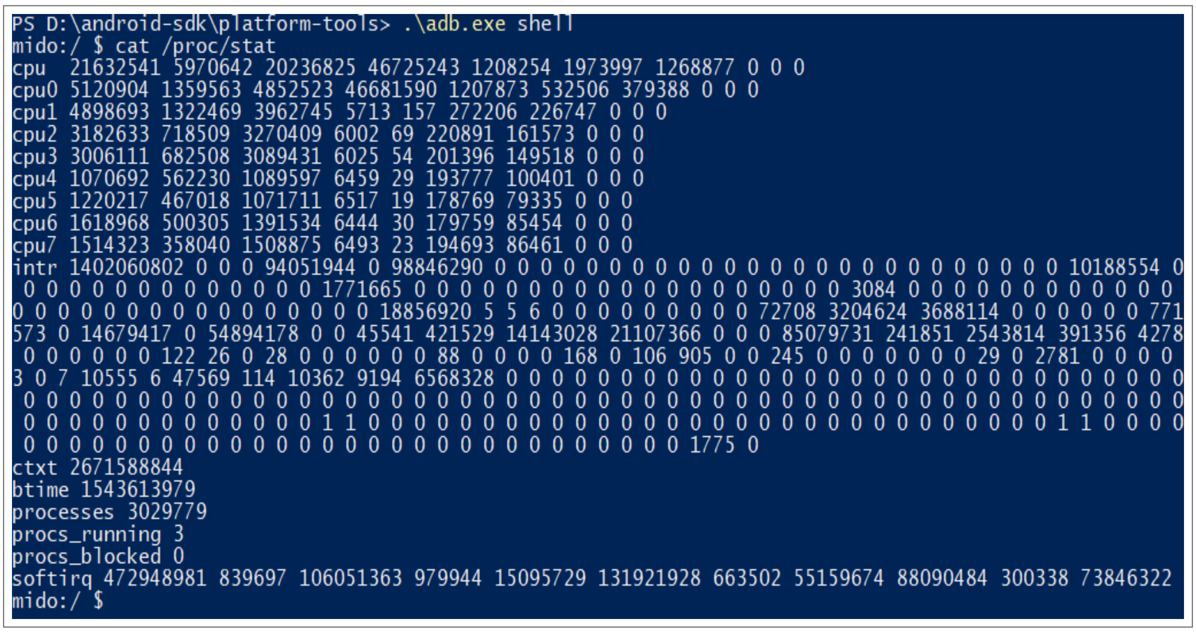
Para exemplificação, seguem abaixo os comandos para desativar o *mpdecision* e definir o *userspace*, executados na *shell* Linux do Android como *root*.

- su -c "stop mpdecision"
- su -c "echo 'userspace' > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor "

Após as definições de permissões e criação do serviço com a notificação, é feito uma checagem se há algum conhecimento já armazenado no dispositivo. A verificação é feita buscando por um arquivo chamado *Q-table.txt*. Se existir, esse arquivo será lido e seu conhecimento transferido para nosso agente inteligente; se o arquivo não existir, será criado e utilizado para armazenar o aprendizado do algoritmo.

Por conseguinte, é realizado o cálculo da utilização da CPU, usando as estatísticas presentes no arquivo */proc/stat*. A Figura 21 apresenta um exemplo da leitura desse arquivo, onde cada coluna representa uma característica de uso da CPU e a linha o atributo em questão.

Figura 21 – Exemplo de leitura do arquivo */proc/stat*.



```
PS D:\android-sdk\platform-tools> .\adb.exe shell
mido:/ $ cat /proc/stat
cpu  21632541 5970642 20236825 46725243 1208254 1973997 1268877 0 0 0
cpu0 5120904 1359563 4852523 46681590 1207873 532506 379388 0 0 0
cpu1 4898693 1322469 3962745 5713 157 272206 226747 0 0 0
cpu2 3182633 718509 3270409 6002 69 220891 161573 0 0 0
cpu3 3006111 682508 3089431 6025 54 201396 149518 0 0 0
cpu4 1070692 562230 1089597 6459 29 193777 100401 0 0 0
cpu5 1220217 467018 1071711 6517 19 178769 79335 0 0 0
cpu6 1618968 500305 1391534 6444 30 179759 85454 0 0 0
cpu7 1514323 358040 1508875 6493 23 194693 86461 0 0 0
intr 1402060802 0 0 0 94051944 0 98846290 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10188554 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1771665 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3084 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 18856920 5 5 6 0 0 0 0 0 0 0 0 0 72708 3204624 3688114 0 0 0 0 0 0 771
573 0 14679417 0 54894178 0 0 45541 421529 14143028 21107366 0 0 0 85079731 241851 2543814 391356 4278
 0 0 0 0 0 0 122 26 0 28 0 0 0 0 0 0 88 0 0 0 0 168 0 106 905 0 0 245 0 0 0 0 0 0 0 29 0 2781 0 0 0 0
3 0 7 10555 6 47569 114 10362 9194 6568328 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1775 0
ctxt 2671588844
btime 1543613979
processes 3029779
procs_running 3
procs_blocked 0
softirq 472948981 839697 106051363 979944 15095729 131921928 663502 55159674 88090484 300338 73846322
mido:/ $
```

Fonte: O autor (2019).

A predição é feita utilizando os dados históricos e a equação presente no Algoritmo 5 referente ao AEWMA-MSE. Após isso, é obtida a frequência atual da CPU para cada núcleo utilizando comandos ADB, porém, nos *smartphones* testados, não há o chaveamento distinto entre núcleos, ou seja, todos os núcleos do processador operam na mesma frequência.

Após isso, inserimos um condicional para melhorar o desempenho do algoritmo. Percebemos que durante alguns experimentos havia muitas aquisições de contexto com a carga de processamento de CPU muito baixa, próximas de zero. Essas taxas muito baixas não incorporam conhecimento significativo no algoritmo de aprendizagem, levando ao mal uso dos recursos do *smartphone*. Dito isso, optamos por excluir algumas dessas iterações para manter as mais representativas. Esse comportamento é justificável pelo tempo ocioso de processamento durante a execução automatizada dos experimentos, ou seja, ao terminar uma execução antes do tempo previsto, é necessário aguardar o tempo pré-definido para iniciar a próxima iteração.

A Figura 22 ilustra as porcentagens de uso da CPU coletadas durante a execução de um experimento. Antes de aplicar a política condicional, é perceptível que há diversas coletas onde a porcentagem de uso está próxima de zero, gerando conhecimento pouco significativo.

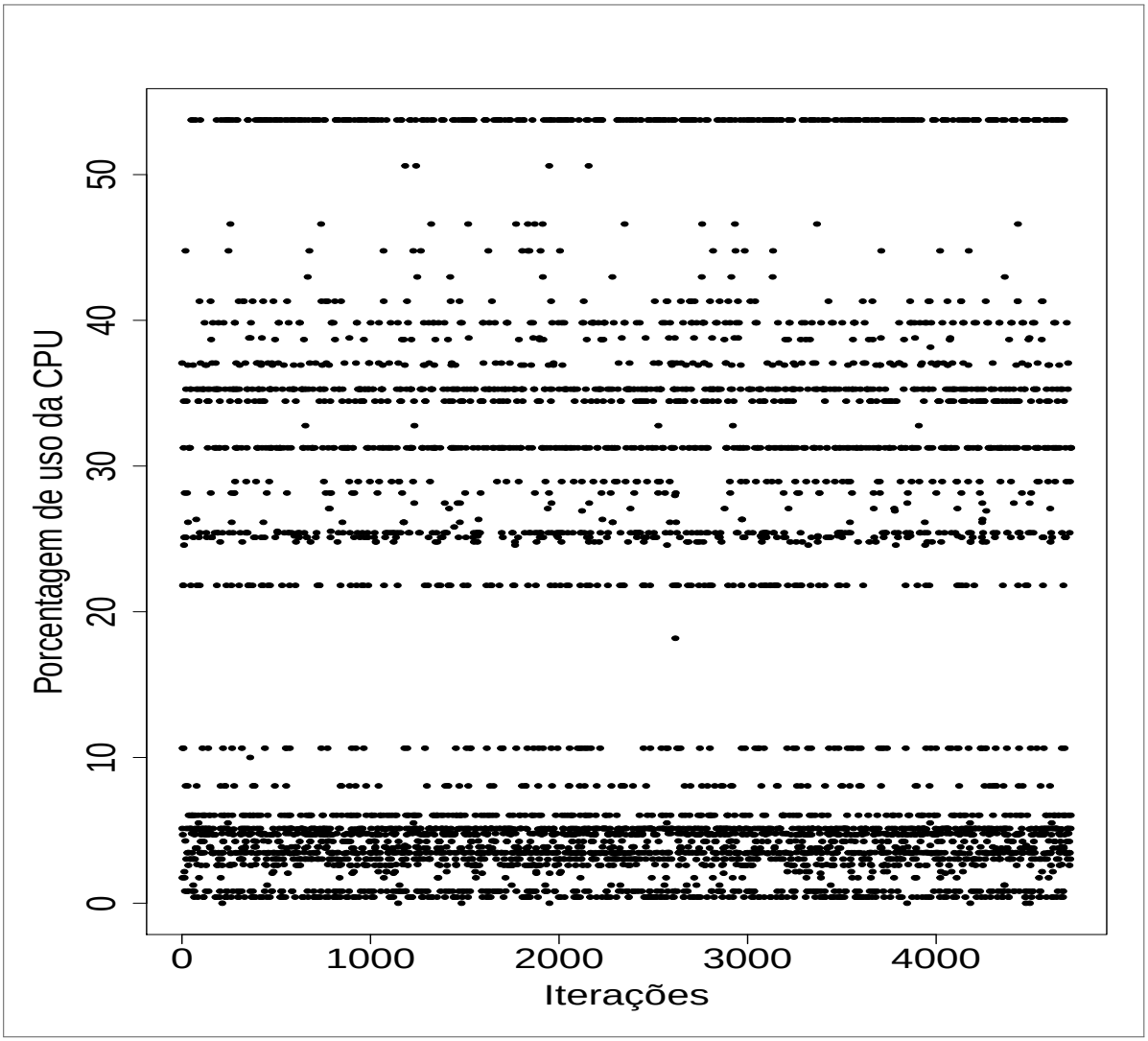
Além disso, a Figura 23 representa o mesmo *benchmark* da Figura 22, sendo executado com a nova abordagem. É perceptível, ao comparar as duas abordagens, que há maior quantidade de amostras abaixo de 10% de uso da CPU na Figura 22, demonstrando que houve uma redução significativa ao aplicar a abordagem com o condicional. Ademais, o fato de ainda existir amostras abaixo de 7%, mesmo com o condicional inserido, se dá pois o condicional é aplicado a carga de processamento predita, sendo os valores presentes nas figuras o valor medido.

Em sequência, no algoritmo, se a carga de processamento estiver abaixo de 7%, é pedido ao processador utilizar a frequência de CPU mais baixa possível. Após isso, retorna-se à execução para o cálculo da utilização média da CPU até que se tenha um valor acima de 7% de utilização.

Ademais, se a carga de processamento for maior que 7% e não houver mudança significativa na carga de processamento predita, é realizado o mapeamento da tupla: carga de processamento predita e frequência de CPU atual para um estado pré-definido no *Q-learning*. Após isso, é escolhida a etapa de acordo com o método *ϵ-greedy* e é selecionado uma ação de exploração ou de explotação.

Em continuidade, é realizado o chaveamento da frequência de CPU, que representa a ação escolhida anteriormente, após isso, entramos no processo de coleta da potência instantânea para a ação tomada. Por conseguinte, realizamos um laço para que o algoritmo só prossiga após a coleta de um valor de potência diferente do valor coletado na iteração anterior, forçando o sensor a atualizar o valor da potência instantânea. Por fim, é realizada

Figura 22 – Exemplo de carga de processamento **antes** da inserção do condicional para cargas menores que 7%.

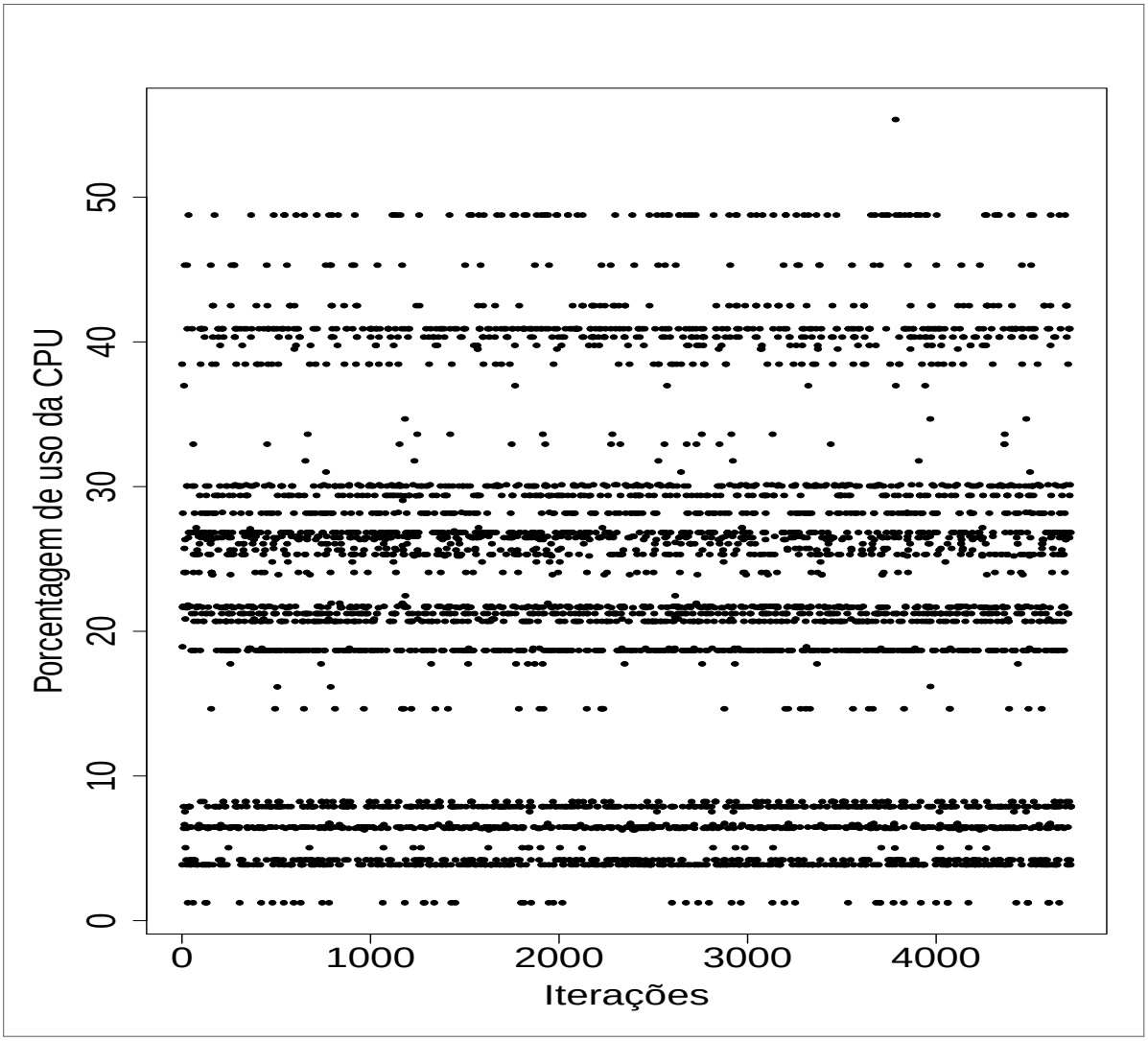


Fonte: O autor (2019).

a leitura do valor de potência e a atualização dos algoritmos *Q-learning* e AEWMA-MSE, finalizando uma iteração.

No próximo capítulo, iremos expor e discutir os resultados alcançados utilizando a abordagem proposta, detalhando cada etapa individualmente e o seu impacto nos resultados alcançados. Além disso, o processo de validação utilizado em cada etapa também será descrito.

Figura 23 – Exemplo de carga de processamento **depois** da inserção do condicional para cargas menores que 7%.



Fonte: O autor (2019).

5 VALIDAÇÃO E RESULTADOS

O objetivo deste capítulo é apresentar os principais resultados obtidos ao utilizar a abordagem proposta. Este capítulo encontra-se dividido em cinco partes principais, ilustradas na Figura 24, e descritas abaixo:

- Etapa (a): é realizada a avaliação dos métodos de predição da carga de processamento EWMA, AEWMA e AEWMA-MSE. São analisadas as variáveis utilização, temperatura e ciclos de CPU. Além disso, são utilizadas as métricas MSE, Erro Médio Absoluto - *Mean Absolute Error* (MAE) e Erro Percentual Médio e Absoluto - *Mean Absolute Percentage Error* (MAPE).

- Etapa (b): é feita a análise da acurácia do modelo de predição de potência, é discutido o balanceamento mais adequado envolvendo o tempo de treinamento e predição para diferentes configurações dos métodos de predição, utilizando regressão linear, ANN e k-NN. Além disso, são analisados os resultados dos *benchmarks* Google Chrome (CH), FB e YT, avaliando os valores preditos com as métricas de erro MSE, MAE e MAPE.

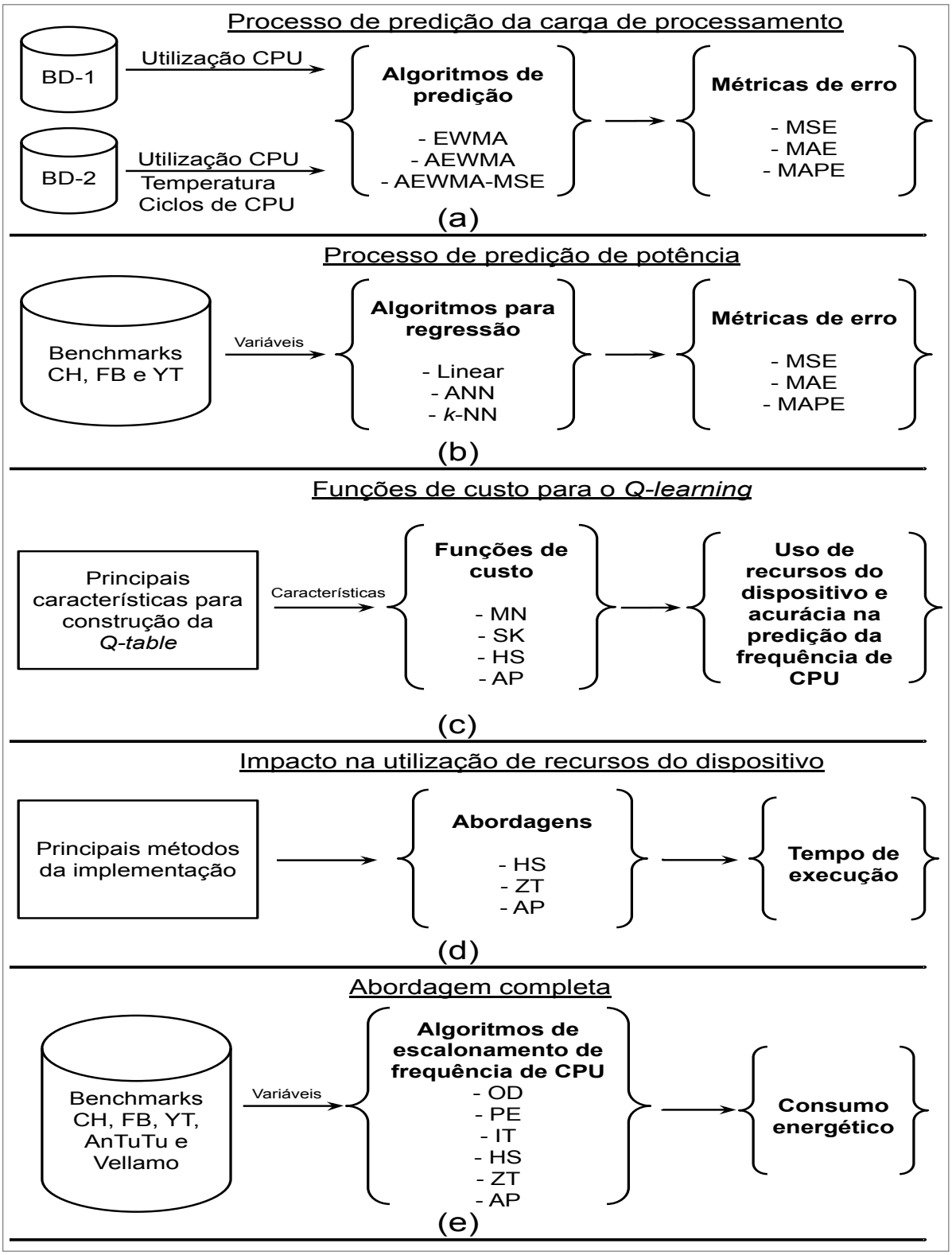
- Etapa (c): é feita a análise da função de custo para o *Q-learning*. Além disso, é investigado o impacto da função de custo nos recursos do dispositivo e as melhorias que ela proporciona quando comparada aos trabalhos recentes da literatura. Por conseguinte, analisamos o impacto da função de custo na predição da frequências da CPU mais indicada para o contexto do dispositivo. A Abordagem Proposta (AP) será comparada com os trabalhos de Maeda-Nunez (2016) (MN), Shafik et al. (2016) (SK) e Shen et al. (2013) (HS).

- Etapa (d): é explanado o impacto do RTM proposto no uso dos recursos do dispositivo, detalhando o tempo de execução dos principais métodos da abordagem. A AP será comparada com os trabalhos de HS e Tian et al. (2018) (ZT).

- Etapa (e): são demonstradas as evidências que indicam a economia de energia quando comparada nossa abordagem aos métodos tradicionais da literatura e indústria. Além disso, é feita a análise, tanto para a abordagem com predição de potência quanto para a abordagem utilizando o sensor de potência nativo no dispositivo. A AP será comparada com as abordagens OD, PE, IT e HS.

Após entender melhor o processo de validação aplicado, vamos detalhar cada etapa da abordagem proposta nas próximas seções, seguindo a ordem: avaliação dos métodos de predição para carga de processamento, modelos para predição de potência, análise da função de custo proposta, economia de energia e uso de recursos no dispositivo.

Figura 24 – Sumário do processo de validação: (a) processo de predição da carga de processamento. (b) processo de predição da potência. (c) processo de análise da função de custo proposta. (d) processo de análise do impacto da abordagem proposta no dispositivo. (e) processo de avaliação da abordagem completa.



Fonte: O autor (2019).

5.1 AVALIAÇÃO DOS MÉTODOS DE PREDIÇÃO PARA CARGA DE PROCESSAMENTO

Nesta seção, é feita a análise dos resultados da acurácia de predição da carga de processamento utilizando os algoritmos EWMA, AEWMA e AEWMA-MSE. Além disso, a análise é feita utilizando duas bases de dados, uma construída a partir de experimentação pelo autor e outra adaptada do trabalho de Walker et al. (2017). Nosso objetivo é analisar a acurácia dos métodos citados anteriormente. Dando continuidade, é analisada a base de dados própria (BD-1) e então a retirada da literatura (BD-2). Ademais, a BD-1 é dividida em três combinações distintas, sendo:

- BD-1.A: gerada a partir do conhecimento de especialistas, sendo uma base sintética adaptada para destacar mudanças bruscas na carga de processamento.
- BD-1.B: composta pela concatenação de atividades de assistir a vídeo no Youtube e não realizar nenhuma atividade. Foi utilizada a sequência de execução de aproximadamente 7 minutos para uma atividade seguida de 7 minutos de outra atividade, sendo 1 minuto para cada frequência de CPU disponível no dispositivo, repetindo até completar 30 minutos.
- BD-1.C: composta pela execução de vídeo no Youtube, seguida de nenhuma atividade, utilizando os algoritmos de chaveamento de frequência (*governors*): IT, OD, PE e PS, sendo alocados 7 minutos de execução para cada.

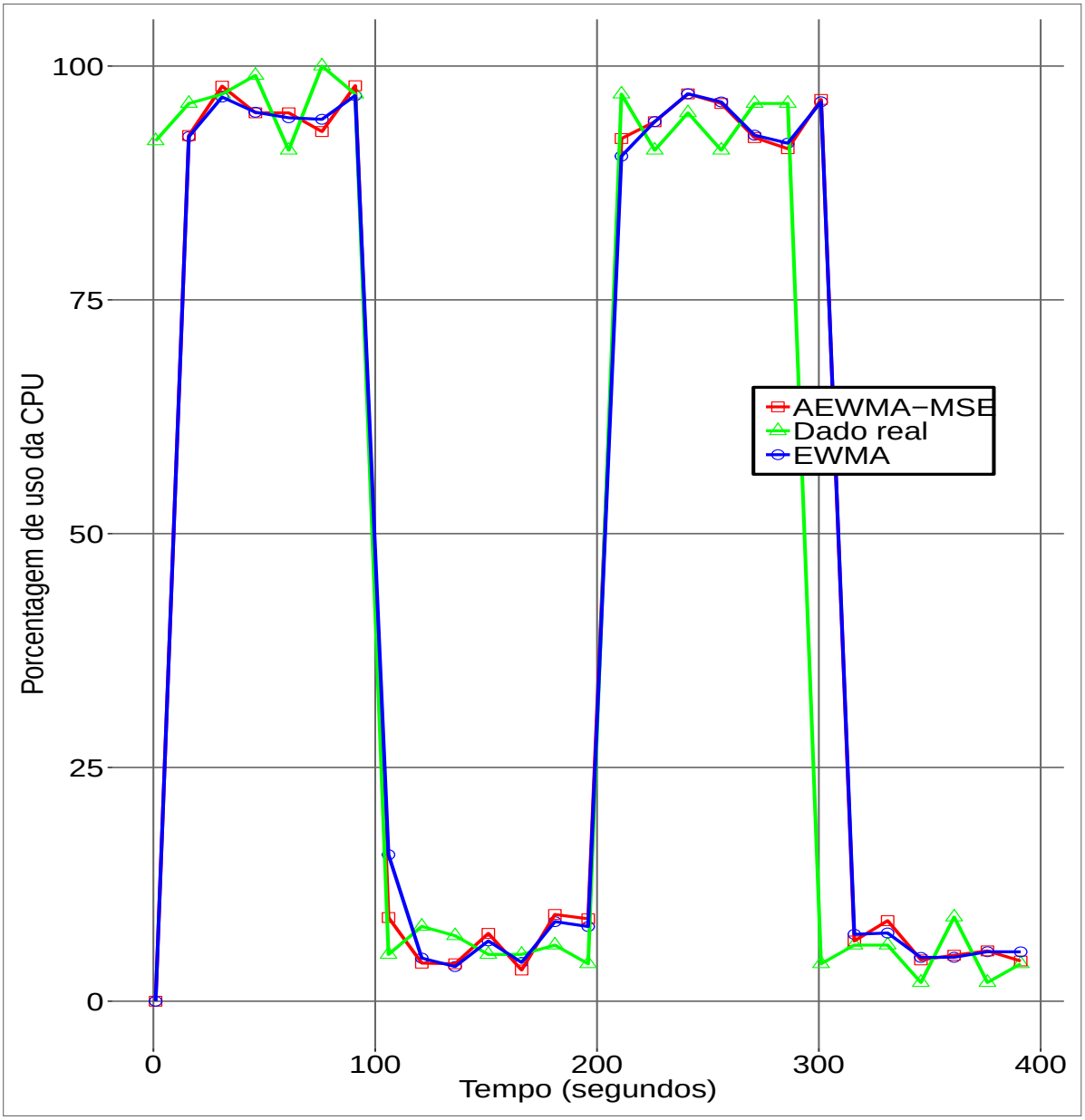
É importante destacar que a BD-1.B difere da BD-1.C, principalmente em: a BD-1.B utiliza uma frequência de CPU fixa até completar o intervalo de tempo de 1 minuto, enquanto na BD-1.C, é utilizado um algoritmo de chaveamento de frequência, onde a frequência da CPU varia de acordo com as heurísticas pré-determinadas e entradas do algoritmo. Além disso, a base de dados DB-1 foi construída utilizando o *smartphone* Motorola XT1033, informações sobre o *smartphone* estão na Tabela 5.

Ademais, é importante relembrar que os algoritmos AEWMA e AEWMA-MSE usam a equação (2.13) para predizer os valores, por isso, somente os dados para o AEWMA-MSE são exibidos nas próximas análises. Em adição, o algoritmo AEWMA-MSE possui a funcionalidade de detecção de mudança na carga de processamento, função essencial usada no algoritmo *Q-learning*.

Além disso, para melhorar a visualização gráfica dos dados, nas Figuras 25, 26, 27, 28, 29 e 30, que envolvem a análise das bases de dados BD-1 e BD-2, somente um pequeno trecho da base de dados foi coletado para a criação dos gráficos. Além disso, os gráficos tiveram alguns pontos não incluídos no procedimento de plotagem, aproximadamente, somente 1 ponto é mostrado para cada $40 \sim 80$ pontos nos dados completos. Em complemento, na análise numérica, todos os pontos são considerados para permitir melhor análise dos dados.

Por conseguinte, a Figura 25 ilustra a base BD-1.A. É perceptível que o AEWMA-MSE se adapta mais rapidamente à nova carga de processamento do que o EWMA para alguns pontos do gráfico. Por exemplo, pode ser visto no momento da mudança da carga de processamento, o primeiro ponto logo após os 100 s. Podemos perceber que o ponto predito pelo AEWMA-MSE está mais próximo do dado real do que o valor predito pelo EWMA. Por esse fenômeno não ser perceptível graficamente nos demais pontos, é necessária a análise numérica, realizada posteriormente a esta análise.

Figura 25 – Base de dados BD-1.A: dado real, EWMA e AEWMA-MSE.

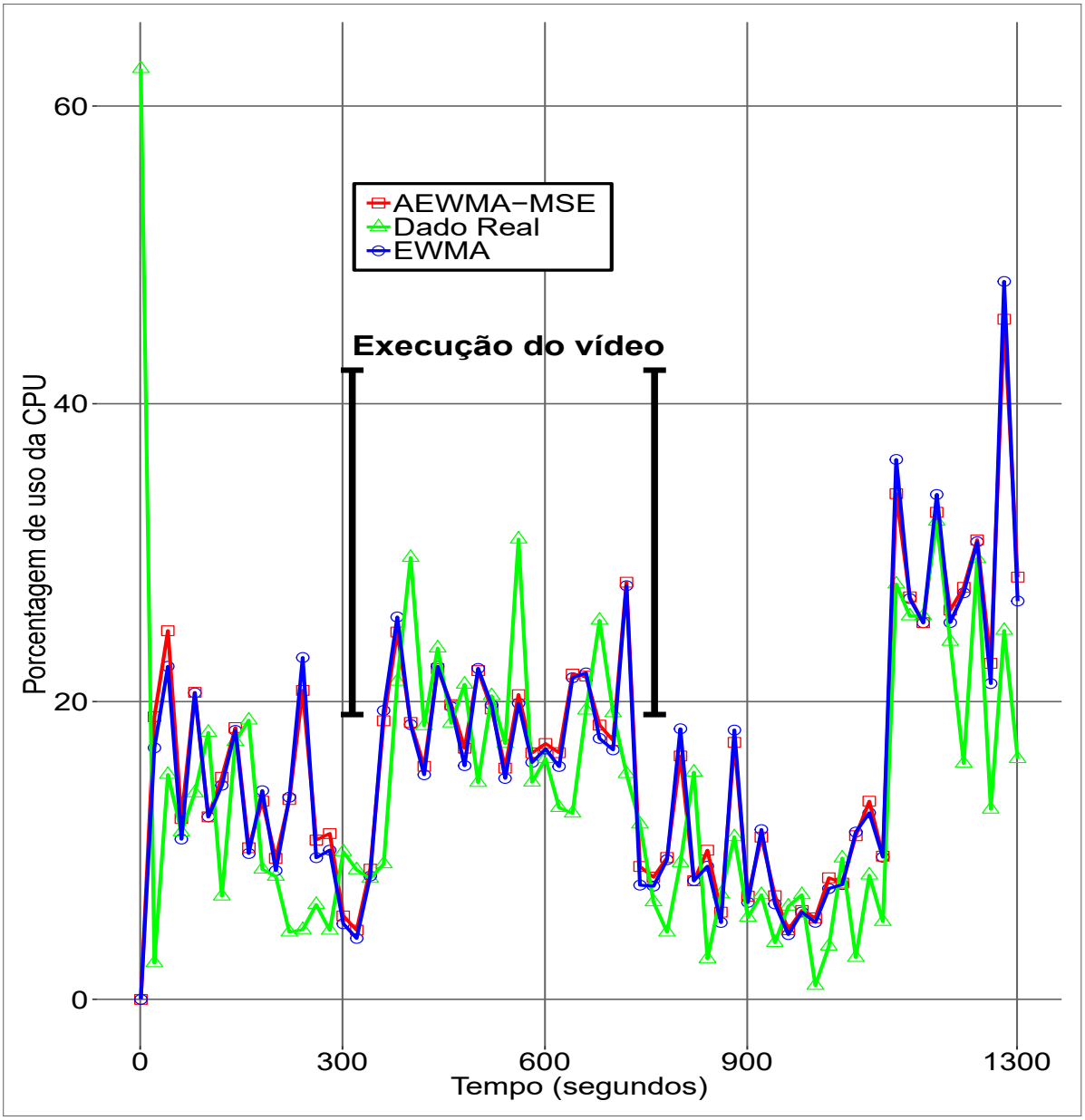


Fonte: O autor (2019).

A Figura 26 ilustra os dados do item BD-1.B. O começo da medição é indicado por um pico entre o tempo 0s até aproximadamente o tempo 35s, seguido pelo registro da ação nenhuma atividade até o tempo 400s. A execução do vídeo está entre o intervalo de

401s até 750s, e assim por diante. O AEWMA-MSE se adapta melhor aos dados reais, mantendo os valores preditos mais próximos da média e evitando os picos.

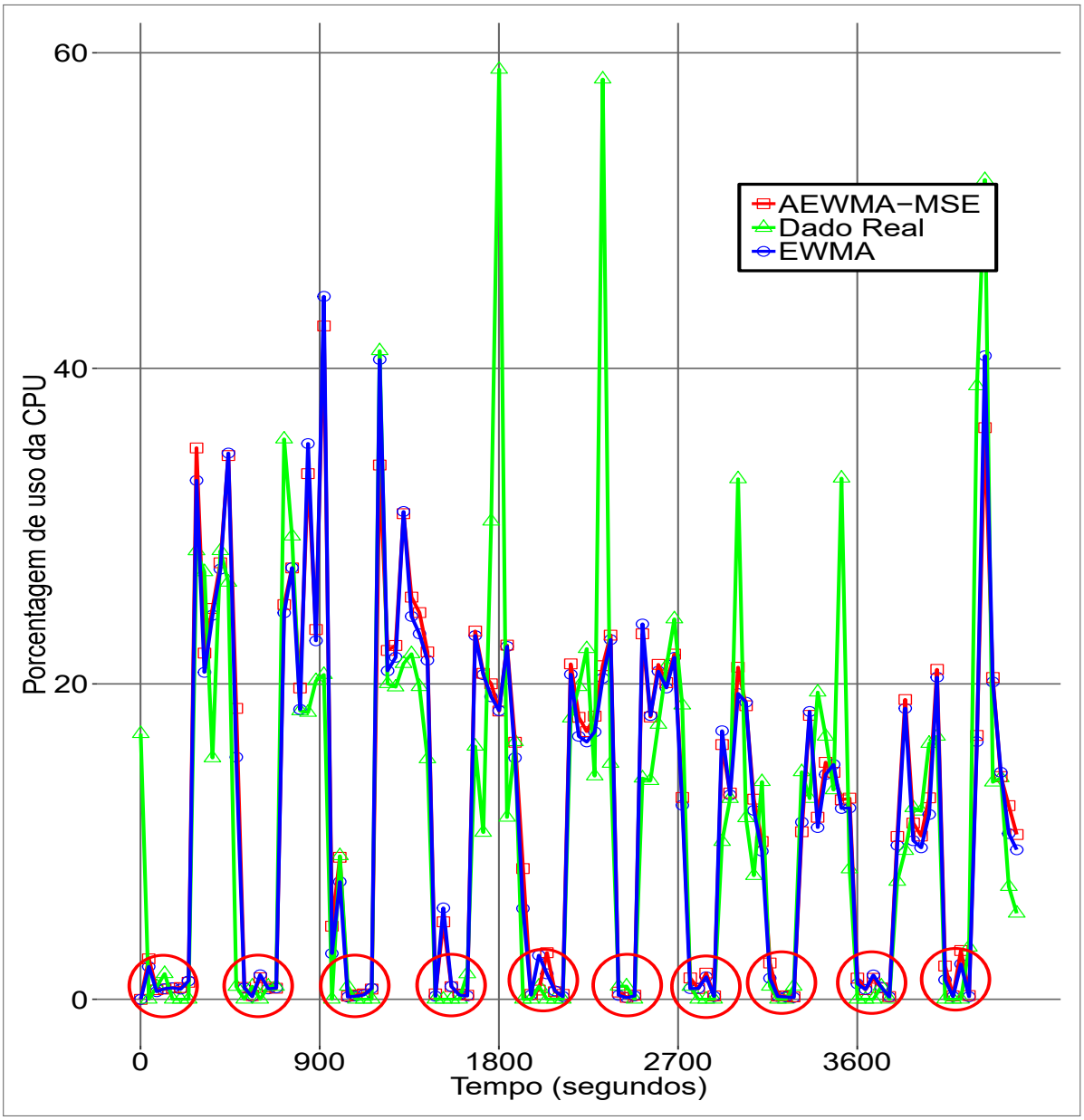
Figura 26 – Base de dados BD-1.B: dado real, EWMA e AEWMA-MSE.



Fonte: O autor (2019).

Além do mais, na Figura 27, é ilustrado o item BD-1.C, no qual é possível ver a oscilação na utilização de CPU dependendo de cada *governor* utilizado. Mesmo quando não há execução de nenhuma atividade em nível de aplicação, ainda há utilização de CPU para manutenção do SO, porém, mais baixa que durante a execução da aplicação. Os pontos circulados em vermelho representam o uso da CPU por processos de manutenção do SO, sendo os pontos acima destes, referentes à execução dos processos a nível de aplicação.

Figura 27 – Base de dados BD-1.C: dado real, EWMA e AEWMA-MSE.



Fonte: O autor (2019).

Diante do exposto, pode-se concluir que o algoritmo AEWMA-MSE aparenta desempenho superior ao EWMA para a maioria dos cenários, mesmo quando o melhor valor de λ é usado no EWMA, valor esse definido para cada combinação da base, variando de acordo com a combinação utilizada. O algoritmo AEWMA-MSE apresentou uma taxa de erro menor para as bases DB-1.A e DB-1.B para as métricas MSE, MAE e MAPE, enquanto para a DB-1.C, o EWMA apresentou uma taxa de erro menor, mas bem próxima dos valores para o AEWMA-MSE. Os erros de predição utilizando as métricas definidas são apresentados na Tabela 6.

Tabela 6 – Erros de predição para os algoritmos EWMA e AEWMA-MSE usando as métricas MSE, MAE e MAPE, considerando a utilização de CPU para DB-1.

	DB-1.A		DB-1.B		DB-1.C	
	EWMA	AEWMA-MSE	EWMA	AEWMA-MSE	EWMA	AEWMA-MSE
MSE	0,013	0,010	0,012	0,011	0,009	0,009
MAE	0,049	0,040	0,068	0,067	0,050	0,050
MAPE (%)	0,637	0,499	0,759	0,748	0,615	0,636

Fonte: O autor (2019).

Tabela 7 – Erros de predição para os algoritmos EWMA e AEWMA-MSE usando as métricas MSE, MAE e MAPE, considerando a base de dados DB-2 em sua forma original e após o uso da validação cruzada com 10-*fold*.

DB-2 original

	Utilização de CPU		Temperatura		Ciclos de CPU	
	EWMA	AEWMA-MSE	EWMA	AEWMA-MSE	EWMA	AEWMA-MSE
MSE	782,434	835,578	27,804	28,250	$9{,}537\times10^{9}$	$1{,}011\times10^{9}$
MAE	16,648	16,835	2,499	2,499	$1{,}651\times10^{8}$	$1{,}653\times10^{8}$
MAPE (%)	2,481	2,529	0,048	0,048	3,289	3,334
DB-2 com validação cruzada						
	EWMA	AEWMA-MSE	EWMA	AEWMA-MSE	EWMA	AEWMA-MSE
MSE	1152,244	1301,006	226,301	257,501	$175{,}64\times10^{15}$	$199{,}65\times10^{15}$
MAE	25,073	26,129	11,777	12,521	$2{,}92\times10^{8}$	$3{,}08\times10^{8}$
MAPE (%)	2,085	2,079	0,249	0,264	2,889	2,905

Fonte: O autor (2019).

Dando continuidade, na BD-2, são coletados 60 tipos de processamentos distintos, retirados das bases de dados *MiBench* (GUTHAUS et al., 2001), *LMBench* (MCVOY; STAELIN, 1996), *Longbottom*, *MediaBench* e mais alguns construídos para complementar as bases. Cada tipo de processamento é repetido com diferentes configurações da CPU, alterando a frequência e a quantidade de núcleos ativos no processador. Para cada execução é calculada a média da repetição de 150 execuções.

Ainda sobre a BD-2, o processador analisado é um *Advanced RISC Machine* (ARM) com arquitetura *big.LITTLE* possuindo dois processadores: um *ARM Cortex A7*, para baixas cargas de processamento; e um *ARM Cortex-A15*, para processamento de alto desempenho. O *Cortex-A15* é o que possui uma maior variabilidade nas cargas de processamento e temperatura, sendo o objeto de estudo das próximas análises.

A Figura 28 ilustra os valores reais dos dados usando a métrica de utilização média da CPU e as predições feitas pelo EWMA e AEWMA-MSE. As operações utilizadas exercitam bem a utilização da CPU, provendo amostras de praticamente todos os valores possíveis de utilização e também utilizando diferentes frequências e quantidades de núcleos ativos no processador.

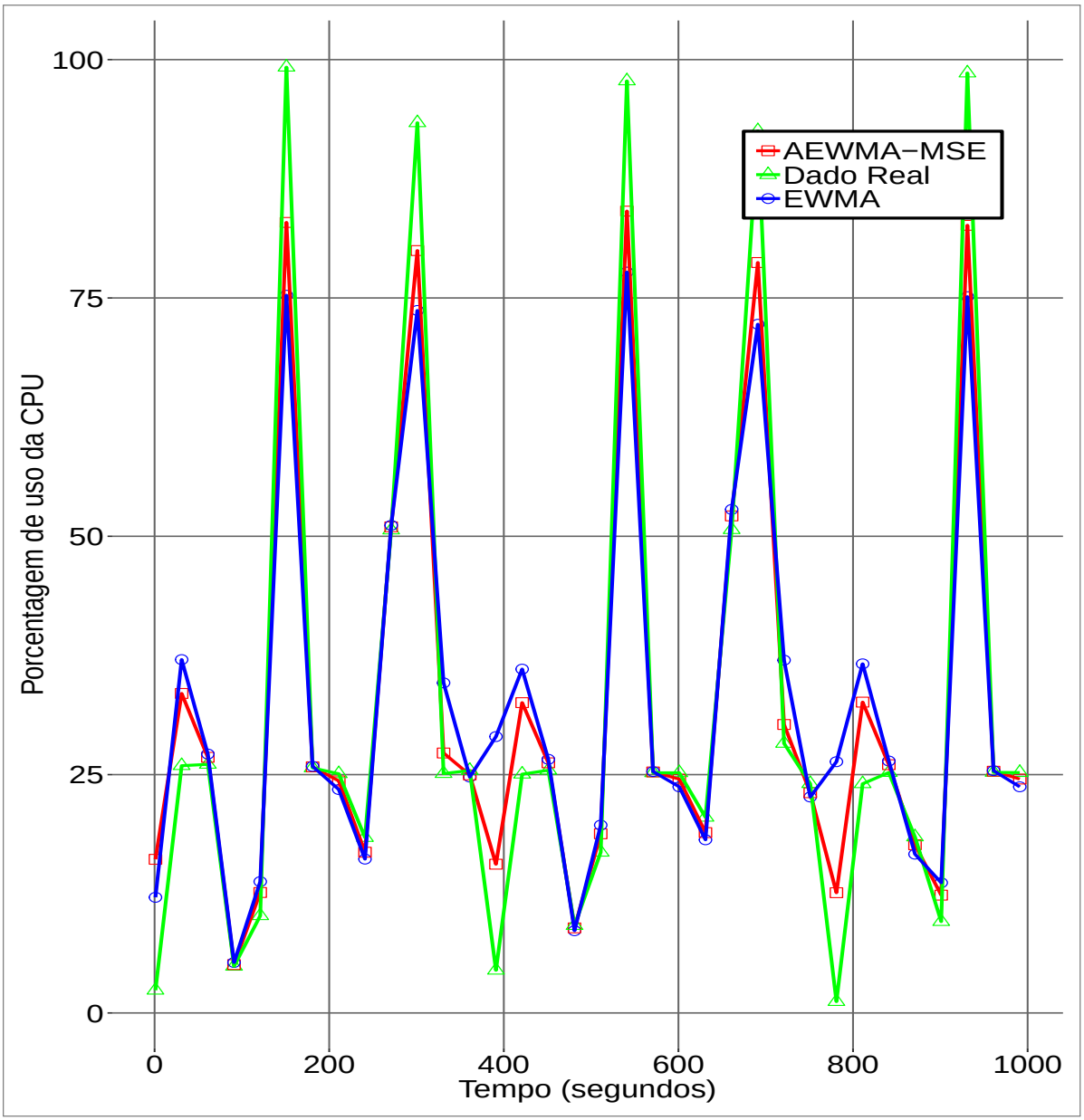
Para também contemplar a predição de outras métricas além da utilização da CPU, foram analisadas as métricas de temperatura e ciclos de CPU. Dito isso, a Figura 29 ilustra os valores de temperatura média da CPU e as predições feitas pelo EWMA e AEWMA-MSE. Como também, a Figura 30 ilustra os valores para a métrica ciclos de CPU.

Em síntese, a Tabela 7 apresenta os valores obtidos no cálculo das métricas de acurácia para toda a base de dados BD-2. Observando a Tabela 7, é possível notar que os valores das métricas de acurácia do EWMA e do AEWMA-MSE são próximas, refletindo os resultados obtidos na análise da BD-1. Assim sendo, exige-se uma validação mais precisa. Para proporcionar uma melhor avaliação, foi feita a verificação através do método de validação cruzada juntamente com o teste de médias de *Wilcoxon* usando postos (REY; NEUHAUSER, 2011).

Ademais, a acurácia dos algoritmos de predição é avaliada usando teste de hipótese estatística. Normalmente, esse teste de médias é utilizado para comparar se há diferença significativa entre os resultados obtidos por diferentes algoritmos. A hipótese nula, representada por H_0, determina que as médias em análise são iguais. No teste de hipótese, é avaliada a probabilidade das amostras terem o mesmo valor do parâmetro real. A hipótese oposta é chamada de hipótese alternativa, representada por H_a e ela contradiz a hipótese nula. Caso a hipótese nula seja rejeitada, a hipótese alternativa é válida.

O teste estatístico é realizado a partir dos dados das amostras e depende das características do problema. Para o nosso, utilizamos o teste de *Wilcoxon*, pois este não assume que os dados sigam uma distribuição de probabilidade. Isto é, ele não requer nenhuma suposição sobre o formato da distribuição. Além disso, o teste estatístico produz uma

Figura 28 – Base de dados BD-2 (utilização de CPU): dado real, EWMA e AEWMA-MSE.

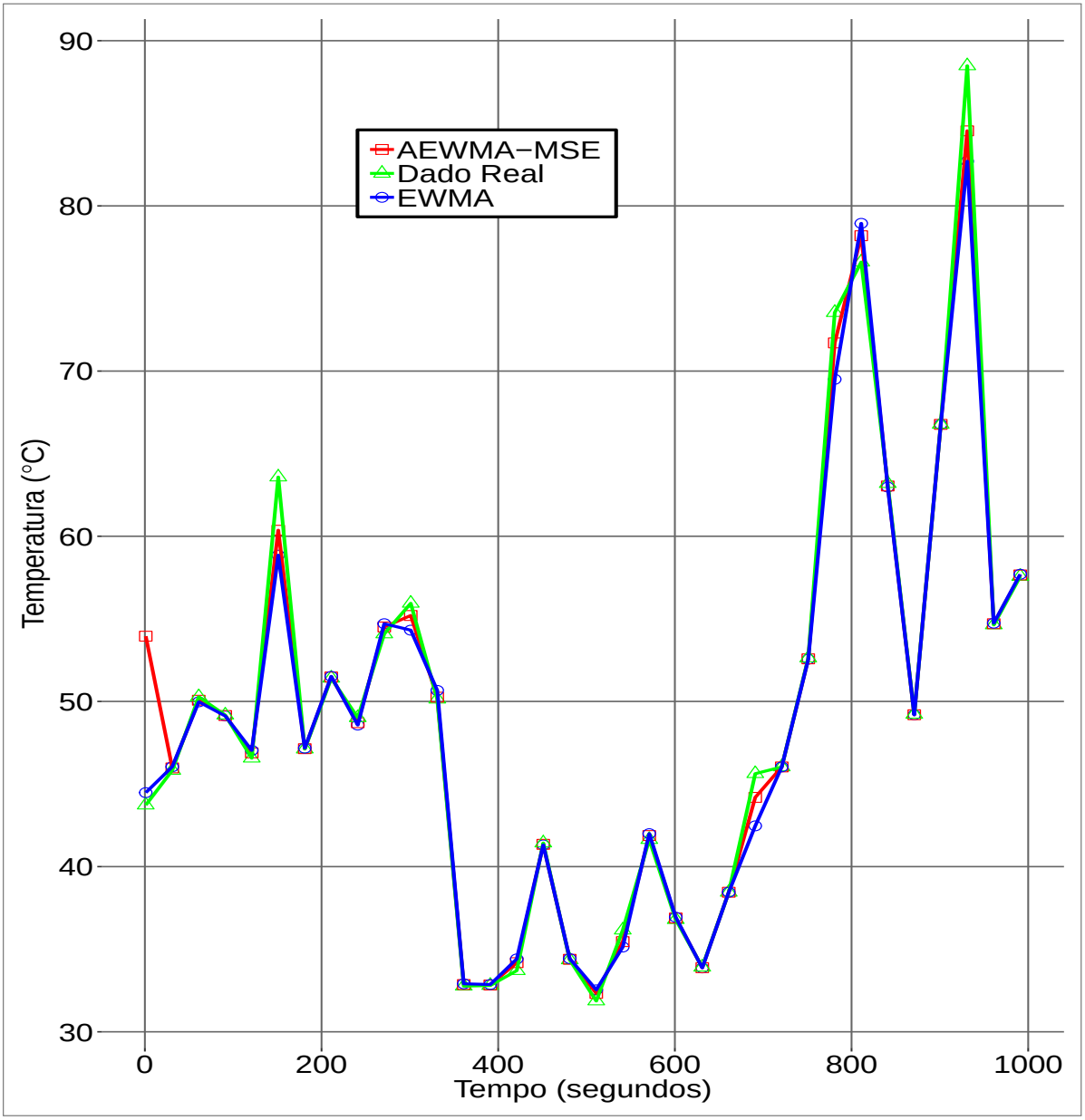


Fonte: O autor (2019).

informação de probabilidade condicional chamada de $p - value$, sendo este a probabilidade de se obter uma estatística de teste mais extrema do que aquela observada em uma amostra, assumindo verdadeira a hipótese nula. Quanto menor o valor do $p - value$, mais evidência se tem contra a hipótese nula.

No teste de hipótese, assume-se um nível de significância para realizar uma decisão sobre a hipótese nula, assim, assumiremos um nível de significância de 1%, de tal forma que se o valor de $p - value$ for menor do que $0,01$ rejeita-se a hipótese nula. De forma resumida, se o valor do $p - value$ for maior que o nível de significância (0,01 para 1%), indica que os algoritmos comparados não possuem diferença estatística nos resultados.

Figura 29 – Base de dados BD-2 (temperatura): dado real, EWMA e AEWMA-MSE.



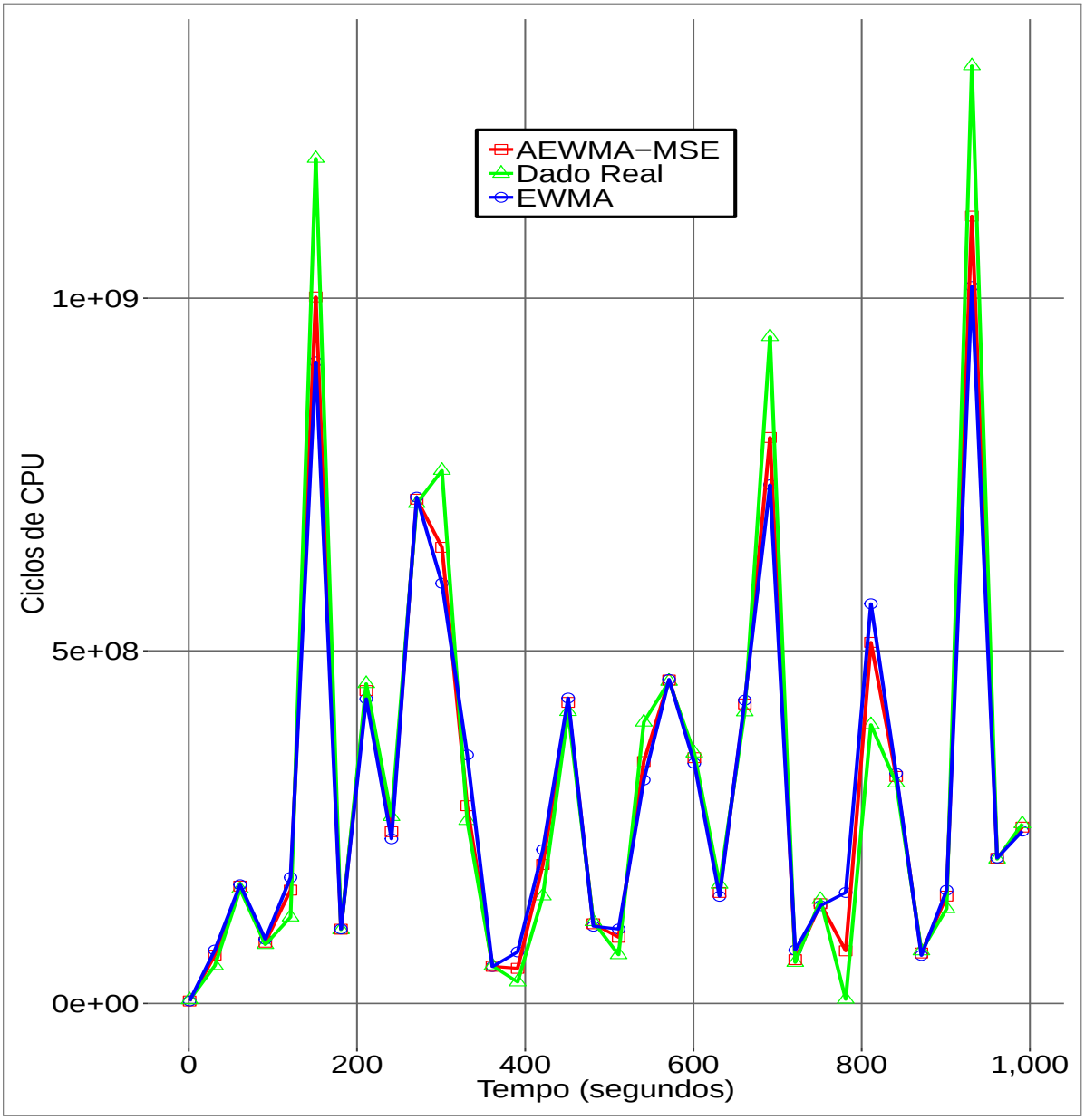
Fonte: O autor (2019).

Também realizamos os testes para significâncias maiores, de 5% e 10%, obtendo resultados similares.

Para gerar as médias amostrais do teste estatístico, usamos a técnica de validação cruzada. Usamos $k = 10$, onde as amostras são dividas em partes iguais ou muito próximas. Após feita a divisão, os algoritmos serão avaliados sob as métricas de acurácia e repetidos 100 vezes cada. Os valores médios obtidos para as 100 repetições serão analisados usando o teste de hipótese de *Wilcoxon* para comparação estatística das médias, resultando na resposta de qual algoritmo possui a maior acurácia ou se são estatisticamente similares.

Após a execução do teste estatístico, foi obtido um $p - value = 0.63122$ usando 1% de precisão, demonstrando que os algoritmos EWMA e AEWMA-MSE possuem grande

Figura 30 – Base de dados BD-2 (ciclos de CPU): dado real, EWMA e AEWMA-MSE.



Fonte: O autor (2019).

similaridade para a base de dados testada.

Dando continuidade, percebe-se uma redução da acurácia para o estudo de caso com a temperatura, quando comparado o erro sem a validação cruzada, sendo o valor do MAPE de $0,0485$ e $0,0481$ e MAPE de $0,2495$ e $0,2642$ para EWMA e AEWMA-MSE, respectivamente. Há um aumento considerável nos erros de predição, sendo justificado pelo seguinte motivo: ao ser usado o embaralhamento dos dados pela validação cruzada, os valores de temperatura ficaram distantes um do outro, uma temperatura de $30°C$ mudando instantaneamente para $90°C$, por exemplo, demonstra uma situação pouco provável no uso comum do dispositivo. Isso levou a um atraso na adaptação dos algoritmos de predição e, consequentemente, ao aumento dos erros.

Diante do exposto, dada a análise detalhada dos dois algoritmos em uma base de dados própria e na base de dados adaptada de Walker et al. (2017), coletadas de maneira distinta, temos evidências suficientes para assumir que o EWMA com λ fixo e otimizado para a base de dados se comporta de maneira similar ao AEWMA-MSE em termos de acurácia de predição para ambientes com distintas cargas de processamento, refletindo cenários reais de uso. É importante destacar que o EWMA necessita de um treinamento para descobrir o valor de λ que minimiza o erro, etapa não necessária no algoritmo proposto, o AEWMA-MSE, pois este se adapta de forma autônoma às diferentes cargas de processamento, descobrindo o melhor valor para λ de forma automática.

O trabalho de Maeda-Nunez (2016) relata atingir melhores resultados com o AEWMA, pois não considera ambientes com variações de carga consideráveis, analisando somente ambientes pontuais de reprodução de vídeos. Além disso, o trabalho de Maeda-Nunez (2016) inviabiliza o uso em ambientes genéricos, caso em que este trabalho se enquadra. Por fim, no trabalho de Maeda-Nunez (2016) não é realizada nenhuma análise estatística para possibilitar maior segurança aos resultados obtidos.

5.2 MODELOS DE PREDIÇÃO DE POTÊNCIA

Nesta seção, são apresentados os resultados da predição de potência para o dispositivo testado. De forma específica, detalhamos a acurácia dos métodos utilizados, as frequências de CPU que consomem menos energia para determinados cenários e os custos computacionais de cada método de predição.

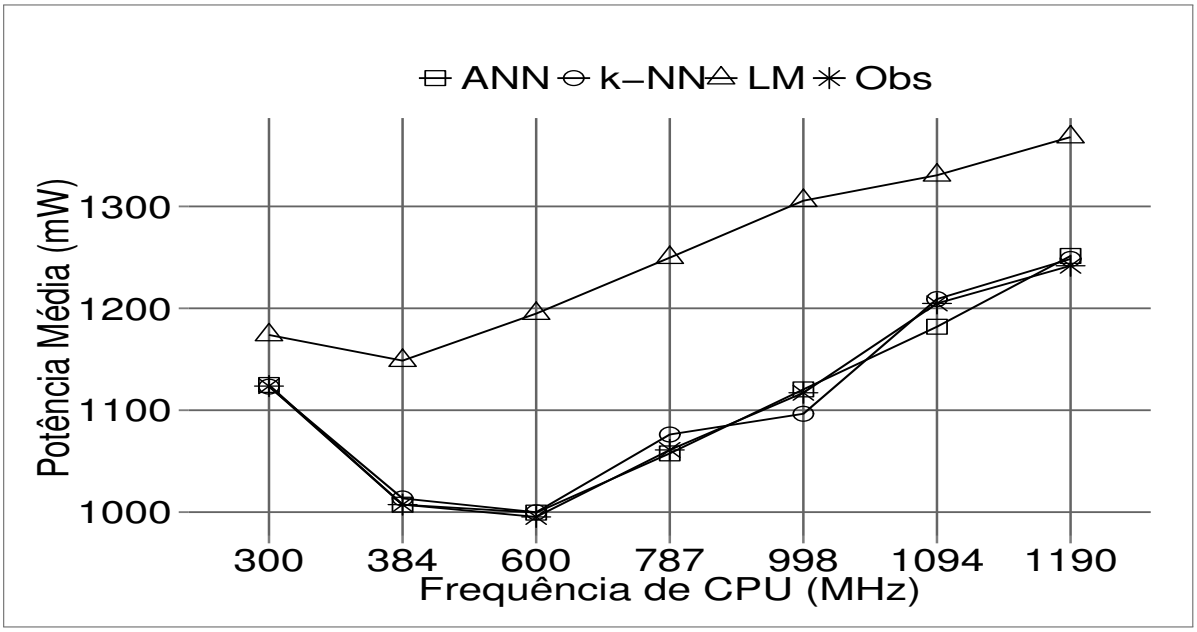
Os métodos analisados são: regressão linear, k-NN e ANN. O objetivo é prever a potência e encontrar a frequência de CPU que consome menos potência para o contexto em um fluxo contínuo de execução da aplicação. A frequência de CPU que consome menos energia é chamada de *Low-power frequency* (f_{LP}).

Diante disso, a Figura 31 ilustra a potência média consumida para todas as frequências de CPU disponíveis no dispositivo, considerando o aplicativo CH. A partir dos dados coletados, pode-se perceber que, a partir da frequência de 600 MHz, o consumo de potência aumenta, mostrando que $f_{LP} = 600\ MHz$ para essa aplicação.

Por outro lado, é observado que os valores preditos pelos métodos não-lineares (k-NN e ANN) se adaptam melhor aos dados reais, também indicando a frequência de 600 MHz como a frequência que exibe o menor consumo de potência. Entretanto, quando utilizado o *Linear Model* (LM), ficou claro não ser uma boa opção, visto que são preditos valores maiores que os reais.

A Figura 32 ilustra o consumo médio de potência para a aplicação FB, também utilizando todas as frequências de CPU disponíveis no dispositivo. Neste caso, pode ser notado um comportamento diferente do observado na aplicação CH para a relação frequência de CPU e potência média, sendo a $f_{LP} = 787\ MHz$ para esta aplicação, indicando a frequência ideal para alcançar o menor consumo de potência.

Figura 31 – Potência média consumida utilizando as frequências de CPU disponíveis para o aplicativo CH.



Fonte: O autor (2019).

Além disso, o preditor linear previu valores próximos do real a partir da frequência de 787 MHz, prevendo valores mais baixos nas demais frequências, enquanto os modelos não-lineares alcançaram valores de potência bem próximos aos reais para todas as frequências, exceto para as frequências 1094 MHz e 1190 MHz, onde o k-NN apresentou um erro maior que a ANN (1094 MHz) e menor que a LM (1190 MHz).

Por fim, a potência média consumida para a aplicação YT é ilustrada na Figura 33. Para esta aplicação, o modelo linear previu a frequência de 300 MHz como a f_{LP}, entretanto, a frequência 384 MHz é a f_{LP} medida. Com isso, tanto ANN quanto k-NN alcançaram bons resultados de predição em todas as frequências analisadas, também obtendo a f_{LP} medida.

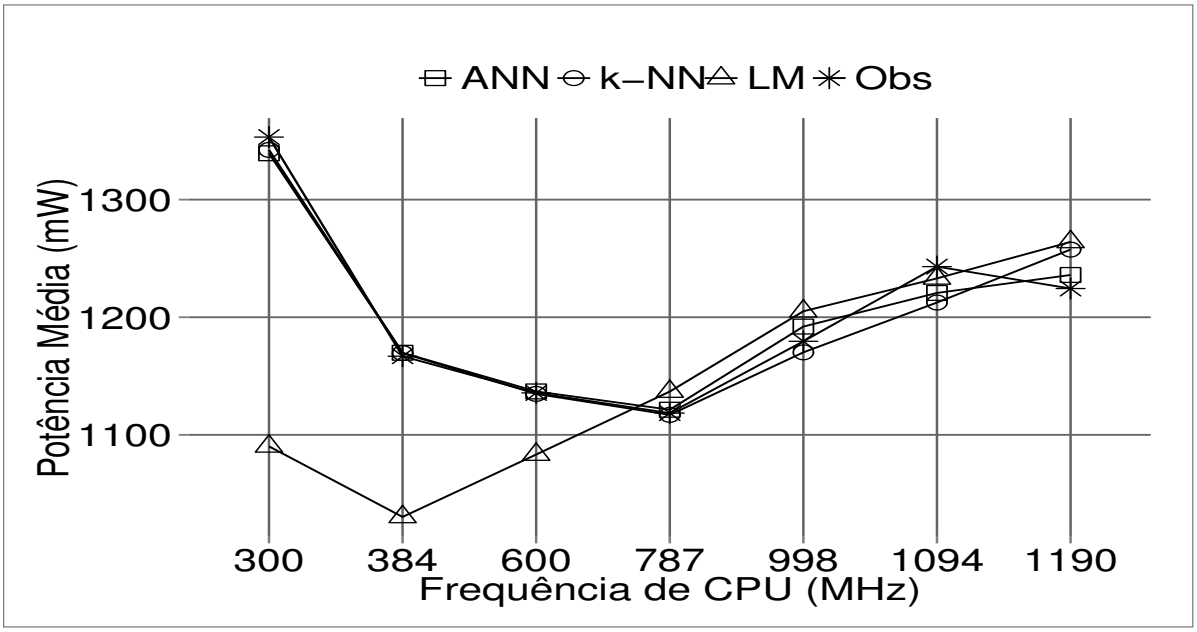
Resumindo o que já foi exposto, a Tabela 8 sumariza a f_{LP} obtida em cada tipo de aplicação. O $f_{LP_{Obs}}$ representa a f_{LP} obtida a partir dos dados reais coletados; a $f_{LP_{ANN}}$ é a f_{LP} predita usando ANN; a $f_{LP_{kNN}}$ é a f_{LP} predita usando k-NN; e a $f_{LP_{LM}}$ é a f_{LP} predita usando LM.

Tabela 8 – Sumário da f_{LP} em MHz por aplicação.

	$f_{LP_{Obs}}$	$f_{LP_{ANN}}$	$f_{LP_{kNN}}$	$f_{LP_{LM}}$
CH	600	600	600	384
FB	786	786	786	384
YT	384	384	384	300

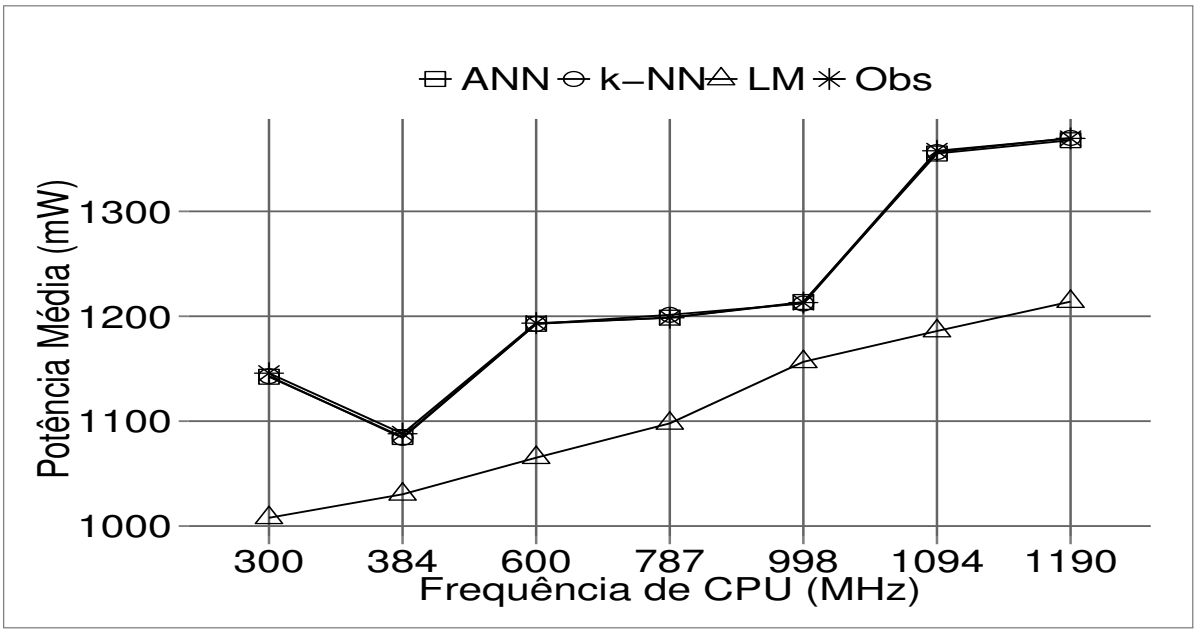
Fonte: O autor (2019).

Figura 32 – Potência média consumida utilizando as frequências de CPU disponíveis para o aplicativo FB.



Fonte: O autor (2019).

Figura 33 – Potência média consumida utilizando as frequências de CPU disponíveis para o aplicativo YT.



Fonte: O autor (2019).

Ademais, é esperado que uma frequência de CPU mais alta consuma mais potência que uma frequência de CPU mais baixa, porém, neste trabalho, é importante destacar que estamos medindo a potência consumida por todo o dispositivo, não somente pelo processador. Logo, podemos ter frequências diversas para cada cenário. Além disso, frequências

mais baixas necessitam fazer uso intenso da CPU (maior porcentagem de uso), podendo levar a um consumo maior de potência (DAUD et al., 2014). Em adição, a CPU com altas cargas de processamento inibe a entrada em estados de descanso profundo, que economizam energia.

Juntamente com as métricas de acurácia, foi aplicada a técnica de validação cruzada *K-fold*, usando $K = 10$ para calcular o erro médio, os resultados são exibidos na Tabela 9, sendo analisado o método com ANN para quatro variações de neurônios: $n = 3, 10, 100$, e 1000. Para o modelo utilizando k-NN, foi avaliado o desempenho com as seguintes variações no número de vizinhos: $k = 3, 10, 100$, e 1000.

Tabela 9 – Erro médio obtido usando validação cruzada com *K-fold*.

Modelo	MSE	MAE	MAPE
LM	47636,47	171,75	0,16
ANN ($n = 3$)	**41042,97**	**115,18**	**0,12**
ANN ($n = 10$)	44783,54	120,21	**0,12**
ANN ($n = 100$)	44537,46	120,11	**0,12**
ANN ($n = 1000$)	46884,86	124,43	0,13
k-NN ($k = 3$)	41246,90	140,17	0,12
k-NN ($k = 10$)	30517,03	120,08	0,10
k-NN ($k = 100$)	**26623,58**	**111,61**	**0,09**
k-NN ($k = 1000$)	36728,76	150,32	0,14

Fonte: O autor (2019).

Sobre a análise do método utilizando ANN, todas as configurações testadas possuem um valor de erro bem próximos, entretanto, a ANN usando três neurônios superou as outras configurações com mais neurônios, obtendo um erro médio menor. Uma explicação possível é que um número grande de neurônios pode causar *over-fitting* em ANNs (LAWRENCE; GILES; TSOI, 1996), então, ao invés de melhorar, muitos neurônios podem piorar a acurácia de predição.

Além disso, quanto à análise para o k-NN, a configuração com $k = 100$ vizinhos superou todas as outras configurações do k-NN e também os demais algoritmos em todas as métricas analisadas, sendo o pior caso atribuído ao LM, obtendo os maiores valores de erro para todas as métricas.

É importante observar que o k-NN usando $k = 1000$ obteve maiores erros de predição quando comparado com as outras configurações do k-NN ($k = 3, 10, 100$), um alto número de vizinhos no cálculo da distância implica em incluir classes muito distantes do ponto analisado, resultando em valores preditos fora do escopo esperado (JAMES et al., 2013).

Dando continuidade às análises, outro fator importante é a complexidade dos modelos, retratada aqui como o tempo de execução. Assim, foram coletados os tempos de treinamento t_{tr} e predição t_{pd}. O t_{tr} é o tempo necessário para ajustar os parâmetros do modelo

até um nível mínimo de erros, enquanto o t_{pd} é o tempo necessário para o cálculo do valor de saída utilizando os valores de entrada e o modelo treinado.

Em suma, a Tabela 10 ilustra os valores médios de t_{tr} e t_{pd} para cada modelo de predição de potência considerado neste trabalho. Os dados são relativos à plataforma utilizada, sendo os valores mínimos de treinamento e predição na ordem dos décimos de segundos.

Ademais, o menor valor de tempo foi escalado para 1 (valor de referência) e os outros valores de tempo foram comparados ao valor de referência. Por exemplo, considerando a tarefa de treinamento, o tempo consumido pelo modelo ANN ($n = 100$) é, aproximadamente, 105 vezes maior que o tempo consumido pelo valor de referência, o método LM.

Tabela 10 – Custo computacional relativo para os modelos de predição.

Modelo	t_{tr}	t_{pd}
LM	1.00	15.00
ANN ($n = 3$)	65.41	4545.00
ANN ($n = 10$)	73.25	4590.00
ANN ($n = 100$)	105.41	4695.00
ANN ($n = 1000$)	293.53	4805.00
k-NN ($k = 3$)	3.07	1.00
k-NN ($k = 10$)	4.02	1.00
k-NN ($k = 100$)	4.43	1.50
k-NN ($k = 1000$)	15.46	1.50

Fonte: O autor (2019).

Analisando os valores, é perceptível que o modelo de predição com o menor tempo de treinamento foi o LM. Entretanto, este modelo apresentou a pior taxa de acertos nas métricas de erros. Além disso, todos os modelos utilizando k-NN possuem aproximadamente o mesmo valor de tempo, sendo os mínimos quando $k = 3$ e $k = 10$, seguido do LM e ANN.

Assim sendo, pode ser visto que o modelo utilizando k-NN possui o melhor balanceamento entre desempenho e acurácia quando comparado aos outros modelos analisados. Entretanto, a análise realizada sugere que a configuração $k = 100$ é a melhor opção dentre todas as analisadas de acordo com a base de dados utilizada, justificado pela melhor acurácia e relativo baixo tempo de treinamento e predição. Deste momento em diante, vamos utilizar o modelo k-NN ($k = 100$) para a realização da predição da potência nos próximos experimentos.

5.3 ANÁLISE DA FUNÇÃO DE CUSTO

A função de custo proposta para o *Q-learning* e utilizada em nosso trabalho é baseada nos valores de potência obtidos a partir do contexto do dispositivo, podendo ser obtido utilizando o preditor de potência com k-NN ($k = 100$) ou diretamente do sensor de potência presente em alguns *smartphones*. Os valores de potência obtidos a partir de cenários específicos do dispositivo são utilizados para guiar o processo de aprendizagem do *Q-learning*.

Para comparar as capacidades de uso de memória da abordagem proposta e da abordagem convencional do *Q-learning*, usamos o tamanho da *Q-table*, S_Q, definido abaixo:

$$S_Q = |A| \cdot |S| \,, \tag{5.1}$$

onde $|A|$ é o número de frequências de CPU e $|S|$ representa o espaço de estados do algoritmo *Q-learning*, dado por

$$|S| = \Delta\mu \cdot \rho \,, \tag{5.2}$$

onde $\Delta\mu$ é o número de partições da carga de processamento e ρ é o número de tipos distintos de carga de processamento. Por exemplo, $\Delta\mu = 5$ significa que a utilização da CPU, um valor pertencente ao intervalo $[0 - 1]$, é dividido em cinco partições. Em outras palavras, isso implica que cada partição é um intervalo da carga de processamento, sendo a primeira partição o subintervalo $[0 - 0,2]$.

A Tabela 11 exibe os tamanhos da *Q-table* para a abordagem convencional do *Q-learning* e o método proposto neste trabalho. As primeiras três colunas (da esquerda para a direita) correspondem aos parâmetros usados para calcular S_Q de acordo com (5.1) e (5.2). É perceptível que a abordagem convencional da *Q-table*, explorada em Shen et al. (2013) e Maeda-Nunez (2016), depende fortemente de ρ. O fato de depender fortemente de ρ resulta em um aumento considerável dos estados possíveis quando se tem muitas cargas de processamento distintas. Como pode ser percebido, para uma carga de processamento dinâmica, a abordagem convencional de modelagem da *Q-table* resulta em $S_Q = 5000$ pares de estado-ação, assumindo $|A| = 10$, $\rho = 50$ e $\Delta\mu = 10$. Possuir pequenas *Q-tables* beneficia a abordagem, permitindo aprendizado e convergência de forma acelerada (SHAFIK et al., 2016).

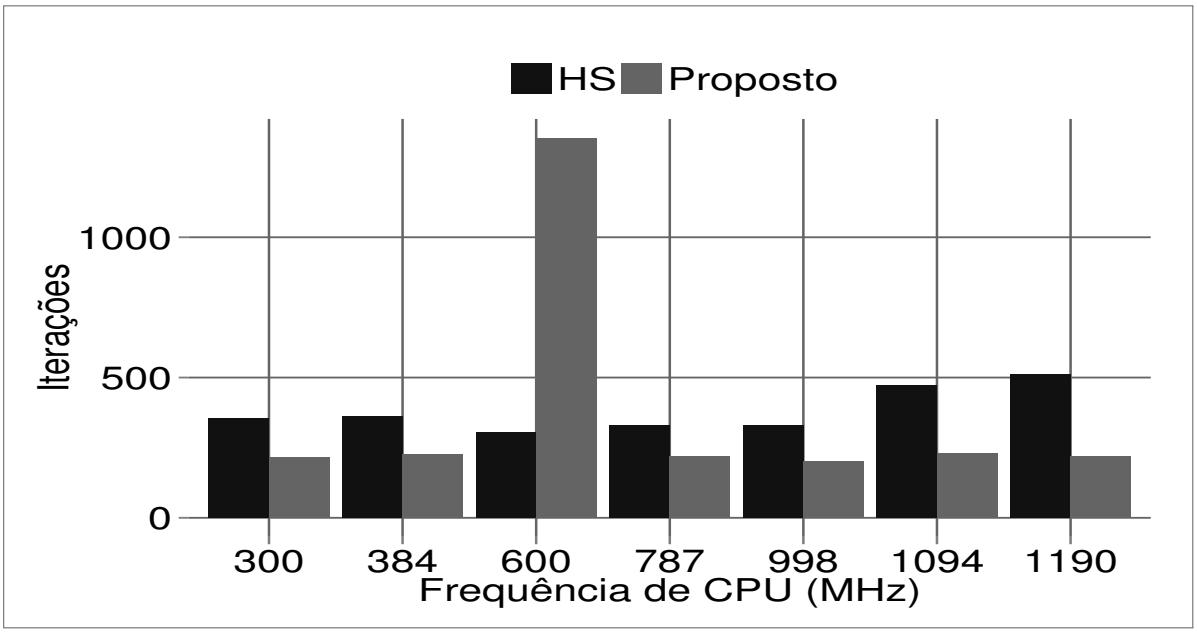
Em nossa abordagem, usamos o preditor de carga de processamento autônomo com detecção de mudança (AEWMA-MSE) para assegurar que $\rho = 1$ para todas as cargas de processamento, isto é, não é mantido histórico de cada aplicação. Em contrapartida, o método proposto aprende durante a execução do algoritmo no dispositivo, reaprendendo a cada mudança na carga de processamento.

Para validar e comparar as abordagens, simulamos um ambiente usando a DB-1 e calculamos o consumo de potência instantânea para diversos instantes em cada abordagem durante o tempo de execução. Em adição, a abordagem ϵ-*greedy* foi aplicada, sendo 20%

de exploração e 80% de explotação para selecionar a melhor ação a ser tomada pelo *Q-learning*. Além disso, um mapeamento com $S_Q = 21$ ($|A| = 7$, $\Delta\mu = 3$) foi utilizado.

A Figura 34 ilustra o histograma das frequências de CPU obtidas para a aplicação CH. O histograma provê informação sobre o impacto do escalonamento de frequência de CPU e do *Q-learning* em nossa proposta e no método de HS. É notável que nossa abordagem usa a frequência de 600 MHz de forma mais intensa que as outras frequências. Além disso, é importante destacar que $f_{LP} = 600\ MHz$ é a frequência de CPU que provê o menor consumo de potência para a aplicação CH, como demonstrado nas seções anteriores.

Figura 34 – Histograma das frequências de CPU usando Shen et al. (2013) (HS) e a função de custo proposta para o aplicativo CH.



Fonte: O autor (2019).

As Figuras 35 e 36 ilustram os histogramas das frequências de CPU obtidas para as aplicações FB e YT, respectivamente. No cenário FB, podemos perceber que na abordagem proposta a frequência de 787 MHz é a f_{LP} para essa aplicação e também a mais utilizada para esta aplicação. Para a aplicação YT, a abordagem proposta usa a frequên-

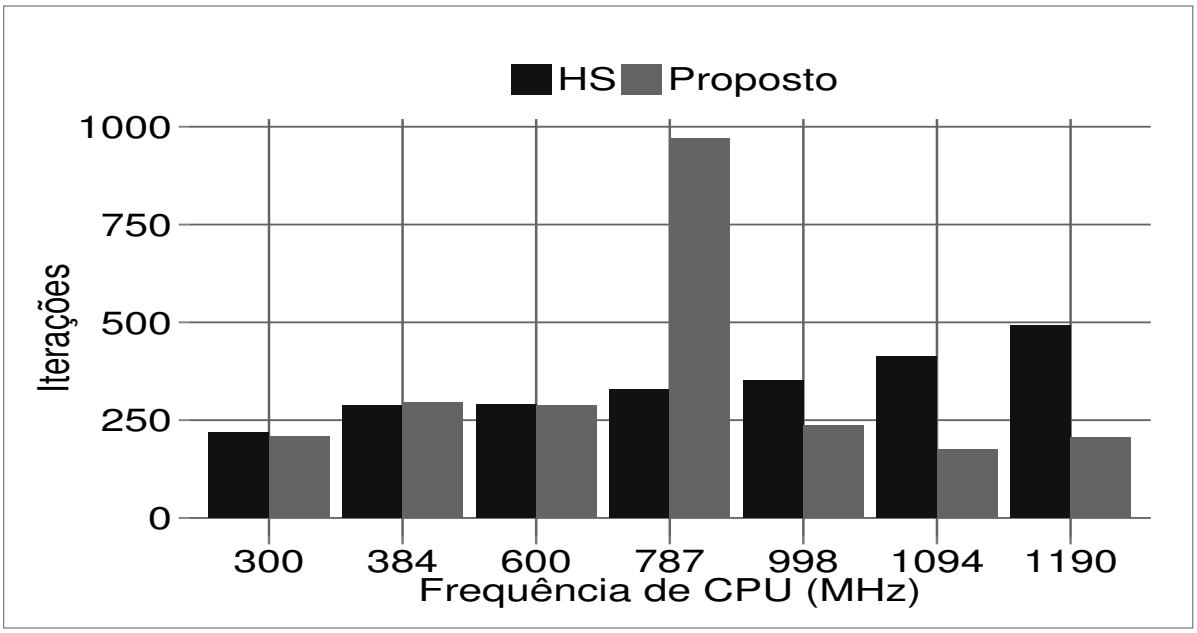
Tabela 11 – Tamanhos da *Q-table* para a abordagem convencional do *Q-learning* e o método proposto neste trabalho.

$\|A\|$	ρ	$\Delta\mu$	Abordagem convencional	Abordagem proposta
2	1	2	4	4
5	1	5	25	25
10	10	10	1000	100
10	50	10	5000	100

Fonte: O autor (2019).

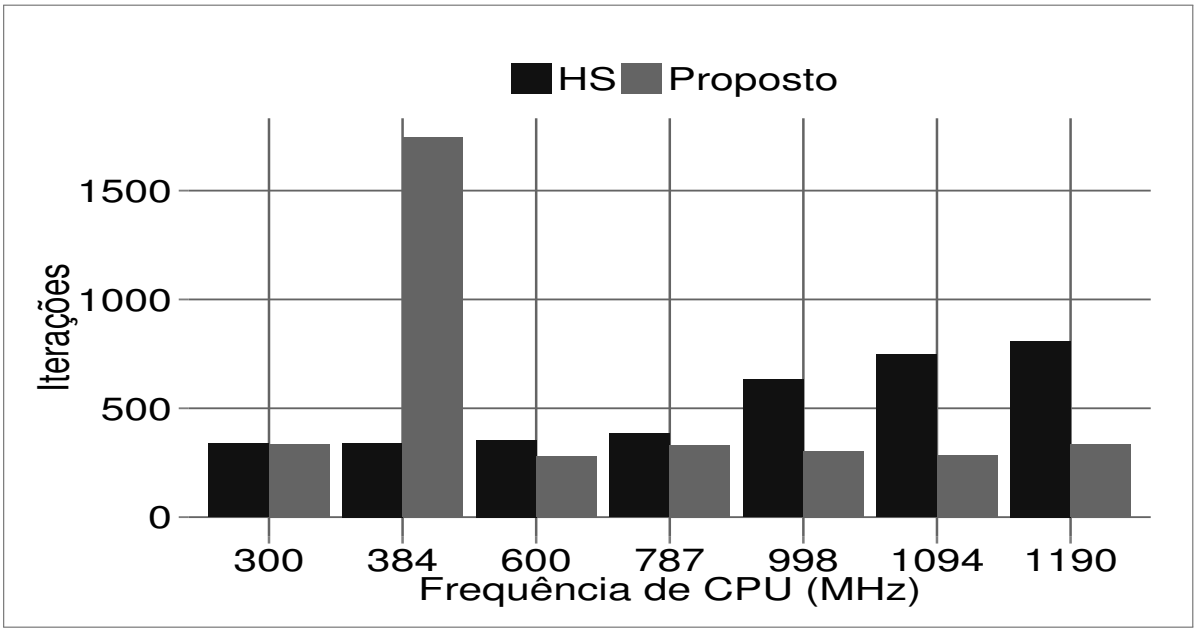
cia de 384 MHz (f_{LP} para a aplicação YT) mais que as demais frequências. Ademais, a abordagem HS usou frequências mais altas, como 998, 1094, e 1190 MHz.

Figura 35 – Histograma das frequências de CPU usando Shen et al. (2013) (HS) e a função de custo proposta para o aplicativo FB.



Fonte: O autor (2019).

Figura 36 – Histograma das frequências de CPU usando Shen et al. (2013) (HS) e a função de custo proposta para o aplicativo YT.



Fonte: O autor (2019).

Como resultado, podemos concluir que o método HS conduz para um uso mais acentuado de frequências mais altas, como 1090 e 1190 MHz. Essas frequências de CPU são

mais prováveis de ocasionar alto consumo de energia, como referenciado anteriormente. Além disso, a função de custo da abordagem proposta usa f_{LP} para todos os cenários analisados na grande maioria do tempo de execução da aplicação.

Por fim, o uso das frequência de CPU diferentes da f_{LP} por nossa abordagem ocorre no intuito de achar melhores frequências de CPU, dado o processo de exploração da abordagem ϵ-*greedy* utilizada juntamente com o *Q-learning*. Foram utilizadas ações de exploração na abordagem ϵ-*greedy* de aproximadamente 20% do número total de escalonamentos de frequência.

5.4 USO DE RECURSOS DA ABORDAGEM PROPOSTA

A abordagem proposta pode ser implementada a nível de aplicação ou *kernel*. Quando implementada a nível de aplicação, possui a vantagem de poder ser utilizada a mesma implementação em diversos dispositivos, bem como a otimização desse aplicativo é mais fácil que em uma abordagem a nível de *kernel*. Por outro lado, a implementação no *kernel* do SO possui o tempo de resposta menor, porém, é necessário maior esforço, dada a complexidade de implementação e adição de módulos no *kernel*.

O trabalho de Holmbacka et al. (2016) demonstrou que realizar DVFS a nível de aplicação tem o tempo de execução médio $2,5$ vezes maior que o tempo de execução da implementação em *kernel*, variando de 6 ms até 13 ms, dependendo da frequência e carga de processamento de CPU. Em complemento, a implementação em *kernel* apresentou um tempo médio de 4 ms para um SoC Exynos 4412 com processador ARM Cortex A9, variando menos que 1 ms. Além disso, realizar muitos chaveamentos de frequência pode levar a um alto consumo de energia devido aos picos de tensão que acontecem ao alterar a frequência de operação do dispositivo.

No trabalho de Shafik et al. (2016), o tempo de DVFS para um processador ARM Cortex A8 é menor que $0,5$ ms e o tempo total de execução do sistema, somado o DVFS, contadores de desempenho e controle de decisões é entre $1,3$ ms até $2,3$ ms, dependendo da abordagem utilizada.

É perceptível uma diferença significativa dos tempos de execução do mesmo procedimento (realizar DVFS) para os trabalhos consultados. Essa diferença entre os trabalhos consultados é justificada pelo tipo de processador analisado ou pela infraestrutura de medição do tempo. Em ambos os casos, é evidente a validade do uso do algoritmo *Q-learning* aplicada ao controle de DVFS em dispositivos móveis, exigindo um custo computacional muito baixo, como vamos ver na próxima seção.

Além disso, percebemos experimentalmente que esse tempo de dezenas ou até centenas de milissegundos não é significativo para afetar negativamente o escalonamento de frequência de CPU, podendo uma mesma frequência ser utilizada em diferentes cenários. Nossa abordagem, mesmo operando em um intervalo de tempo maior que o citado na literatura, consegue atingir níveis significativos de economia de energia.

Em adição, na seção seguinte, analisamos o tempo de execução de duas abordagens da literatura, sendo: Shen et al. (2013) e Tian et al. (2018). Essas abordagens que também fazem uso de algoritmos de aprendizagem por reforço são comparadas à abordagem proposta neste trabalho. Nesta análise, são avaliados o tempo total de execução da abordagem e o tempo de execução dos principais componentes da abordagem.

5.4.1 Análise do tempo de execução

Para facilitar o agrupamento e reconhecimento da fonte de maior consumo de tempo, dividimos o tempo de execução das abordagens em quatro categorias: Dispositivo, Predição de Variáveis de Desempenho (PVD), Escolher Melhor Ação (EMA) e Atualizar Algoritmos (AA).

A categoria Dispositivo contém os métodos dependentes do hardware do dispositivo em análise, sendo os tempos para obter a utilização média de CPU, a frequência de CPU no momento da coleta e realizar DVFS.

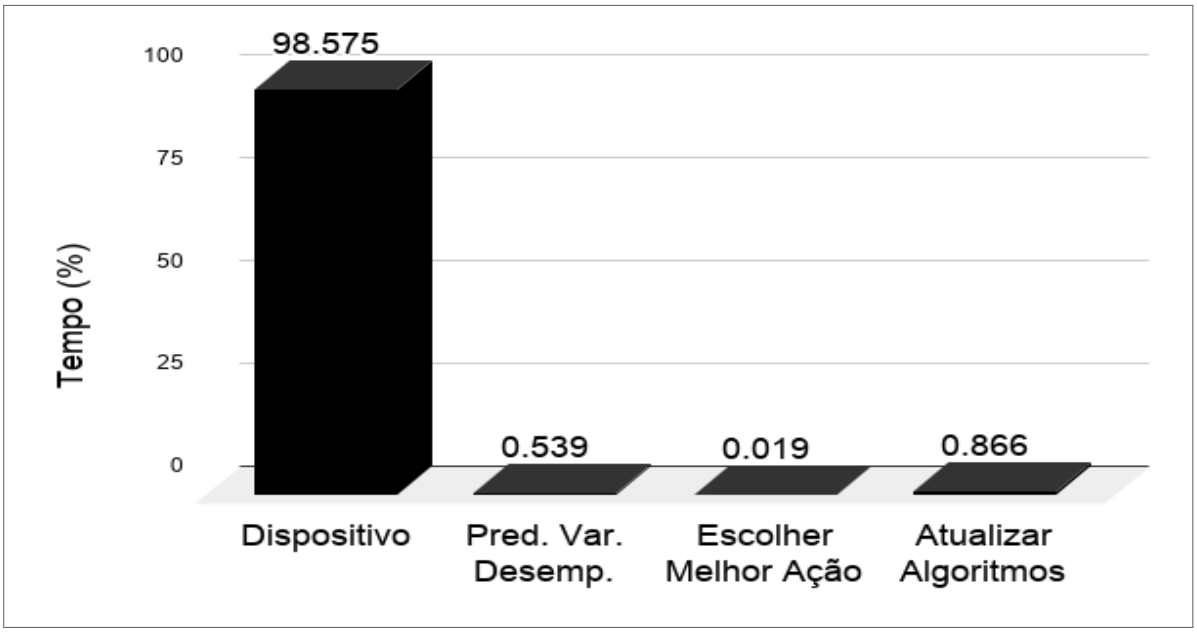
A categoria PVD contém os métodos de predição e detecção de mudança da carga de processamento da CPU. Ademais, a categoria EMA é responsável pela categorização do contexto atual do dispositivo e seleção da frequência de CPU que minimiza o consumo de energia (execução do *Q-learning*). Por fim, a categoria AA contém os métodos para atualização dos valores nos métodos AEWMA-MSE e *Q-learning*.

A Figura 37 ilustra o tempo gasto em cada categoria para uma iteração do algoritmo proposto, os valores estão expressos em porcentagem. Os métodos que pertencem à categoria Dispositivo consomem 98,5% do total do tempo gasto em uma iteração. Sendo os métodos inerentes ao algoritmo proposto quase desprezíveis quando comparados ao tempo total para uma iteração.

A Figura 38 ilustra o tempo de execução da abordagem proposta excluindo os atrasos proporcionados pelo dispositivo, descritos na categoria Dispositivo. Analisando o gráfico, é perceptível o baixo consumo de tempo na etapa relacionada à EMA, etapa que abriga os métodos de RL para consulta do conhecimento adquirido, sendo a etapa de AA contendo os trechos para atualizar o algoritmo de RL e AEWMA-MSE, onde a função de custo é utilizada.

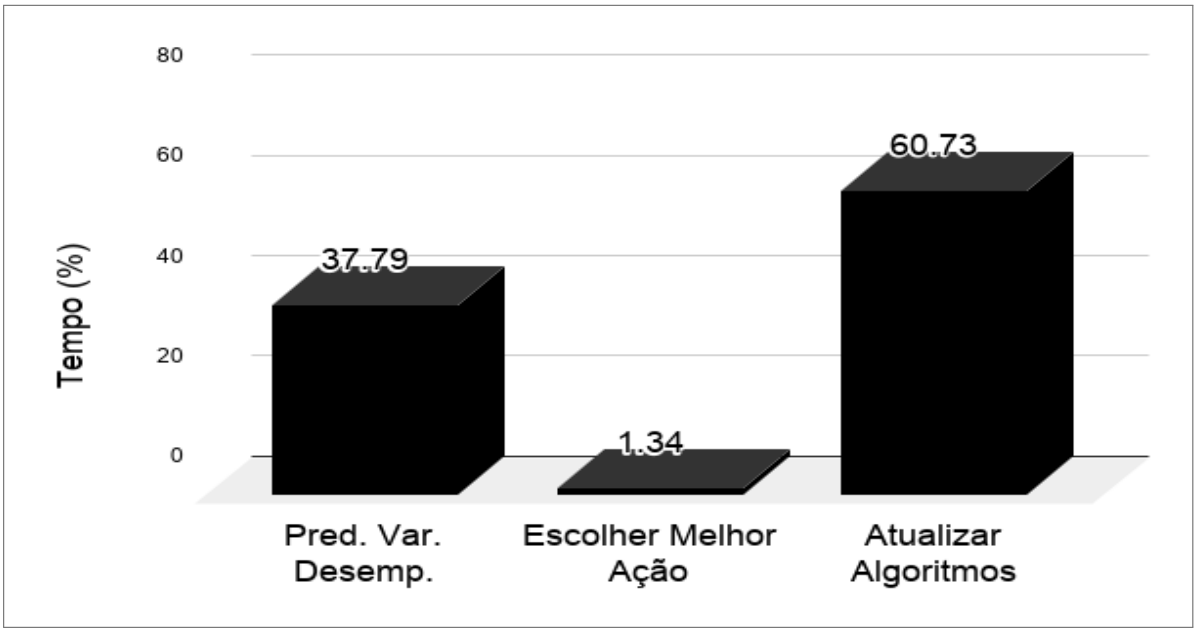
A Figura 39 ilustra o tempo consumido comparando somente os métodos relacionados à categoria Dispositivo. A maior parte do tempo consumido para as fases da categoria Dispositivo é do método de obter a utilização média da CPU, isso acontece pois é necessário esperar um intervalo de tempo para poder calcular o uso da CPU, mesmo não havendo processamento por parte do algoritmo proposto, uma *thread* fica observando por 500 ms o uso do processador. Além disso, esse valor pode ser adaptado para mais ou menos, sendo definido de acordo com restrições do dispositivo ou do algoritmo que realiza o DVFS. Os métodos de obter a frequência de CPU e realizar o DVFS são menores que 6% do tempo total dos métodos da categoria Dispositivo.

Figura 37 – Tempo de execução total da abordagem proposta dividido em categorias.



Fonte: O autor (2019).

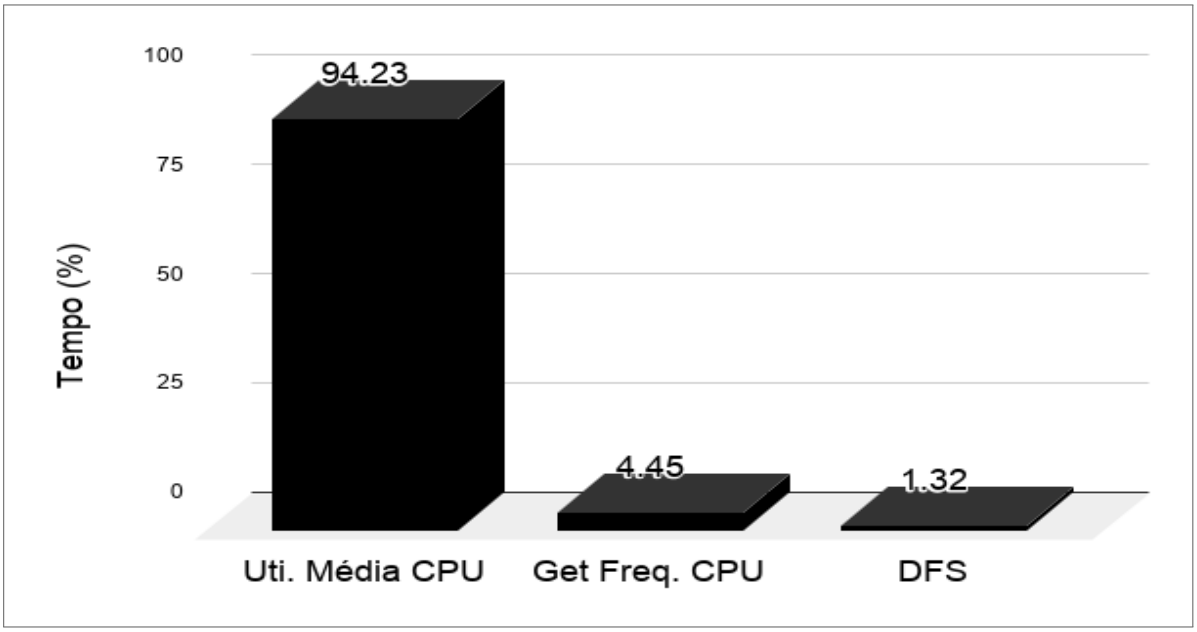
Figura 38 – Tempo de execução da abordagem proposta não considerando atrasos do dispositivo.



Fonte: O autor (2019).

A Figura 40 ilustra o comparativo entre a abordagem proposta e as de HS (SHEN et al., 2013) e ZT (TIAN et al., 2018) construídas a partir das informações disponibilizadas nos artigos, de forma pública, visto que o código fonte não é disponibilizado. Os trabalhos de HS (SHEN et al., 2013) e ZT (TIAN et al., 2018) foram implementados a nível de *kernel*. Nós expandimos essas implementações para serem também utilizadas a nível de serviços

Figura 39 – Tempo de execução da abordagem proposta considerando somente atrasos do dispositivo.



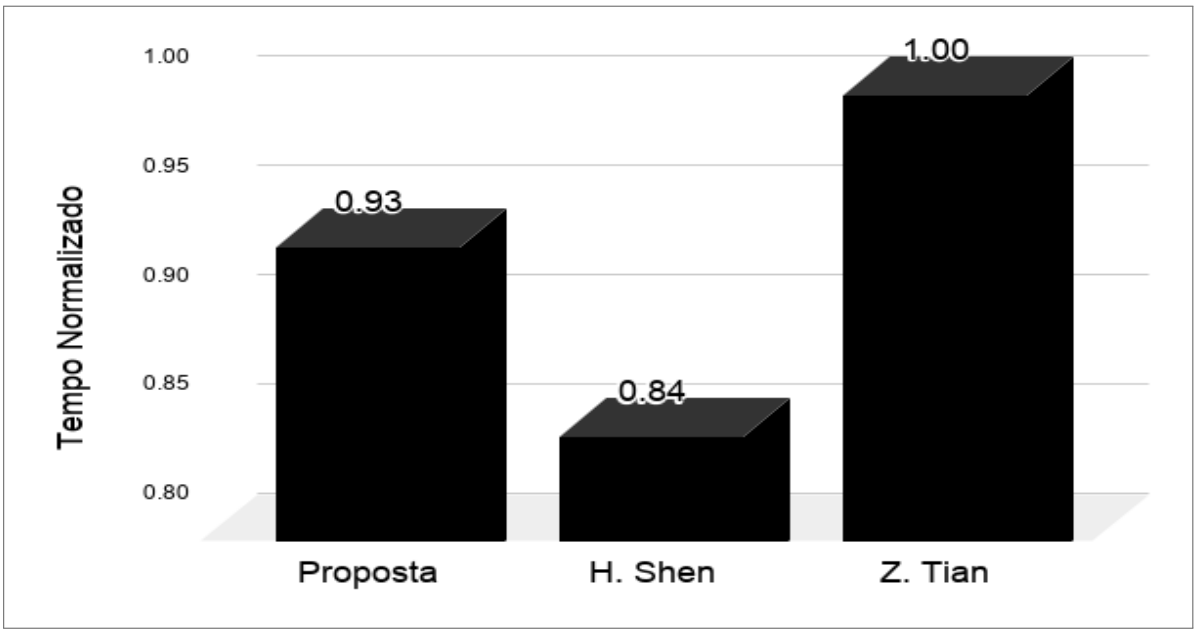
Fonte: O autor (2019).

no Android, utilizando as funções de custo propostas por estes trabalhos, visando uma comparação mais justa com nossa abordagem.

Os trabalhos foram normalizados pelo maior valor de tempo de execução, ou seja, o trabalho de ZT, necessitando de $44,11$ ms para completar uma iteração. Para chegar a esse valor, a operação foi repetida 100 vezes para cada abordagem e a média calculada. Nossa abordagem ficou em segundo lugar, consumindo $0,93$ x o tempo usado por ZT. Por fim, a abordagem HS consumiu $0,84$ x quando comparado a ZT. O relativo alto consumo da nossa abordagem, quando comparada à de HS, é pelo fato da complexidade exigida para calcular os valores futuros e fazer uso de métodos que proporcionam uma maior acurácia na predição.

A Tabela 12 detalha os tempos de execução dos principais métodos da implementação proposta, informando a categoria a que o método pertence, o método, o tempo de execução e a descrição do método.

Figura 40 – Comparativo do tempo de execução da abordagem proposta com os trabalhos relacionados HS (SHEN et al., 2013) e ZT (TIAN et al., 2018).



Fonte: O autor (2019).

Tabela 12 – Tempos de execução dos principais métodos das implementações.

Categoria	Método	Tempo (ms)	Descrição
Dispositivo	getMeanCpuUtilization	504,1626	Calcular a utilização média de todos os núcleos do processador. Foi utilizado um intervalo de 500 ms para calcular o tempo de utilização da CPU; cada núcleo foi calculado individualmente.
Pred. Var. Desemp	AewmaPredict	0,0227	Realizar a predição da carga de processamento para o instante $i + 1$ utilizando o algoritmo AEWMA-MSE e o histórico observado.
Pred. Var. Desemp	AewmaWlChangedMSE	2,9035	Verificar se houve mudança significativa na utilização de CPU predita para o instante $i + 1$, utilizando o algoritmo AEWMA-MSE e o histórico observado.
Dispositivo	GetCurrentCpuFrequency	23,8057	Obter a frequência de CPU que está ativa no momento da consulta.
Escolher Melhor Ação	MapToAnState	0,0758	Mapear a utilização de CPU predita e a frequência de CPU atual para um estado pré-definido do algoritmo de RL.
Escolher Melhor Ação	GetMinQtableAction	0,0279	Obter a ação que fornece a melhor recompensa, ou seja, o menor consumo de energia. A ação é uma frequência de CPU que minimiza o consumo de energia para o cenário atual do dispositivo.
Dispositivo	TakeActionDoDFS	7,0807	Realizar o chaveamento da frequência de CPU a partir da frequência escolhida na etapa "Escolher Melhor Ação".
Atualizar Algoritmos	UpdateRLAlgorithm	4,6763	Realizar a atualização dos valores da *Q-table* no algoritmo de RL usando a função de custo.
Atualizar Algoritmos	UpdateAewmaAlgorithm	0,0258	Realizar a atualização dos valores do algoritmo de predição AEWMA-MSE.

Fonte: O autor (2019).

5.5 ECONOMIA DE ENERGIA

Nesta seção, detalhamos o procedimento de apuração da economia de energia ao utilizar o RTM (escalonador de frequência de CPU) proposto neste trabalho. Primeiramente, é feita a análise utilizando a abordagem proposta com predição de potência, utilizando k-NN ($k = 100$) descrito nas seções anteriores. Na subseção posterior, é utilizada a abordagem que faz uso do sensor de potência, obtendo os valores de potência diretamente deste sensor, sem a necessidade de predição.

Para a abordagem com predição, foi utilizado o smartphone Motorola XT1033 (Tabela 5). Na abordagem com sensor de potência, foi utilizado um *smartphone* mais moderno, fabricante Xiaomi e modelo Redmi Note 4, lançado em 2017 (Tabela 13). Todos os experimentos foram realizados com o número máximo de núcleos do processador ativos.

5.5.1 Abordagem com predição de potência

Nesta subseção, analisamos o consumo de energia obtido a partir da execução dos *governors* padrões do Android, o método HS e a abordagem proposta, focando na economia de energia. Para os métodos HS e abordagem proposta, assumimos um erro de predição máximo δ de 9% para avaliar o consumo de energia.

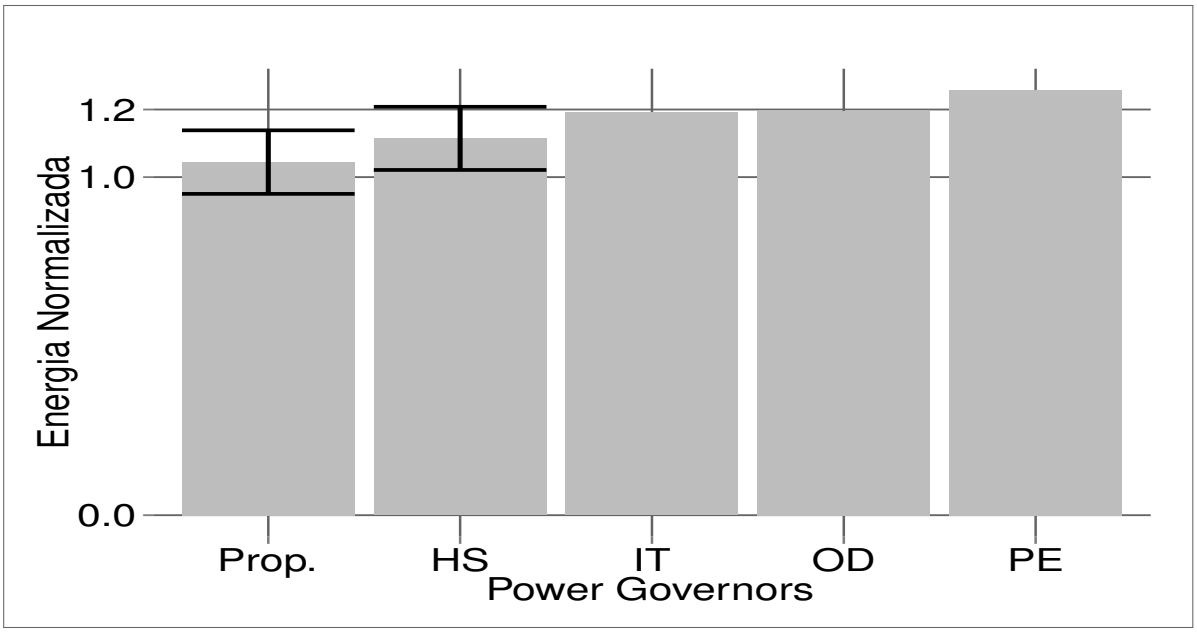
A Figura 41 exibe o consumo de energia normalizado para o aplicativo CH. É perceptível que a abordagem proposta tem o menor consumo quando comparada aos demais algoritmos, mesmo considerando o erro de predição máximo ($\delta = 9\%$), que é representado pelas barras no gráfico. Em média, a abordagem proposta é melhor que o método HS e todos os demais *governors* para o aplicativo CH. Analisando os *governors* Android, o IT e OD possuem um comportamento similar, sendo o PE o que mais consome energia.

A Figura 42 ilustra os resultados obtidos para o aplicativo FB, detalhando o consumo de energia. Podemos perceber que, em média, a abordagem proposta supera (menor consumo) todos os *governors* Android analisados e a abordagem HS. Os *governors* IT e PE apresentam um valor de consumo de energia bem próximo, enquanto o OD possui o menor consumo de energia dos *governors* Android.

Em sequência, a Figura 43 exibe o consumo de energia normalizado para o aplicativo YT. Enquanto a abordagem proposta supera todos os demais algoritmos, os *governor* IT e PE apresentam o maior consumo de energia para essa aplicação. Ademais, a abordagem proposta e o método HS foram avaliados em um ambiente experimental, enquanto os demais algoritmos foram avaliados utilizando medição por *hardware*.

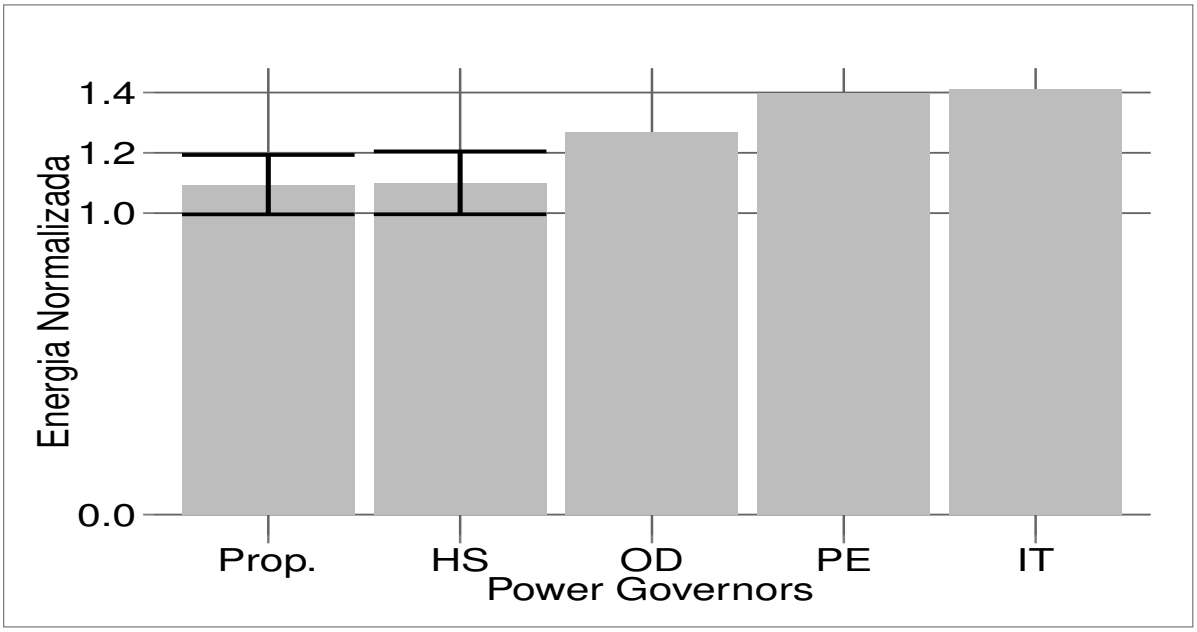
Para os cenários analisados, o RTM proposto neste trabalho consome menos energia, quando comparado às demais abordagens, inclusive para o HS. De forma geral, nossa abordagem pode prover uma economia média de energia de 21% quando comparada à abordagem IT em todos os aplicativos. Em adição, para o aplicativo FB, a abordagem proposta pode alcançar uma economia de até 29%, quando comparada ao *governor* IT.

Figura 41 – Consumo de energia normalizado usando o método HS, *governors* padrões do Android OD, PE e IT, como também a abordagem proposta (Prop.) para o aplicativo CH.



Fonte: O autor (2019).

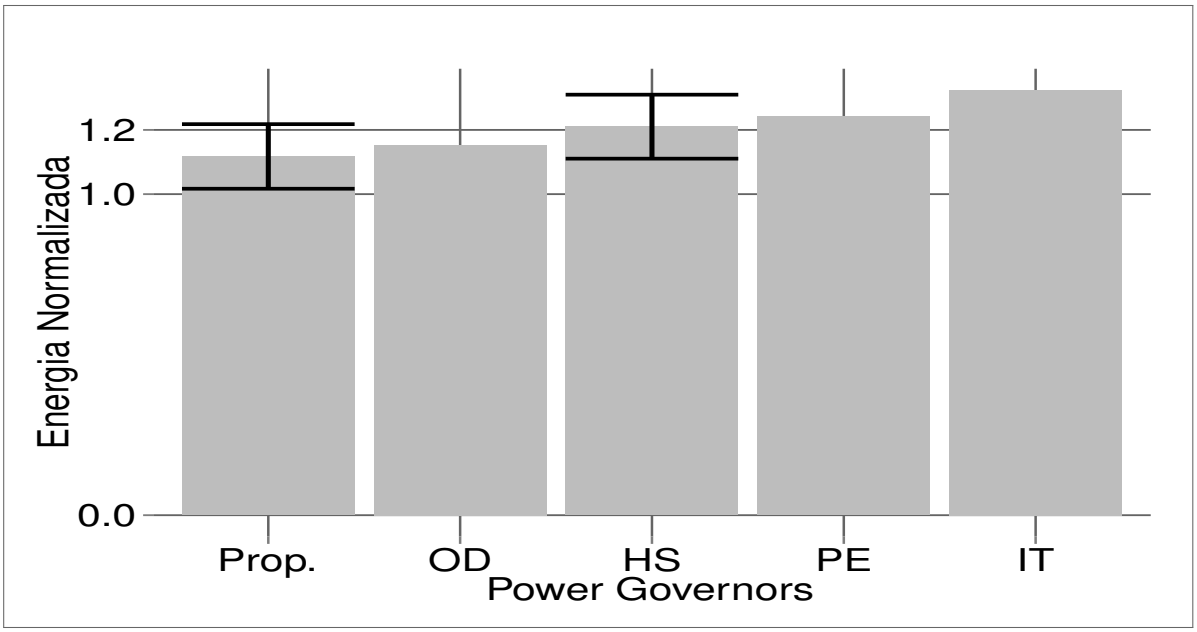
Figura 42 – Consumo de energia normalizado usando o método HS, *governors* padrões do Android OD, PE e IT, como também a abordagem proposta (Prop.) para o aplicativo FB.



Fonte: O autor (2019).

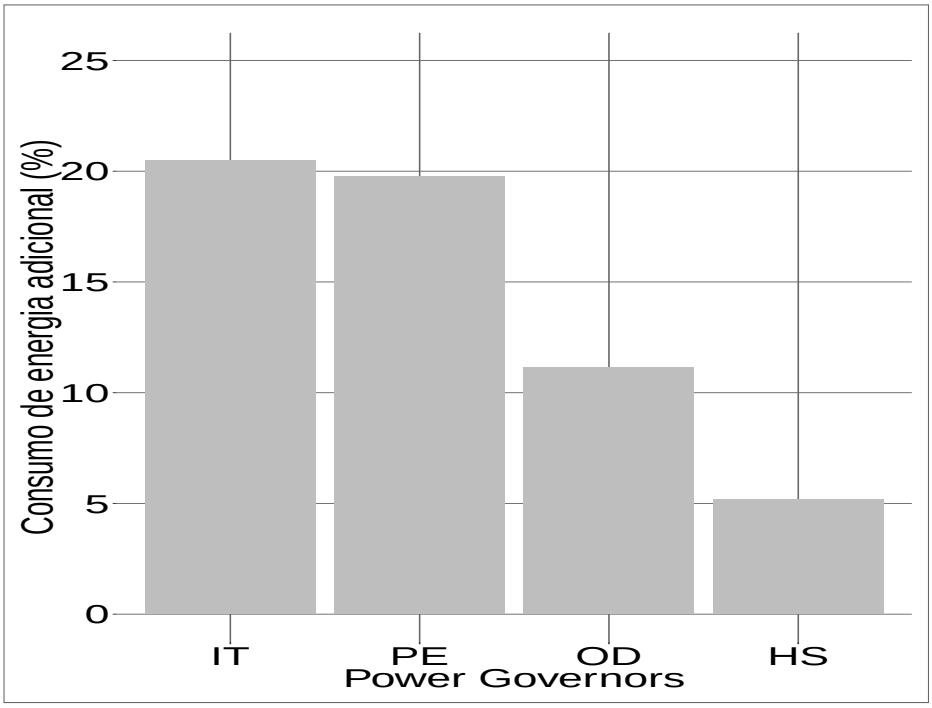
A Figura 44 ilustra o consumo de energia percentual médio adicional de cada *governor* (OD, PE e IT) juntamente com o método HS em relação à abordagem proposta neste trabalho. Por exemplo, o consumo de energia para o aplicativo FB é 299.70 J e 232.57 J

Figura 43 – Consumo de energia normalizado usando o método HS, *governors* padrões do Android OD, PE e IT, como também a abordagem proposta (Prop.) para o aplicativo YT.



Fonte: O autor (2019).

Figura 44 – Consumo de energia adicional de cada Android *governor* (OD, PE e IT), como também o método HS em relação à abordagem proposta para todos os aplicativos.



Fonte: O autor (2019).

para os algoritmos IT e a abordagem proposta, respectivamente. Então, o IT consome quase 29% mais energia que nossa abordagem. Considerando os aplicativos CH e YT, o *governor* IT consome mais energia que a nossa abordagem, 14% e 18% para CH e YT, respectivamente. De acordo com os dados, podemos perceber que o *governor* IT é, em média, 21% menos eficiente que a nossa abordagem em termos energéticos.

5.5.2 Abordagem com leitura do sensor de potência

Nesta seção, é detalhado o ganho energético e o desempenho alcançado utilizando as diversas abordagens da literatura para o escalonamento de frequência da CPU. Para os experimentos seguintes, utilizamos um *smartphone* Xiaomi Redmi Note 4, as principais características estão expostas na Tabela 13. Além

Tabela 13 – Características do *smartphone* Xiaomi Redmi Note 4.

	Redmi Note 4
SoC	Snapdragon 625
Processador	ARM Cortex A53
Frequências (MHz)	652 - 2020
Número de Núcleos	8
Versão do Android	7.0
Capacidade da bateria (mAh)	4100

Fonte: O autor (2019).

Além disso, as diferentes abordagens foram validadas em diversos *benchmarks* públicos e representativos. Utilizamos o *Vellamo Mobile Benchmark* (CENTER, 2017), desenvolvido pela Qualcomm e utilizado por mais de 500 mil usuários, de acordo com a *Google Play*. Por conseguinte, o *AnTuTu Benchmark*, desenvolvido pela chinesa *AnTuTu*, empresa especializada em software de medição de desempenho. De acordo com a *Google Play*, foi utilizado por mais de 10 milhões de usuários. Estes *benchmarks*, ao final de cada execução, mostram um número que mede a qualidade de cada execução, fornecendo um *score*. Porém, a análise dos valores de *score* não fazem parte do escopo deste trabalho, nosso esforço é na análise da redução do consumo de energia.

As principais características dos *benchmarks* estão descritas nas tabelas 14 e 15. O *Vellamo* é dividido em três grandes categorias: *Browser, Metal e Multicore*, detalhados nas tabelas abaixo. Mais detalhes quanto às implementações dos *benchmarks* podem ser obtidos no aplicativo oficial, disponibilizado na *Google Play*, ou no site oficial. O *Vellamo Browser* possui 15 microbenchmarks, o *Vellamo Metal* possui 6 e o *Vellamo Multicore* possui 8 microbenchmarks.

Além disso, o *AnTuTu Benchmark* provê uma descrição mais simplória, não especificando quais *microbenchmarks* são utilizados em cada categoria. As categorias são:

operações envolvendo CPU e Unidade de Processamento Gráfico - *Graphics Processing Unit* (GPU), Experiência do Usuário - *User Experience* (UX) e memória. A categoria UX, além de avaliar métricas comuns na literatura, como operações por tempo ou similares, também inclui métricas que refletem a experiência do usuário com o dispositivo, provendo dados de conforto no uso do dispositivo pelo usuário.

Cada *benchmark* foi avaliado utilizando os seguintes algoritmos de escalonamento de frequência de CPU: AP g=0,1, AP g=0,5, AP g=0,9, HS, ZT, OD, IT e PE. O atributo g, representado nas variações da abordagem proposta, simboliza o valor de γ, o fator de desconto no algoritmo *Q-learning*, descrito nas seções anteriores.

A métrica consumo de energia relativo, foi medida utilizando o seguinte roteiro:

1. Desativar todos os periféricos e ativar o modo avião. Somente para o *Vellamo Browser* o Wifi permanece ativo.
2. Proibir todos os aplicativos de exibir notificações, exceto o escalonador e o *benchmark*.
3. Carregar o dispositivo até a bateria marcar 100% de carga.
4. Ainda conectado no carregador, limpar dados de *cache* do aplicativo.
5. Esperar a temperatura da bateria ficar próxima da temperatura ambiente (ambiente em 22° C).
6. Ativar mecanismos de automação da execução do teste (a primeira execução é supervisionada, porém com a automação ativada).
7. Repetir a execução do teste até a bateria atingir um valor percentual abaixo de 20%.

É importante destacar que a cada iteração são coletados diversos atributos da execução, listados abaixo:

- *Printscreen* das telas de resultados após o final da execução da iteração pelo *benchmark*.
- Número de segundos desde 1 de janeiro de 1970, usando UTC/GMT (*Unix timestamp*).
- Frequência de *CPU* atual e futura, juntamente com a ação do *Q-learning*.
- Valores de corrente e tensão da bateria, capturados a partir do sensor do dispositivo.
- Porcentagem da bateria e o tempo de início e fim da iteração.
- Quantidade de mudanças na carga de processamento, seus respectivos valores e valores do MSE associado.

- Valores a *Q-table* para cada iteração do *Q-learning*

Por conseguinte, definimos executar os experimentos para medir o consumo de energia de [100-20]% de uso da bateria, ou seja, 80% de uso, pois ao atingir um valor abaixo de 20%, o SO ativa mecanismos de economia de energia, podendo atrapalhar a execução correta do aplicativo de escalonamento de frequência e automação dos experimentos.

Ao finalizar cada sequência de experimentos, utilizamos a métrica abaixo para calcular o consumo de energia relativo:

$$E_{rel} = \frac{80}{Total\ de\ iterações}, \tag{5.3}$$

onde E_{rel} representa a relação de 1% de bateria para cada n iterações executadas para determinado *benchmark*. O objetivo dos algoritmos é minimizar esta métrica, economizando energia. Após o cálculo da métrica, normalizamos os valores em uma escala $[0-100]$ em relação à abordagem AP com $g = 0,9$, exibidos nos próximos gráficos.

Dentre as configurações da abordagem proposta analisadas, a que alcançou o melhor custo benefício entre economia de energia e tempo de execução foi a abordagem AP com $g = 0,9$, apresentando menor consumo de energia e menor tempo de execução, sendo a escolhida para comparação com os demais algoritmos.

Tabela 14 – Síntese dos componentes do *benchmark Vellamo Browser*.

Microbenchmarks	Descrição
1. Deep crossfader	Operação de composição em imagens, é a base dos efeitos gráficos vistos em um navegador.
2. Kruptein	Codifica, decodifica e valida imagens usando biblioteca JavaScript (JS) *cryptographic*.
3. Image Re-focus	Mede a velocidade para a manipulação dos *pixels* quando refocando a imagem.
4. Pixel Blender	Injeta imagens coloridas e mede a largura de banda do *hardware*. É medido o número de pixels/seg.
5. Aquarium Canvas	Mede o número de objetos que podem ser desenhados e animados usando HTML5 e mantendo 30 frames/sec.
6. CSS 3D Fish	Mede a velocidade para renderizar animações 3D com CSS.
7. WebGL Jellyfish	Mede a velocidade/experiência do usuário no uso de gráficos 3D com WebGL.
8. DOM Node Surfer	Mede o desempenho das páginas Web utilizando funções JS.
9. Surf Wax Binder	Mede a quantidade de vetores acessados por segundo com dados randômicos e de pixels.
10. See the Sun	Rotacionada uma imagem em 2,5D usando HTML5 e mede a quantidade de frames/seg.
11. Ocean Scroller	Quantifica a experiência de rolar uma página real, medindo a quantidade de frames/seg.
12. Page Load Perf.	Mede o tempo em que uma página demora para carregar todos os recursos.
13. Text Reflo	Mede o número de vezes por segundo que o texto é colocado na páginas Web.
14. Sun Spider 1.0.2	*Benchmark* popular JS desenvolvido pelo *webkit.org*.
15. Octane v1	*Benchmark* popular JS desenvolvido pelo Google.

Fonte: O autor (2019).

Tabela 15 – Síntese dos componentes dos *benchmarks Vellamo Metal*, *Vellamo Multicore* e *AnTuTu General*

.

Microbenchmarks	Descrição
Vellamo Metal	
1. Dhrystone 2.1	Testa o desempenho da CPU para realizar operações inteiras.
2. Linpack	Mede o desempenho da CPU ao resolver uma sequência de equações lineares.
3. Branch-K	Avalia o tempo em que um programa pula de um espaço de memória para outro.
4. Stream 5.9	Testa a velocidade da banda da memória em MB/seg.
5. RamJam	Mede a velocidade para executar ações de leitura e escrita na memória RAM.
6. Storage I/O	Mede a velocidade que o dispositivo leva para ler e gravar na memória não-volátil.
Vellamo Multicore	
1. MT Linpack (native)	Testa o desempenho da CPU ao realizar operações com ponto flutuante, implementado em C.
2. MT Linpack (Java)	Testa o desempenho da CPU ao realizar operações com ponto flutuante, implementado em Java.
3. MT Stream 5.10	Uma variação do STREAM adaptado para multicores, testa o desempenho da memória RAM.
4. Membench	Consiste de outros quatro: Atomics, Mutex, Barrier and Corememlat.
5. Sysbench	Consiste de outros dois: Finepar and Launch, avaliando a adaptação da CPU a carga de processamento.
6. Threadbench	Consiste de outros dois: Pthread and Syscall, envolvendo *threads* e chamadas ao SO.
7. Parsec (Custom)	Avalia o desempenho das arquiteturas paralelas.
8. Inter Process Communication	Avalia quão rápido um processo cliente e servidor consegue se comunicar.
AnTuTu	
1. CPU	Realiza operações matemáticas, uso comum de CPU e operações com multicores.
2. GPU	Realiza operações envolvendo manipulação 3D: *Marooned, Coastline e Refinery*
3. Experiência de usuário (UX)	Avalia segurança de dados, processamento de dados, processamento de imagens e experiência do usuário.
4. Memória	Operações diversas envolvendo memória RAM e ROM.

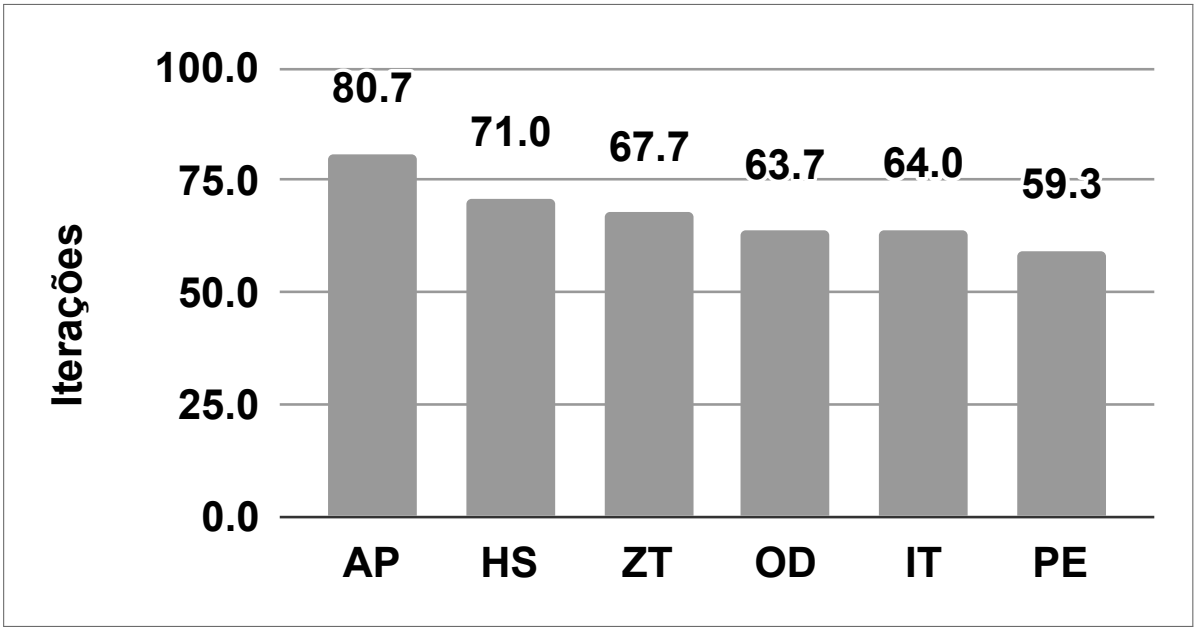
Fonte: O autor (2019).

Os resultados ilustrados a seguir representam os valores médios da repetição dos experimentos. Executamos três repetições para cada configuração; uma configuração é composta de um algoritmo de escalonamento e um *benchmark*. Por exemplo: a configuração V-BW-AP ilustra a execução do *benchmark Vellamo Browser* utilizando a abordagem proposta. Dentro de cada configuração, o mesmo *benchmark* é repetido até que seja completada uma carga de bateria, sendo representado pela quantidade de iterações de cada configuração. Uma carga de bateria é o consumo de 100% até 20% da bateria pelo *smartphone*. Os valores numéricos de todos os gráficos estão sintetizados ao final da seção.

A Figura 45 ilustra a quantidade média de repetições alcançadas na execução do *benchmark Vellamo Browser* utilizando uma carga de bateria. É notável que a AP conseguiu o maior número de repetições para a mesma quantidade de energia utilizada, calculada a partir de uma carga de bateria. A AP alcançou, em média, 80,7 iterações: a maior quantidade de execuções demonstra um consumo de energia menor quando comparada às demais abordagens. Como esperado, o *governor* PE obteve o maior consumo de energia, atingindo somente 59,3 iterações do *benchmark Vellamo Browser*.

As abordagens HS e ZT obtiveram o resultado mais próximo da AP, enquanto os *governors* padrões do Android como OD, IT e PE obtiveram resultados piores, consumindo mais energia.

Figura 45 – Quantidade média de iterações do *benchmark Vellamo Browser* para uma carga de bateria utilizando as abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS, ZT, OD, IT e PE.



Fonte: O autor (2019).

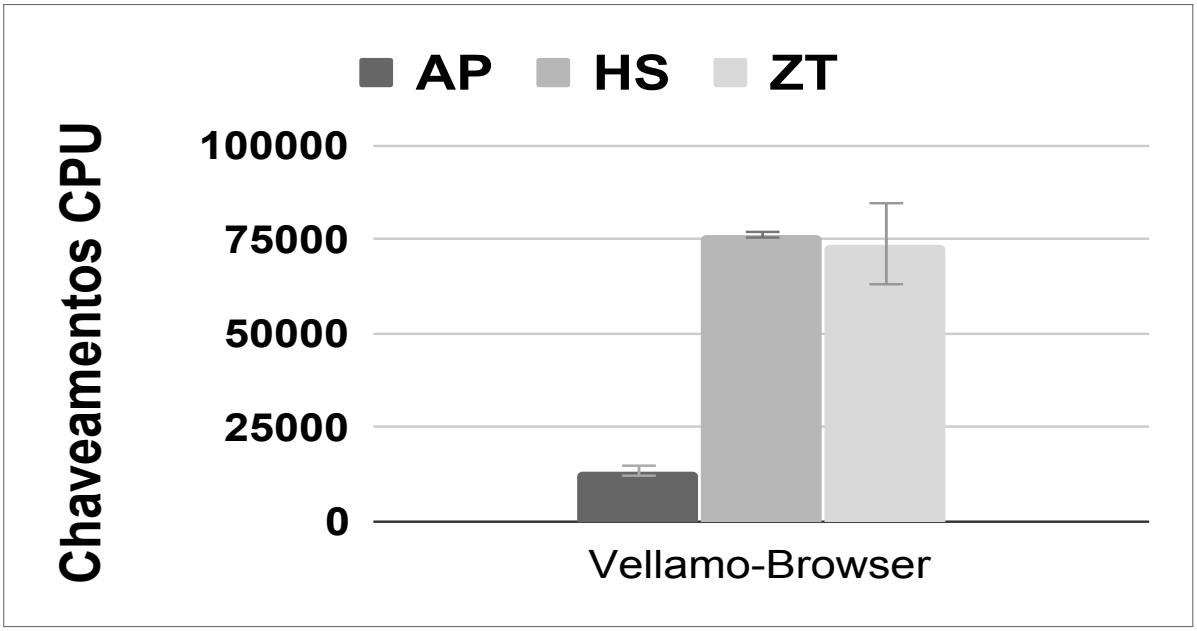
A Figura 46 ilustra a quantidade de chaveamentos de CPU realizada por cada abordagem, ilustrando a AP e as abordagens da literatura HS e ZT. É perceptível a redução considerável do número de chaveamentos alcançada pela AP, auxiliando na redução do

consumo de energia. É sabido que muitos chaveamentos de frequência de CPU desnecessários podem implicar em um alto consumo de energia, dado o custo necessário para realizar o chaveamento.

A Figura 46 ilustra o valor médio e o desvio padrão obtido a partir de três repetições do experimento. Note que, mesmo no pior caso, considerando o valor de chaveamentos máximo da AP, esta ainda é pelo menos 5x menor do que as abordagens da literatura analisadas, demonstrando a eficácia da abordagem proposta.

As abordagens nativas do Android como OD, IT e PE não permitiram a contagem dos chaveamentos por termos utilizado a versão nativa do algoritmo de escalonamento, implementada no sistema operacional e executada a nível de *kernel*.

Figura 46 – Quantidade de chaveamentos de frequência de CPU média utilizada para executar o *benchmark Vellamo Browser* para uma carga de bateria utilizando as abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS e ZT.

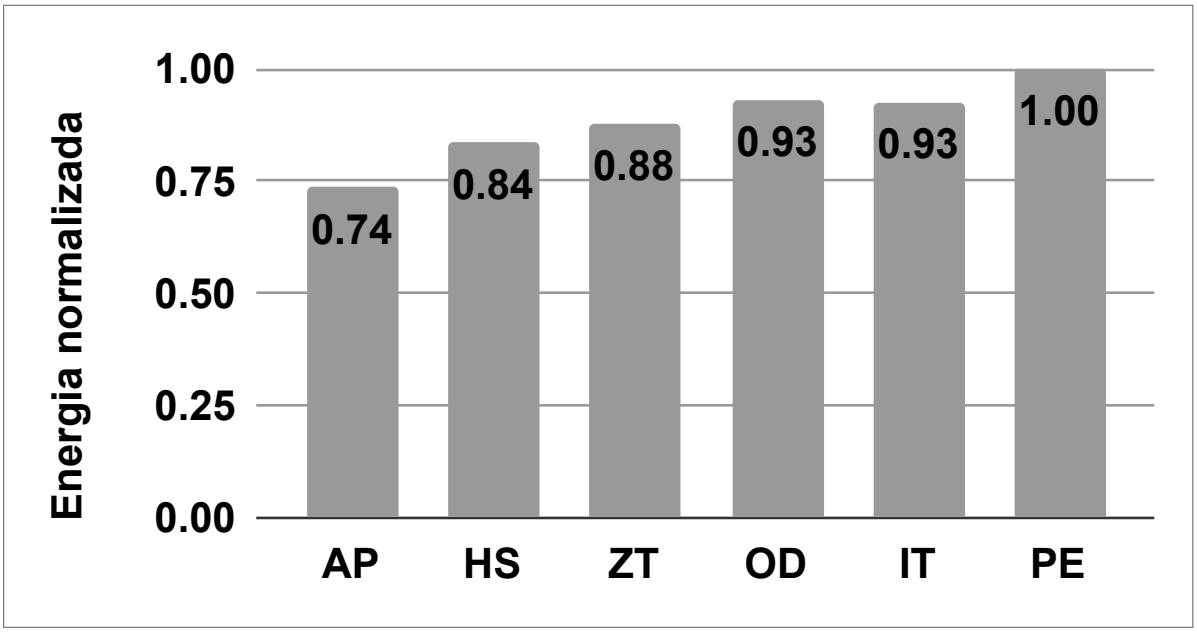


Fonte: O autor (2019).

Na sequência, a Figura 47 ilustra a quantidade de energia consumida por cada abordagem analisada. Os valores médios de energia normalizados foram obtidos a partir da carga da bateria do *smartphone*, demonstrando que a AP obteve os melhores resultados, com o menor consumo de energia dentre todas as abordagens analisadas.

As abordagens da literatura HS e ZT obtiveram o segundo e terceiro menor consumo de energia, respectivamente. As abordagens padrões do Android obtiveram consumos energéticos maiores que as demais abordagens, sendo o *governor* PE o que obteve o maior consumo de energia, sendo o menos recomendado para este *benchmark*, quando o objetivo é economizar energia.

Figura 47 – Consumo de energia médio normalizado para o *benchmark Vellamo Browser* utilizando as abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS, ZT, OD, IT e PE.



Fonte: O autor (2019).

Por conseguinte, a Figura 48 exibe os valores percentuais de economia de energia comparando a abordagem proposta com as demais abordagens. Para chegar aos valores ilustrados na Figura 48, repetimos a execução das abordagens por três vezes, ou seja, cada abordagem foi executada para o *benchmark Vellamo Browser* por três vezes, sendo que, em cada repetição, tivemos uma quantidade de repetições do *benchmark* em questão. A Tabela 16 sintetiza os valores de repetição da abordagem proposta, visando reduzir a chance de erro nas afirmações sobre a economia de energia, indicando que há um ganho real da abordagem proposta.

Tabela 16 – Dados e estatísticas das repetições das abordagens utilizadas para o *benchmark Vellamo Browser*.

Repetição / Qtd. iterações	AP	HS	ZT	OD	IT	PE
1	82	71	70	63	65	59
2	80	71	66	64	64	59
3	80	71	67	64	63	60
Estatísticas						
Média:	80,7	71,0	67,7	63,7	64,0	59,3
Mediana:	80	71	67	64	64	59
Desvio padrão:	1,15	0,00	2,08	0,58	1,00	0,58

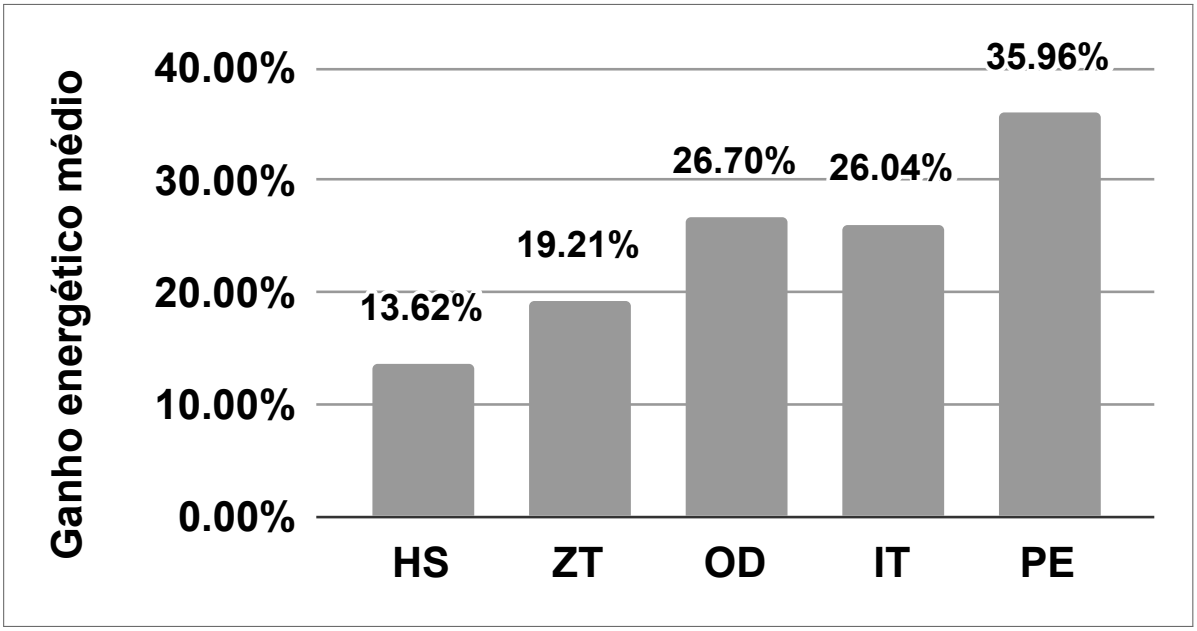
Fonte: O autor (2019).

É perceptível que as variações quanto ao número de repetições do *benchmark* para uma carga de bateria são mínimas, mantendo um desvio padrão entre 0 e 2,08.

Para entender o cálculo do ganho energético, observe o exemplo a seguir: a AP executou em média 80,7 vezes o mesmo *benchmark Vellamo Browser* e a abordagem PE executou somente 59,3. Ao aplicar a equação 5.3, chegamos ao valor aproximado de 1 $(80 \div 80,7)$ para a AP *Vellamo Browser*, ao dividir a porcentagem utilizada da bateria do *smartphones* (80%) por a quantidade de iterações do *benchmark*. Além disso, chegamos ao valor de 1,34 $(80 \div 59,3)$ para o algoritmo PE utilizando o *benchmark Vellamo Browser*. Ao normalizarmos o consumo de energia da AP com o maior consumo (abordagem PE), temos: $(80 \div 80,7 - 0) \div (80 \div 59,3 - 0) \approx 0,74$. Esses valores demonstram que a abordagem proposta é aproximadamente 35% $(1 \div 0,74)$ mais energeticamente econômica do que a abordagem PE. Os valores calculados para os demais algoritmos HS, ZT, OD e IT estão ilustrados na Figura 48.

Nossa abordagem atingiu no máximo 35% de economia quando comparada à abordagem PE e, no mínimo, 13% quando comparada à abordagem HS. Em média, nossa abordagem é 24,3% mais econômica para o *benchmark Vellamo Browser* quando comparada com as demais abordagens, este valor foi obtido a partir da média numérica dos ganhos percentuais presentes na Figura 48.

Figura 48 – Ganho energético relativo percentual médio para as abordagens de escalonamento de frequência de CPU: HS, ZT, OD, IT e PE em relação à abordagem proposta (AP) para o *benchmark Vellamo Browser*.



Fonte: O autor (2019).

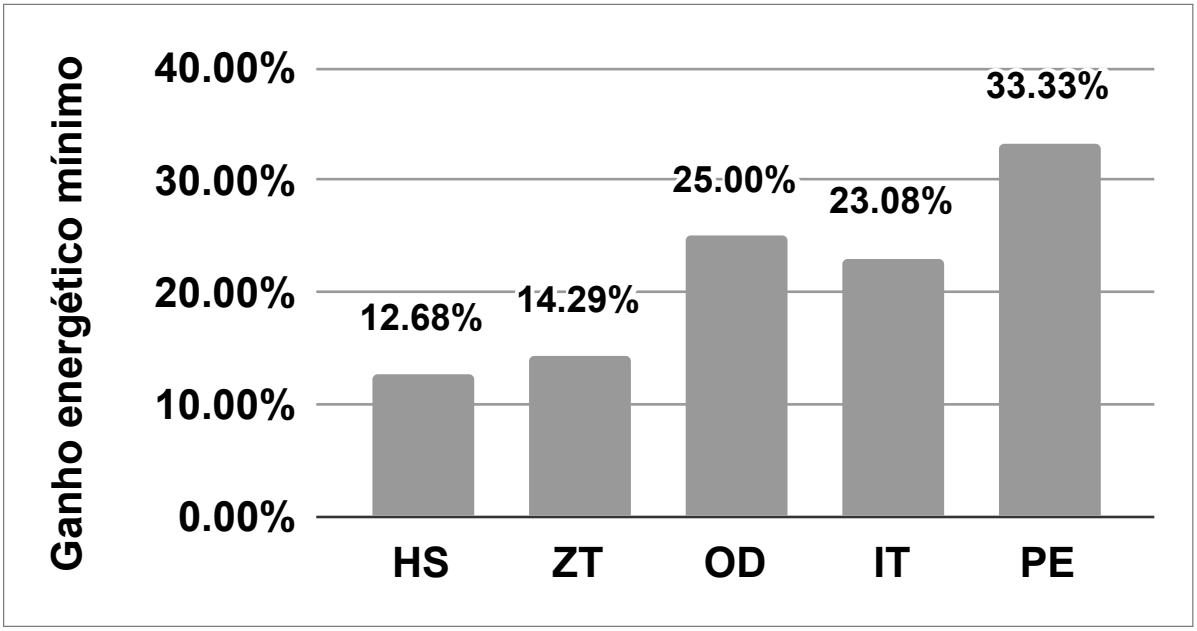
Ao realizar as repetições dos experimentos, prezamos por manter as mesmas condições físicas do ambiente onde o *smartphone* se encontrava, bem como, todo o processo de inicialização e execução do *benchmark* foi automatizado para evitar interferências hu-

manas. Todos estes procedimentos foram repetidos para todas as execuções de todos os *benchmarks* nos diferentes algoritmos de escalonamento.

Em continuidade à análise, o desvio padrão de $1,15$ pontos equivale a somente 1,42% do ganho energético médio, ou seja, assumimos que as 80 iterações executadas pela AP no *benchmark* representam os 100%, logo, ao aplicar a proporcionalidade, temos: $(1,15 \times 100) \div 80 \approx 1,42\%$.

Aplicando este mesmo conceito às demais abordagens, chegamos aos valores presentes na Figura 49, ilustrando o ganho mínimo esperado pela abordagem proposta, considerando o pior cenário, onde a nossa abordagem atinge o pior desempenho na economia de energia e o melhor cenário das demais abordagens. Mesmo sendo o pior cenário, nossa abordagem ainda é relevante por manter um valor muito próximo da média calculada, obtendo no mínimo 33% de economia de energia quando comparado à abordagem PE e 12% quando comparada à abordagem HS, sendo em média 21,67% mais econômica para o *benchmark Vellamo Browser*, quando comparada a todas as abordagens analisadas.

Figura 49 – Ganho energético relativo percentual mínimo para as abordagens de escalonamento de frequência de CPU: HS, ZT, OD, IT e PE em relação à abordagem proposta (AP) para o *benchmark Vellamo Browser.*



Fonte: O autor (2019).

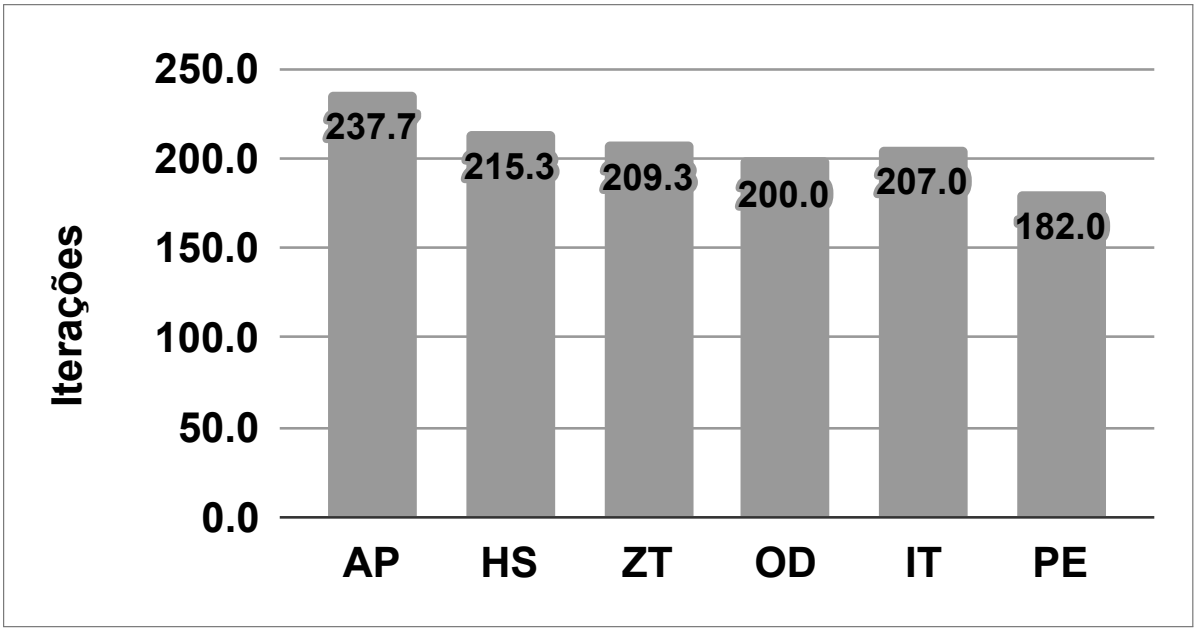
Todos os procedimentos demonstrados anteriormente serão aplicados aos demais *benchmarks* analisados, na sequência de aparição são: *Vellamo Metal, Vellamo Multicore e AnTuTu*. A seguir, vamos ilustrar os resultados para o *benchmark Vellamo Metal*.

Na sequência de análises, a Figura 50 ilustra a quantidade média de repetições alcançadas na execução do *benchmark Vellamo Metal* utilizando uma carga de bateria. É perceptível que a AP alcançou a maior quantidade de iterações do *benchmark* para uma

carga de bateria. O *governor* PE obteve o maior consumo de energia, atingindo somente 182 iterações do *benchmark Vellamo Metal*.

As abordagens HS e ZT obtiveram o resultado mais próximo da AP, enquanto os *governors* padrões do Android como OD, IT e PE obtiveram resultados piores, consumindo mais energia, como aconteceu no *benchmark Vellamo Browser*.

Figura 50 – Quantidade média de iterações do *benchmark Vellamo Metal* para uma carga de bateria utilizando as abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS, ZT, OD, IT e PE.



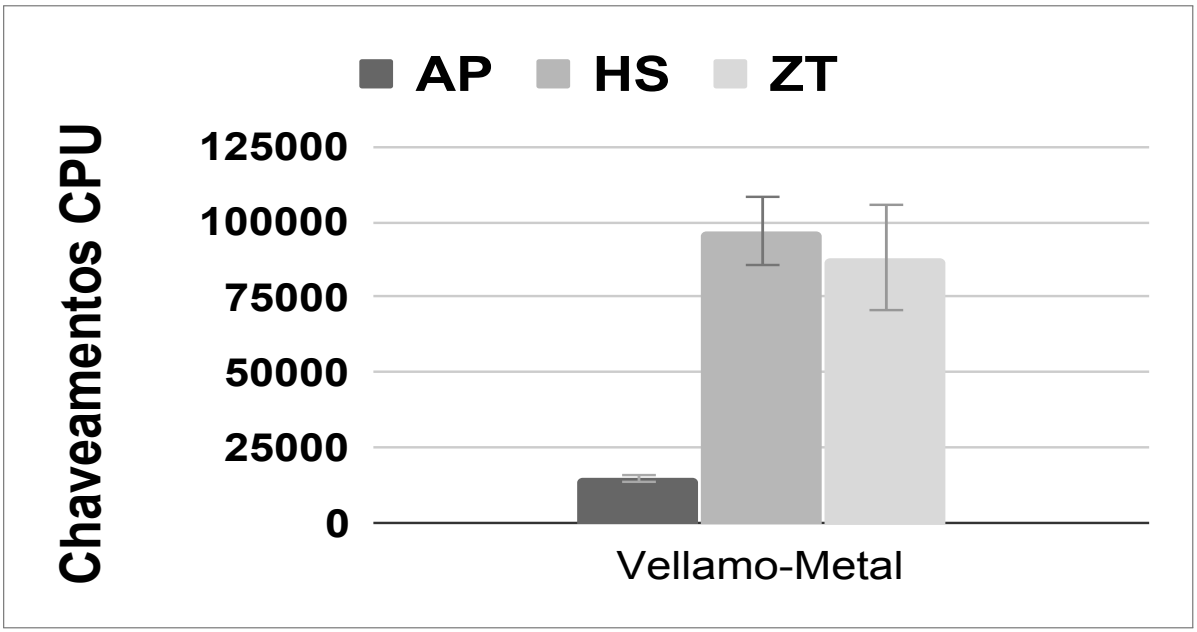
Fonte: O autor (2019).

O gráfico da Figura 51 ilustra o valor médio e o desvio padrão obtido a partir das repetições do experimento *Vellamo Metal*. Novamente, a quantidade de chaveamentos da AP é mínimo quando comparada às abordagens HS e ZT, corroborando para a economia de energia. A quantidade de chaveamentos da AP chega a ser 6x menor do que as demais abordagens analisadas. Como no *benchmark Vellamo Browser*, não foi possível a medição da quantidade de chaveamentos para os algoritmos nativos do sistema operacional, como o OD, IT e PE.

Na sequência, a Figura 52 ilustra a quantidade de energia consumida para as abordagens analisadas. A AP obteve o menor consumo de energia, seguida da abordagem HS, ZT, IT, OD e PE, respectivamente. Os valores foram normalizados de acordo com o maior consumo, referente à abordagem PE.

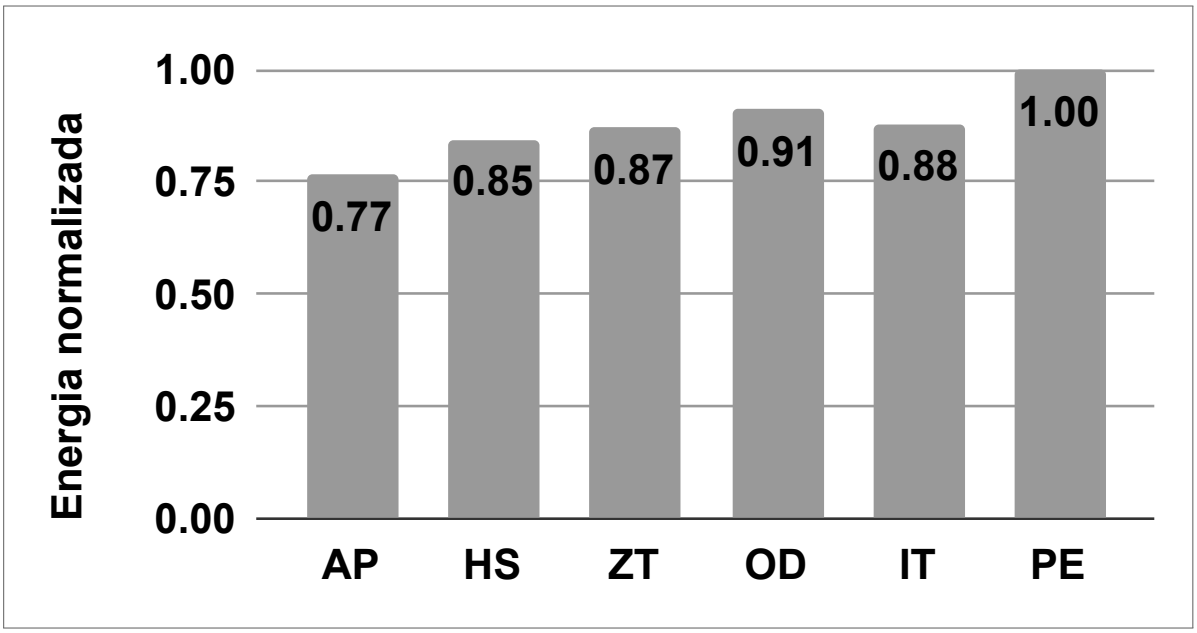
O PE é o algoritmo menos recomendado para o *benchmark Vellamo Metal* quando se deseja economizar energia. Por conseguinte, a Figura 53 exibe os valores percentuais de economia de energia comparando a abordagem proposta com as demais abordagens. A Tabela 17 sumariza os valores de repetição da abordagem proposta para o *benchmark Vellamo Metal*.

Figura 51 – Quantidade de chaveamentos de frequência de CPU média utilizada para executar o *benchmark Vellamo Metal* para uma carga de bateria utilizando as abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS e ZT.



Fonte: O autor (2019).

Figura 52 – Consumo de energia médio normalizado para o *benchmark Vellamo Metal* utilizando as abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS, ZT, OD, IT e PE.



Fonte: O autor (2019).

Podemos notar que as variações no número de repetições do *benchmark* para uma carga de bateria e utilizando a abordagem AP são mínimas, estando entre 236 e 240

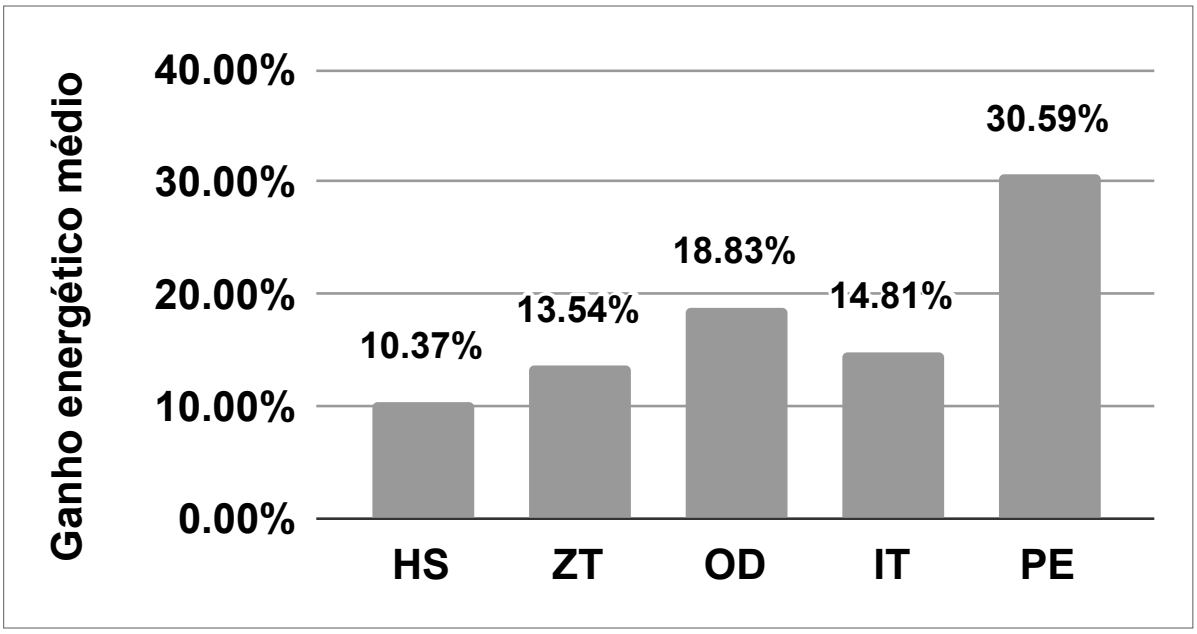
Tabela 17 – Dados e estatísticas das repetições das abordagens utilizadas para o *benchmark Vellamo Metal*.

Repetição / Qtd. iterações	AP	HS	ZT	OD	IT	PE
1	240	217	215	211	219	194
2	237	214	206	197	205	179
3	236	215	207	192	197	173
Estatísticas						
Média:	237,7	215,3	209,3	200,0	207,0	182,0
Mediana:	237	215	207	197	205	179
Desvio padrão:	2,08	1,53	4,93	9,85	11,14	10,82

Fonte: O autor (2019).

iterações.

Figura 53 – Ganho energético relativo percentual médio para as abordagens de escalonamento de frequência de CPU: HS, ZT, OD, IT e PE em relação à abordagem proposta (AP) para o *benchmark Vellamo Metal*.



Fonte: O autor (2019).

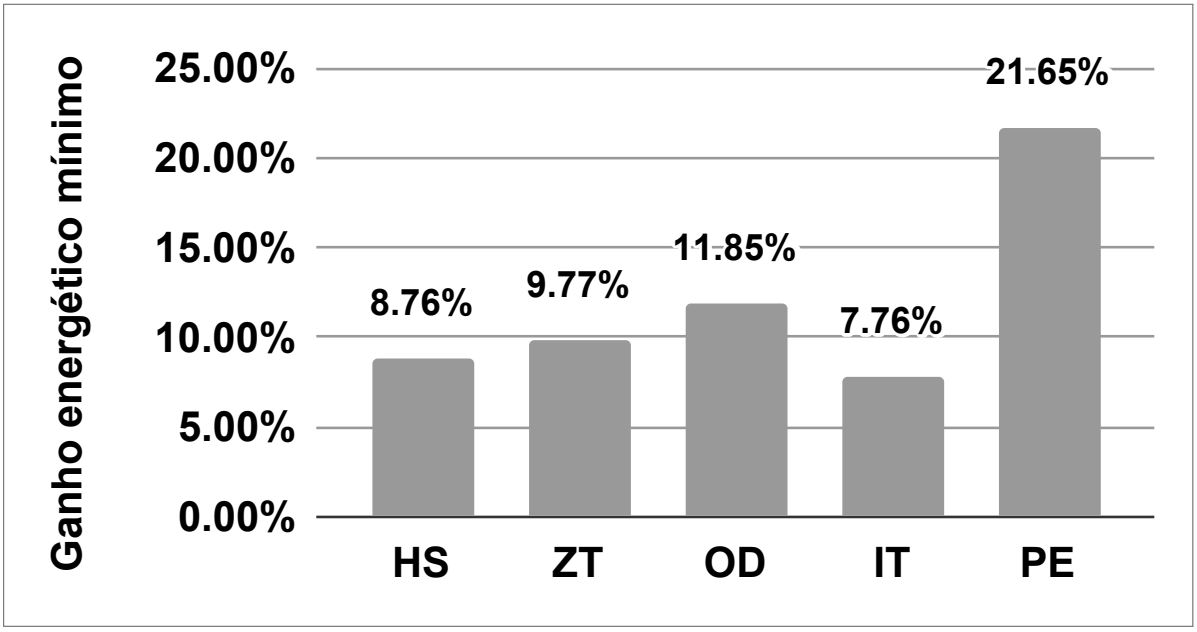
A abordagem proposta atingiu no máximo 30% de economia quando comparada à abordagem PE; e, no mínimo, 10% quando comparada à abordagem HS. Em média, nossa abordagem é 17,6% mais econômica para o *benchmark Vellamo Metal* quando comparada com as demais abordagens, de acordo com a média numérica dos ganhos percentuais presentes na Figura 48.

Ao aplicar o cenário do pior caso para as demais abordagens analisadas, chegamos aos valores presentes na Figura 54. A Figura 54 ilustra o ganho mínimo alcançado com o uso

da abordagem proposta. Mesmo utilizando a análise do pior cenário, nossa abordagem alcançou valores significativos de economia de energia, sendo, no mínimo, 7,7% de economia de energia quando comparada à abordagem IT e, no máximo, 21,6% de economia quando comparada à abordagem PE, sendo em média 11,9% mais econômica para o *benchmark Vellamo Metal* quando comparada a todas as abordagens analisadas e no cenário do pior caso.

Outro ponto interessante a ser destacado quanto ao consumo de energia, é que o *governor* IT, utilizado atualmente na maioria dos dispositivos que utilizam *Android*, apresentou a melhor economia de energia quando comparado aos demais algoritmos testados, exceto quando comparado às abordagens da literatura (HS e ZT) e a AP. Como já vimos, o *benchmark Vellamo Metal* objetiva o uso intensivo da CPU e o algoritmo IT propõe o alto desempenho com menor atraso de resposta para o usuário, muitas vezes, este atraso é gerado pela dormência da CPU, ativada pelo sistema operacional para economizar energia.

Figura 54 – Ganho energético relativo percentual mínimo para as abordagens de escalonamento de frequência de CPU: HS, ZT, OD, IT e PE em relação à abordagem proposta (AP) para o *benchmark Vellamo Metal.*

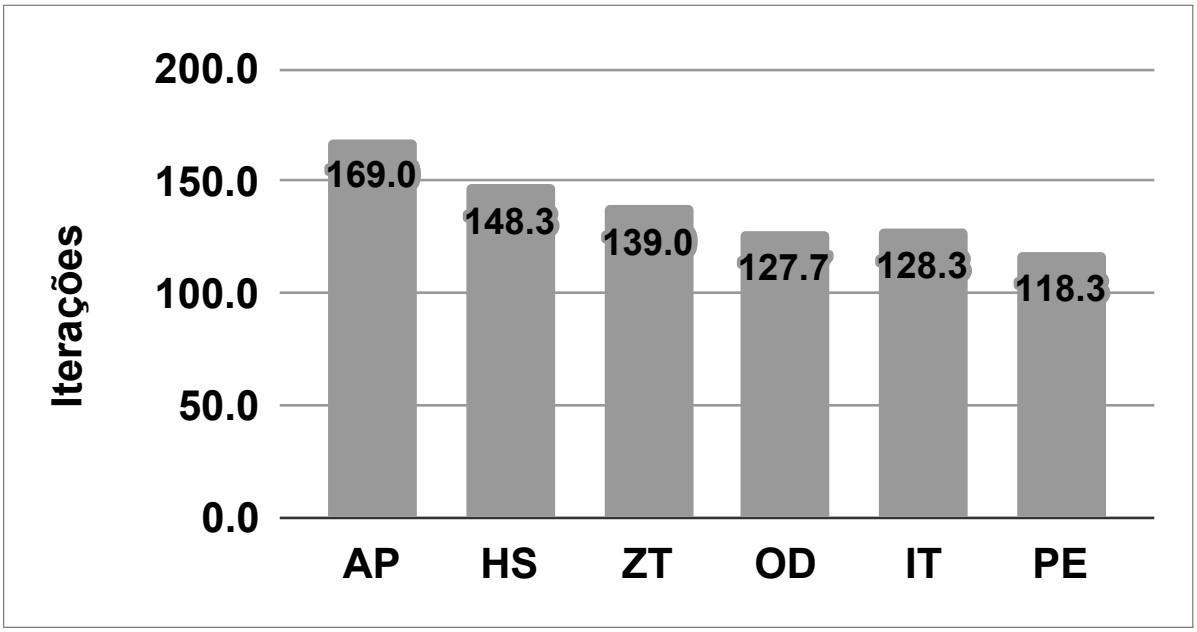


Fonte: O autor (2019).

Em continuidade, as figuras a seguir ilustram os resultados obtidos para o *benchmark Vellamo Multicore*. A Figura 55 ilustra a quantidade de repetições alcançadas na execução do *benchmark Vellamo Multicore* utilizando uma carga de bateria. Novamente, a AP alcançou a maior quantidade de iterações do *benchmark* para uma carga de bateria. O *governor* PE obteve o maior consumo de energia, atingindo somente 118,3 iterações do *benchmark Vellamo Multicore*, enquanto a AP conseguiu realizar 169 iterações.

As abordagens HS e ZT obtiveram o resultado mais próximo da AP, enquanto os *governors* padrões do Android como OD, IT e PE obtiveram resultados piores, consumindo

Figura 55 – Quantidade de iterações do *benchmark Vellamo Multicore* para uma carga de bateria utilizando às abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS, ZT, OD, IT e PE.



Fonte: O autor (2019).

mais energia, como aconteceu no *benchmark Vellamo Browser* e no *benchmark Vellamo Metal*.

O gráfico da Figura 56 ilustra o valor médio e o desvio padrão obtido a partir das repetições do experimento *Vellamo Multicore*. A quantidade de chaveamentos da AP novamente é baixíssima quando comparada às abordagens HS e ZT, colaborando para a economia de energia. A quantidade de chaveamentos da AP chega a ser 5x menor do que as demais abordagens analisadas. Como no *benchmark Vellamo Browser* e *benchmark Vellamo Metal*, não foi possível a medição da quantidade de chaveamentos para os algoritmos nativos do sistema operacional, como o OD, IT e PE.

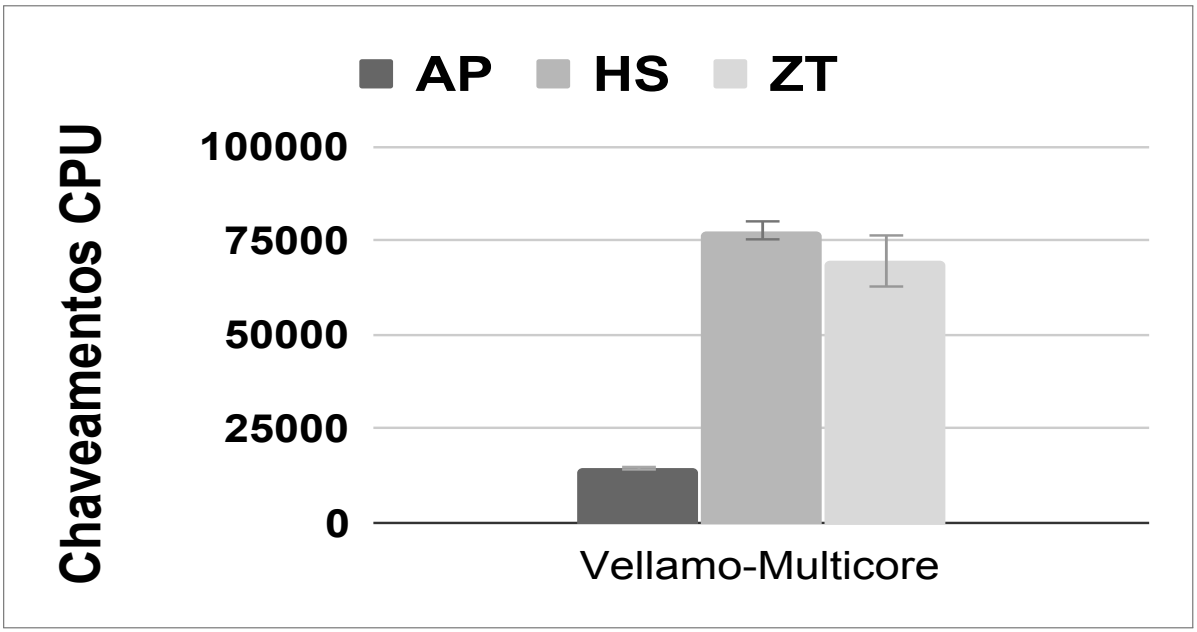
Na sequência, a Figura 57 ilustra a quantidade de energia consumida para às abordagens analisadas. A AP obteve o menor consumo de energia, seguida da abordagem HS, ZT, OD, IT e PE, respectivamente. Os valores foram normalizados de acordo com o maior consumo, referente à abordagem PE.

O PE é o algoritmo menos recomendado para o *benchmark Vellamo Multicore* quando se deseja economizar energia. Por conseguinte, a Figura 58 exibe os valores percentuais de economia de energia comparando a abordagem proposta com as demais abordagens. A Tabela 18 sumariza os dados e estatísticas das repetições das abordagens analisadas para o *benchmark Vellamo Multicore*.

Podemos notar que as variações no número de repetições do *benchmark* para uma carga de bateria também são mínimas, variando entre 166 e 173 iterações para a AP.

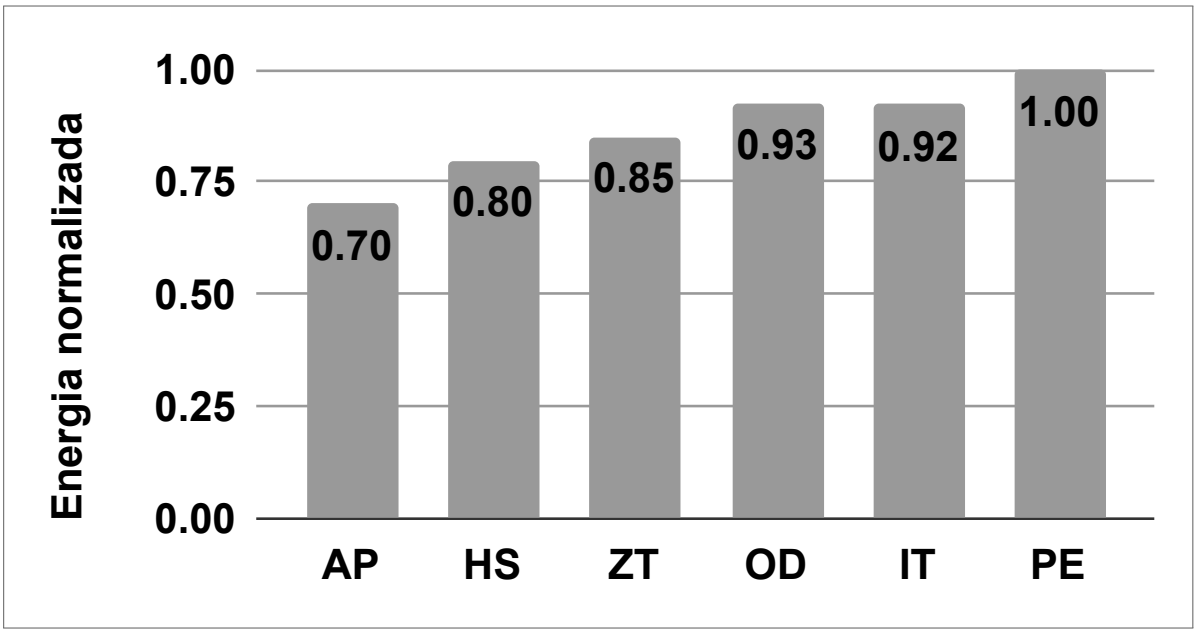
Nossa abordagem atingiu no máximo 42% de economia de energia quando comparada

Figura 56 – Quantidade de chaveamentos de frequência de CPU média utilizada para executar o *benchmark Vellamo Multicore* para uma carga de bateria utilizando às abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS e ZT.



Fonte: O autor (2019).

Figura 57 – Consumo de energia normalizado para o *benchmark Vellamo Multicore* utilizando às abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS, ZT, OD, IT e PE.



Fonte: O autor (2019).

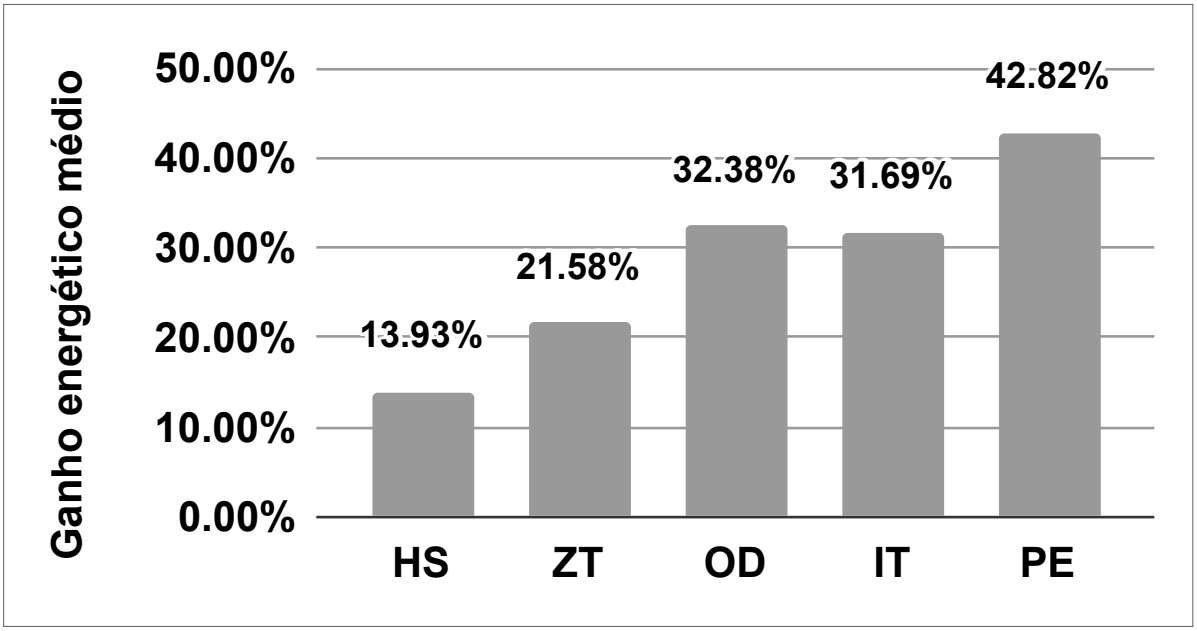
a abordagem PE e no mínimo 13% quando comparada a abordagem HS. Em média, nossa abordagem é 28,4% mais econômica para o *benchmark Vellamo Multicore* quando

Tabela 18 – Dados e estatísticas das repetições das abordagens utilizadas para o *benchmark Vellamo Multicore*.

Repetição / Qtd. iterações	AP	HS	ZT	OD	IT	PE
1	173	152	142	138	142	117
2	168	148	139	123	122	122
3	166	145	136	122	121	116
Estatísticas						
Média:	169,0	148,3	139,0	127,7	128,3	118,3
Mediana:	168	148	139	123	122	117
Desvio padrão:	3,61	3,51	3,00	8,96	11,85	3,21

Fonte: O autor (2019).

Figura 58 – Ganho energético relativo percentual para as abordagens de escalonamento de frequência de CPU: HS, ZT, OD, IT e PE em relação à abordagem proposta (AP) para o *benchmark Vellamo Multicore*.

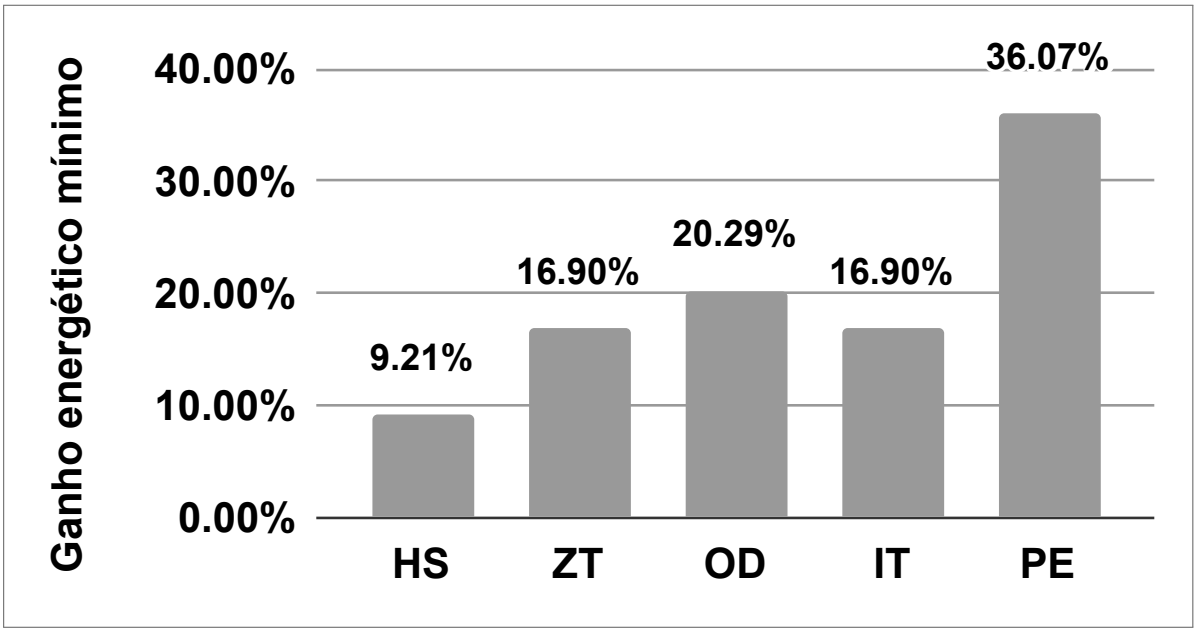


Fonte: O autor (2019).

comparada com as demais abordagens, de acordo com a média numérica dos ganhos percentuais presentes na Figura 58. Os valores para os demais algoritmos: HS, ZT, OD e IT estão ilustrados na Figura 58.

Ao aplicar o cenário do pior caso para as demais abordagens analisadas, chegamos aos valores presentes na Figura 59. A Figura 59 ilustra o ganho mínimo alcançado com o uso da abordagem proposta. Mesmo utilizando a análise do pior cenário, nossa abordagem alcançou valores significativos de economia de energia, sendo até 36% de economia de energia quando comparada à abordagem PE e 9%, quando comparada à abordagem HS, sendo em média 19,8% mais econômica para o *benchmark Vellamo Multicore*, quando

Figura 59 – Ganho energético relativo percentual mínimo para às abordagens de escalonamento de frequência de CPU: HS, ZT, OD, IT e PE em relação à abordagem proposta (AP) para o *benchmark Vellamo Multicore.*



Fonte: O autor (2019).

comparada a todas as abordagens analisadas no pior cenário para a AP e no melhor cenário para as demais abordagens.

Outro ponto interessante a ser destacado é que as abordagens da literatura mais recentes, como HS e ZT obtiveram a maioria dos resultados melhores que as abordagens padrões utilizadas no Android, sendo inferior somente à abordagem proposta neste trabalho, destacando a linha de evolução das abordagens e consequente melhoramento energético desenvolvido na literatura da área dos últimos anos.

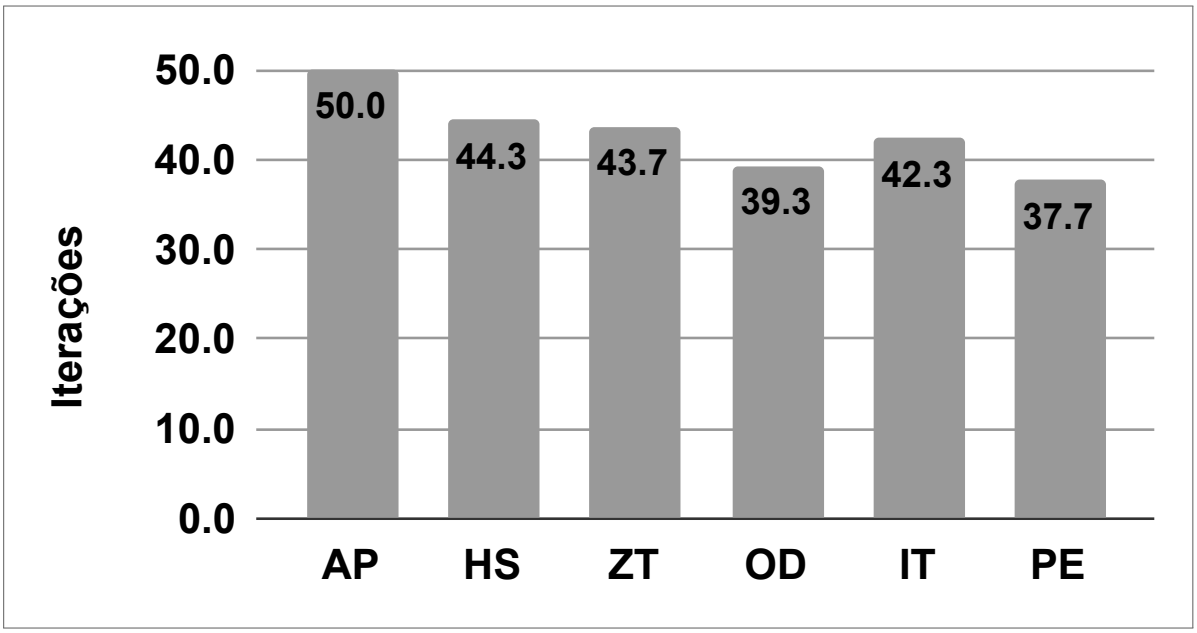
Os fatos expostos demonstram os diversos indícios de economia de energia que a abordagem proposta por este trabalho pode alcançar para os diversos *benchmarks* realizados, envolvendo a suíte *Vellamo*. Além disso, a qualidade e cobertura dos *microbenchmarks* permitem que nossa abordagem possa ser implementada em diversos outros dispositivos e, mesmo assim, alcançar efetivas economias de energia para o usuário do dispositivo móvel, principalmente os *smartphones*.

Por último, vamos analisar os resultados obtidos para o *benchmark AnTuTu*. Cada iteração do *AnTuTu* demora aproximadamente 14 *min* enquanto cada iteração do *Vellamo Browser* leva 9 *min*, do *Vellamo Metal* aproximadamente 4 *min* e o *Vellamo Multicore*, $4,5$ *min*.

As ilustrações a seguir ilustram os cenários para o *benchmark AnTuTu*. A Figura 60 ilustra a quantidade de repetições alcançadas na execução do *benchmark AnTuTu* utilizando uma carga de bateria. Novamente, a AP alcançou a maior quantidade de iterações do *benchmark* para uma carga de bateria. O *governor* PE obteve o maior consumo de

energia, atingindo em média 37,7 iterações do *benchmark AnTuTu* enquanto a AP conseguiu realizar 50 iterações, seguida das abordagens HS e ZT com 44,3 e 43,7 iterações, respectivamente.

Figura 60 – Quantidade de iterações do *benchmark AnTuTu* para uma carga de bateria utilizando às abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS, ZT, OD, IT e PE.



Fonte: O autor (2019).

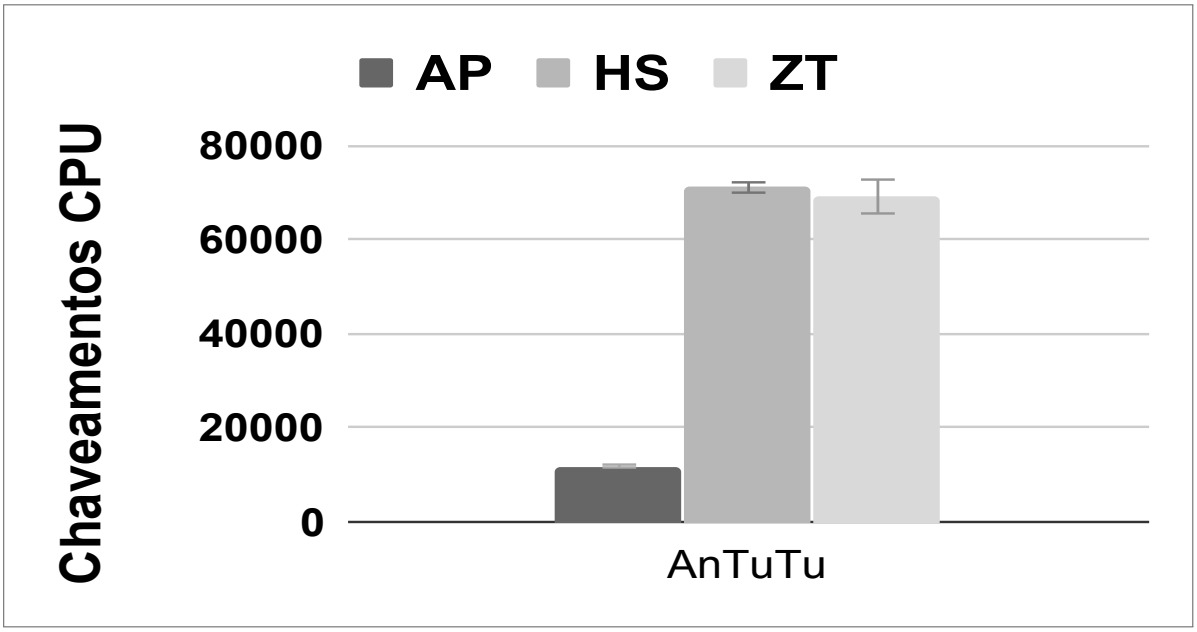
As abordagens HS e ZT obtiveram o resultado mais próximo da AP, enquanto os *governors* padrões do Android como OD, IT e PE obtiveram resultados piores, consumindo mais energia, como aconteceu no *benchmark Vellamo Browser*, *benchmark Vellamo Metal* e *benchmark Vellamo Multicore*.

A Figura 61 ilustra a quantidade média de chaveamentos utilizados para executar o *benchmark AnTuTu* utilizando uma carga de bateria. Novamente, a quantidade de chaveamentos necessários para executar a AP é mínima, quando comparada às demais abordagens da literatura, HS e ZT. A AP chega a possuir até 5x menos chaveamentos do que as abordagens da literatura analisadas, indicando economia de energia desperdiçada com chaveamentos desnecessários.

Na sequência, a Figura 62 ilustra a quantidade de energia consumida para as abordagens analisadas. A AP obteve o menor consumo de energia, seguida da abordagem HS, ZT, IT, OD e PE, respectivamente. Os valores foram normalizados de acordo com o maior consumo, referente à abordagem PE.

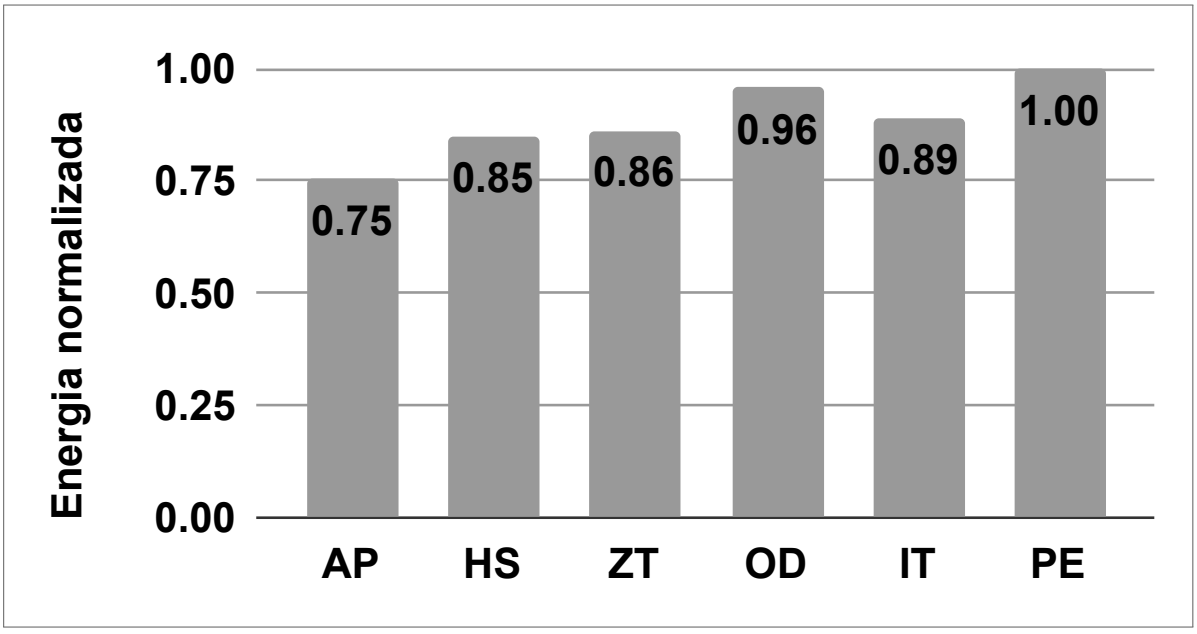
O PE é o algoritmo menos recomendado para o *benchmark AnTuTu* quando se deseja economizar energia. Por conseguinte, a Figura 63 exibe os valores percentuais de economia de energia comparando a abordagem proposta com às demais abordagens. A Tabela 19 sumariza os valores de repetição da abordagem proposta para o *benchmark AnTuTu*.

Figura 61 – Quantidade de chaveamentos de frequência de CPU média utilizada para executar o *benchmark AnTuTu* para uma carga de bateria utilizando as abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS e ZT.



Fonte: O autor (2019).

Figura 62 – Consumo de energia normalizado para o *benchmark AnTuTu* utilizando as abordagens de escalonamento de frequência de CPU: abordagem proposta (AP), HS, ZT, OD, IT e PE.



Fonte: O autor (2019).

É possível notar que as variações no número de repetições do *benchmark* para uma carga de bateria também são mínimas, a mais baixa de todos os *benchmarks* analisados,

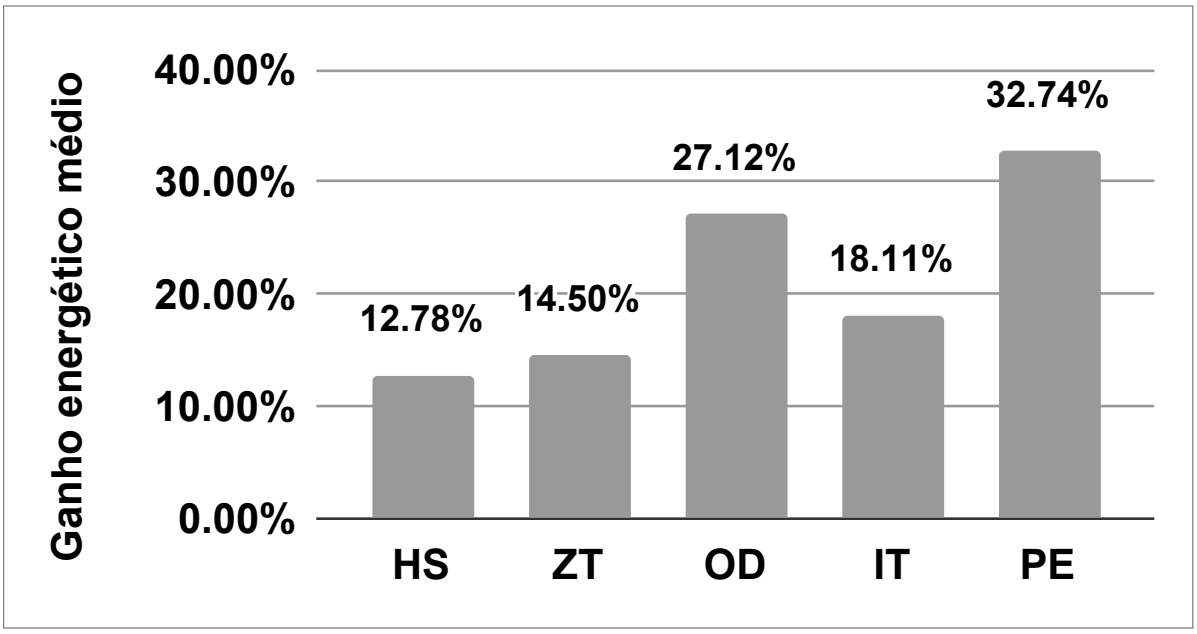
Tabela 19 – Dados e estatísticas das repetições das abordagens utilizadas para o *benchmark AnTuTu*.

Repetição / Qtd. iterações	AP	HS	ZT	OD	IT	PE
1	51	45	45	38	41	36
2	50	44	42	39	43	38
3	49	44	44	41	43	39
Estatísticas						
Média:	50,0	44,3	43,7	39,3	42,3	37,7
Mediana:	50	44	44	39	43	38
Desvio padrão:	1,00	0,58	1,53	1,53	1,15	1,53

Fonte: O autor (2019).

seguida do *Vellamo Browser*.

Figura 63 – Ganho energético relativo percentual para as abordagens de escalonamento de frequência de CPU: HS, ZT, OD, IT e PE em relação à abordagem proposta (AP) para o *benchmark AnTuTu*.



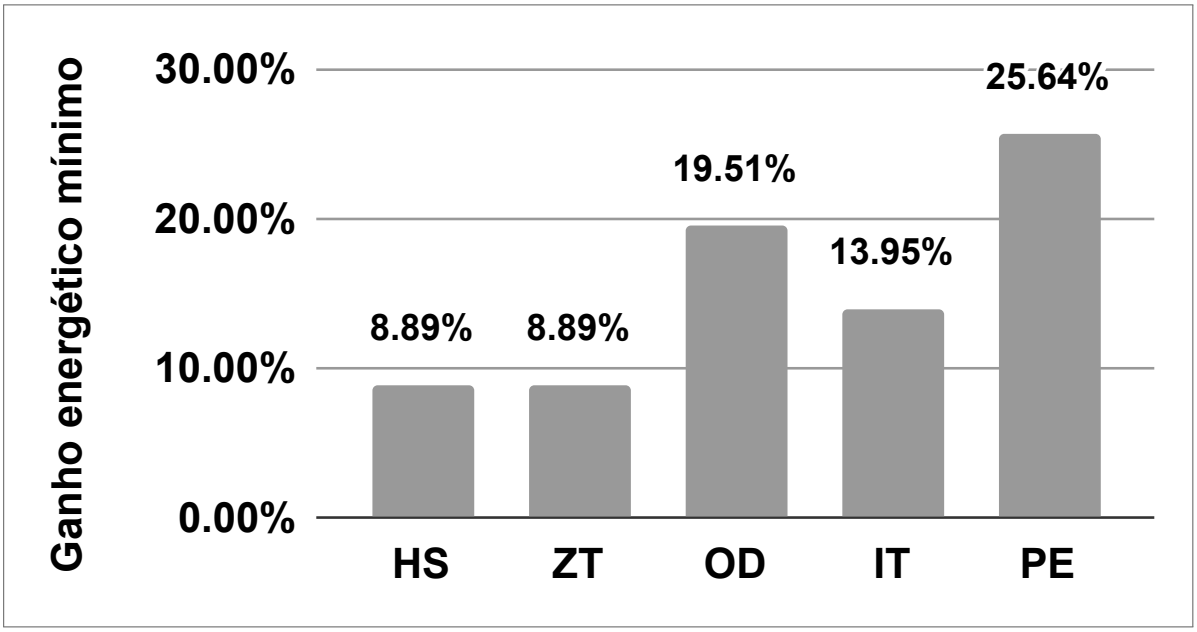
Fonte: O autor (2019).

Nossa abordagem atingiu, no máximo, 32,7% de economia quando comparada à abordagem PE e, no mínimo, 12,7% quando comparada à abordagem HS. Em média, nossa abordagem é 21% mais econômica para o *benchmark AnTuTu* quando comparada com as demais abordagens, de acordo com a média numérica dos ganhos percentuais presentes na Figura 63.

Ao aplicar o cenário do pior caso para as demais abordagens analisadas, chegamos aos valores presentes na Figura 64. A Figura 64 ilustra o ganho mínimo alcançado com o uso

da abordagem proposta. Mesmo utilizando a análise do pior cenário, nossa abordagem alcançou valores significativos de economia de energia, sendo 25,6% de economia de energia quando comparada à abordagem PE e, no mínimo, 8,8% quando comparada à abordagem ZT, sendo em média 15,2% mais econômica para o *benchmark AnTuTu*, quando comparada a todas as abordagens analisadas.

Figura 64 – Ganho energético relativo percentual mínimo para as abordagens de escalonamento de frequência de CPU: HS, ZT, OD, IT e PE em relação à abordagem proposta (AP) para o *benchmark AnTuTu*.



Fonte: O autor (2019).

A Tabela 20 sintetiza os resultados obtidos para os *benchmarks Vellamo Browser*, *Vellamo Metal*, *Vellamo Multicore* e *AnTuTu*. A Tabela 20 possui o nome do *benchmark* na primeira coluna e, em sequência: a quantidade de iterações (Qnt. ite.); a quantidades média de chaveamentos na CPU (M. chav.); a energia normalizada (E. norma.); o ganho energético da abordagem proposta quando comparada aos demais algoritmos (G. Energ.) e o ganho energético mínimo (Ganho e. mín.) referente à abordagem proposta e às demais abordagens.

O ganho energético da abordagem proposta ficou evidente para todos os *benchmarks* analisados. Em média, nossa abordagem economizou: 24,2% de energia para o *benchmark Vellamo Browser*; 17,5% de energia para o *benchmark Vellamo Metal*; 28,4% de energia para o *benchmark Vellamo Multicore* e; 21,0% de energia para o *benchmark AnTuTu*.

Por fim, o banco de informações, contendo todos os arquivos gerados pela execução dos *benchmarks* utilizando os algoritmos, citados anteriormente, está integralmente disponível ao público com o propósito de auxiliar outras pesquisas. O arquivo pode ser acessado no seguinte endereço: Banco de informações (clique aqui) sobre DOI *10.5281/zenodo.2532705*.

A seguir, vamos discutir o trabalho proposto com os principais trabalhos relacionados encontrados durante a produção deste documento, comparando suas características e onde a abordagem proposta melhora os trabalhos da literatura.

Tabela 20 – Síntese dos resultados dos *benchmarks Vellamo Browser*, *Vellamo Metal*, *Vellamo Multicore* e *AnTuTu*

Benchmark	Qnt. ite.	M. chav.	E. norma.	G. energ.	Ganho e. mín.
Vellamo Browser					
V-BW-AP	80,7	13608	0,74	n.d.	n.d.
V-BW-HS	71,0	76237	0,84	13,6%	12,6%
V-BW-ZT	67,7	73850	0,88	19,2%	14,2%
V-BW-OD	63,7	n.d.	0,93	26,7%	25,0%
V-BW-IT	64,0	n.d.	0,93	26,0%	23,0%
V-BW-PE	59,3	n.d.	1,0	35,9%	33,3%
Média	n.d.	n.d.	n.d.	24,2%	21,6%
Vellamo Metal					
V-MT-AP	237,7	14845	0,77	n.d.	n.d.
V-MT-HS	215,3	96975	0,85	10,3%	8,7%
V-MT-ZT	209,3	88168	0,87	13,5%	9,7%
V-MT-OD	200,0	n.d.	0,91	18,8%	11,8%
V-MT-IT	207,0	n.d.	0,88	14,8%	7,7%
V-MT-PE	182,0	n.d.	1,0	30,5%	21,6%
Média	n.d.	n.d.	n.d.	17,5%	11,9%
Vellamo Multicore					
V-MC-AP	169,0	14598	0,70	n.d.	n.d.
V-MC-HS	148,3	77730	0,80	13,9%	9,2%
V-MC-ZT	139,0	69602	0,85	21,5%	16,9%
V-MC-OD	127,7	n.d.	0,93	32,3%	20,2%
V-MC-IT	128,3	n.d.	0,92	31,6%	16,9%
V-MC-PE	118,3	n.d.	1,0	42,8%	36,0%
Média	n.d.	n.d.	n.d.	28,4%	19,8%
AnTuTu					
AT-AP	50,0	11933	0,75	n.d.	n.d.
AT-HS	44,3	71079	0,85	12,7%	8,8%
AT-ZT	43,7	69163	0,86	14,5%	8,8%
AT-OD	39,3	n.d.	0,96	27,1%	19,5%
AT-IT	42,3	n.d.	0,89	18,1%	13,5%
AT-PE	37,7	n.d.	1,0	32,7%	25,6%
Média	n.d.	n.d.	n.d.	21,0%	15,2%

Fonte: O autor (2019).

5.6 COMPARAÇÃO COM OS TRABALHOS RELACIONADOS

A Tabela 21 exibe um resumo dos principais trabalhos citados e da abordagem proposta nesta tese. A coluna Aprendizado indica o tipo de técnica utilizada no aprendizado da política de escalonamento de frequência, podendo ser *online* ou *offline*. Na coluna Política, é exibido o tipo de política usada para aplicar o escalonamento de frequência, podendo ser reativa ou proativa. Os *governors* padrões no sistema Android são reativos, enquanto alguns trabalhos recentes demonstraram que os algoritmos que utilizam a abordagem proativa são mais eficientes.

A coluna Predição de carga (Pred. Carga) ilustra se alguma técnica é utilizada para a predição da carga de processamento e especifica qual. A coluna Ajuste λ ilustra como o λ é refinado para uma versão adaptativa do algoritmo de predição AEWMA. A coluna Detecção de mudança (Det. mudança) contém a técnica utilizada para detecção da mudança da carga de processamento.

A coluna Predição de potência (Pred. pot.) exibe se alguma estratégia é usada para prever a potência do dispositivo, informando se o método de predição é linear ou não-linear. Adicionalmente, a mesma coluna ilustra se a abordagem suporta o uso de sensor de potência (Pot. real (sensor)) para alimentar o algoritmo de aprendizagem. Esse sensor está se tornando um componente comum nos *smartphones* atuais; utilizar o sensor de potência para ensinar ao algoritmo, inibe os erros de predição presentes nos modelos de predição de potência.

Ademais, o ambiente experimental utilizado é descrito na coluna Amb. exp. onde é destacado o tipo de ambiente utilizado nos experimentos, como *smartphones* ou protótipos como infraestrutura de validação ou se foram utilizados computadores pessoais ou servidores. Além disso, a coluna economia de energia (Eco. energia) exibe a porcentagem de economia de energia alcançada pelas abordagens. Por fim, a coluna Ano exibe o ano de publicação do trabalho analisado.

Tabela 21 – Comparativo entre os trabalhos relacionados.

	Aprendizado	Política	Pred. carga	Ajuste λ	Det. mudança	Pred. pot.	Pot. real (sensor)	Amb. exp.	Eco. energia	Ano
Proposta	*Online*	Proativa	AEWMA	Offline	MSE	Não-linear	Sim	Móvel	Até 42%	2019
Tian et al. (2018)	*Online*	Reativa	n.d.	n.d.	n.d.	Linear	Não	Móvel	Até 10%	2018
Wang et al. (2017)	*Online*	Reativa	n.d.	n.d.	n.d.	n.d.	Não	Servidor	Até 28%	2017
Das et al. (2016)	*Online*	Proativa	EWMA	n.d.	KL	n.d.	Não	Móvel	n.d.	2016
Shafik et al. (2016)	*Online*	Proativa	AEWMA	Decai. exp.	APP $\leftrightarrow$ *Governor*	n.d.	Não	Móvel	Até 33%	2016
Pasricha, Donohoo e Ohlsen (2015)	*Online*	Reativa	n.d.	n.d.	n.d.	Linear	Não	Móvel	Até 29%	2015
Carroll e Heiser (2014)	*Offline*	Reativa	n.d.	n.d.	n.d.	n.d.	Não	Móvel	Até 26%	2014
Shen et al. (2013)	*Online*	Reativa	n.d.	n.d.	n.d.	Linear	Não	Pessoal	n.d.	2013

Fonte: O autor (2019).

6 CONCLUSÕES E TRABALHOS FUTUROS

Vimos que a popularização dos dispositivos móveis, principalmente os *smartphones*, evidenciou a necessidade de dispositivos que consigam realizar mais computação com um menor consumo de energia, proporcionando um maior tempo de uso para o usuário final. Além disso, o tempo de uso proporcionado pelos dispositivos atuais é menor do que os dispositivos de alguns anos atrás.

Diante deste alto consumo de energia nos dispositivos móveis, a CPU é um dos componentes que mais demandam energia, drenando a bateria rapidamente. Embora a eficiência das baterias tenha sido otimizada, provendo mais capacidade de carga e maior tempo de vida, esses avanços são inferiores à evolução dos demais componentes, dentre eles, os microprocessadores.

Para combater esse problema do reduzido tempo de uso dos dispositivos por restrições energéticas, algumas tecnologias foram desenvolvidas para redução do consumo de energia. Essas tecnologias têm o objetivo de reduzir o consumo de energia com foco em alguns componentes do dispositivo. Na CPU, uma das técnicas mais utilizadas é o DVFS. Neste trabalho, utilizamos essa técnica juntamente com algoritmos de predição e aprendizagem de máquina para prover economia energética.

Além disso, dada a grande diversidade de dispositivos e aplicações existentes atualmente, mostrou-se necessária uma abordagem que não exigisse conhecimento prévio do ambiente, podendo ser aplicada em diversos cenários. Para cumprir este quesito, identificamos um ramo da aprendizagem de máquina, a aprendizagem por reforço, que pode proporcionar uma solução a esta necessidade.

O problema encontrado foi: como descobrir as frequências de CPU que reduzem o consumo de energia do dispositivo para determinados cenários? Essa pergunta deve ser respondida levando em consideração a grande quantidade de cenários distintos, resultados de muitos aplicativos e diferentes intensidades de processamento. A partir disto, propomos uma abordagem autônoma, capaz de se adaptar aos diferentes cenários em tempo de execução do dispositivo.

Para identificar a frequência de CPU mais indicada para um determinado cenário, mapeamos a carga de processamento da CPU como uma característica deste cenário. Percebemos que uma abordagem proativa é mais eficiente, realizando a predição da carga de processamento e permitindo que a frequência de CPU que economiza energia seja definida antes da carga de processamento chegar. Ademais, propomos um algoritmo de predição da carga de processamento baseado no AEWMA, chamado de AEWMA-MSE, que, além de proporcionar acurácia similar na predição da carga de processamento, possui a característica de detecção de mudanças significativas na carga de processamento. Essa característica é importante para ser utilizada em conjunto com a aprendizagem de

máquina, mais especificamente o algoritmo *Q-learning*, permitindo o aprendizado a cada nova mudança significativa, de forma autônoma.

Em continuidade, foi desenvolvido um aplicativo para Android que realiza a coleta de 65 características do dispositivo, chamado de CL. O aplicativo foi executado em um *smartphone* Motorola juntamente com diversos *benchmarks* que simulavam ações comuns de um usuário de *smartphone* para coleta de dados. A partir dessas execuções, foi criado um banco de dados que foi essencial para a produção dos modelos de predição de potência.

Em sequência, os modelos de predição de potência foram avaliados utilizando métodos lineares e não-lineares. Quanto aos métodos de predição de potência analisados, concluímos que para os cenários analisados, ficou evidente que o modelo *k-NN* possui o melhor balanceamento entre desempenho e acurácia, quando comparado aos outros modelos analisados, sendo recomendado o *k-NN* com configuração $k = 100$ como a melhor opção entre o modelo linear e a ANN, justificado pelo melhor balanceamento entre acurácia de predição e baixo tempo de treinamento e predição.

Após isso, uma abordagem integrando os modelos de predição e aprendizagem por reforço foi proposta para identificar a frequência de CPU que proporciona o menor consumo de energia para o cenário atual do dispositivo. A abordagem proposta é autônoma e se adapta aos diversos cenários proporcionados pela grande quantidade de aplicações e intensidades de cargas de processamento.

A abordagem proposta utilizando predição de potência com k-NN na função de custo para o *Q-learning* pode alcançar até 29% de economia de energia para os cenários de uso testados. Em uma variação da abordagem proposta, notamos que alguns *smartphones* possuem um sensor de potência que pode ser acessado em tempo de execução para prover o valor de potência energética instantânea.

A variação da abordagem com utilização do sensor de potência foi testada em um smartphone Xiaomi com diversos *benchmarks* populares e representativos. Nossa abordagem foi confrontada com cinco abordagens para chaveamento de frequência da CPU, sendo três *governors* padrões no Android (OD, IT e PE) e duas da literatura atual (HS e ZT). Para essa variação da abordagem, foi alcançada uma economia energética em todos os *benchmarks* utilizados, alcançando até 42% de economia, a depender da abordagem e *benchmark* em comparação. É importante destacar que a metodologia utilizada para avaliar o consumo de energia e comparar os algoritmos é uma inovação desta tese, podendo ser utilizada como arcabouço para novas metodologias.

A partir deste trabalho, foram alcançadas publicações internacionais, em conferências e periódicos de alto impacto na tabela CAPES, listados a seguir.

- S. A. L. Carvalho, R. N. Lima, D. C. Cunha, e A. G. Silva-Filho. *A hardware and software Web-based environment for Energy Consumption analysis in mobile devices. In proceedings of 15th International Symposium on Network Computing and Applications* (NCA), p. 242–245, 2016.

- S. A. L. Carvalho, D. C. Cunha, e A. G. Silva-Filho. *On the use of nonlinear methods for low-power cpu frequency prediction based on android context variables. In proceedings of 15th International Symposium on Network Computing and Applications* (NCA), p. 250–253, 2016.
- S. A. L. Carvalho, D. C. Cunha, e A. G. Silva-Filho. *Autonomous Power Management for Embedded Systems using a Non-Linear Power Predictor. In proceedings of Euromicro Conference on Digital System Design* (DSD), p. 22–29, 2017.
- S. A. L. Carvalho, D. C. Cunha, e A. G. Silva-Filho. *Autonomous power management in mobile devices using dynamic frequency scaling and reinforcement learning for energy minimization. Microprocessors and Microsystems, 64 (2019)* (MICPRO), p. 205–220, 2019.

Por fim, na próxima seção iremos discursar sobre as principais continuações do trabalho proposto, indicando caminhos promissores de pesquisa para os próximos anos.

6.1 TRABALHOS FUTUROS

Os projetos listados a seguir podem ser realizados para agregar mais conhecimento á abordagem proposta, desenvolver novas funcionalidades, otimizar funções existentes ou criar novas visões do problema exposto por este trabalho.

- Analisar o impacto de outros algoritmos da aprendizagem por reforço para o problema especificado neste trabalho, por exemplo, substituir o *Q-learning* por SARSA. Após isso, é necessário executar os experimentos novamente e confrontar os resultados na economia de energia.
- Avaliar o uso da divergência de *Kullback–Leibler* para detecção de mudança na no algoritmo AEWMA-MSE. Esse foi utilizado no trabalho de Das et al. (2016) e apresentou bons resultados para um problema similar.
- Avaliar a inclusão da temperatura como uma restrição da função de custo no algoritmo de aprendizagem por reforço. Além disso, analisar o impacto no consumo de energia do dispositivo e avaliar a predição dessa variável, que impacta diretamente o consumo de energia do processador.
- Inserir uma restrição de desempenho na função de custo do algoritmo de aprendizagem por reforço, podendo priorizar o desempenho para alguns cenários ou a economia energética para outros. Esse melhoramento vai permitir que seja feito o balanceamento entre esses dois recursos.
- Avaliar a evolução da inteligência da *Q-table*, comparando a frequência que economiza energia com o ótimo global, obtido a partir de medição por hardware.

- Otimizar as etapas de exploração e explotação utilizada pelo algoritmo de aprendizagem por reforço. Sugiro criar uma abordagem adaptativa do algoritmo ϵ-*greedy*, permitido que a partir de algumas iterações, a taxa de exploração vá diminuindo gradativamente, melhorando as perdas por exploração após a solução ótima ser alcançada.

- Analisar o impacto de uma variação do modelo linear proposto no trabalho de Walker et al. (2017) para predição de potência ou da carga de processamento. Walker et al. (2017) obteve um erro médio de apenas 3% ao combinar diversos modelos de regressão linear.

- Implementar a abordagem proposta no *kernel* Linux, permitindo que seja executado como um *governor* padrão no Android. A implementação é realizada na linguagem C, seguindo as normas do Linux. Após a implementação, é necessário que esse arquivo seja adicionado como um módulo do *kernel* e então o SO deve ser compilado com essa nova funcionalidade. Além disso, é importante analisar as proteções inseridas pelos fabricantes nos *smartphones* comerciais antes de gerar a nova *build* do Android com o *governor* incorporado. Para facilitar os testes da implementação, sugiro o uso de uma placa protótipo, por exemplo, a Odroid-XU4, com sistema operacional Android e SoC Exynos 5422, utilizada em *smartphones* comerciais.

REFERÊNCIAS

ABOUGHAZALEH, N.; FERREIRA, A.; RUSU, C.; XU, R.; LIBERATO, F.; CHILDERS, B.; MOSSE, D.; MELHEM, R. *Integrated CPU and l2 cache voltage scaling using machine learning*. 2007. Disponível em: <http://dl.acm.org/citation.cfm?id=1254773>.

AKAN, O. B.; CETINKAYA, O.; KOCA, C.; OZGER, M. Internet of hybrid energy harvesting things. *IEEE INTERNET OF THINGS JOURNAL*, v. 5, n. 2, p. 736–746, 2018.

ARDUINO. *Arduino Uno Board*. 2015. Disponível em: <http://arduino.cc/en/Main/arduinoBoardUno>.

ARM. *ARM big.LITTLE*. 2013. Disponível em: <https://developer.arm.com/technologies/big-little>.

BHATTI, K.; BELLEUDY, C.; AUGUIN, M. Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. In: *Proc. IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. [S.l.: s.n.], 2010. p. 184–191.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. [S.l.]: Springer, 2006.

CAPIZZI, G.; MASAROTTO, G. An adaptive exponentially weighted moving average control chart. *Technometrics*, v. 45, n. 3, p. 199–207, 2003.

CARROLL, A.; HEISER, G. An analysis of power consumption in a smartphone. In: *Proc. USENIX annual technical conference*. [S.l.: s.n.], 2010.

CARROLL, A.; HEISER, G. The systems hacker's guide to the galaxy energy usage in a modern smartphone (apsys). In: *Proc. 4th Asia-Pacific Workshop on Systems*. [S.l.: s.n.], 2013. p. 1–7.

CARROLL, A.; HEISER, G. Unifying dvfs and offlining in mobile multicores. In: *Proc. RTAS*. [S.l.: s.n.], 2014. p. 287–296.

CARVALHO, S. A. L.; CUNHA, D. C.; SILVA-FILHO, A. G. On the use of nonlinear methods for low-power cpu frequency prediction based on android context variables. In: *Proc. NCA*. [S.l.: s.n.], 2016. p. 250–253.

CARVALHO, S. A. L.; CUNHA, D. C.; SILVA-FILHO, A. G. Autonomous power management for embedded systems using a non-linear power predictor. In: *Proc. Euromicro Conference on Digital System Design (DSD)*. [S.l.: s.n.], 2017. p. 1–8.

CENTER, Q. I. *Vellamo Mobile Benchmark*. 2017. Disponível em: <https://play.google.com/store/apps/details?id=com.quicinc.vellamo>.

CHAN, M. *Linux Kernel Patch: cpufreq: interactive: New 'interactive' governor*. 2015. Disponível em: <https://lwn.net/Articles/662209/>.

CHANG, M.-F.; LIANG, W.-Y. Learning-directed dynamic voltage and frequency scaling for computation time prediction. In: *Proc. of the IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM)*. [S.l.: s.n.], 2011. p. 1023–1029.

CHAPELLE, O.; SCHLKOPF, B.; ZIEN, A. Semi-supervised learning. *IEEE Transactions on Neural Networks*, v. 20, p. 542–542, 2009.

CHEN, D.-R.; CHEN, Y.-S.; CHEN, L.-C.; HSU, M.-Y.; CHIANG, K.-F. A machine learning method for power prediction on the mobile devices. *Journal of Medical Systems - Mobile Systems*, p. 1–11, 2015.

CHEN, X.; CHEN, Y.; MA, Z.; FERNANDES, F. C. A. How is energy consumed in smartphone display applications? In: *Proc. 14th Workshop on Mobile Computing Systems and Applications*. [S.l.: s.n.], 2013. p. 1–6.

DAS, A.; MERRETT, G. V.; TRIBASTONE, M.; AL-HASHIMI, B. M. Workload change point detection for runtime thermal management of embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, v. 35, n. 8, p. 1358–1371, 2016.

DAUD, S.; AHMAD, R. B.; LYNN, O. B.; KAREEM, Z. I. A.; KAMARUDDIN, L. M.; EHKAN, P.; WARIP, M. N. M.; OTHMAN, R. R. The effects of cpu load & idle state on embedded processor energy usage. In: *Proc. ICED*. [S.l.: s.n.], 2014. p. 30–35.

DEVELOPER, A. *Android 5.0 Behavior Changes*. 2015. Disponível em: <https://developer.android.com/about/versions/android-5.0-changes.html>.

DEVELOPER, A. *Measuring Device Power*. 2017. Disponível em: <https://source.android.com/devices/tech/power/device>.

DHIMAN, G.; ROSING, T. Dynamic power management using machine learning. In: *Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. [S.l.: s.n.], 2006. p. 747–754.

DHIMAN, G.; ROSING, T. S. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In: *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*. [S.l.: s.n.], 2007. p. 207–212.

DHIMAN, G.; ROSING, T. S. System-level power management using online learning. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, v. 28, n. 5, p. 676–689, 2009.

FATHABADI, A. S.; MAEDA-NUNEZ, L. A.; BUTLER, M. J.; AL-HASHIMI, B. M.; MERRETT, G. V. Generation of run-time power management for embedded systems using formal methods. In: *Proc. of the IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. [S.l.: s.n.], 2015. p. 104–111.

FLAUTNER, K.; REINHARDT, S.; MUDGE, T. Automatic performance setting for dynamic voltage scaling. In: *Proc. of the 7th Annual International Conference on Mobile Computing and Networking*. [S.l.: s.n.], 2001. p. 260–271.

FREEH, V. W.; LOWENTHAL, D. K.; PAN, F.; KAPPIAH, N.; SPRINGER, R.; ROUNTREE, B. L.; FEMAL, M. E. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Transactions on Parallel and Distributed Systems*, v. 18, n. 6, p. 835–848, Jun 2007.

GARG, N.; GARG, R. Energy harvesting in iot devices: A survey. In: *Proc. International Conference on Intelligent Sustainable Systems*. [S.l.: s.n.], 2017. p. 127–131.

GARTNER. *Gartner Says Five of Top 10 Worldwide Mobile Phone Vendors Increased Sales in Second Quarter of 2016*. 2016. Disponível em: <http://www.gartner.com/newsroom/id/3415117>.

GARTNER. *Gartner Says Huawei Secured No. 2 Worldwide Smartphone Vendor Spot, Surpassing Apple in Second Quarter 2018*. 2018. Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2018-08-28-gartner-says-huawei-secured-no-2-worldwide-smartphone-vendor-spot-surpassing-apple-

GUO, J.; POTKONJAK, M. Coarse-grained learning-based dynamic voltage frequency scaling for video decoding. In: *Proc. of the 26th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. [S.l.: s.n.], 2016. p. 84–91.

GUTHAUS, M.; RINGENBERG, J.; ERNST, D.; AUSTIN, T.; MUDGE, T.; BROWN, R. Mibench: A free, commercially representative embedded benchmark suite. In: *Proc. IEEE International Workshop on Workload Characterization*. [S.l.: s.n.], 2001. p. 3–13.

HERBERT, S.; MARCULESCU, D. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In: *Proc. of the International Symposium on Low Power Electronics and Design (ISLPED)*. [S.l.: s.n.], 2007. p. 38–43.

HERBERT, S.; MARCULESCU, D. Variation-aware dynamic voltage/frequency scaling. In: *Proc. of the 15th International Symposium on High Performance Computer Architecture (HPCA)*. [S.l.: s.n.], 2009. p. 301–312.

HODGES, A. *Alan Turing: the enigma*. New York: Simon and Schuster, 1983. ISBN 978-0-671-49207-6 978-0-671-52809-6.

HOLMBACKA, S.; KELLERB, J.; EITSCHBERGERB, P.; LILIUSA, J. Accurate energy modeling for many-core static schedules with streaming applications. *Microprocessors and Microsystems*, v. 43, p. 14–25, Jun. 2016.

HSU, C.-H.; FENG, W.-C. Effective dynamic voltage scaling through cpu-boundedness detection. In: *Proc. of the 4th International Conference on Power-Aware Computer Systems (PACS)*. [S.l.: s.n.], 2004. p. 135–149.

INDUSTRIES, A. *INA219 High Side DC Current Sensor Breakout*. 2015. Disponível em: <http://www.adafruit.com/products/904>.

ISCI, C.; ; BUYUKTOSUNOGLU, A.; CHER, C. yong; BOSE, P.; MARTONOSI, M. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In: *Proc. of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. [S.l.: s.n.], 2006. p. 347–358.

ISLAM, F. M. M. ul; LIN, M. Hybrid dvfs scheduling for real-time systems based on reinforcement learning. *IEEE Systems Journal*, v. 11, n. 2, p. 931–940, 2015.

JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. *An Introduction to Statistical Learning with Applications in R.* [S.l.]: Springer New York Heidelberg Dordrecht London, 2013. 39-42 p.

JUNG, H.; PEDRAM, M. Supervised learning based power management for multicore processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 29, n. 9, p. 1395–1408, 2010.

KANA, E. Y. Y.; CHANB, W. K.; TSEA, T. H. Eclass: An execution classification approach to improving the energy-efficiency of software via machine learning. *Journal of Systems and Software*, v. 85, n. 4, p. 960–973, 2012.

KEATING, M.; FLYNN, D.; AITKEN, R.; SHI, A. G. andKaijian. *Low Power Methodology Manual For System-on-Chip Design.* [S.l.]: Springer, 2008.

KERNEL, T. L. *The /proc filesystem.* 2009. Disponível em: <https://www.kernel.org/doc/Documentation/filesystems/proc.txt>.

KIM, J. M.; KIM, Y. G.; CHUNG, S. W. Stabilizing cpu frequency and voltage for temperature-aware dvfs in mobile devices. *IEEE Transactions on Computers*, v. 64, n. 1, p. 286–292, 2015.

KROAH-HARTMAN, G. *Linux Kernel in a Nutshell.* [S.l.]: O'REILLY, 2006.

KUHN, M.; JOHNSON, K. *Applied Predictive Modeling.* [S.l.]: New York: Springer, 2013.

LAWRENCE, S.; GILES, C. L.; TSOI, A. C. *What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation.* College Park, MD 20742, 1996. 35 p.

LI, S.; AHN, J. H.; STRONG, R. D.; BROCKMAN, J. B.; TULLSEN, D. M.; JOUPPI, N. P. Mcpat: An integrated power area and timing modeling framework for multicore and manycore architectures. In: *Proc. International Symposium on Microarchitecture (MICRO).* [S.l.: s.n.], 2009. p. 469–480.

LI, X.; YAN, G.; HAN, Y.; LI, X. Smartcap: User experience-oriented power adaptation for smartphone's application processor. In: *Proc. of the Conference on Design, Automation and Test (DATE).* [S.l.: s.n.], 2013. p. 57–60.

MAEDA-NUNEZ, L. A. *System-Level Power Management using Online Machine Learning for Prediction and Adaptation.* Tese (PhD thesis) — Faculty of Physical Sciences and Engineering Electronics and Computer Science, 2016.

MAEDA, R. K. V.; YANG, P.; WU, X.; WANG, Z.; XU, J.; WANG, Z.; LI, H.; DUONG, L. H. K.; WANG, Z. Jade: a heterogeneous multiprocessor system simulation platform using recorded and statistical application models. In: *Proc. International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems.* [S.l.: s.n.], 2016. p. 1–6.

MAHESRI, A.; VARDHAN, V. Power consumption breakdown on a modern laptop. *Lecture Notes in Computer Science (LNCS)*, v. 3471, n. 12, p. 165–180, 2005.

MARCULESCU, D. Application adaptive energy efficient clustered architectures. In: *Proc. of the International Symposium on Low Power Electronics and Design (ISLPED)*. [S.l.: s.n.], 2004. p. 344–349.

MAUERER, W. *Professional Linux kernel architecture*. [S.l.]: Wiley, 2010.

MCCULLOCK, J. *Q-learning tutorial*. 2013. Disponível em: <http://mnemstudio.org/path-finding-q-learning-tutorial.htm>.

MCVOY, L.; STAELIN, C. lmbench: portable tools for performance analysis. In: *Proc. conference on USENIX Annual Technical Conference*. [S.l.: s.n.], 1996. p. 23.

MIFTAKHUTDINOV, R.; EBRAHIMI, E.; PATT, Y. N. Predicting performance impact of dvfs for realistic memory systems. In: *Proc. of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*. [S.l.: s.n.], 2012. p. 155–165.

MNEMSTUDIO.ORG. *Path finding Q-learning tutorial*. 2018. Disponível em: <http://mnemstudio.org/path-finding-q-learning-tutorial.htm>.

MOENG, M.; MELHEM, R. Applying statistical machine learning to multicore voltage and frequency scaling. In: *Proc. ACM International Conference on Computing Frontiers*. [S.l.: s.n.], 2010. p. 277–286.

NIROOMAND, M.; FOROUGHI, H. R. A rotary electromagnetic microgenerator for energy harvesting from human motions. *Journal of Applied Research and Technology*, v. 14, p. 259–267, July 2016.

PALLIPADI, V.; STARIKOVSKIY, A. The ondemand governor. In: *Proc. Linux Symposium*. [S.l.: s.n.], 2006. p. 215–230.

PASRICHA, S.; DONOHOO, B. K.; OHLSEN, C. A middleware framework for application-aware and user-specific energy optimization in smart mobile devices. *Pervasive and Mobile Computing*, v. 20, n. C, p. 47–63, July 2015.

POPLI, S.; JHA, R. K.; JAIN, S. A survey on energy efficient narrowband internet of things (nbiot): Architecture, application and challenges. *IEEE Access*, v. 7, p. 16739–16776, 2018.

PRABHA, V. L.; MONIE, E. C. Hardware architecture of reinforcement learning scheme for dynamic power management in embedded systems. *EURASIP Journal on Embedded Systems*, v. 2007, p. 1–6, 2007.

REY, D.; NEUHAUSER, M. Wilcoxon-signed-rank test. *International Encyclopedia of Statistical Science*, p. 1658–1659, 2011.

ROUNTREE, B.; LOWENTHAL, D. K.; SCHULZ, M.; SUPINSKI, B. R. de. Practical performance prediction under dynamic voltage frequency scaling. In: *Proc. of the International Green Computing Conference and Workshops (IGCC)*. [S.l.: s.n.], 2011. p. 1–8.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: a modern approach*. [S.l.]: Prentice Hall, 2003. 269-272 p.

SAEZ, J. C.; PRIETO, M.; FEDOROVA, A.; BLAGODUROV, S. A comprehensive scheduler for asymmetric multicore systems. In: *Proc. of the 5th European Conference on Computer Systems (EUROSYS)*. [S.l.: s.n.], 2010. p. 139–152.

SALZMAN, P. J. *The Linux Kernel Module Programming Guide*. [S.l.]: Peter Jay Salzman, 2007.

SEEKER, V. *User Experience Driven CPU Frequency Scaling On Mobile Devices Towards Better Energy Efficiency*. Tese (PhD thesis) — School of Informatics at University of Edinburgh, 2017.

SEMERARO, G.; MAGKLIS, G.; BALASUBRAMONIAN, R.; ALBONESI, D. H.; DWARKADAS, S.; SCOTT, M. L. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In: *Proc. of the 8th International Symposium on High-Performance Computer Architecture (HPCA)*. [S.l.: s.n.], 2002. p. 29–.

SHAFIK, R. A.; YANG, S.; DAS, A.; MAEDA-NUNEZ, L. A.; MERRETT, G. V.; AL-HASHIMI, B. M. Learning transfer-based adaptive energy minimization in embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, v. 35, n. 6, p. 877–890, 2016.

SHEN, H.; TAN, Y.; LU, J.; WU, Q.; QIU, Q. Achieving autonomous power management using reinforcement learning. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, v. 18, n. 2, p. 24–32, 2013.

SINHA, S.; SUH, J.; BAKKALOGLU, B.; CAO, Y. Workload-aware neuromorphic design of the power controller. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, v. 1, n. 3, p. 381–390, Sep. 2011.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. [S.l.]: MIT Press, 2012.

TALPES, E.; MARCULESCU, D. Toward a multiple clock/voltage island design style for power-aware processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 13, n. 5, p. 591–603, May 2005.

TARKOMA, S.; SIEKKINEN, M.; LAGERSPETZ, E.; XIAO, Y. *Smartphone Energy Consumption*. [S.l.]: Cambridge University Press, pp. 18-35, 2014.

TIAN, Z.; WANG, Z.; LI, H.; YANG, P.; MAEDA, R. K. V.; XU, J. Multi-device collaborative management through knowledge sharing. In: *Proc. 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. [S.l.: s.n.], 2018. p. 22–27.

TORCHIANO, M.; ARDITO, L.; PROCACCIANTI, G.; MIGLIORE, G. Profiling power consumption on mobile devices. In: *Proc. Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*. [S.l.: s.n.], 2013. p. 101–106.

TUTOR, P. *Power Tutor Software*. 2015. Disponível em: <http://ziyang.eecs.umich.edu/projects/powertutor/>.

VENKATACHALAM, V.; FRANZ, M. Power reduction techniques for microprocessor systems. *ACM Computing Surveys*, v. 37, n. 3, p. 195–237, Sep 2005.

VOGELEER, K. D.; MEMMI, G.; JOUVELOT, P.; COELHO, F. The energy/frequency convexity rule: Modeling and experimental validation on mobile devices. *Parallel Processing and Applied Mathematics*, v. 1, n. 1, p. 793–803, 2014.

WALKER, M. J.; DIESTELHORST, S.; HANSSON, A.; DAS, A. K.; YANG, S.; AL-HASHIMI, B. M.; MERRETT, G. V. Accurate and stable run-time power modeling for mobile and embedded cpus. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 36, n. 1, p. 106–119, Jan. 2017.

WANG, Z.; TIAN, Z.; XU, J.; MAEDA, R. K. V.; LI, H.; YANG, P.; WANG, Z.; DUONG, L. H. K.; WANG, Z.; CHEN, X. Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system. In: *Proc. 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. [S.l.: s.n.], 2017. p. 684–689.

WATKINS, C. J. C. H. *System-Level Power Management using Online Machine Learning for Prediction and Adaptation*. Tese (PhD thesis) — Cambridge University, 1989.

WEISER, M.; WELCH, B.; DEMERS, A.; SHENKER, S. Scheduling for reduced cpu energy. In: *Proc. of the 1st USENIX Conference on Operating Systems Design and Implementation*. [S.l.: s.n.], 1994. p. 101–106.

WESTE, N. H. E.; HARRIS, D. M. *CMOS VLSI Design: A Circuits and Systems Perspective*. [S.l.]: Pearson, 2011.

WIERING, M.; OTTERLO, M. van. *Reinforcement Learning: State-of-the-Art*. [S.l.]: Springer-Verlag, 2012. 31 p.

XIAN, C.; LU, Y.-H.; LI, Z. Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In: *Proc. of the 44th Annual Design Automation Conference (DAC)*. [S.l.: s.n.], 2007. p. 664–669.

YADAV, S.; SHUKLA, S. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In: *Proc. 6th International Advanced Computing Conference*. [S.l.: s.n.], 2016. p. 78–83.

YUAN, W.; NAHRSTEDT, K. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In: *Proc. of the Nineteenth ACM Symposium on Operating Systems Principles*. [S.l.: s.n.], 2003. p. 149–163.

ZHU, X.; GOLDBERG, A. B. Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, v. 3, n. 1, p. 1–130, 2009.