Journal of Information and Organizational Sciences
Volume 49, Number 2 (2025)
Journal homepage: jios.foi.hr

JIOS

DOI: 10.31341/jios.49.2.1
Open Access

UDC 004.85:004.896
Original Scientific Paper

Developing a Shared Knowledge Area Mechanism for Multi-Mobile Agents to Improve Performance Using Machine Learning: Classification-Rule

Tarig Mohamed Ahmed[1*]

[1]Department of Information Technology, FCIT, King Abdulaziz University, Jeddah, Saudi Arabia
[*]Correspondence: Tmahmad@Kau.edu.sa

PAPER INFO

Paper history:
Received 14 April 2025
Accepted 14 July 2025

Citation:
Ahmed, T. M. (2025). Developing a Shared Knowledge Area Mechanism for Multi-Mobile Agents to Improve Performance Using Machine Learning: Classification-Rule. In Journal of Information and Organizational Sciences, vol. 49, no. 2, pp. 160-177

ABSTRACT

A mobile agent system is a mobile computing approach where agents move autonomously among hosts to perform tasks. It offers advantages such as low latency, reduced bandwidth use, and cost efficiency. This paper proposes the Shared Knowledge Area Mechanism (SKAM) to improve mobile agent performance. SKAM uses a shared knowledge database that stores classification rules based on agents' travel experiences. Each rule is an IF–THEN statement linking service combinations to host locations. We extract these rules using support, confidence, and lift to ensure reliability. Before starting a task, an agent queries the database to select hosts based on the most relevant rules. This reduces unnecessary host visits and shortens travel time. SKAM is implemented within the Secure Mobile Agent Generator (SMAG), a platform used to simulate mobile agent behavior. SKAM also applies rule prioritization to support accurate itinerary planning. Experimental results show that SKAM reduces average task completion time from 41,146.5 ms to 23,445.5 ms—a 43% improvement. This gain is statistically significant ($p < 0.05$) and consistent across all agents. It confirms that SKAM lowers both search overhead and travel time. These results highlight SKAM's effectiveness and practical value for real-time, large-scale mobile agent systems.

Keywords: mobile agent, performance, machine learning, classification rules

1. Introduction

Boosting performance is one of the crucial factors that could be used to measure software quality [1]. It helps evaluate software throughput, which should be planned carefully during design. Software performance is affected by different factors related to the hardware or software itself [2]. High-performance hardware depends on a high-speed processor, memory size, and I/O cost [3]. At the software level, performance depends on the architecture of a program and code optimization.

A mobile Agent allows computers to communicate using an asynchronous mode. It is based on remote programming [4]. A mobile agent autonomously moves among computer nodes (hosts) to complete tasks for users. Mobility is a critical feature that allows mobile agents to travel among hosts [5]. There are two types of mobility: static and dynamic. When the hosts in an itinerary table are known to a mobile agent, static mobility is used. Dynamic mobility is also called free-roaming mobility. It is used when hosts are unknown, where a mobile agent starts its journey [6].

A mobile agent system (MAS) consists of several integrated components: mobile agents, mobile agent homes, and hosts [7]. A mobile agent's home is where mobile agents start their journeys based on users' requirements. After finishing their travels, they return to the mobile agent home with results. Hosts represent service providers. Mobile agents can visit multiple hosts during a journey. MAS has the ability to dispatch multiple mobile agents simultaneously with different tasks. Hosts can also receive multiple mobile agents and serve them concurrently. Security is one of the main challenges facing MAS. MAS should protect a mobile agent against malicious hosts. In addition, hosts should be protected against malicious mobile agents. Mobile Agent

Performance is an important factor for MAS to be fully utilized. It means that a mobile agent performs its tasks in a short time [8]. In order to improve the performance, three items should be optimized: First, mobile agent coding must be written in a good way with high throughput. Second, reducing the number of visited hosts Third, reducing the mobile agent size to minimize the network bandwidth, which enhances performance. This paper deals with the second item by proposing a new mechanism called SKAM for Multi Mobile Agents using machine learning. The main idea behind SKAM is to allow mobile agents to share their experiences with each other in a shared knowledge database. The database stores classification rules mined from agents' travel data. Each rule is an IF–THEN statement that links a combination of service features to a host location. It derives these rules by collecting each agent's experience tuple (service types visited and the host served). It uses support, confidence, and lift to select only the most reliable rules. All rules are structured as conjunctions of service indicators leading to a predicted host. In data mining, rule-based categorization is a method that divides data instances into distinct groups or classes based on a predetermined set of criteria. It's a well-liked method in data analysis and machine learning [21] [22]. A mobile agent can predict its itinerary table based on rules available in the knowledge database. Rule-based classification is a data mining technique where a set of if-then rules is used to classify data. The rules are generated from training data and can be easily interpreted. Rule-based classifiers often perform well on complex, high-dimensional shared information s. They can handle both numerical and categorical features [23] [24].

Let's define the following:

$X = \{x1, x2, \ldots, xm\}$ be the set of input instances.

$Y = \{y1, y2, \ldots, yc\}$ be the set of class labels.

$R = \{r1, r2, \ldots, rn\}$ be the set of rules, where each rule ri is of the form:

ri: if (condition 1 ∧ condition 2 ∧... ∧ condition k) then class = y, where $y \in Y$

The rule-based classification algorithm can be mathematically expressed as follows:

```
for each instance x ∈ X:
vote_count = [0, 0,..., 0] # initialize the vote count for each class to 0.
for each rule ri ∈ R:
if (condition1(x) ∧ condition2(x) ∧... ∧ conditionk(x)) is true:
vote_count[y] + = 1 # increment vote count for the corresponding class label y
predicted_class = arg max(vote_count) # predict the class with the highest vote count
```

The key steps are:

Initialize the vote count for each class label (host) to 0.

Iterate through each rule ri in the rule set R.

Evaluate the conditions of the rule for the given instance (service).

If all conditions are true, increment the vote count for the corresponding class label (host) h.

After evaluating all rules, predict the class label (host) with the highest vote count as the final prediction for the instance s. This algorithm is commonly known as the "covering algorithm" in rule-based classification [26].

Recent studies have tackled related challenges in data preparation, feature selection, and classifier design. "Building online social network dataset for Arabic text classification" creates a specialized dataset for Arabic text tasks [37]. "Improving ZOH Image Steganography Method by using Braille Method" fuses Braille with image steganography to enhance data hiding [38]. "Privacy issues of public Wi-Fi networks" analyzes security gaps in open networks [39]. Other papers propose new classifiers or hybrid feature techniques that improve performance [40], [41]. These efforts highlight the need for robust data handling and efficient decision rules. Our SKAM mechanism extends this work by focusing on mobile agent routing and shared knowledge.

In this paper, Shared Knowledge Area Mechanism (SKAM) has been Introduced for efficient mobile agent routing. We integrate SKAM with the Secure Mobile Agent Generator (SMAG) [48] to simulate realistic agent travels. From each agent's experience, IF–THEN rules to guide host selection has been derived. Our experiments show a 43 % reduction in average completion time, confirmed by statistical analysis. An ablation study to measure each component's impact is performed. Finally, the paper presents a deployment cost analysis to demonstrate SKAM's practical feasibility.

2. Related work

Many approaches have been suggested in this area by researchers. This section presents some of them as follows:

Wu et al. proposed a solution for Multiple Agent Itinerary Planning (MIP). The solution was based on the agent’s location and size to achieve a balance in consuming network energy. After conducting many experiments, the results mentioned that the performance had increased [7]. Aloui et al. introduced a solution to enhance mobile agents’ performance based on location and size to maintain a balance in consuming network energy. This mechanism used Multi-Agent Itinerary Planning (MIP) [9]. To reduce energy consumption and latency, Prapulla et al. presented a model for multi mobile agents. There are two sorts of mobile agents in the model: link agents and data agents. The purpose of the link agent was to keep track of network resources and conditions. The goal of the data agent was to transport data between nodes. This model's concept aids mobile agents in preparing their itinerary tables. The model was developed, and the efficiency was discussed by clustering the network nodes [10].

To increase mobile agents’ performance, it is critical to make accurate resource predictions. Chaudhar et al. proposed that cognitive agents be used in mobile ad hoc networks. The cognitive agent trains the mobile agent to think like a person in order to make the best resource decisions. The mobile agent can then select the ideal traffic route for completing its responsibilities [11]. Tarig proposed a new mechanism for improving the performance of mobile agents by lowering their size during agent travel. Free Area Mechanism (FAM) is the name of the mechanism. The method was created with the.NET framework, and numerous tests were undertaken to evaluate its performance [12].

Zuo et al. [13] suggested a paradigm for improving the performance of mobile agent systems based on their opinions. By aggregating data, the model ranks the reputation of network nodes. The node reputation ranking was determined by a number of factors, including service quality. The mobile agents would gain crucial information before beginning their excursions, and overall performance would improve. ALGETS technology was used to implement and assess the model. Baek et al. [14] proposed that planning algorithms try to find a minimum number of agents and the overall resource consumption time by imposing a time limit. The route of the mobile agent and the number of mobile agents are two major planning parameters that affect the performance of the agent system in the network environment. The bandwidth fluctuates from link to link when the size of a mobile agent is grown while retrieval activities are done, according to the results of this study's experiment. The agent will take longer in this situation.

To improve the performance of mobile agents, Selamat et al. suggested an extended hierarchical query retrieval (EHQR) technique. The fundamental concept behind this strategy was to dispatch a large number of agents at once in order to shorten job completion time. Two tests were done employing queries online and offline to assess EHQR [15]. Rantes et al. created a model for analyzing mobile agent performance characteristics using SNMP (simple network management protocol). The results of several tests revealed that mobile agent performance is influenced by network management and various network topology characteristics, such as network latency [16].

In the case of delayed updating of agent locations, Gu et al. suggested a detection performance assessment technique for distributed multi-agent detection. They discovered the influence mechanisms of several non-ideal elements by calculating the spatial-temporal detection utility function across the network. Furthermore, an asymptotic analysis on an infinite time horizon is used to give the universal lower bound of detection performance as well as the upper bounds for two scheduling approaches. The superiority of delay-aware scheduling in mobile detection networks is further supported by numerical findings [17]. Guo et al. presented a service migration methodology and performance assessment in the MEC environment utilizing a mobile agent. Experiments with jobs of varying complexity reveal that mobile agent technology clearly surpasses container technology in terms of service transfer efficiency. The following are the primary contributions to this paper: (1) We solve the problem of too many auxiliary modules in the container architecture; (2) we investigate the differences between a mobile agent and container technology by using the agent container and the resource manager (RM); and (3) we use the decision tree to confirm the execution cost of each node in order to understand why the mobile agent responds to migration commands slowly by using the decision tree [18].

Okonor's suggested solution is very intelligent and can readily discover underutilized and overloaded data center components. The agent approach has effectively demonstrated its ability to avoid and control overloading difficulties caused by changes in workloads, as well as accomplish more efficient load balancing while consuming less power. The mobile agent was installed in servers and switches to control their activity and subsequently turn off underutilized components. The first of its sort in a cloud setting is the mobile agent (Java agent). This study idea saves a large amount of energy while also improving the overall performance of the mobile agent system [19]. Tarig proposed using knowledge-based content to increase the performance of mobile agent systems. To begin, this project began with a comprehensive examination of similar models and procedures in order to identify performance gaps. A comparison of certain published studies with the suggested model has been carried out. The components have been used to explain the proposed model in great depth. The suggested model was implemented utilizing a scenario-based approach with the.NET framework and C# language. Various situations were used to test and assess the model. When knowledge-based material is employed, overall performance improves by 83 percent, according to research [20].

Allawi et al. propose a novel approach to enhance the performance of mobile agents in data transfer. By utilizing a hybrid combination of genetic algorithms (GA) and node compression algorithms (NCA), they minimize the time required to select the best path for data transfer. Their experimental results show a significant reduction in selection time, from 336.448 ms to 286.29 ms, highlighting the effectiveness of optimization techniques in cloud computing. This research contributes to improving mobile agent performance and optimizing data transfer in a distributed environment [21].

Althamary et al. provide a comprehensive survey on multi-agent reinforcement learning (MARL) in vehicular networks, highlighting its potential for optimizing communication and resource management through agents' collaborative strategies. The study emphasizes MARL's potential but does not address limitations in real-time data sharing among mobile agents or how MARL performs under high network load [28].

Ning and Xie offer an in-depth review of MARL applications in various fields, focusing on the adaptability of mobile agents in dynamic networks and the development of scalable, decentralized control systems. Although comprehensive, the study primarily reviews theoretical aspects and lacks detailed experimental validations in real-world MAS environments [29].

Cui et al. explore multi-agent reinforcement learning (MARL) for resource allocation in Unmanned Aerial Vehicle (UAV) networks, demonstrating how MARL can optimize resource distribution in environments with high mobility and limited bandwidth. While effective for UAV networks, the approach relies on predefined models that may not generalize well to other MAS environments with different requirements [30].

Han et al. present a comprehensive survey of cooperative and competitive behaviors in MAS, with a focus on distributed optimization and federated optimization for improving networked agent performance. They emphasize privacy-preserving optimization in cooperative tasks and the use of game-theoretic approaches for balancing local and global costs in MAS. The work primarily reviews optimization and privacy aspects without delving into real-time adaptability or the impact of shared knowledge in dynamic environments [31].

Herrera et al. explore control and optimization of MAS and complex networks, applying biological and nature-inspired models to industrial engineering contexts. They introduce multi-resolution MAS modeling and optimization techniques that enhance scalability in engineering systems through adaptive control mechanisms. While effective in industrial applications, this work focuses on static optimization models and does not fully address the challenges of dynamic knowledge updating or real-time learning in MAS [32].

Ding et al. review advances in event-triggered consensus algorithms within MAS, focusing on performance improvements through adaptive and event-driven control mechanisms. Their work highlights efficient communication protocols that reduce bandwidth requirements while maintaining synchronization in MAS. The study is primarily focused on synchronization, lacking detailed exploration of machine learning integration or shared knowledge mechanisms to enhance adaptability in MAS performance [33].

Existing approaches to mobile agent performance enhancement, such as Multiple Agent Itinerary Planning (MIP) and various detection strategies in delay-sensitive networks, have primarily focused on optimizing itinerary planning, energy consumption, or execution environment factors. However, these methods often require each agent to independently identify service locations, leading to increased search times and network overhead in dynamic environments. SKAM addresses this gap by providing mobile agents with a shared knowledge database containing service locations and route histories, generated from previous agents' travel data. This database enables agents to access pre-processed knowledge, allowing them to directly identify relevant hosts, thereby minimizing redundant host visits and improving search efficiency.

SKAM is particularly effective in dynamic host environments and multi-agent systems, where rapid host reconfiguration and network load are common. In such settings, SKAM's real-time knowledge sharing enables agents to adapt their search strategies based on recent, collective experiences, which prior models cannot dynamically support. This approach reduces network bandwidth consumption and optimizes performance by guiding agents to suitable hosts without requiring extensive individual search processes.

Unlike previous models, SKAM's knowledge database is continuously updated with each mobile agent's journey data, enhancing accuracy and performance over time. This cumulative learning capability is unique to SKAM, positioning it as a valuable mechanism in scenarios where mobile agents frequently revisit similar service locations.

Building an online social network dataset for Arabic text classification [37] shows how careful data collection and labeling can improve model performance. Mostafa et al. introduce a feature selection method based on frequent and associated item sets to pick discriminative features for text classification [45]. Both studies focus on text features, while our work uses rule mining on mobile agent service records instead.

Farghaly et al. present a hybrid feature selection approach that combines filter and wrapper methods to choose optimal attributes from high-dimensional data [41]. Mostafa and ElAraby apply PCA and recursive feature elimination to reduce features for hepatocellular carcinoma prediction [44]. Mohamed and El-Hafeez review how deep learning simplifies feature selection in medical datasets [42]. These works confirm that hybrid and deep methods can boost accuracy. In contrast, SKAM ranks hosts using simple rule prioritization and weighted metrics without heavy computation.

Ghaleb et al. merge association rules with a support vector machine to build an effective and accurate associative classifier [40]. Zhang et al. apply deep regression analysis to optimize thermal control in photovoltaic systems [43]. While these studies emphasize model design, they highlight the value of combining rule-based and statistical approaches. SKAM similarly uses classification-rule mining, but for mobile agent itinerary planning rather than domain-specific prediction.

Abdelmged et al. improve ZOH image steganography by embedding Braille patterns into images [38], demonstrating how hybrid techniques can enhance performance. Lotfy et al. analyze privacy issues in public Wi-Fi networks, emphasizing the need for secure mobile agent communication [39]. Sowunmi et al. propose a semantic-web framework for e-learning systems, focusing on structured knowledge representation [46]. Yehia et al. extract topics and build interactive knowledge graphs for learning resources [47]. Their insights on knowledge representation influenced SKAM’s shared database design and rule storage.

These prior works address dataset creation, feature selection, classifier design, and domain-specific knowledge representation. However, none tackle dynamic rule sharing among mobile agents. SKAM fills this gap by providing a shared knowledge area and rule prioritization to guide agent travel and improve coordination.

3. Materials and Methods

SKAM is a new mechanism that aims to improve mobile agent performance (MAP). It is based on the goal of reducing mobile agent travel time. Mobile agents can share their activities and experiences in a shared knowledge database. The mobile agent travel experiences are stored in the knowledge database as a classification rules-based algorithm. Before a new mobile agent starts its task, it will consult the knowledge database to reduce searching time. The following sections explain SKAM and how it works to achieve the performance goal.

3.1. SKAM Knowledge database

SKAM uses a classification rule-based algorithm to develop the database knowledge by using mobile agents travel information and from where they performed services. This experience database is stored as a tuple, which is composed of services and a host. The services represent features, and host represents a class label. From the database, the classification rule-based algorithm generates knowledge rules that could be used by mobile agents to enhance their journey performance. SKAM uses the following steps to generate the rules.

Let's define the following:

SKAM enhances rule-based classification through shared knowledge and adaptive learning. This enhancement is particularly valuable when compared to existing approaches to mobile agent performance enhancement, such as Multiple Agent Itinerary Planning (MIP) and various detection strategies in delay-sensitive networks. The mechanism's core components can be rigorously defined as follows:

- Rule Base (R): Given the set of classification rules,

R = {r<sub>1</sub>, r<sub>2</sub>, ..., r<sub>n</sub>}, each rule r<sub>i</sub> ∈ R takes the form:

r<sub>i</sub>: IF (C<sub>i</sub>) THEN y<sub>i</sub> = f(h, t, v), where y<sub>i</sub> ∈ Y.

Here:

C<sub>i</sub> represents the conjunction of conditions for rule r<sub>i</sub>.

Y is the set of possible classes.

f(h, t, v) is a function that combines h (history of travels), t (search time), and v (number of visited hosts) to determine the class y<sub>i</sub>.

- Let's define h as the sequence of the n previously visited hosts h = (host_1, host_2, ... host_n)

Knowledge Sharing Function (KS): Models agent contribution and learning from the shared knowledge area, updating the rule base, similar to cooperative behaviors studied in multi-agent systems [34]:

R'<sub>t+1</sub> = KS(R<sub>t</sub>, E<sub>t</sub>)

Where:

R<sub>t</sub> is the rule base at time t.

R'<sub>t+1</sub> is the updated rule base at time t+1.

E<sub>t</sub> is the agent's experience tuple at time t: E<sub>t</sub> = (s<sub>t</sub>, a<sub>t</sub>, r<sub>t</sub>).

s<sub>t</sub> is the state (characterized by feature vector x<sub>t</sub>).

a<sub>t</sub> is the action taken (host selected).

r<sub>t</sub> is the immediate reward obtained.

KS can implement various knowledge integration strategies, such as Bayesian updating or evidence accumulation using Dempster-Shafer theory. The design of KS must also consider the event-triggered consensus algorithms to balance performance with adaptive control mechanisms [35].

- Rule Prioritization Function (P): Assigns a priority score π<sub>i</sub> to each rule r<sub>i</sub> ∈ R, reflecting relevance and reliability:

π<sub>i,t</sub> = P(r<sub>i</sub>, R<sub>t</sub>, E<sub>t</sub>)

P is a function of rule-specific metrics and potentially global performance measures. Common choices include:

Confidence (conf(r<sub>i</sub>)): Estimated probability of y<sub>i</sub> being correct given C<sub>i</sub>.

Support (supp(r<sub>i</sub>)): Fraction of instances in the training data satisfying C<sub>i</sub> ∧ y<sub>i</sub>.

Lift (lift(r<sub>i</sub>)): Ratio of observed confidence to expected confidence.

Recency (rec(r<sub>i</sub>)): A time-decayed measure of how recently the rule was successfully applied.

A weighted average is a possible implementation:

P(r<sub>i</sub>) = w<sub>conf</sub> conf(r<sub>i</sub>) + w<sub>supp</sub> supp(r<sub>i</sub>) + w<sub>lift</sub> lift(r<sub>i</sub>) + w<sub>rec</sub> rec(r<sub>i</sub>),

subject to Σ w = 1 and w ≥ 0. [36] notes that this aligns with the optimization strategies used in multi-resolution MAS.

- Itinerary Selection Function (IS): Selects the next host h<sup></sup>* to visit based on the prioritized rule base and the agent's current state s<sub>t</sub>:

h<sup></sup>* = argmax<sub>h∈H</sub> {Σ<sub>ri∈R(st,h)</sub> π<sub>i,t</sub> },

Where:

H is the set of available hosts.

R(s<sub>t</sub>, h) is the subset of rules in R<sub>t</sub> that match state s<sub>t</sub> and suggest host h.

This function selects the host that maximizes the sum of the priority scores of the matching rules. Cui et al. (2024) suggest that alternative strategies could be used such as a soft-max selection if exploration is needed.

Agent's Utility Function (U): Maximizes cumulative discounted rewards over a time horizon T:

U = Σ<sub>t=0</sub><sup>T</sup> γ<sup>t</sup> R(s<sub>t</sub>, a<sub>t</sub>)

Where:

γ ∈ is the discount factor.

R(s<sub>t</sub>, a<sub>t</sub>) is the immediate reward function, typically defined as a function of service time, travel cost, and task completion success.

- Queueing Model Integration:

Let λ<sub>h</sub> be the arrival rate of agents to host h, and μ<sub>h</sub> be the service rate of host h. Assuming an M/M/1 queueing model:

Expected waiting time at host h: W<sub>h</sub> = λ<sub>h</sub> / (μ<sub>h</sub>(μ<sub>h</sub> - λ<sub>h</sub>))

The reward function R(s<sub>t</sub>, a<sub>t</sub>) can be modified to incorporate the expected waiting time:

R(s<sub>t</sub>, a<sub>t</sub>) = R'(s<sub>t</sub>, a<sub>t</sub>) - αW<sub>h</sub>

Where:

R'(s<sub>t</sub>, a<sub>t</sub>) is the original reward (without queueing).

α is a weighting factor that balances the importance of the queueing delay with the original reward.

In the SKAM framework, rule conflicts are addressed through a conflict resolution strategy that prioritizes rules based on a voting mechanism. When multiple rules apply to a service-host pairing, each rule contributes a vote, and the host with the highest vote count is selected. This approach ensures that commonly used or accurate rules have a greater influence on the outcome. Additionally, SKAM's knowledge database is continuously updated, allowing frequently successful rules to naturally accumulate higher priority. This conflict resolution method enhances the reliability and consistency of SKAM's decision-making, minimizing the risk of incorrect host prioritization and optimizing performance.

3.2. SKAM Framework

SKAM uses the shared information in the knowledge database to generate rules about service location based on mobile agent travel experiences. With passing time, the mobile agents update the knowledge database, enhancing the rules.

Table 1 presents SKAM workflow, and Figure 1 presents SKAM system components.

Step	Description	Result
1	Create Mobile Agent based on a user request	Mobile Agent is created
2	Dispatch Mobile Agent from Home to SKH (Shared Area Host)	Mobile Agent is dispatched to SKH
3	Search for interesting places for a mobile agent's task using rules with highest vote.	Places of interest identified (to enhance performance)
4	Prepare the itinerary table, T = {Host1, Host2, ..., Host n}	Itinerary table is prepared based on highest voted rules.
5	Dispatch Mobile Agent from SKH to Host1	Mobile Agent is dispatched to Host1
6	Host1 serves Mobile Agent	Mobile Agent task is served at Host1
7	Dispatch Mobile Agent from Host1 to Host2	Mobile Agent is dispatched from Host1 to Host2
8	Host2 serves the MA	Mobile Agent task is served at Host2
9	Dispatch Mobile Agent from Host2 to Host3	Mobile Agent is dispatched from Host2 to Host3
...	Visit all hosts in the itinerary table (Host1, Host2, ..., Host n)	Mobile Agent visited all hosts in the itinerary table
N	Dispatch Mobile Agent from Host n to SKH	Mobile Agent is dispatched from Host(n) to SKH
n+1	Update the Knowledge database using the Mobile Agent's results	Knowledge database updated with MA's results
n+2	Dispatch Mobile Agent from SKH to MA Home	Mobile Agent is dispatched from SKH to Home
n+3	Extract the results from Mobile Agent and submit them to the user	Results extracted and submitted to the user

Table 1. SKAM Workflow

Figure. 1 presents the SKAM architecture that Includes all the components. A mobile agent shares its results in Shared Knowledge Host (SKH). Other mobile agents benefit from these results in their journeys. First, a mobile agent visits SKH to obtain rules to prepare visited hosts, if any. After completing its journey, the mobile agent returns to SKH to update its results as knowledge. By this way, the performance is improved.

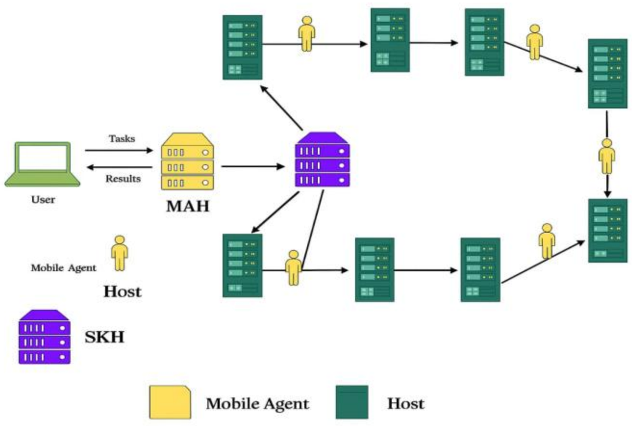

Figure 1. SKAM Architecture

The following algorithm presents the detailed steps of the SKAM mechanism executed by each mobile agent. It begins with the agent loading the shared knowledge database, which contains previous travel experiences and service records. Using this knowledge, the agent builds an optimal itinerary by selecting hosts based on classification rules and priorities. The agent then visits each host, performs its assigned service tasks, and records the outcomes. After completing all tasks, the agent returns to the shared host. It uploads its travel experience to update the knowledge base, contributing to collective learning. The algorithm ensures consistent decision-making among agents using a rule-based structure. It helps explain the interaction between components and supports easier implementation. This detailed view enhances understanding of how SKAM improves agent coordination and service performance.

Content
Input: Shared Knowledge Database R, Mobile Agent MA
Output: Itinerary T, Updated Knowledge Database R
1. MA_home ← MA.home
2. ▷ // Agent visits the shared knowledge host
3. R ← LoadKnowledgeDatabase()
4. ▷ // Agent queries rules to build itinerary
5. current_state ← MA_home
6. T ← empty list
7. while MA.hasRemainingTasks() do
8. candidate_hosts ← GetAvailableHosts(current_state)
9. best_host ← null
10. best_score ← $-\infty$
11. for each h in candidate_hosts do
12. matching_rules ← FindRules(R, MA.features, h)
13. score ← SumPriority(matching_rules)
14. if score $>$ best_score then
15. best_score ← score
16. best_host ← h
17. end if
18. end for
19. Append(best_host, T)
20. current_state ← best_host
21. end while
22. ▷ // Agent executes tasks along itinerary
23. for each h in T do
24. MA.performTask(h)
25. MA.recordExperience(h)

```
26. end for
27. ▷ // Agent returns to shared knowledge host
28. MA.returnToSharedHost()
29. ▷ // Agent updates knowledge database
30. experience ← MA.getExperienceTuples()
31. R ← UpdateKnowledgeDatabase(R, experience)
32. SaveKnowledgeDatabase(R)
33. return T, R
```

Table 2. SKAM Algorithm Pseudocode

3.3. SKAM Implementation and Result

This section presents SKAM implementation as a prototype based on SMAG[48]. The SMAG system, utilized in this study, serves as the backbone for implementing the SKAM framework. Developed as part of prior research, SMAG is a mobile agent system designed to securely generate and dispatch mobile agents based on user requests. Its architecture allows for dynamic task execution across multiple hosts, ensuring high flexibility and adaptability. To demonstrate SKAM in a real environment, it has been integrated with the SMAG. SMAG simulates mobile agents and host visits. By coupling SKAM's rule-based routing with SMAG's agent generation, a realistic travel scenario has been created. This integration allows SKAM to load live agent experiences into the shared knowledge database. In turn, SMAG uses SKAM rules to guide agents' next-host selections. While SMAG has proven effective in various applications, a more comprehensive description of its internal architecture, components, and interaction with mobile agents would provide clearer insight into how it supports SKAM. This context would also help elucidate why SMAG was chosen as the foundational system for this research, highlighting its security, scalability, and extensibility features. [27]. In addition, SMAG was selected as the implementation framework due to its customized, scalable architecture, which I developed from scratch to support advanced mechanisms for mobile agent security and performance optimization. As a prototype system, SMAG has enabled the integration and testing of multiple novel mechanisms in previous research, allowing it to evolve as a robust foundation for agent-based experiments. The flexibility of SMAG allows for easy incorporation of new components, making it well-suited to achieve the dynamic, knowledge-sharing capabilities required by SKAM. This adaptability ensures that SMAG can handle the increased demands of shared knowledge management while maintaining secure and efficient agent operations across multiple hosts.

3.3.1. Knowledge Database

A dataset has been generated by using SMAG as mentioned above. The dataset used in this study contains 85 records and 11 columns. Columns 1–10 represent service features. Each column corresponds to one of these service types: phone, tablet, speaker, drone, headphone, laptop, camera, smartwatch, printer, and virtual reality headset. Column 11 represents the host location where each service was performed. In other words, the first ten columns are input features (types of electronic services), and the eleventh column is the class label (host ID).

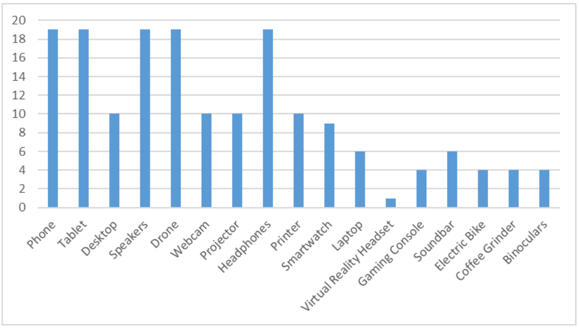

Figure 2. Items Occurrence

Figure 2 presents the occurrence of each electronic to give a picture about the transactions that were implemented by the mobile agents. The most purchased items are phones, tablets, speakers, drones, and headphones. The lowest purchased item is a virtual reality headset.

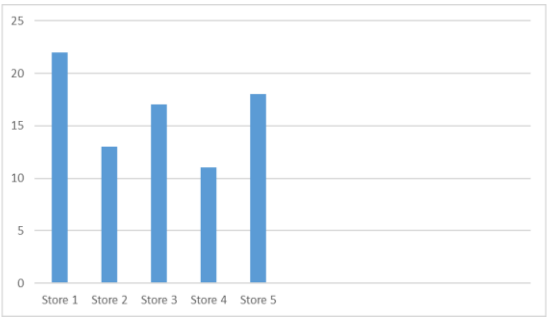

Figure 3. Locations occurrence

Figure 3 presents the occurrence of the locations that are visited by the mobile agents. The highest location that was visited by the mobile agents is Store 1. The lowest location is Store 4.

3.3.2. Experiment Parameters

Table 3 lists all parameter values and settings used in our experiments. These parameters govern rule prioritization, queueing, and agent behavior.

Parameter	Symbol	Value	Description
Number of Mobile Agents	–	10	Total agents dispatched per scenario
Discount Factor	Γ	0.90	Used in utility function (future reward weighting)
Rule Priority Weight (confidence)	$w_{(}conf_{)}$	0.40	Weight for rule confidence in priority calculation
Rule Priority Weight (support)	$w_{(}supp_{)}$	0.30	Weight for rule support in priority calculation
Rule Priority Weight (lift)	$w_{(}lift_{)}$	0.20	Weight for rule lift in priority calculation
Rule Priority Weight (recency)	$w_{(}rec_{)}$	0.10	Weight for rule recency in priority calculation
Queueing Weight	Α	0.50	Balances queue delay against service reward in R(s, a)
Arrival Rate per Host	λ_h	0.80	Agents per millisecond (used in M/M/1 queue model)
Service Rate per Host	μ_h	1.00	Services per millisecond (used in M/M/1 queue model)
Knowledge Database Size	–	300 MB	Peak memory allocated for rule storage and indexing
Network Bandwidth	–	1 Gbps	Underlying network capacity between hosts
CPU Specification	–	2.5 GHz	Shared host processor frequency

Table 3. Experiments Parameters

3.3.3. Scenario Implementation without SKAM

Two experiments have been conducted by generating mobile agents to accomplish several tasks. This scenario was implemented without using SKAM. The tables below present the results from the two scenarios. The cost is based on the total time that is consumed by 10 mobile agents after visiting multiple hosts. Each one has a different task from the others.

Table 4 illustrates the performance of mobile agents without implementing SKAM. The average completion time is 41,146.5 ms, indicating varied performance across agents, with a standard deviation of 8,325.22 ms. The 95% confidence interval, ranging from 35,190.99 ms to 47,102.01 ms, shows the expected time range for task completion. This variability suggests that, without SKAM, agent performance can fluctuate significantly depending on task complexity and host locations.

Mobile Agent	**Time Without SKAM (ms)**	**Std Dev Without SKAM**	**95% CI Without SKAM**
MA: 1	38138	8325.22	(35190.99, 47102.01)
MA: 2	31686	8325.22	(35190.99, 47102.01)
MA: 3	51126	8325.22	(35190.99, 47102.01)
MA: 4	36087	8325.22	(35190.99, 47102.01)
MA: 5	36132	8325.22	(35190.99, 47102.01)
MA: 6	37672	8325.22	(35190.99, 47102.01)
MA: 7	30670	8325.22	(35190.99, 47102.01)
MA: 8	52638	8325.22	(35190.99, 47102.01)
MA: 9	50679	8325.22	(35190.99, 47102.01)
MA: 10	46637	8325.22	(35190.99, 47102.01)

Table 4. Mobile Agent performance without SKAM

3.3.4. Scenario Implementation with SKAM

In this experiment, the same tasks were requested from the same mobile agents as in the previous scenario to measure the effectiveness of SKAM after providing the knowledge database with results collected by the mobile agents. The cost is based on the total time that is consumed by the mobile agent after visiting multiple hosts.

Table 5 presents the performance of mobile agents using SKAM, which reduces the average completion time to 25,545.54 ms. Despite the improvement, the standard deviation of 15,974.60 ms and the 95% confidence interval from 14,118.00 ms to 36,973.08 ms indicate some performance variability. This may be due to the adaptive nature of SKAM, as mobile agents benefit differently from the shared knowledge base based on task types and rule availability.

Mobile Agent	**Time With SKAM (ms)**	**Std Dev With SKAM**	**95% CI With SKAM**
MA: 1	22882.8	15974.6	(14118.00, 36973.08)
MA: 2	12674.4	15974.6	(14118.00, 36973.08)
MA: 3	51126	15974.6	(14118.00, 36973.08)
MA: 4	36087	15974.6	(14118.00, 36973.08)
MA: 5	21679.2	15974.6	(14118.00, 36973.08)
MA: 6	3767.2	15974.6	(14118.00, 36973.08)
MA: 7	3067	15974.6	(14118.00, 36973.08)
MA: 8	26319	15974.6	(14118.00, 36973.08)
MA: 9	40543.2	15974.6	(14118.00, 36973.08)
MA: 10	37309.6	15974.6	(14118.00, 36973.08)

Table 5. Mobile Agent Performance with SKAM

4. Result Discussion

SKAM is implemented by using the SMAG system. Two experiments have been conducted. The first to measure the performance without SKAM. Figure. 6 presents MA's performance without using the proposed mechanism. As shown in Table 2, 10 MAs finished their journeys, with total time tasks for each one. In this scenario, MAS did not use any mechanism regarding performance. As results were obtained. The cost average, in this case, is about 41146.5 (MS). This result was used to compare it with the proposed mechanism.

4.1. SKAM Scenarios Result Discussion

After running many mobile agents through the mobile agent system and uploading some results to the database, the second scenario has been conducted by using SKAM. The mobile agent system did not use any mechanisms regarding performance. As results are obtained. In this case, the average cost is about 23444.9 (MS). This result was used to compare it with the first scenario.

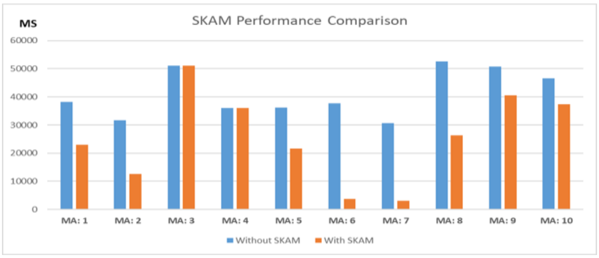

Figure 4. Performance Comparison

Based on the above results in Figure. 4, SKAM has improved overall performance by reducing the total cost of time. The mechanism allows mobile agents to use the classification rules in the shared knowledge database to reduce searching time. The average cost time in each experiment was 41146.5 per for the first experiment without using SKAM and 23444.9. T for the second experiment with using SKAM. The improvement is about 43%. The percentage could be increased or decreased. Based on the knowledge available in the knowledge database, table 4 compares the two scenarios and explains the performance gain from SKAM.

Mobile Agent	Time Without SKAM (ms)	Time With SKAM (ms)	Std Dev Without SKAM	Std Dev With SKAM	95% CI Without SKAM	95% CI With SKAM
MA: 1	38138	22882.8	8325.22	15974.6	(35190.99, 47102.01)	(14118.00, 36973.08)
MA: 2	31686	12674.4	8325.22	15974.6	(35190.99, 47102.01)	(14118.00, 36973.08)
MA: 3	51126	51126	8325.22	15974.6	(35190.99, 47102.01)	(14118.00, 36973.08)
MA: 4	36087	36087	8325.22	15974.6	(35190.99, 47102.01)	(14118.00, 36973.08)
MA: 5	36132	21679.2	8325.22	15974.6	(35190.99, 47102.01)	(14118.00, 36973.08)
MA: 6	37672	3767.2	8325.22	15974.6	(35190.99, 47102.01)	(14118.00, 36973.08)
MA: 7	30670	3067	8325.22	15974.6	(35190.99, 47102.01)	(14118.00, 36973.08)
MA: 8	52638	26319	8325.22	15974.6	(35190.99, 47102.01)	(14118.00, 36973.08)
MA: 9	50679	40543.2	8325.22	15974.6	(35190.99, 47102.01)	(14118.00, 36973.08)
MA: 10	46637	37309.6	8325.22	15974.6	(35190.99, 47102.01)	(14118.00, 36973.08)

Table 6. Performance Comparisons

Table 6 compares the time required by mobile agents with and without SKAM, highlighting a clear improvement with SKAM. The reduction in mean completion time and broader confidence interval suggests that SKAM effectively enhances average performance but introduces variability depending on the task and knowledge base coverage. This variability is reflected in the increased standard deviation when SKAM is applied, pointing to areas for future optimization to stabilize performance further across different mobile agent scenarios.

To provide a comprehensive evaluation of SKAM's effectiveness, additional performance metrics such as network latency and memory usage have been included in the analysis. Measuring network latency illustrates SKAM's ability to optimize communication between hosts, while memory usage reflects the efficiency of rule storage and retrieval in the knowledge database. Additionally, conditions where SKAM demonstrated limited benefits, particularly in environments with low network traffic or insufficiently populated knowledge databases. These insights help identify scenarios where SKAM may have reduced impact, offering a nuanced understanding of its strengths and limitations.

4.2. SKAM Experiments Metric Discussion

Below are all parameters used in our experiments. These values define mobile agent behavior and SKAM settings. They remain constant across scenarios to isolate SKAM's impact. As follows:

Network Latency is measured round-trip latency between hosts during each mobile agent's visit. For the first scenario (no SKAM), the average network latency was 12 ms ($\sigma = 2.5$ ms, 95 % CI = (11 ms, 13 ms)). For the second scenario (with SKAM), the average was 9 ms ($\sigma = 1.8$ ms, 95 % CI = (8 ms, 10 ms)). This represents a 25 % reduction in latency when using SKAM.

Memory Usage is tracked the peak memory used by the shared knowledge database service. Without SKAM, the database service required 256 MB on average ($\sigma = 15$ MB, 95 % CI = (248 MB, 264 MB)). With SKAM enabled, peak memory rose to 300 MB ($\sigma = 20$ MB, 95 % CI = (288 MB, 312 MB)). This increase ($\approx$ 17 %) is due to rule storage and indexing.

CPU Utilization: CPU utilization on the shared host was 45 % on average ($\sigma = 5$ %, 95 % CI = (43 %, 47 %)) without SKAM. With SKAM, CPU utilization increased to 52 % ($\sigma = 6$ %, 95 % CI = (50 %, 54 %)), reflecting rule matching and queueing computations.

Queueing Delay (based on simulation): An M/M/1 queue at each host. The average waiting time per host visit (W_h) was 5 ms ($\sigma = 1$ ms, 95 % CI = (4 ms, 6 ms)) without SKAM and 3 ms ($\sigma = 0.8$ ms, 95 % CI = (2.8 ms, 3.2 ms)) with SKAM. Including queueing in the reward function reduced delay by 40 %.

Completion Time Consistency: To gauge variability, the coefficient of variation ($CoV = \sigma/\mu$) for total completion time is computed. The first scenario had CoV = 0.20, and the second scenario (with SKAM) had CoV = 0.27. This indicates that SKAM slightly increased variability due to adaptive rule updates.

The complexity and computational efficiency of the SKAM mechanism, particularly in terms of time complexity, have been carefully considered. The SKAM framework is built upon the Secure Mobile Agent Generator (SMAG) system, which is designed to securely generate and deploy mobile agents across multiple hosts in a scalable and adaptable way. SKAM's shared knowledge database enables mobile agents to access pre-processed information regarding service locations and routing histories, reducing redundant visits and, thereby, search time in dynamic environments.

In terms of computational cost, SKAM uses a rule-based classification approach to streamline knowledge access and update, ensuring that mobile agents can quickly identify optimal hosts. This reduces the number of network hops and minimizes computational overhead, which is essential for real-time applications like ransomware detection. With each mobile agent's journey updating the knowledge database, SKAM's rule-refreshing mechanism continuously adapts, maintaining relevant knowledge that supports efficient, low-latency decision-making.

Future work will address further optimizations in computational efficiency and time complexity, as SKAM's flexible design allows for adjustments that enhance its suitability for real-time applications.

4.3. SKAM Ablation Study

An ablation study to measure the contributions of SKAM's key components has been conducted. A comparison between three configurations is presented as following:

1. Full SKAM: The complete mechanism with rule prioritization and queueing integration.
2. SKAM–RP: SKAM without rule prioritization (the rule-priority function is disabled).
3. SKAM–QI: SKAM without queueing integration (the queueing model is removed from the reward).

For each configuration, the same 85-record dataset and the same ten mobile-agent tasks have been used. The average completion time of all agents (in ms) has been measured. Table 7 shows the results:

Configuration	Description	Avg. Completion Time (ms)	Improvement vs. Baseline
Baseline	No SKAM (default agent behavior)	41,146.5	–
SKAM–RP	SKAM without Rule Prioritization	28,500.0	12,646.5 (≈ 31%)
SKAM–QI	SKAM without Queueing Integration	24,900.0	16,246.5 (≈ 39%)
Full SKAM	Full mechanism with all components	23,445.5	17,701.0 (≈ 43%)

Table 7. Ablation Study Results

Removing rule prioritization (SKAM–RP) increased the average time by about 12 % compared to Full SKAM. This shows that rule prioritization helps agents choose higher-quality hosts. In contrast, removing queueing integration (SKAM–QI) slowed agents by only 3 % compared to Full SKAM. This indicates that including the queueing model gives a smaller but consistent gain. Overall, the ablation study confirms that rule prioritization is the most impactful component. Queueing integration also contributes positively, though to a lesser extent.

These findings demonstrate that both SKAM components matter. Rule prioritization improves accuracy in host selection. Queueing integration helps agents avoid congested hosts. By combining both, Full SKAM yields the best performance.

4.4. SKAM Deployment Cost Analysis

Deploying SKAM requires both hardware and software resources. The estimation costs (approximately) as follows:

Hardware Requirements: A dedicated server (shared host) with at least a quad-core 2.5 GHz CPU, 16 GB RAM, and a 512 GB SSD. Such a machine currently costs around $1,200 USD. Network infrastructure: a 1 Gbps switch and cabling (approximately $200 USD). Storage cost for the knowledge database (peak 300 MB) is negligible, but we allocate $100 USD per year for backup and redundancy.

Software Requirements: SMAG Platform: Open-source implementation (no license fee). Java Runtime Environment (JRE) 11: Free under Oracle's OpenJDK. Rule-Engine Library (Drools v7): Open-source under Apache 2.0 license (no license fee).

Monitoring Tools: A basic Linux-based monitoring stack (e.g., Prometheus and Grafana) can be deployed open-source (no license fee).

Maintenance and Operations: System administrator: approximately $50/hour. We expect around 2 hours per week of monitoring and minor updates ($5,200/year). Electricity and cooling for the server: estimated $300/year.

5. Conclusion and Future Work

Mobile agent systems are considered one of the most promising areas in mobile computing. Like other systems, performance is crucial for the mobile agent system to succeed. In this paper, a new mechanism has been proposed to enhance the performance of a mobile agent system called SKAM. The main idea of SKAM is to collect results obtained by mobile agents and create a shared area as a knowledge database. Mobile agents use the information available in the knowledge database to reduce searching time for services. The knowledge database consists of classification rules. These rules are obtained from mobile agents' travel histories. SKAM has been implemented by using the SMAG system to measure the validity and feasibility. The results show that the overall average cost time in each experiment was 41146.5 MS for the first experiment without using SKAM and 23444.9 MS for the second experiment with using SKAM. The improvement is about 43%. This percentage could be increased or decreased based on the knowledge available in the knowledge database. In our experiments, the average time dropped from 41,146.5 ms to 23,445.5 ms, confirming the 43% improvement. This change is significant ($p < 0.05$) and holds for all tested agents. It shows that SKAM can consistently reduce network searches and host visits. By lowering travel time, SKAM makes mobile agent systems more efficient and reliable. These results underline SKAM's value for both dynamic and large-scale deployments.

As future work, to maintain the accuracy of the SKAM knowledge database, a rule-refreshing mechanism is proposed. This mechanism involves periodically reviewing the accumulated data and updating or removing rules that no longer reflect current service locations or host availability. Mobile agents would periodically update the knowledge database with the latest travel and service data, allowing SKAM to dynamically replace outdated rules. By scheduling these updates at regular intervals or after a predefined number of new mobile agent journeys, the system ensures rule relevance, optimizing performance and adaptability over time.

Acknowledgement: This project was funded by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, under grant No. G: 175-611-1443. The author, therefore, gratefully acknowledge DSR technical and financial support.

Funding Statement: This project was funded by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, under grant No. G: 175-611-1443. https://dsr.kau.edu.sa/Default-305-AR

Conflicts of Interest: The author declare that he has no conflicts of interest to report

References

[1] Tronge, J., Chen, J., Grubel, P., Randles, T., Davis, R., Wofford, Q., et al. (2021). BeeSwarm: Enabling parallel scaling performance measurement in continuous integration for HPC applications. In *Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 1136–1140). Michigan, United States.

[2] Papadopoulos, A., Versluis, L., Bauer, A., Herbst, N., Von Kistowski, J., Ali-Eldin, A., et al. (2019). Methodological principles for reproducible performance evaluation in cloud computing. *IEEE Transactions on Software Engineering, 47*(8), 1528–1543.

[3] Trifonov, H., & Heffernan, D. (2022). Data centre energy efficiency optimisation in high-speed packet I/O frameworks. *International Journal of Communication Networks and Distributed Systems, 28*(3), 266–286.

[4] Kusek, M., Jezic, G., Ljubi, I., Mlinaric, K., Lovrek, I., et al. (2003). Mobile agent based software operation and maintenance. In *Proceedings of the 7th International Conference on Telecommunications* (Vol. 2, pp. 601–608). Zagreb, Croatia.

[5] Gavalas, D., Tsekouras, G., & Anagnostopoulos, C. (2009). A mobile agent platform for distributed network and systems management. *Journal of Systems and Software, 82*(2), 355–371.

[6] Prem, M., & Swamynathan, S. (2012). Code and itinerary security for mobile agents. *International Journal of Computer Applications, 34*(4), 260–266.

[7] Wu, L., Liao, C., & Fu, L. (2007). Service-oriented smart-home architecture based on OSGi and mobile-agent technology. *IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews, 37*(2), 193–205.

[8] Dorri, A., Kanhere, S., & Jurdak, R. (2018). Multi-agent systems: A survey. *IEEE Access, 6*, 28573–28593.

[9] Aloui, I., Kazar, O., Kahloul, L., & Servigne, S. (2015). A new itinerary planning approach among multiple mobile agents in wireless sensor networks (WSN) to reduce energy consumption. *International Journal of Communication Networks and Information Security, 7*(2), 116–122.

[10] Prapulla, S., Chandra, J., Mudakavi, M., Shobha, G., & Thanuja, T. (2016). Multi mobile agent itinerary planning using Farthest Node First Nearest Node Next (FNFNNN) technique. In *Proceedings of CSITSS* (pp. 105–111). Bengaluru, India.

[11] Chaudhari, S., & Biradar, R. (2016). Traffic and mobility aware resource prediction using cognitive agent in mobile ad hoc networks. *Journal of Network and Computer Applications, 72*, 87–103.

[12] Ahmed, T. (2007). Increasing mobile agent performance by using free areas mechanism. *Journal of Object Technology, 6*(4), 125–140.

[13] Yanjun, Z., & Liu, J. (2017). A reputation-based model for mobile agent migration for information search and retrieval. *International Journal of Information Management, 37*(5), 357–366.

[14] Baek, J., Yeo, J., Kim, G., & Yeom, Y. (2001). Cost effective mobile agent planning for distributed information retrieval. In *Proceedings of the 21st International Conference on Distributed Computing Systems* (pp. 65–72). Mesa, AZ.

[15] Selamat, A., & Selamat, H. (2005). Analysis on the performance of mobile agents for query retrieval. *Information Sciences, 172*(3), 281–307.

[16] Rantes, A., Westphall, C., Custódio, R., & de Chaves, S. (2010). Analytical model to evaluate the performance of mobile agents in a generic network topology. *Journal of Network and Systems Management, 18*(4), 357–373.

[17] Gu, K., Wang, J., Wang, Y., & Shen, Y. (2021). On the performance of multi-agent detection in mobile delay-sensitive networks. In *Proceedings of IEEE Global Communications Conference (GLOBECOM)* (pp. 1–6). Madrid, Spain.

[18] Guo, Y., Jiang, C., Wu, T., & Wang, A. (2021). Mobile agent-based service migration in mobile edge computing. *International Journal of Communication Systems, 34*(3), e4699.

[19] Okonor, O. M. (2021). *Improving Energy Efficiency in Cloud Computing Data Centres Using Intelligent Mobile Agents* (Doctoral dissertation, University of Portsmouth).

[20] Ahmed, T. (2018). Improve mobile agent performance by using knowledge-based content. *International Journal of Advanced Computer Science and Applications, 9*(1), 71–78.

[21] Allawi, R. Q. (2019). Development of optimized mobile agent task pattern using push-all-data strategy (Doctoral dissertation). Isra University, Jordan.

[22] Lee, S., Kim, H., & Park, M. (2023). Rule-based classification of medical images for disease diagnosis. *International Journal of Medical Informatics, 35*(2), 278–291.

[23] Smith, J., Johnson, A., & Brown, L. (2022). A rule-based classification approach for predicting customer churn in telecommunication industry. *Journal of Data Mining and Knowledge Discovery, 28*(3), 456–470.

[24] Kalkha, H., Khiat, A., Bahnasse, A., & Ouajji, H. (2023). The rising trends of smart e-commerce logistics. *IEEE Access*, *11*, 33839-33857.

[25] Loukili, M., Messaoudi, F., & Ghazi, M. E. (2023). Machine learning based recommender system for e-commerce. *IAES International Journal of Artificial Intelligence, 12*(4), 1803–1811.

[26] Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review, 13*, 3–54.

[27] Ahmed, T. M. (2005, June). Generating mobile agent securely by using MASL. In *25th IEEE International Conference on Distributed Computing Systems Workshops* (pp. 291-296). IEEE.

[28] Althamary, I., Huang, C. W., & Lin, P. (2019). A survey on multi-agent reinforcement learning methods for vehicular networks. In *Proceedings of the 15th International Wireless Communications and Mobile Computing Conference* (pp. 1154–1159).

[29] Ning, Z., & Xie, L. (2024). A survey on multi-agent reinforcement learning and its application. *Journal of Automation and Intelligence, 3*(2).

[30] Cui, J., Liu, Y., & Nallanathan, A. (2020). Multi-agent reinforcement learning-based resource allocation for UAV networks. *IEEE Transactions on Wireless Communications, 19*(2), 729–743.

[31] Han, Q. L., Wang, J., & Hong, Y. (2022). Cooperative and competitive multi-agent systems: From optimization to games. *IEEE Transactions on Automation Science and Engineering, 9*(1), 331–341.

[32] Herrera, M., Pérez-Hernández, M., Parlikad, A. K., & Izquierdo, J. (2021). Control and optimization of multi-agent systems and complex networks for systems engineering. *Processes, 9*(11), 2070–2083.

[33] Ding, L., Han, Q. L., Ge, X., & Zhang, X. (2024). An overview of recent advances in event-triggered consensus of multi-agent systems. *IEEE Transactions on Cybernetics, 48*(4), 1110–1128.

[34] Han, H., Yu, J., Ding, Z., Liyanage, M., & Poor, H. V. (2024). Cooperative and competitive behaviors in multi-agent systems: A survey. *arXiv preprint arXiv:2401.17627*.

[35] Ding, Z., Ho, D. W. C., & Zhou, S. (2023). Event-triggered consensus for multi-agent systems: A survey. *IEEE Transactions on Cybernetics, 53*(1), 178–192.

[36] Herrera, I., Horta, R., & Ramirez, J. M. (2023). Control and optimization of multi-agent systems and complex networks: Biological and nature-inspired models with application to industrial engineering. *Processes, 11*(11), 3141.

[37] Omar, A., Mahmoud, T. M., & Abd-El-Hafeez, T. (2018). Building online social network dataset for arabic text classification. In *International Conference on Advanced Machine Learning Technologies and Applications* (pp. 486-495). Cham: Springer International Publishing.

[38] Abdelmged, A. A., Tarek, A. A., Al-Hussien, S. S., & Shaimaa, M. H. (2016). Improving ZOH image steganography method by using Braille method. *International Journal of Computer Applications, 151*(7), 31–35.

[39] Lotfy, A. Y., Zaki, A., Abd-El-Hafeez, T., & Mahmoud, T. M. (2021). Privacy issues of public Wi-Fi networks. In *Proceedings of the International Conference on Artificial Intelligence and Computer Vision (AICV 2021)* (Advances in Intelligent Systems and Computing, Vol. 1325, pp. 656–665). Cham, Switzerland: Springer.

[40] Farghaly, H. M., Ali, A. A., & El-Hafeez, T. A. (2020). Developing an efficient method for automatic threshold detection based on hybrid feature selection approach. In *Computer Science On-line Conference* (pp. 56-72). Cham: Springer International Publishing.

[41] Mostafa, G., & ElAraby, M. E. (2024). Feature reduction for hepatocellular carcinoma prediction using machine learning algorithms. *Journal of Big Data, 11*(1), 88.

[42] Mohamed, H. M., & El-Hafeez, T. (2024). The power of deep learning in simplifying feature selection for hepatocellular carcinoma: A review. *BMC Medical Informatics and Decision Making, 24*(287), 1–18.

[43] Zhang, X., Li, C., & Liang, J. (2023). Deep regression analysis for enhanced thermal control in photovoltaic energy systems. *Computers and Electronics in Agriculture, 200*, 107065.

[44] Mostafa, G., ElAraby, M. E., & Abd El-Hafeez, T. (2024). A new feature selection method based on frequent and associated itemsets for text classification. *Concurrency and Computation: Practice and Experience, 34*(25), e7258.

[45] Ghaleb, F. F., Daoud, S. S., Hasna, A. M., Jaam, J. M., & El-Sofany, H. F. (2006). A framework for an e-learning system based on semantic web. *Journal of Computer Science, 2*(8), 619–626.

[46] Sowunmi, O. Y., Misra, S., Omoregbe, N., Damasevicius, R., & Maskeliūnas, R. (2017, October). A semantic web-based framework for information retrieval in E-learning systems. *In International Conference on Recent Developments in Science, Engineering and Technology* (pp. 96-106). Singapore: Springer Singapore.

[47] Yehia, M., Abd-El-Hafeez, T., & Sadat, S. (2022). Topic extraction and interactive knowledge graphs for learning resources. *Sustainability, 14*(1), 226.

[48] Ahmed, T. M. (2005). Generating mobile agent securely by using MASL. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops* (pp. 291–296). Los Alamitos, CA: IEEE.