Received 18 December 2023, accepted 27 January 2024, date of publication 1 February 2024, date of current version 8 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3361317

RESEARCH ARTICLE

A Resource Aware Memory Requirement Calculation Model for Memory Constrained Context-Aware Systems

MUMTAZ ALI[1], MUHAMMAD ARSHAD[1], IJAZ UDDIN[1], GAUHAR ALI[2], MUHAMMAD ASIM[2,3], AND MOHAMMED ELAFFENDI[2]
[1]Department of Computer Science, City University of Science and Information Technology, Peshawar 25000, Pakistan
[2]EIAS Data Science and Blockchain Laboratory, College of Computer and Information Sciences, Prince Sultan University, Riyadh 11586, Saudi Arabia
[3]School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China

Corresponding author: Mumtaz Ali (mumtazali@cusit.edu.pk)

ABSTRACT Smart spaces are physical environments equipped with sensors, actuators, and other computing devices to gather data and provide intelligent services to users. These spaces are made possible by ubiquitous computing, particularly context-aware computing. Although these systems are mainly implemented on mobile and other resource-constrained wearable devices, different techniques have been adopted for their implementation. Rule-based reasoning is a relatively easy-to-implement approach that can solve real-world problems. Rule-based systems rely on a set of assertions that constitute the working memory and a set of rules that govern what should be done with the set of assertions. Despite its relative simplicity, the working memory size is a critical factor in developing these systems, particularly for resource-constrained devices. In this paper, we propose techniques for efficiently calculating the working memory size. Our results show that all three techniques, DWM, APS, and SAPS, performed well in different ways. However, APS and SAPS consumed from 25% to 100% less memory than existing techniques.

INDEX TERMS Context-aware systems, rule-based reasoning, working memory.

I. INTRODUCTION

A Rule-Based System (RBS) is a powerful computing system frequently employed in numerous artificial intelligence applications. It is designed to incorporate expert knowledge in the form of rules, closely mimicking human reasoning and thinking [1]. Numerous studies have supported the RBS as a viable alternative to human thinking and problem-solving. While facts are thought to function as short-term memory, rules behave as long-term memory [2], [3], [4]. Rules are combined to form a knowledge base, each containing a small amount of knowledge. The context-aware technique is used to make the computer aware of its surroundings, just as a human can perceive its environment through his or her senses of sight, touch, smell, and hearing. A device with several sensors attached to it can sense its surroundings and provide the rule engine with this context in the form of facts. This enables the system to react to changes made to its surroundings or as intended otherwise. A complete system with reasoning and sensing capabilities is created when rule-based and context-aware systems are combined [5], [6]. Figure 1 illustrates the Rule-based system, where multiple components work together as a single unit. The inference engine and memory are the two essential parts. Logically, the rules and facts are stored in memory. The area where the facts are kept is conceptually further split into two parts: dynamic and static. The static portion of the RBS keeps the initial facts, which are essential for any system to start up and cannot be withdrawn or changed, while the dynamic portion of the RBS stores newly derived contexts. The work by [7] has a detailed discussion and further information about the RBS and its environment.

The associate editor coordinating the review of this manuscript and approving it for publication was Yichuan Jiang.

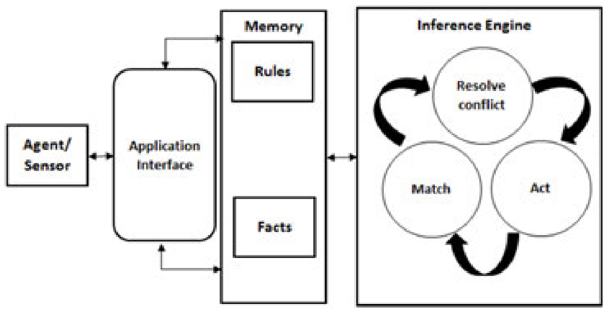


FIGURE 1. Rule engine components logical connection [8].

Naturally, we cannot retain contexts as they enter and as frequently as they arise when the system is resource constrained, mainly when the working memory is relatively minor. As a result, we must establish algorithms/standards to regulate this situation. The present study has further refined the previous models of context-aware systems [8], which were developed for resource-constrained devices using rule-based systems based on the logical framework [9]. The improvement in this research lies in developing new calculation methods for memory allocation. To the best of our knowledge, the existing techniques lack any such computation method. Most of them are RETE-based, and the authors of [9], who introduced the preference sets-based methods, claim the memory randomly without any specific memory size calculation method.

This paper is an extended version of our previous published work [10], which is on an abstract level and only proposed the idea; there was no algorithm design and no implementation. This paper has properly designed the algorithms and their implementation on two different systems. The paper proposes three possible solutions for the same problem in three different scenarios, which are (i) Distinct Working Memory (DWM), (ii) Average of the Preference Sets (APS) and (iii) Smart Average of the Preference Sets (SAPS). The DWM is recommended for situations where time is more constrained as compared to space. APS and SAPS are similar in space requirements; however, SAPS is more time-efficient when the diversity in the sizes of preference sets is less than the threshold provided.

The rest of this paper is organized as follows: Section II provides a review of relevant literature. Section III outlines the fundamental components of rule-based systems and multi-agent rule-based reasoning. Section IV presents an approach to modeling smart space systems. Section V delves into the management of working memory and presents proposed algorithms for calculating its size. Section VI is about the results and their discussion. Lastly, Section VII summarizes the paper's key findings and suggests potential areas for future research.

II. RELATED WORK

In the early works in Pervasive and Ubiquitous Computing, context awareness is referred to as the location of the objects and/or people [11]. In recent years, the context has been extended to other factors, such as the social and physical aspects of an entity and the activities of the user [8], [12], [13]. Similarly, [11], [14], [15], [16], [17], [18], [19] evaluated the user's identity and social situation, as well as the user's location, environmental and/or temporal information, the objects and people in their immediate surroundings, as well as any changes to those objects. Furthermore, social networks are playing a significant role in this regard, as the users provide, or in the current smart world, these networks and their applications are intelligent enough to collect information like the user's preferences and their likes and dislikes. In other words, these applications can collect or provide different user information, like contextual information and behavioral activities towards the surroundings [20], [21]. These gadgets/applications are currently a good and prominent source of data regarding user preferences. An illustrative example is the SociaCircuit Platform [22], which aids in monitoring diverse social interactions among users and subsequently modifies their preferences. The authors of [23] used data mining tools and techniques to monitor user interaction. In [24], an academic advisor expert system is developed. It is a monotonic system that returns the same answer for the same input every time. The interface is connected with a rule set, and there is no capacity to run a different rule set. Sociometric badges tracked and evaluated employee activity patterns over time [25] in an organization. Using this information, they predicted the employee's job satisfaction and the effectiveness of employee interactions within an organization. Similarly, in [26], they monitor the users' activities on mobile sensors like call logs, a location visited, etc. Based on this monitoring information, they further tried to find significant locations based on social activities and other relevant information.

Significant efforts have been undertaken to introduce mature expert systems to the Android platform, as highlighted in the book "Build Android-Based Smart Applications" [27]. The book provides comprehensive coverage of rule engines that can be effectively employed on the Android platform. It offers in-depth insights into the functionality and utilization of these rule engines within Android-based applications. However, these rule engines have limitations in terms of context awareness, resource efficiency, and dynamic context utilization with preferences. The authors of the book also identified additional limitations with these engines. For instance, certain rule engines such as Jruleengine and Zilonis do not support the usage of OR operators; Termware necessitates writing rules in code, making the process of updating them challenging; and Roolie mandates each rule to be coded in a separate file, which is cumbersome and impractical for larger systems.

During porting various rule engines, technical difficulties were encountered. Eclipse faced memory constraints while converting files to Dalvik format due to the memory-intensive nature of Drools. Take necessitates a Java compiler at runtime, and JLisa encounters a stack overflow error on Android. Jess, although costly and incompatible with Android, consumes excessive memory resources.

Table 1 shows that most rule-based engines developed are RETE algorithm-based. RETE is a well-known algorithm in the area of rule-based engines. However, according to the literature, it is a memory-intensive algorithm unsuitable for memory-constrained devices. RETE loads all the rules in the memory at the very start of the application's loading time and checks them one by one to see if they should be fired [28]. In addition, it is a time-consuming job to check each and every rule for each instance. Owing to this, the authors of [28] introduced preferences. They divide the whole rule base into preference sets and load only those rules that are in the preference set. Although at loading time, it works very efficiently due to preference sets, over time, it does not work with the same efficiency as it does not remove the already loaded rule sets. Similarly, [7] does not have any systematic way for memory allocation. They claim a fixed-size memory for execution that may lead to some issues. Like both in [7] and [28], there is no technique to remove the already loaded rules from the memory once it reaches its limits. They randomly remove rule(s) that can be critical and may soon be required for execution. This scenario can lead the system into an infinite loop.

III. RULES-BASED MULTI-AGENT SYSTEMS

Multi-agent systems are decentralized systems designed to solve comparatively challenging problems compared to centralized systems. Multi-agent systems make it possible to divide the problem into sub-parts, and the structure/organization of agents imitate human interaction in an organization. Every agent has responsibilities and sufficient autonomy to perform the specific task allotted and communicate with other agents. The primary goal is to work, act smartly like a human, and smartly perform the task.

An agent is a software system or device (a sensor or robot) composed of perception and reasoning that acts accordingly. Simply speaking, agents are designed and programmed to solve a specific problem(s). Similarly, multi-agent systems are designed and programmed to solve complex problems. The author contends in [29] that it enables the development of a more natural and human-like process to solve an issue. This also allows agents to be more autonomous in their tasks and to change physical space into a smart and interactive environment with a decentralized structure.

Agents that are designed to implement a rule-based application are known as rule-based agents. Various authors have proposed a variety of smart rule-based agents [30], [31], [32], [33], [34], [35]. The behaviour of the proposed systems was smart as compared to the traditional agents, where the behaviour of the agents is fixed and works in a fixed environment. Moreover, rule-based agents are dynamic and are created and modeled so that they can acquire information from their surroundings, reason about it, and alter their behaviour accordingly.

IV. MODELING SMART SPACE SYSTEMS

It has long aimed to create intelligent, autonomous, and adaptable systems that function in complex and dynamic environments. Discussion and studies on this subject have evolved in recent years [36], [37], [38], [39], [40]. An interoperable, heterogeneous environment is offered by a smart space for users, devices, and services to communicate in. Users are finally given unobtrusive support based on contextual information in such a scenario [41]. As numerous different hardware and software components are present in smart spaces, interoperability is essential. The receiving device must accurately translate the data into the form that the transmitting device intended. Semantics Web technology is one of the techniques to accomplish this task [42], especially for mobile and embedded devices. However, one must keep in mind the resource limitations and unique characteristics of both devices.

Smart spaces [43] rely heavily on contextually aware systems, and context modeling is a crucial first step in creating such systems. Context refers to any information, whether it be physical or conceptual, that can be employed in the process of determining the state of an entity. An entity can be a person, a place, a physical thing, or a computational object. Entities can also be abstract concepts. This context reflects the connection between a user and an application [44]. A smart space provides a service to its users by sensing and analyzing the scenario in which the users are currently located, determining the demands of the users, and supplying the necessary functionality based on the resources that are accessible during that situation. "Ontology-based context modeling and rule-based context reasoning are two of the most common approaches utilized to facilitate semantic interoperability and analyze user context in smart spaces" [7]. The process of getting context entails gathering information about the context in its raw form from a diverse range of available sensors. Moreover, the smart system may provide the user with the choice to manually input contextual information, allowing them to contribute to the context-gathering process [45]. An ontology that represents the generic concepts on a more advanced level is required in order to model a context and make it applicable to any domain. The context model needs to be able to provide hierarchical frameworks for enhancing specific context information. A standardized language provided by context ontology can be used to express knowledge about a domain and describe particular scenarios that take place within a domain [46]. Contextual reasoning's goal is to create higher-level contexts from perceived low-level contexts. Contextual reasoning also aims to provide new pertinent information for users and applications from many context-data sources [10]. This is the primary stage, which is mostly in charge of determining what's happening in a smart environment and how to assist its users [47]. The OWL is a descriptive ontology-based language. The semantic web researchers [48] described the web ontology language (OWL) as a semantic markup language for ontologies with formal

TABLE 1. Comparison of various rule engines.

Name	Generic	Context Aware	For Resource Bounded Devices	Native	RETE Based	Porting Issue	Context Reusability
Clips	YES	NO	NO	NO	YES	YES	NA
Drools	YES	NO	NO	NO	YES	YES	NA
FMES	NO	NO	NO	YES	NO	NA	NO
Jlisa	YES	NO	NO	NO	NO	YES	NA
JruleEngine	YES	NO	NO	NO	YES	YES	NA
JEOPS	YES	NO	NO	NO	YES	YES	NA
KBAM	NO	YES	NO	YES	NO	NA	NO
SociaCircuit	NO	NA	NO	NO	NA	NA	YES
Zilionis	YES	NO	NO	NO	YES	YES	NA
Proposed by [28]	YES	YES	YES	NO	NO	NO	YES
TermWare	YES	NO	NO	NO	YES	NO	NO

syntax and semantics. OWL 1 and 2 are W3C standards. OWL 1 also comes in Lite, DL, and Full flavours. The descriptive logic-based OWL DL is more expressive than OWL Lite. OWL fully expresses more. The authors define OWL 2 as a more efficient and manageable reasoning standard. There are a total of four subsets of OWL 2 within the DL subset; these are EL, QL, and RL. The authors define rules and ontologies in their work using the OWL 2 RL and SWRL languages. Both OWL 2 RL and SWRL [49] are applicable for context modeling and reasoning. It has been shown that OWL 2 RL is a good fit for developing rule-based systems, which can then be implemented with the help of rule-based reasoning engines [9]. A set of Horn clause rules can be created using an OWL 2 RL ontology, claims [50]. SWRL enables us to build rules using OWL principles, even for more complex rule-based concepts. According to our method, a context-aware system consists of a collection of rule-based agents and a set of rules that have the potential to infer new information to represent the system's behaviour and predict how the context will change. Additional information on this method is available in [51]. The rule-based reasoning method and working memory management and updating will be the main focus in the subsequent.

V. WORKING MEMORY MANAGEMENT

As mentioned earlier, working memory has been divided into two parts. Each part carries out its specific function. The significance of dynamic working memory is the primary focus of this paper. A context that has just been created can be added to the dynamic working memory by firing a rule instance or a context that has been received as a message from another device or agent. Before proposing novel techniques to replace the existing framework for efficient working memory utilization, we must analyze our current working mechanisms to apply this new technique or methodology.

A. UPDATING WORKING MEMORY

The working memory functions as a repository for current contexts, enabling context-aware reasoning. Given the limited availability of memory, it becomes a crucial resource during system design and implementation. To ensure efficient usage, we impose a size restriction on the working memory, preventing it from exceeding its capacity to hold contexts at any given time. However, new contexts can emerge with each iteration, and it is vital to preserve the most essential ones for execution. Our approach modifies working memory as a fixed-size container with static and dynamic components. Each memory unit in the dynamic portion is just large enough to store one context. When the agent's memory is full or a contradictory context presents itself, only facts currently recorded in the dynamic memory can be overwritten. Conflicts are resolved even if the memory is not fully utilized by swapping out the conflicting context for the new one. Conflicts are found by comparing newly added contexts with those already present. If there is no conflict, the new context is introduced to the working memory by replacing a context chosen randomly if the memory is already full. If there is an unreachable goal and no means to halt the inference engine, the system may continue running indefinitely because the dynamic memory has limited capacity. For instance, if there is just one memory unit and rule r1 generates a context that can activate rule r2, and vice versa, the system will run unabated until it is stopped. We set the number of iterations to equal the number of rules to address this problem. This ensures that every rule is tested, and the system stops instead of acting suddenly if no matches are discovered. This approach also conserves the resources of the host system.

B. WORKING MEMORY SIZE ESTIMATION

Estimating working memory size can help minimize context loss, particularly when critical information needs to be retained for further processing or decision-making. Here are some techniques that can be used to estimate working memory size.

1) DISTINCT WORKING MEMORY(DWM)

The "distinct" operator in a database eliminates duplicate values so that each value appears only once. Similarly, when determining the working memory size, it is sufficient to consider the number of unique consequences of the rules rather than every occurrence of a consequence described in Algorithm 1. Let RB represent the total number of rules in the rule base, RC represents the total number of consequences of these rules, and RDC represents the number of rules with distinct consequences, where RDC is less than or equal to

RC and greater than zero ($RDC \leq RC$ and $RDC > 0$). The appropriate size for the working memory can then be determined as RDC.

Algorithm 1 Distinct Working Memory (DWM)

```
Require: [$RB$: Rule-base, $RC$: Rule Consequent, $RDC$: Rule
    Distinct Consequent, $RS$: Array to hold Rule Set, $DRS$:
    Array to hold Distinct Rule Set]
Ensure: [$RWM$: Required Working Memory Size, $TPM$:
    Time for Performance Measurement]
    START
 1: start-TPM to start measuring the time
 2: create RS
 3: create DRS
 4: $RS \longleftarrow$ Push all the rules from $RB$
 5: for all each Rule in $RS$ do
 6:     if DRS array does not include RC then
 7:         $DRS \longleftarrow$ Push Rule
 8:     end if
 9: end for
10: $RWM \longleftarrow$ size of DRS
11: end-TPM
12: $TPM =$ start-TPM $-$ end-TPM
13: return $RWM, TPM$
    END
```

2) AVERAGE OF THE PREFERENCE SETS (APS)

This technique considers sets of preferences to personalize resource-limited context-aware applications based on user preferences. While implementing preferences can be more complex with this method, it is space-saving as it considers the rules of several different preference groups. By considering the overall average of rule base sizes, Algorithm 2 uses a method to determine the average working memory size. The number of preference sets with working memory (WM) requirements greater than the estimated average is then calculated. The second stage involves checking these preference sets for distinctive values in their consequent parts. The system will seek a WM size equivalent to the estimated average if these preference sets tested for WM have fewer distinct consequent parts than average. In contrast, if the number is higher than average, the system will ask for a WM size equivalent to that amount. To avoid losing crucial context, the greatest determined value in both cases is chosen as the WM size. Suppose we have a rule base R with n rules and m preference sets P1, P2,..., Pm with varied preference methods. In that case, the size of the working memory will be $\frac{(|P_1|+|P_2|+\cdots+|P_m|)}{m}$, where $|P_i|$ is the size of the preference set Pi for $1 \leq i \leq m$. The memory size may be larger than the calculated average if any preference set requires more memory, as discussed above.

3) SMART AVERAGE OF THE PREFERENCE SETS (SAPS)

The SAPS technique, outlined in Algorithm 3, focuses on preference sets and shares similarities with APS. However,

Algorithm 2 Average of the Preference Sets (APS)

```
Require: [RB: Rule-base, RC: Rule Consequent, RDC:
    Rule Distinct Consequent, RS: Array to hold Rule Set,
    DRS: Array to hold Distinct Rule Set, NPS: Number
    of Preference Set, PS: Preference Set, APS: Average
    of Preference sets, PSHR: Preference Set with Highest
    Number of Rules]
Ensure: [RWM: Required Working Memory Size, TPM:
    Time for Performance Measurement]
    START
 1: start-TPM to start measuring the time
 2: $NPS \longleftarrow$ Check RB for the number of preference sets
 3: $PS[i] \longleftarrow$ Check RB for the number of rules in each
    preference set
 4: $Average \longleftarrow$ Calculate the average/mean of the prefer-
    ence sets
 5: if (Average < PSHR) then
 6:     Create DRS
 7:     for each Rule in PSHR do
 8:         if RS does not include RC then
 9:             Push rule to DRS
10:         end if
11:     end for
12: end if
13: if Number of rules in DRS > Average then
14:     $RWM \longleftarrow$ DRS
15: else
16:     $RWM \longleftarrow$ Average
17: end if
18: end-TPM
19: $TPM =$ start-TPM $-$ end-TPM
20: return RWM, TPM
    END
```

it incorporates certain conditions that can optimize computation. The first step involves calculating the available preference sets' standard deviation (SD). A small SD suggests values closely clustered around the mean, while a large SD indicates a wider spread of values.

The system will select the preference set from the available sets with the most memory units when the SD is low (less than a predetermined threshold value of 2). Since the size of the chosen preference set is sufficient for all other preference sets in this situation, no additional calculations are required.

Conversely, when the SD is large (greater than or equal to the threshold value of 2), the technique operates in a manner similar to APS, considering the individual characteristics of each preference set.

4) TIME COMPLEXITY OF THE ALGORITHMS

The time complexity of an algorithm is a measure of how long it takes to run as a function of the input size. This metric provides an upper limit on the algorithm's running time, particularly in a worst-case scenario. For DWM, APS, and SAPS, the time complexity is O(n) each.

Algorithm 3 Smart Average of the Preference Sets (SAPS)

```
Require: [RB: Rule-base, RC: Rule Consequent, RDC: Rule Distinct Consequent, RS: Array to hold Rule Set, DRS: Array to hold Distinct Rule Set, NPS: Number of Preference Set, PS: Preference Set, SDPS: Standard Deviation of Preference sets, APS: Average of Preference sets, PSHR: Preference Set with Highest Number of Rules]
Ensure: [RWM: Required Working Memory Size, TPM: Time for Performance Measurement]
    START
 1: start-TPM to start measuring the time
 2: $NPS \longleftarrow$ Check RB for the number of preference sets
 3: $PS[i] \longleftarrow$ Check RB for the number of rules in each preference set
 4: $SDPS \longleftarrow \sqrt{\frac{1}{\text{NPS}}\sum_{i=1}^{\text{NPS}}(\text{PS}[i]-\text{APS})^2}$
 5: if (SDPS $\leq$ 2) then
 6:     $RWM \longleftarrow$ size of PSHR
 7: else
 8:     $RWM \longleftarrow$ Size calculated through APS
 9: end if
10: end-TPM
11: $TPM$ = start-TPM − end-TPM
12: return RWM, TPM
    END
```

VI. EXPERIMENTAL RESULTS

The proposed algorithms have been evaluated using three different rule sets, and the overall combination of the three rule sets "Smart Home, Smart Patient, Smart Office and Overall" developed by the authors of [28]. We combine all the rules as it can be a multi-agent smart solution and check the algorithm on the maximum number of rules available in the rule sets selected. The rule sets comprise a total of 153 rules. The algorithms were evaluated, and results were generated on two different systems. Each result is averaged over 10 iterations for accuracy. The specification of the first system is described in Table 2, and the second one is shown in Table 3.

A. TIME AND SPACE REQUIREMENT

The results generated on both systems are discussed in the subsequent.

1) RESULTS GENERATED ON CORE I5 SYSTEM:

Table 2 shows the performance results of the three distinct methods: DWM, APS, and SAPS on an Intel Core i5-4570 system with a CPU 3.20 GHz and 8 GB of RAM for the following rule sets: Smart Home, Smart Patient, Smart Office, and Overall.

Based on the data in Table 2, it is evident that the DWM approach routinely outperforms APS and SAPS in terms of execution time (in milliseconds (ms)). In the Smart Patient category, for example, DWM completes tasks in 1.984 ms, whereas APS and SAPS take 2.58 ms and 2.998 ms, respectively. DWM leads APS and SAPS in the Overall category, with completion times of 1.998 ms, 2.999 ms, and 2.999 ms, respectively. If we observe the average run-time of the three methods, DWM has the lowest average run-time, 1.741 ms, APS follows closely behind with an average run-time of 2.319 ms, while SAPS has the most extended average run-time of 2.495 ms.

When it comes to memory usage (in kilobytes (KB)), DWM uses more memory than APS and SAPS in almost every category. In the Smart Home category, for example, DWM uses 0.22 KB, whereas APS and SAPS both utilize 0.136 KB. The Smart Office category has the largest memory use, with DWM consuming 0.46 KB and APS and SAPS each consuming 0.176 KB. On average, DWM consumed 0.25 KB of memory. In contrast, the APS and SAPS requirements were only 0.147 KB each. These methods have a significant difference of almost 52% with DWM. The results and the difference in the result of the algorithms can also be observed in Figure 2.

2) RESULTS GENERATED ON RASPBERRY PI:

The results generated on Raspberry PI Model 3B are shown in Table 3. Time is measured in milliseconds (ms), and memory is measured in bytes. The result is similar in terms of memory requirement. That is DWM consumes the most memory in the Smart Home rules set at 188 bytes. APS and SAPS use a lesser amount of memory, each utilizing only 124 bytes. DWM consumes the most memory in the Smart Patient rules set as well, at 236 bytes. APS and SAPS are more efficient, each using 156 bytes. The major difference can be viewed in the Overall rules set, where DWM used 460 bytes, and APS and SAPS used only 156 bytes. On average, DWM has the highest memory requirement, which is 0.244 KB, and APS and SAPS memory requirements were found to be the same at 0.132 KB. In this case, the APS and SAPS consumed almost 60% less memory as compared to DWM.

There is a significant difference in the calculation running time between the methods as compared to the result generated on the core i5 system. That is DWM is the fastest approach in the Smart Home rules set, finishing tasks in 7.38 ms. APS and SAPS are both slower, taking 20.48 and 23.97 ms, respectively. Similarly, DWM maintains its speed lead in the Smart Patient rules set, with an execution time of 8.29 ms. At 25.16 ms and 30.55 ms, respectively, APS and SAPS are slower. Similar trends are found in the Smart Office and Overall rules sets. The average runtime for DWM is 8.115 ms, 24.096 ms for APS, and 28.520 ms for SAPS. The results can also be observed in Figure 3.

B. NUMBER-OF-RULES BASED RESULTS

The time and memory requirement is directly proportional to the number of rules required to run a system on a specific rule base. Table 5 provides information on the maximum number of rules required to run the system properly using five different rule-based systems/algorithms over four rule sets. This section will enhance the clarity and comprehensibility

TABLE 2. Results generated on core i5-4570.

Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz, 8 GB RAM												
	Smart Home			Smart Patient			Smart Office			Overall		
	DWM	APS	SAPS	DWM	APS	SAPS	DWM	APS	SAPS	DWM	APS	SAPS
Time (ms)	1.984	1.999	1.986	1.984	2.58	2.998	0.999	1.699	1.999	1.998	2.999	2.999
Space (KB)	0.22	0.136	0.136	0.22	0.176	0.176	0.1	0.14	0.1	0.46	0.176	0.176

TABLE 3. Results generated on raspberry PI model 3B.

Raspberry PI Model 3B with 1 GB RAM												
	Smart Home			Smart Patient			Smart Office			Overall		
	DWM	APS	SAPS	DWM	APS	SAPS	DWM	APS	SAPS	DWM	APS	SAPS
Time (ms)	7.38	20.48	23.97	8.29	25.16	30.55	6.43	13.27	11.59	10.35	37.48	47.98
Space (B)	188	124	124	236	156	156	92	92	92	460	156	156

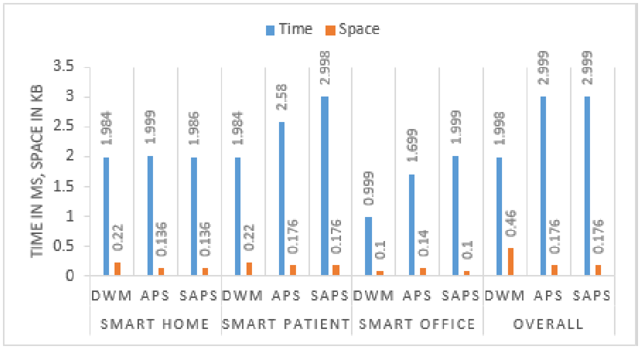


FIGURE 2. Results generated on core i5-4570.

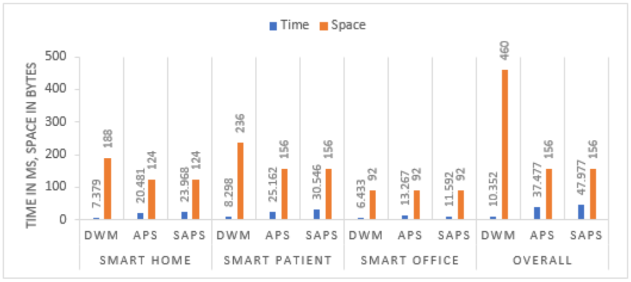


FIGURE 3. Results generated on raspberry PI model 3B.

of the results. In this table, the Rule Base (RB) column shows the total number of rules per rule set, and we consider it equivalent to RETE algorithm-based systems because the RETE algorithm loads all the rules in memory and compares them one by one during processing [8].

The Preference Set Based Method (PSBM) technique was introduced by the authors of [28]. This column shows the maximum number of rules required from the rule set based on preference sets. Both RB and PSBM methods do not have any specific method for calculating the required memory for rules. The former loads all rules into memory, while the latter requests a random working memory size. The remaining three proposed methods calculate the working memory for the given rule sets.

For the "Smart Home" rule set, RB needs to load 47 rules at maximum, the PSBM will load 21 rules, the DWM method needs space for 16 rules, APS and SAPS will need space for 17 rules each to operate the system on the same rule set. The requirement of the other rule sets can also be observed in Figure 4.

On average, DWM requires 44 rules, while APS and SAPS require 22 rules each. RB and PSBM have an average of 77 and 28 rules, respectively. APS and SAPS performed well and required space for less number of rules to operate a specific rule set on a given resource-constrained device as compared to the other three methods.

In terms of the "overall" number of rules, RB has the highest number with 153, followed by DWM with 97, PSBM with 40, and APS and SAPS with 30 each.

C. THE PERCENTAGE DIFFERENCE

The percentage difference in both the memory and time requirements are discussed hereunder.

1) MEMORY-WISE PERCENTAGE DIFFERENCE:

Based on the percentage differences in required memory space, it appears that APS and SAPS performed better than DWM, PSBM, and RB for each of the rule sets evaluated. Specifically, APS and SAPS required less space for the same rule set than the other methods. While DWM outperformed RB, it did not perform well as compared to PSBM. This can be observed in Figure 5.

On average, the percentage difference between APS and SAPS with RB was 100%, indicating that these methods required only half as much memory as RB. The percentage difference with PSBM is 24.5% and 65% with DWM.

2) TIME-WISE PERCENTAGE DIFFERENCE:

The results obtained on the Core i5, as illustrated in Figure 2, exhibit notable percentage differences. Firstly, when comparing DWM with APS, on average, there is a significant gap of 29.57%. This means that DWM operates approximately 29.57% faster than APS. Similarly, on average, DWM showcases a time-saving advantage of 35.85% when compared to SAPS. There is a discernible

TABLE 4. The percentage difference.

Comparative Overview - Percentage Differences in Results												
	Smart Home			Smart Patient			Smart Office			Overall		
	DWM	APS	SAPS	DWM	APS	SAPS	DWM	APS	SAPS	DWM	APS	SAPS
RB	98.41	93.75	93.75	61.53	95.65	95.65	27.03	80	62.5	44.8	134.42	134.42
PSBM	27.03	21.05	21.05	-11.76	28.57	28.57	-37.04	20	0	-83.21	28.57	28.57

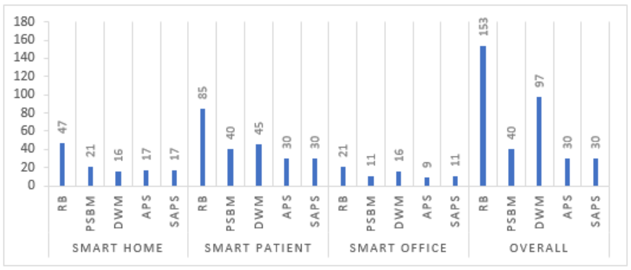
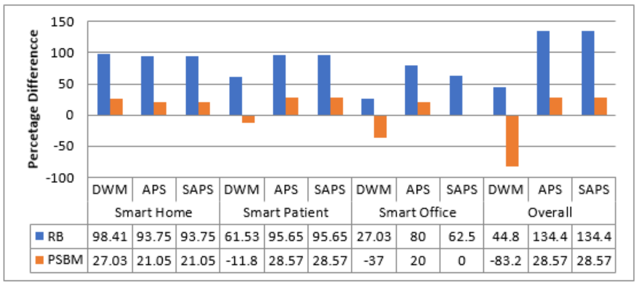


FIGURE 4. Number of rules results (Rule-sets-wise).



	DWM	APS	SAPS	DWM	APS	SAPS	DWM	APS	SAPS	DWM	APS	SAPS
	Smart Home			Smart Patient			Smart Office			Overall		
■ RB	98.41	93.75	93.75	61.53	95.65	95.65	27.03	80	62.5	44.8	134.4	134.4
■ PSBM	27.03	21.05	21.05	-11.8	28.57	28.57	-37	20	0	-83.2	28.57	28.57

FIGURE 5. Percentage difference in results.

TABLE 5. Number of rules-based results.

Rule Based System	Smart Home	Smart Patient	Smart Office	Overall
RB	47	85	21	153
PSBM	21	40	11	40
DWM	16	45	16	97
APS	17	30	11	30
SAPS	17	30	9	30

difference of 7.47% between APS and SAPS. This suggests that APS and SAPS present variations in their performance levels, with APS demonstrating a 7.47% advantage over SAPS.

Likewise, the results obtained on the Raspberry Pi, as depicted in Figure 3, reveal a significant average percentage difference. When comparing DWM with APS, the percentage difference is remarkable at 99.19%. This indicates that DWM operates nearly twice as fast as APS, resulting in a substantial reduction in processing time. Furthermore, DWM showcases an even greater advantage of 111.35% over SAPS, implying that DWM completes tasks in approximately 111.35% less time compared to SAPS. Additionally, the average difference between APS and SAPS amounts to 16.80%.

The analysis of time-wise percentage differences reveals that, on average, APS holds an advantage over SAPS. However, a closer examination of Figures 2 and 3 reveals an interesting observation in the "Smart Office" rule set: under specific conditions favouring SAPS, namely when the standard deviation is less than 2, SAPS outperforms APS in terms of processing time. In the case of the Raspberry Pi, SAPS demonstrates a time-saving advantage of 13.52%, while on the Core i5, SAPS exhibits a more significant advantage of 33.33%. This suggests that SAPS can be particularly efficient and faster than APS in specific scenarios with lower standard deviations.

In summary, the study evaluated five rule-based systems across four different rule sets. The results showed that the DWM method had the highest memory requirement, while APS and SAPS required significantly less memory. However, DWM had the shortest calculation runtime, outperforming APS and SAPS. Regarding the overall number of rules, RB had the highest number, followed by DWM, PSBM, and APS/SAPS. The individual rule-set categories showed that RB had the highest number of rules in the Smart Home and Smart Office categories, while PSBM had the highest number in Smart Patient. On average, APS and SAPS required from 25% to 100% less memory compared to RB and PSBM, indicating that they are more memory-efficient, can be observed in Table 4. Both the APS and SAPS generate almost similar results. On the overhead of standard deviation calculation, SAPS can save time by abstaining from additional processing/checks for distinct rules within the preference set. This situation can be observed in Figure 3 in the Smart Office rule set.

Overall, the study provides valuable insights into the performance of different rule-based systems and methods for working memory calculation, which can help guide the selection of appropriate systems for specific applications.

VII. CONCLUSION AND FUTURE WORK

The presented paper proposes a solution to the problem of calculating dynamic working memory size in resource-constrained devices. The proposed solution enables the calculation of dynamic working memory size for fixed-size rule sets, which is a critical problem in many intelligent systems. However, it is important to consider resource constraints while designing more complex solutions.

In the future, the authors intend to focus on the rules write-up and generation strategies to improve the efficiency of the proposed solution further. Even though the rules are well written, there is room for improvement in reducing the memory requirements for rule sets and making the process more efficient. The authors plan to explore new techniques and approaches to address these challenges, enabling the implementation of more complex and intelligent systems on resource-constrained devices. Likewise, there is potential for enhancing the functionality of the working memory updating process, and we plan to incorporate these improvements in the upcoming version of our paper.

ACKNOWLEDGMENT

The authors would like to thank Prince Sultan University and the EIAS Laboratory for their valuable support and also would like to thank Prince Sultan University for paying the article processing charges (APC) of this publication.

REFERENCES

[1] E. Lagun, "Evaluation and implementation of match algorithms for rule-based multi-agent systems using the example of jadex," M.S. thesis, Faculty Math., Comput. Sci. Natural Sci., Univ. Hamburg, Hamburg, Germany, 2009.

[2] J. C. Giarratano and G. Riley, *Expert Systems: Principles and Programming*. Boston, MA, USA: Thomson Course Technology, 2005.

[3] G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, 6th ed. USA: Pearson Education, 2016.

[4] K. Ahmed, A. Altaf, N. S. M. Jamail, F. Iqbal, and R. Latif, "ADAL-NN: Anomaly detection and localization using deep relational learning in distributed systems," *Appl. Sci.*, vol. 13, no. 12, p. 7297, Jun. 2023.

[5] J.-Y. Hong, E.-H. Suh, and S.-J. Kim, "Context-aware systems: A literature review and classification," *Expert Syst. Appl.*, vol. 36, no. 4, pp. 8509–8522, May 2009.

[6] S. Matalonga, D. Amalfitano, A. Doreste, A. R. Fasolino, and G. H. Travassos, "Alternatives for testing of context-aware software systems in non-academic settings: Results from a rapid review," *Inf. Softw. Technol.*, vol. 149, Sep. 2022, Art. no. 106937.

[7] I. Uddin, H. M. U. Haque, A. Rakib, and M. R. S. Rahmat, "Resource-bounded context-aware applications: A survey and early experiment," in *Proc. Int. Conf. Nature Comput. Commun.* Rach Gia, Vietnam: Springer, 2016, pp. 153–164.

[8] A. Rakib and I. Uddin, "An efficient rule-based distributed reasoning framework for resource-bounded systems," *Mobile Netw. Appl.*, vol. 24, no. 1, pp. 82–99, Feb. 2019.

[9] A. Rakib and H. M. U. Haque, "A logic for context-aware non-monotonic reasoning agents," in *Human-Inspired Computing and Its Applications*. Tuxtla Gutierrez, Mexico: Springer, 2014, pp. 453–471.

[10] I. Uddin, A. Rakib, M. Ali, and P. C. Vinh, "Memory-constrained context-aware reasoning," in *Proc. 10th EAI Int. Conf. Context-Aware Syst. Appl.* Ho Chi Minh City, Vietnam: Springer, 2021, pp. 133–146.

[11] B. N. Schilit and M. M. Theimer, "Disseminating active map information to mobile hosts," *IEEE Netw.*, vol. 8, no. 5, pp. 22–32, Sep. 1994.

[12] P. Dourish, "What we talk about when we talk about context," *Pers. Ubiquitous Comput.*, vol. 8, pp. 19–30, Dec. 2003.

[13] I. H. Sarker, "Context-aware rule learning from smartphone data: Survey, challenges and future directions," *J. Big Data*, vol. 6, no. 1, pp. 1–25, Dec. 2019.

[14] P. J. Brown, J. D. Bovey, and X. Chen, "Context-aware applications: From the laboratory to the marketplace," *IEEE Pers. Commun.*, vol. 4, no. 5, pp. 58–64, Oct. 1997.

[15] N. S. Ryan, J. Pascoe, and D. R. Morse, "Enhanced reality fieldwork: The context-aware archaeological assistant," in *Computer Applications in Archaeology*. Canterbury, U.K.: Univ. of Kent at Canterbury, 1998.

[16] P. J. Brown, "The stick-e document: A framework for creating context-aware applications," *Electronic*, vol. 8, pp. 259–272, Sep. 1995.

[17] A. Burinskiene, "Context-aware service support efficiency improvement in the transport system," in *Development of Smart Context-Aware Services for Cargo Transportation* (International Series in Operations Research & Management Science). Cham, Switzerland: Springer, 2022, pp. 179–227.

[18] A. Ward, A. Jones, and A. Hopper, "A new location technique for the active office," *IEEE Pers. Commun.*, vol. 4, no. 5, pp. 42–47, Oct. 1997.

[19] A. O. Bang and U. P. Rao, "Context-aware computing for IoT: History, applications and research challenges," in *Proc. 2nd Int. Conf. Smart Energy Commun.* Jaipur, India: Poornima Institute of Engineering and Technology, 2021, pp. 719–726.

[20] I. H. Sarker, "Mobile data science: Towards understanding data-driven intelligent mobile applications," 2018, *arXiv:1811.02491*.

[21] I. H. Sarker, "BehavMiner: Mining user behaviors from mobile phone data for personalized services," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2018, pp. 452–453.

[22] I. Chronis, A. Madan, and A. Pentland, "SocialCircuits: The art of using mobile phones for modeling personal interactions," in *Proc. Workshop Multimodal Sensor-Based Syst. Mobile Phones Social Comput.*, Nov. 2009, p. 1.

[23] Q. Meng, B. Liu, H. Zhang, X. Sun, J. Cao, and R. K.-W. Lee, "Temporal-aware and multifaceted social contexts modeling for social recommendation," *Knowl.-Based Syst.*, vol. 248, Jul. 2022, Art. no. 108923.

[24] W. M. Aly, K. A. Eskaf, and A. S. Selim, "Fuzzy mobile expert system for academic advising," in *Proc. IEEE 30th Can. Conf. Electr. Comput. Eng. (CCECE)*, Apr. 2017, pp. 1–5.

[25] D. O. Olguin, B. N. Waber, T. Kim, A. Mohan, K. Ara, and A. Pentland, "Sensible organizations: Technology and methodology for automatically measuring organizational behavior," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 39, no. 1, pp. 43–55, Feb. 2009.

[26] N. Eagle and A. S. Pentland, "Reality mining: Sensing complex social systems," *Pers. Ubiquitous Comput.*, vol. 10, no. 4, pp. 255–268, May 2006.

[27] C. Mukherjee, *Build Android-Based Smart Applications: Using Rules Engines, NLP and Automation Frameworks*. Karnataka, India: Apress, 2017. [Online]. Available: https://books.google.com.my/books?id=plJDDwAAQBAJ

[28] I. Uddin, "A rule-based framework for developing context-aware systems for smart spaces," Ph.D. dissertation, School Comput. Sci., Malaysia Campus, Univ. Nottingham, Nottingham, U.K., 2019.

[29] N. R. Jennings and S. Bussmann, "Agent-based control systems," *IEEE Control Syst.*, vol. 23, no. 3, pp. 61–74, Jun. 2003.

[30] J. Ramos, J. A. Castellanos-Garzón, A. González-Briones, J. F. de Paz, and J. M. Corchado, "An agent-based clustering approach for gene selection in gene expression microarray," *Interdiscipl. Sci., Comput. Life Sci.*, vol. 9, no. 1, pp. 1–13, Mar. 2017.

[31] A. González, J. Ramos, J. F. De Paz, and J. M. Corchado, "Obtaining relevant genes by analysis of expression arrays with a multi-agent system," in *Proc. 9th Int. Conf. Practical Appl. Comput. Biol. Bioinf.* Spain: Univ. of Salamanca, 2015, pp. 137–146.

[32] A. González-Briones, J. Ramos, J. F. De Paz, and J. M. Corchado, "Multi-agent system for obtaining relevant genes in expression analysis between young and older women with triple negative breast cancer," *J. Integrative Bioinf.*, vol. 12, no. 4, pp. 1–14, Dec. 2015.

[33] A. González-Briones, G. Villarrubia, J. F. De Paz, and J. M. Corchado, "A multi-agent system for the classification of gender and age from images," *Comput. Vis. Image Understand.*, vol. 172, pp. 98–106, Jul. 2018.

[34] P. Praitheeshan, L. Pan, X. Zheng, A. Jolfaei, and R. Doss, "SolGuard: Preventing external call issues in smart contract-based multi-agent robotic systems," *Inf. Sci.*, vol. 579, pp. 150–166, Nov. 2021.

[35] A. Bregar, "Implementation of a multi-agent multi-criteria negotiation protocol for self-sustainable smart grids," *J. Decis. Syst.*, vol. 29, no. 1, pp. 87–97, Aug. 2020.

[36] M. Alirezaie, J. Renoux, U. Köckemann, A. Kristoffersson, L. Karlsson, E. Blomqvist, N. Tsiftes, T. Voigt, and A. Loutfi, "An ontology-based context-aware system for smart homes: E-care@home," *Sensors*, vol. 17, no. 7, p. 1586, Jul. 2017.

[37] R. Abdur, "Smart space system interoperability," in *Proc. 3rd Int. Workshop (Meta) Model. Healthcare Syst.*, 2019, pp. 16–23.

[38] I. Uddin, A. Rakib, H. M. U. Haque, and P. C. Vinh, "Modeling and reasoning about preference-based context-aware agents over heterogeneous knowledge sources," *Mobile Netw. Appl.*, vol. 23, no. 1, pp. 13–26, Feb. 2018.

[39] N. Streitz, D. Charitos, M. Kaptein, and M. Böhlen, "Grand challenges for ambient intelligence and implications for design contexts and smart societies," *J. Ambient Intell. Smart Environments*, vol. 11, no. 1, pp. 87–107, Jan. 2019.

[40] P. N. Mahalle and P. S. Dhotre, *Context-aware Pervasive Systems and Applications*. New York, NY, USA: Springer, 2020.

[41] D. Cook and S. K. Das, *Smart Environments: Technology, Protocols, and Applications*, vol. 43. Hoboken, NJ, USA: Wiley, 2004.

[42] M. Ushold, C. Menzel, and N. Noy, "Semantic integration & interoperability using RDF and OWL," in *Proc. W3C Editor's Draft*, vol. 3, 2005.

[43] F. Yang, J. Huang, A. Bhardwaj, A. Hussain, A. A. A. El-Latif, and K. Yu, "Adaptive modulation based on nondata-aided error vector magnitude for smart systems in smart cities," *IEEE Internet Things J.*, vol. 10, no. 21, pp. 18672–18685, Nov. 2023.

[44] A. K. Dey, "Understanding and using context," *Pers. Ubiquitous Comput.*, vol. 5, no. 1, pp. 4–7, Feb. 2001.

[45] I. Uddin and A. Rakib, "A preference-based application framework for resource-bounded context-aware agents," in *Mobile and Wireless Technologies*. Kuala Lumpur, Malaysia: Springer, 2017, pp. 187–196.

[46] M. Alenezi, "Ontology-based context-sensitive software security knowledge management modeling," *Int. J. Electr. Comput. Eng.*, vol. 10, no. 6, p. 6507, Dec. 2020.

[47] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung, "Ontology based context modeling and reasoning using OWL," in *Proc. IEEE Annu. Conf. Pervasive Comput. Commun. Workshops*, Mar. 2004, pp. 18–22.

[48] A. Rakib, R. U. Faruqui, and W. MacCaull, "Verifying resource requirements for ontology-driven rule-based agents," in *Foundations of Information and Knowledge Systems*. Kiel, Germany: Springer, 2012, pp. 312–331.

[49] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A semantic web rule language combining OWL and RuleML. Acknowledged W3C submission, standards proposal research report: Version 0.6," W3C, Tech. Rep., 2004.

[50] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker, "Description logic programs: Combining logic programs with description logic," in *Proc. 12th Int. Conf. World Wide Web*, 2003, pp. 48–57.

[51] A. Rakib, H. M. Ul Haque, and R. U. Faruqui, "A temporal description logic for resource-bounded rule-based context-aware agents," in *Proc. 2nd Int. Conf. Context-Aware Syst. Appl.* Phu Quoc Island, Vietnam: Springer, 2013, pp. 3–14.

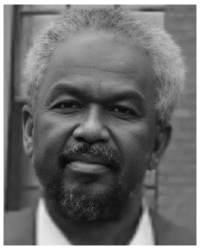
MUMTAZ ALI received the M.Sc. degree in computer science from the University of Peshawar, Pakistan, and the M.S. degree from the Institute of Management Sciences Peshawar, Pakistan. From 2005 to 2012, he was a Computer Network Administrator with the Institute of Management Sciences, Peshawar. Currently, he is an Assistant Professor with the Department of Computer Science, City University of Science and IT, Peshawar, Pakistan. With over 15 years of experience in research and academia, his areas of expertise encompass machine learning, the IoT, computer networks, and AI.

MUHAMMAD ARSHAD received the Ph.D. degree in computer science from Liverpool John Moores University, Liverpool, U.K., in 2010. From 2010 to 2013, he was an Assistant Professor with City University Peshawar, Pakistan. In 2013, he joined Sohar University, Sohar, Oman, as an Assistant Professor. Since 2019, he has been an Associate Professor with City University Peshawar. He has over 14 years of experience in research and academics. His current research interests include peer-to-peer networks, networked appliances, agent-based modeling and simulation, the Internet of Things, MANET's, and service-oriented architecture.

IJAZ UDDIN received the M.S. degree in computer science from the Institute of Management Sciences, Peshawar, Pakistan, and the Ph.D. degree in computer science from the University of Nottingham, Malaysia, in 2019. He has been an active Researcher and a Developer, since 2007. He is currently an Assistant Professor with the City University of Science and IT, Peshawar. He has published a number of research articles. His research interests include the Internet of Things, context-aware systems, and multi-agent systems.

GAUHAR ALI received the M.S. degree in computer science from the Institute of Management Sciences, Peshawar, Pakistan, in 2012, and the Ph.D. degree in computer science from the University of Peshawar, Peshawar. He is currently a Postdoctoral Researcher with the EIAS Data Science and Blockchain Laboratory, College of Computer and Information Sciences, Prince Sultan University, Riyadh, Saudi Arabia. His research interests include the Internet of Things, access control, blockchain, machine learning, wireless sensor networks, intelligent transportation systems, formal verification, and model checking.

MUHAMMAD ASIM received the M.S. degree in mathematics from the University of Peshawar, Peshawar, Pakistan, in 2013, the M.Phil. degree in mathematics from the Kohat University of Science and Technology, Kohat, Pakistan, in 2016, and the Ph.D. degree in computer science and technology from Central South University, Changsha, China, in 2022. Currently, he is conducting his postdoctoral research with the EIAS Data Science Laboratory, College of Computer and Information Sciences, Prince Sultan University, Riyadh, Saudi Arabia. His current research interests include artificial intelligence, computational intelligence techniques, cloud computing, edge computing, 5G/6G communication systems, and autonomous vehicles. He has been awarded as an Outstanding International Graduate of Central South University, in 2022.



MOHAMMED ELAFFENDI is currently a Professor in computer science with the Department of Computer Science, College of Computer and Information Sciences (CCIS), Prince Sultan University, Riyadh, Saudi Arabia. He is also the Founder and the Director of the Center of Excellence in CyberSecurity (CYBEX), the Founder and the Director of the EIAS Data Science Research Laboratory, AIDE to the Rector, the Director of the Institutional Policy and Development Unit, and the former Dean of CCIS. His research interests include machine learning, natural language processing, emerging distributed architectures, next-generation computer systems, and complexity science.