International Journal on Semantic Web and Information Systems
Volume 18 • Issue 1

Tiny-UKSIE:
An Optimized Lightweight Semantic Inference Engine for Reasoning Uncertain Knowledge

Daoqu Geng, College of Automation, Chongqing University of Posts and Telecommunications, China*
Haiyang Li, College of Automation, Chongqing University of Posts and Telecommunications, China
Chang Liu, College of Automation, Chongqing University of Posts and Telecommunications, China

ABSTRACT

The application of semantic web technologies such as semantic inference to the field of the internet of things (IoT) can realize data semantic information enhancement and semantic knowledge discovery, which plays a key role in enhancing data value and application intelligence. However, mainstream semantic inference engines cannot be applied to IoT computing devices with limited storage resources and weak computing power and cannot reason about uncertain knowledge. To solve this problem, the authors propose a lightweight semantic inference engine, Tiny-UKSIE, based on the RETE algorithm. The genetic algorithm (GA) is adopted to optimize the Alpha network sequence, and the inference time can be reduced by 8.73% before and after optimization. Moreover, a four-tuple knowledge representation method with probability factors is proposed, and probabilistic inference rules are constructed to enable the inference engine to infer uncertain knowledge. Compared with mainstream inference engines, storage resource usage is reduced by up to 97.37%, and inference time is reduced by up to 24.55%.

KEYWORDS

IoT Edge Computing, Rete Algorithm, Semantic Inference Engine, Uncertain Knowledge

INTRODUCTION

List of Notations and acronyms:

- IoT: Internet of Things
- GA: Genetic Algorithms
- GSMA: Groupe Speciale Mobile Association
- OWL: Ontology Wed Language
- DL: Description Logic
- DS: Direct Style

DOI: 10.4018/IJSWIS.300826 *Corresponding Author

- RDF: Resource Description Framework
- RDFS: Resource Description Framework Schema
- RS: RDFS Style
- JVM: Java Virtual Machine

The IoT plays an important role in the new generation of information and communication technology. As more and more devices are connected to the Internet, the data generated by these IoT devices is also increasing. According to the report released by Groupe Speciale Mobile Association (GSMA) in 2019, the total number of global IoT connections reached 9.1 billion in 2018. The total number of global IoT connections is expected to reach 25.2 billion by 2025 (GSMA, 2019). Therefore, the effective use of data has become a research focus in recent years. More and more researchers are introducing Semantic Web technologies into IoT cloud servers to transform IoT data into knowledge. To solve the problem of semantic computing overload, in recent years, many researchers have tried to apply semantic web technology to IoT computing devices to share the computing tasks of servers. For example, semantic ontology is used in smart medical systems to improve the interoperability between medical devices and sensors (Rahmani, 2018); Semantic rules and sensor-based semantic ontology are used in an intelligent medical system to improve the robustness, efficiency, and comprehensibility of medical and health care system(Radhika, 2022) Semantic annotation and inference technologies are used in gateways (Al-Osta, 2019), and an event-driven model is designed to improve the efficiency of data processing. In the gateway system, the researchers realize the annotation and reasoning of sensor data streams and the real-time processing of events. Due to the limited resources of the IoT gateway, to reduce the weight of the annotated sensor data on networks, Urkude et al. (2021) established semantic data management by using semantic reasoner rules to reduce the number of triples from the semantic sensor data employing the unambiguous latent context information of a triple term. Kalatzis et al. (2019) propose a design principle for the specification of interoperability enabling solutions and verify the design principles in the experiment. Semantic annotation technology and the ontology are used in the edge devices of IoT to realize early warning of fire; Ali et al. (2017) design a lightweight ontology, which describes smartphones and sensors from different aspects, including platform, deployment, measurement functions, and attributes, data fusion and context modeling to realize the design of a smartphone sensor body for context-aware applications; the semantic inference technology is used in edge devices and cloud devices to realize stand-alone reasoning, distributed reasoning, mobile reasoning, and their synthetic reasoning (Ai, 2017), etc. There are many advantages to using Semantic Web technologies on IoT devices: 1) Reduce semantic computing tasks in the Cloud; 2) Reduce energy consumption in data transmission; 3) Improve the real-time performance of data processing; 4) Improve data security; 5) Enhance interoperability between IoT devices; 6) Develop more IoT solutions based on Semantic Web technologies, such as local event detection and early warning, fault handling, etc.

However, IoT computing devices usually have fewer storage resources and lower computing power, which cannot meet the operation of current inference engines that require higher storage resources and computing power. Therefore, current inference engines cannot be directly ported into IoT computing devices. On the other hand, IoT systems usually contain uncertain knowledge, and current reasoning engines do not support the representation and reasoning of uncertain knowledge, especially online real-time reasoning. Based on the above problems, the authors propose a lightweight semantic reasoning engine Tiny-UKSIE, which can realize the reasoning of uncertain knowledge, and is well suitable for resource-constrained IoT computing devices.

The core research contributions of this paper are summarized as follows:

- Since the mainstream semantic inference engines occupy more computing and storage resources and are not suitable for resource-constrained devices, a lightweight semantic inference engine

(Tiny-UKSIE) is proposed, which is very suitable for resource-constrained IoT devices and solves the problem that no inference engine is available for resource-constrained devices.
- Aiming at the difficulty of expressing uncertain knowledge, a four-tuple representation method with probabilistic factors is proposed, and the probability value is calculated by using the Bayesian formula and the total probability formula, which can well solve the problem of expressing uncertain knowledge in IoT applications.
- The proposed Tiny-UKSIE adopts the GA to optimize the alpha network of the RETE algorithm, and its inference time is shorter than that of the traditional inference engine based on the RETE algorithm.

BACKGROUND

Inference Engine of Semantic Web Technologies

Since the birth of the Semantic Web, many different inference methods have been represented. According to different ways of inference, they mainly can be divided into two types including Description Logic (DL) tableau inference engine and forward chaining rule-entailment inference engine.

DL tableau inference engine converts established OWL entailments to DL which satisfies the checking based on semantic connections between the two (Horrcks, 2003). The principle of the DL tableau is to convert the OWL ontology into a DL knowledge base. If a set of consistent conversion rules can be used to convert the DL knowledge base into a consistent model, then the conversion rules meet the DL knowledge base, otherwise, it is not. HermiT (Motik, 2009), Pellet (Evren, 2008), and FaCT++ (Tsarkov, 2006) are DL tableau inference engines. DL tableau inference engine uses hard coding in the execution process, which hinders the flexibility and scalability of this inference (Li F., 2020).

In the field of Semantic Web, the reasoning object of rule-entailments inference engines is usually ontology. In the process of reasoning, the rule-entailments inference engine uses a set of OWL semantic profiles to generate rules through forward-chaining and calculates the schema relationship between the OWL ontology triples and the rules. These rules are often termed semantic entailment rules (Tai, 2015). Moreover, the ontology inferred by the inference engine can be replaced at any time according to the needs of the application. As long as the inference rules are given, it can infer the given ontology. Therefore, this kind of inference engine has strong scalability and high flexibility. In addition, the rule-entailments inference engine is easy to understand, obtain and manage, and is widely adopted in the application field of Semantic Web. According to the different ways of interpreting OWL semantics, the rule-entailment inference engine can be regarded as direct style (DS) and RDFS style (RS). DS rules are mainly used for reasoning assertional data; however, RS rules can reason both terminological and assertional data, which is the reason that the authors chose to design a rule-based inference engine. O-DEVICE (Meditskos, 2008) and DLEJena (Meditskos, 2010) are Rule-entailment inference engines. Other inference engines usually combine the characteristics of the DL tableau inference engine and the Rule-entailment inference engine. Such as Minerva (Zhou, 2006), which combines a DL table approach with a resolution-based approach, Jena (Carroll, 2004), which incorporates a rule-implication approach with a resolution-based approach, and the Pellet rule processing engine, which incorporates forward linking rules.

Semantic Inference Engine Based on RETE Algorithm

Among the rule-based reasoning methods, the RETE algorithm is used in a variety of inference engines, including Drools, JESS, Jena, O-DEVICE. Every rule can be described by the formula (1), P_r represents the premise of the rule and C_r representing the conclusion of the rule:

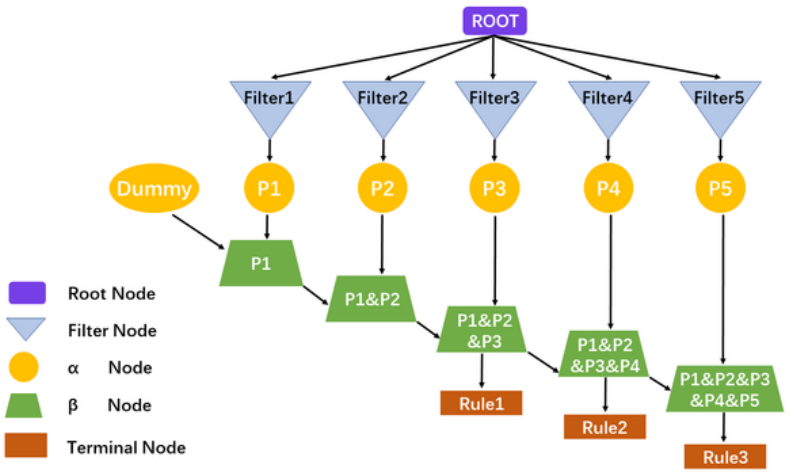
$$P_r \xrightarrow{r} C_r \tag{1}$$

RETE algorithm is a fast rule matching method, and its matching speed is independent of the number of rules (Bonatti, 2020). As long as the rule set is set sufficiently comprehensive, the relationship between knowledge in the ontology can be deeply explored. This is very suitable for the IoT application scenario with large amounts of data. The RETE algorithm transforms the ruleset into a RETE network for rule matching, and its network can be divided into Alpha network and Beta network. There are five types of nodes in the network: root nodes, filter nodes, alpha nodes, beta nodes, and terminal nodes (Forgy, 1982). As shown in Fig. 1, The root node is the entrance of the network, and all instance objects to be reasoned must enter the network based on the root node. The Alpha network consists of many α nodes, which are mainly used to find all triple instances that satisfy the premise of the rule. Each α node has a memory for storing triplet instances. In addition, each α node corresponds to a filter node, and these filter nodes are used to filter out incomplete triplet instances. The Beta network consists of β nodes, where β nodes are gradually constructed by α nodes. During the rule matching process, if the rules defined by the beta network are satisfied, the corresponding rules will be activated. The activated rules are stored in the terminal nodes.

Before describing the RETE algorithm, some Notations are defined. Let U denote the set of URI constants, B represents the set of blank nodes, V represents the set of variables, L represents the set of literals, and $T = U \cup V \cup B$ represents the set of terms. Let $TPI = \{(s,p,o) \mid s \in U \cup B, p \in U, o \in U \cup B \cup L\}$ denote the set of triple instances and $TPP = \{(s,p,o) \mid s \in U \cup V, p \in U \cup V, o \in U \cup V \cup L\}$ denote the set of triple patterns. Let R denote the set of rules and r denote one of the rules. So $r = (PRS, CNS)$ denotes a rule including a premise sequence (PRS) and a conclusion sequence (CNS), where $PRS = \{tpp_i \in TPP\}$ and $CNS = \{tpp_i \in TPP\}$. The notation $r.PRS.i$ denotes the *i-th* premise of rule r, and similarly, the $r.CNS.i$ denotes the *i-th* conclusion of rule r. Let $O = \{tpi \mid tpi_i \in TPI\}$ denote the ontology. Next, the Alpha node and Beta node are defined.

$tokenize()$ is a function that turns a triple instance into a token. An Alpha node corresponds to a specific token that comes from the return value of the $tokenize(tpp)$ function, written as α_i^{tpp},

Figure 1. The structure of the RETE network (Forgy, 1982)



where *i* represents the *i-th* Alpha node. Then the α_i^{tpp} is defined as for formula (2). The Alpha Network can be obtained by operating formula (2) on each tpp in the ruleset:

$$\alpha_i^{tpp} = \{tokenize(tpp) \mid tpp \in r.PRS \wedge r \in R\} \tag{2}$$

The *i-th* Beta node of rule r is defined recursively as for formula (3), where $\oplus$ represents the joining operation, which means that adds the current Alpha node to the other network. The authors can see that each Beta node is composed of the previous Beta node and the next Alpha node in the network:

$$\beta^{r.i} = \begin{cases} \alpha^{r.PRS.i} \oplus \alpha^{r.PRS.(i+1)}, & i = 1 \\ \beta^{r.(i-1)} \oplus \alpha^{r.PRS.(i+1)}, & 2 \le i \le \left|r.PRS\right| - 1 \end{cases} \tag{3}$$

After the Alpha network and the Beta network are constructed, the rule matching can be performed. The steps of rule matching include:

Step 1: tpi enters the RETE network through the root node, and is stored in the alpha cache after being filtered by the filter nodes.
Step 2: If the current α node is the starting point of the match, skip to step 3, otherwise skip to step 4.
Step 3: Find its public node based on an alpha node and another alpha node with a common connection, which is the β node, and put the matched triple instance into the β memory. If there is a matching complete rule, then activate the rule.
Step 4: Find the next connection node of the current β node and the next α node with a matching relationship, whose common connection node is the next β node, and put the matched triple instance into the β memory. If there is a complete rule that matches, then activate the rule.
Step 5: If there is no next matching β node, then the matching ends. Otherwise, repeat step 4.

The Improved Method for RETE Algorithm

The node sharing and state caching of the RETE algorithm greatly improve the matching efficiency, but it also has many disadvantages:(1) There is a problem of repeated state storage. For example, the fact that patterns A and B have been matched must be stored in the node caches of pattern A and pattern B at the same time, which will take up more space and affect the matching efficiency. (2) The processing logic is limited, only supporting first-order Boolean logic. (3) It is inefficient when dealing with large-scale data and rapidly changing data. In the study (Tai, 2015), an inference engine composition approach is proposed for problem (1) which includes a selective rule loading algorithm and a Two-phase RETE algorithm. The selective rule loading algorithm is that only when each premise of a rule exists in the target ontology or as a follow-up premise of other rules, the rule can be regarded as a candidate rule to be selected. Through this algorithm, unnecessary rules can be reduced, thereby reducing the complexity of the RETE network. The Two-phase RETE algorithm includes a most specific condition first and a pre-evaluation of join connectivity to diminish the RETE network. The study in (Van Woensel, 2019) proposed an OWL2 RL optimization including leaving out redundant rules, dividing the rule set based on rule purpose and references, and removing resource-heavy rules to reduce the size of the Beta network. The researchers in (Xin, 2017) propose an improved RETE algorithm using a shared degree model. This method gives each Alpha node a share value by counting the number of occurrences of Alpha nodes in the Alpha network and arranges the Alpha network in descending order of the share value. However, when there are a large

number of Alpha nodes with the same shared degree value, the effect of this shared degree model will be significantly reduced. To solve this problem, the authors have carried out in-depth research and proposed improvement strategies.

Methods of Uncertain Knowledge Representation and Reasoning

The traditional Semantic Web technologies are not able to represent and reason with uncertain knowledge which is one of the characteristics of the world. The study (Li, 2019) shows an evolutionary reasoning method of emergency plan based on ontology clustering and using a Bayesian network to realize conditional probabilistic reasoning. The researchers in (Fareh, 2019) propose a method of modeling incomplete knowledge in the classical OWL ontology using Bayesian networks, which include five steps: (a) Extracting terminological and assertion parts. (b) Constructing structure of Bayesian network. (c) Constructing statistical table with incomplete knowledge. (d) Learning parameter with EM algorithm. (e) Inference in Bayesian network. The study in (Hlel, 2018) shows a new method of constructing probabilistic ontology by integrating uncertainty to elements of an OWL ontology, which includes four phases: (a) Specification of requirements which means determining the domain and the purpose of ontology. (b) Building classic ontology. (c) Determination of probabilities that will be associated with the knowledge of classic ontology. (d) Integration of uncertainty to classic ontology. The researchers in (Setiawan, 2019) represent a framework called ByNowLife, which is a novel approach for integrating BN with OWL by providing an interface for retrieving probabilistic information through SPARQL queries. They think that the system must use a separate knowledge base, separate processing, or third-party applications so that adapts to systems that already have a running ontology. The study (Riali, 2019) shows a fuzzy probabilistic ontology (FPO) which copes with probabilistic events even with incomplete evidence and models events whose results cannot be predicted with certainty (randomness) and provides a fuzzy probabilistic inference under fuzzy evidence. To sum up, the above researches are devoted to extending the existing ontologies and realizing the representation of uncertain knowledge. However, their disadvantage is that they cannot reason about uncertain knowledge dynamically. To solve this problem, the authors optimize the RETE network structure and add a probability calculation module to realize the dynamic inference of probability.

THE DESIGN OF TINY-UKSIE

Optimizing Alpha Network by Using GA

As mentioned in section “The Improved Method for RETE Algorithm”, although the shared degree model in (Xin, 2017) can reduce the size of the Beta network, when there are a large number of Alpha nodes with the same shared degree, this method cannot determine the specific order of these Alpha nodes. Therefore, in Tiny-UKSIE the authors adopted the shared degree model and used a GA algorithm to optimize the order of the wnodes with the same sharing degree. the GA contains six processes:(1) Encoding; (2) Initializing the population; (3) Evaluating the individual; (4) Selecting individual; (5) Cloning individuals; (6) Crossover among individuals; (7) Individual variation. Before the GA algorithm is described, notations are defined: Firstly, let $\alpha = (tpp_1, tpp_2, tpp_3, \ldots, tpp_{n-1}, tpp_n)$ denote a normal sequence of an Alpha network, and α_{std} denote the sequence of an Alpha network which is obtained by sorting α using the shared degree model. Moreover, define $EQsubseq_i$ as the *i-th* subsequence of α_{std}, where each triple pattern in the $EQsubseq_i$ has the equal shared degree, $i = 1,2,\cdots,m$, and m is is the number of subsequences of α_{std}. Lastly, let $Chrs$ denote the set of chromosomes, $Chrs.i$ represent the *i-th* chromosome, and N_{chr} denote the size of the population.

- **Encoding:** The encoding result is to treat the sequence of α_{std} as a chromosome and a triple pattern as a gene.
- **Initializing the population:** In this step, N_{chr} chromosomes are initialized based on α_{std}. Specifically, these chromosomes can be obtained by changing the order of all triple patterns in an $EQsubseq_i$ respectively, where $i \in \{1,2,\cdots,m\}$.
- **Evaluating the individual:** In this step, a fitness function is used to evaluate the chromosome and return a percentage denoting the fitness of the current chromosome. In this evaluation, the running time of one matching and the memory consumption of the IoT device are recorded. The shorter the time and the smaller the memory consumption, the higher the fitness score of the current chromosome.
- **Selecting individual:** In this step, individuals with high fitness are selected, while those with low fitness are eliminated.
- **Cloning individual:** In this step, individuals are randomly cloned to expand the population size to N_{chr}.
- **Crossover among individuals:** At this stage, individual pairs and crossover points are randomly selected and the two chromosomes in the individual pair are crossed to generate a new chromosome.
- **Individual variation:** In this step, individuals are randomly selected and random genes in $EQsubseq_i$ are changed to generate new individuals, where $i \in \{1,2,\cdots,m\}$.

Before describing the process of GA, the authors define α_{targ} as the optimal Alpha network, $F_{\exp}$ as the expected value of the fitness evaluation, F_{cur} as the fitness valule of the current individual, and F_{best} as the best fitness value of the current population. Moreover, let N_{cur} represent the current size of the population and P_{cur} denote the current population. Let Ite_{cur} represent the current number of iterations and $Ite_{\max}$ the maximum number of iterations. The steps of GA optimized Alpha Network are shown in Algorithm 1. To obtain the α_{targ}, firstly, the shared degree of Alpha nodes is counted to obtaine a sequence $\alpha^{'}$ (corresponding function $statisticsShareDgree(\alpha)$), and the $\alpha^{'}$ sequence is sorted in descending order (corresponding function $descendingSort(\alpha^{'})$) to generate the α_{std} network. Then the number of population is expanded from one to N_{chr} by using α_{std} (corresponding function $generateChromosome(\alpha_{std})$), and the current population $P_{cur} = \{\alpha_1, \alpha_2, \cdots, \alpha_i, \cdots \alpha_{N_{chr}}\}$, where $i = 1,2 \cdots N_{chr}$. Furthermore, a fitness evaluation function for each individual is performed (corresponding function $fitnessEvaluation(\alpha_i)$). Once the value of the fitness evaluation reaches $F_{\exp}$ or the number of iterations reaches the upper limit, the algorithm terminates. Otherwise, it continues. Next, the roulette wheel selection method is adopted to eliminate individuals (corresponding function $eliminatePopulation(P_{cur})$), and the individuals are randomly cloned until the number of populations is up to N_{chr} (corresponding function $clone(P_{cur})$). Then, two chromosomes and an intersection of the two chromosomes in the population are randomly selected. Subsequently, the two chromosomes are crossed (corresponding function $crossover(P_{cur})$). In the next step, two genes on a chromosome are arbitrarily chosen and exchanged as a way to mutate (corresponding function $variation(P_{cur})$). Finally, each individual is evaluated again and the steps are looped until the loop conditions are not met. Then the algorithm ends.

Table 1. Algorithm 1: GA optimized Alpha Network

Input: α , the original Alpha sequence generated by rule set

Output: α_{targ} , the optimal Alpha network

$\alpha_1 = statisticsShareDgree(\alpha)$

$\alpha_{std} = descendingSort(\alpha_1)$

$N_{cur} = 1$

for $N_{cur} \le N_{chr}$ **do**

    $\alpha_{N_{cur}} = generateChromosome(\alpha_{std})$

    $N_{cur} = N_{cur} + 1$

end for

$Ite_{cur} = 1$

$F_{best} = 0$

$endwhileflag = false$

while $Ite_{cur} \le Ite_{\max}$ **do**

    $i = 1$

    for $i \le N_{chr}$ **do**

        $F_{cur} = fitnessEvaluation(\alpha_i)$

        if $F_{cur} \ge F_{\exp}$ **then**

            $\alpha_{targ} = \alpha_i$

            $endwhileflag = true$

            break

        end if

        if $F_{cur} > F_{best}$ **then**

            $F_{best} = F_{cur}$

            $\alpha_{targ} = \alpha_i$

        end if

        $i = i + 1$

continued on following page

Table 1. Continued

```
end for
if endwhileflag == true then
    break
end if
eliminatePopulation(P_cur)
clone(P_cur)
crossover(P_cur)
variation(P_cur)
Ite_cur = Ite_cur + 1
end while
```

Proposed Four-tuple With Probability Factors and Probability Rules

The representation and reasoning of uncertain knowledge can better express the real IoT system. Many methods have been proposed and studied described in section "Methods of Uncertain Knowledge Representation and Reasoning". However, current research is insufficient. For example, (Li, 2019) and (Riali, 2019) respectively studied the Bayesian network and ontology for OWL DL, rather than OWL RS. Moreover, they calculate and store the probability value through documents. This is not suitable for IoT application scenarios where data changes dynamically. In our study, the authors propose a method of using the four-tuple to express possibility knowledge and define novel probability rules to realize online probability calculation in the semantic inference engine.

In the classic semantic technologies, the triples are used to represent the knowledge, such as $?c, rdf:subClassOf, ?d$ representing that subject *c* is a subclass of predicate *d*, $?x, rdf:type, ?c$ representing that the type of x is the same as c, and $?x, rdf:type, ?d$ representing that the type of x is the same as x. Therefore, these three triples make up a rule seen as formula (4), where the left side of the arrow indicates the premises (PRS) and the right side indicates the conclusions (CNS):

$$(?c, rdf:subClassOf, ?d) \wedge (?x, rdf:type, ?c) \rightarrow (?x, rdf:type, ?d) \tag{4}$$

However, the triples cannot represent uncertain knowledge, so the authors proposed a method that uses the four-tuple to represent uncertain knowledge. Let $ftp = \{?s,?p,?o,?pb\}$ denote the four-tuple. Compared with the triples, the four-tuple add a probability factor at the end. For example, $?a,leadto,?b,?pb$ represents that the occurrence of the subject (a) can lead to the occurrence of the object (b), and the probability value of occurrence of *b* under the condition that *a* occurs is *pb*. In mathematics, the probability factor is defined as $pb{=}P(b \mid a)$. To be compatible with triples, the authors split the four-tuple into two parts: the triple part and the probability factor part. the authors

can also express ftp as $ftp = \{(?s,?p,?o),(?pb)\}$. Therefore, these two equations $ftp.left = tpp = (?s,?p,?o)$, $ftp.right = (?pb)$ are established.

Let's introduce the method with an example. The authors assume a scenario that the robotic arm is malfunctioning. And the authors define the concept of causing robotic arm failure and the problems caused by robotic arm failure. And the simple Bayesian network of this scenario is shown in Fig. 2.

Next, the authors describe the method for constructing the probabilistic rule. These can be seen that three factors, the event A_1 (denoting locating sensor failure), A_2 (representing program bug), A_3 (denoting controller output failure) can lead to the event B (denoting robotic arm location failure). Similarly, the event B is also a factor that can lead to C_1 (denoting unqualified product) and C_2 (denoting production accident). Therefore, the authors can define two types of rules to calculate the probability. One is, when event B occurs, used to calculate these probabilities of the event set ($A_1, A_2 \cdots\cdots A_n$) which can lead to the occurrence of event B. The other is, when event B does not occur, used to calculate the probability of event B resulting from the occurrence of every event in the set $A_1, A_2 \cdots\cdots A_n$. The rules are shown in Tab. 2.

To calculate the value of the probability factor in probability rules, the authors divide the probability rules into two categories: (1) Total probability model; (2) Bayesian rule model. The total probability model is used for multiple causes (multiple events) leading to one result (one event) (Mughal, 2021), and the Bayesian model is used for the analysis of each cause under the condition that the result occurs (Cantone, 2021). The probability calculation formula is as follows:

$$P(B) = \sum_{i=1}^{n} P(A_i)P(B \mid A_i) \tag{5}$$

When the set ($A_1, A_2 \cdots\cdots A_n$) is the complete set leading to event *B*, the formula (5) can be used to calculate the probability of event *B*. When looking for the cause of event *B* under the condition that event *B* has already occurred, formula (6) can be used to calculate the probability:

$$P(A_i \mid B) = \frac{P(A_i)P(B \mid A_i)}{\sum_{j=1}^{n} P(A_j)P(B \mid A_j)} \tag{6}$$

In the probabilistic rule, the authors use the four-tuple pattern $?s,:has\,\mathrm{Pr}obabilityValueOf,?pb,?pb$ to represent the probability of the occurrence of the subject $?s$ and use $?s,:leadTo,?o,?pb$ to

Figure 2. The relationships of robotic arm failure concepts

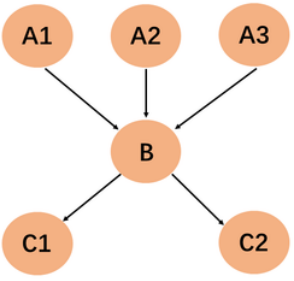


Table 2. Rule 1 (Left) and Rule 2 (Right)

Rule1	Rule2
PRS:	***PRS:***
? A_1 ,:leadTo, ?B, ?pb ∧	*? A_1 ,:leadTo, ?B, ?pb* ∧
? A_2 ,:leadTo, ?B, ?pb ∧	*? A_2 ,:leadTo, ?B, ?pb* ∧
? A_3 ,:leadTo, ?B, ?pb ∧	*? A_3 ,:leadTo, ?B, ?pb* ∧
? A_1 ,:hasprobabilityValueOf, ?pb, ?pb ∧	*? A_1 ,:hasprobabilityValueOf, ?pb, ?pb* ∧
? A_2 ,:hasprobabilityValueOf, ?pb, ?pb ∧	*? A_2 ,:hasprobabilityValueOf, ?pb, ?pb* ∧
? A_3 ,:hasprobabilityValueOf, ?pb, ?pb ∧	*? A_3 ,:hasprobabilityValueOf, ?pb, ?pb* ∧
?B,:ishappened, true	*?B,:ishappened, false*
CNS:	***CNS:***
?B, causedBy, ? A_1 , ?P(A_1 |B)	*?B,:hasProbabilityValueOf, ?P(B), ?P(B)*
?B, causedBy, ? A_2 , ?P(A_2 |B)	
?B, causedBy, ? A_3 , ?P(A_3 |B)	

indicate that the event $?s$ can lead to the event $?o$ and the probability of the event $?o$ occurring under the condition that the event $?s$ occurs is *?pb*. To simplify the representation, probability symbols are used in the probability rules to represent the corresponding value of probability.

The authors have designed two probability rules, Rule 1 and Rule 2, as shown in Tab. 1. They correspond to the Bayesian model and total probability model respectively. When calculating the probability rule, Rule 1 uses the formula (6), and Rule 2 uses the formula (5). For the example shown in Figure 2, when calculating the probability of occurrence of A_1 under the condition that B occurs, the formula (7) can be used. The method for calculating the probability of occurrence of other factors such as A_2 is the same as that for A_1. Ruler 2 implement the reasoning from the causes (A_1 , A_2 , A_3) to the result(B), and the probability formula (8) is used to calculate the probability of the result B :

$$P(A_1 \mid B) = \frac{P(B \mid A_1)P(A_1)}{P(B \mid A_1)P(A_1) + P(B \mid A_2)P(A_2) + P(B \mid A_3)P(A_3)} \tag{7}$$

$$P(B) = P(B \mid A_1)P(A_1) + P(B \mid A_2)P(A_2) + P(B \mid A_3)P(A_3) \tag{8}$$

The Implementation of Tiny-UKSIE

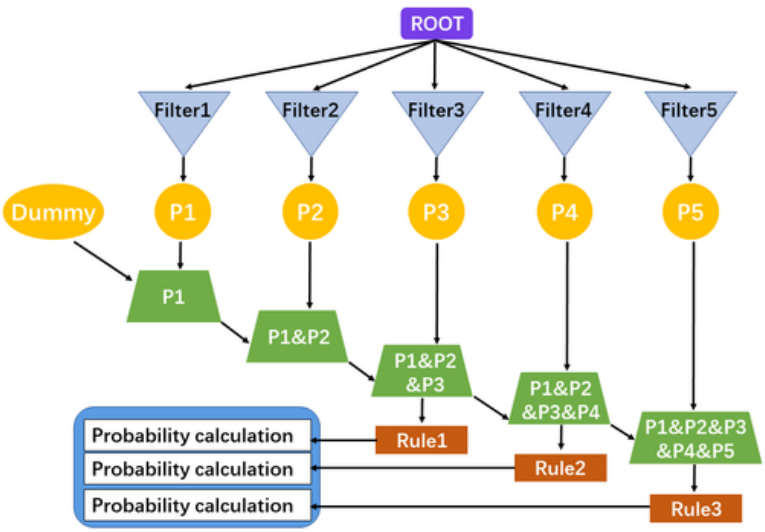
The implementation of the classic RETE network requires some modifications due to the introduction of quaternions and probabilistic rules. These changes mainly include three aspects:

1. **Adjusting the storage structure of nodes:** Due to the introduction of the four-tuple with the probability factor, it is necessary to build Alpha and Beta networks suitable for the four-tuple. Specifically, the α memory corresponding to the α node will store the matched four-tuple, and the β memory corresponding to the β node will also store the matched four-tuple. Therefore, the authors adjusted the storage structure of two types of nodes α and β, respectively, and realized the storage of quadruples.
2. **Adding a probability calculation module:** To realize the reasoning of uncertain knowledge, the authors add a probability calculation module based on Fig. 1, which is used to calculate the probability of uncertain knowledge. As shown in Fig. 2, the structure of the proposed RETE network remains unchanged, and still includes two parts: alpha networks and beta networks. Moreover, the probability calculation module is linked with the rule activation module. In the process of rule matching and inference, probabilistic rule detection is performed immediately after the rule is activated. If the activated rule is probabilistic, the corresponding probability model is selected for probability calculation.
3. **Tailoring and Designing Inference Engines:** Based on the mainstream inference engine architecture, auxiliary functions are tailored to achieve a lightweight design. Specifically, the authors removed these functions including ontology construction, knowledge graph storage, and consistency check, then implemented the inference engine using C++ language[1]. All the source codes are available in GitHub[2].

EXPERIMENT AND EVALUATION

In this section, the authors implement Tiny-UKSIE and apply it to an edge gateway in IoT for performance comparison tests such as resource usage and inference time, and then analyze the experimental results.

Figure 3. The proposed structure of the RETE network



Experimental Environment and Architecture

The architecture of the experimental system is shown in Fig. 4, including the wireless sensor network, gateway, and server. The gateway is designed based on an EXYNOS-4412 processor with 2G memory, 1.5GHz clock frequency, 4G eMMC, and running Linux 3.5 operating system. The server model is A840-G1012, with 512 GB of memory and running CentOS 6.5 operating system. Data from the sensor network is transmitted to the gateway via wireless communication, and the data is processed by the gateway and transmitted to the server. Specifically, the gateway includes modules for multi-protocol adaptation, data collection, ontology module, inference engine (Tiny-UKSIE), event processing, etc. Based on semantic processing technology, it realizes the update of sensor network ontology and industrial equipment ontology, data annotation, and inference based on built-in rules and custom rules. In addition, the ontology used in the gateway is built on the server-side.

Experimental Preparation

Building Ontology

The underlying sensor network mainly detects AGV robot status data including speed, rotation angle, load, etc., the mechanical arm status data includes six-axis angle data and torque data, etc., the station status data includes load, power, working status, etc., and environmental status including temperature, humidity, harmful gas, etc. The construction tool of ontology is Top Braid Composer (TBC)[3]. The ontology view of sensor networks and industrial equipment based on Gruff software (a visualization software for RDF) is shown in Fig. 5. In the construction of the ontology, some semantics standards of W3C[4] were cited, such as using the Web Ontology Language 2 (OWL2[5]) and the related concepts defined in Semantic sensor network (SSN[6]) ontology, and the name-space like the SOSA[7], SSN, and Time[8].

Conversion From Triples to Four-Tuples

In the section "Proposed Four-tuple with Probability Factors and Probability Rules", the concept of the four-tuples is proposed, but the ontology constructed by TBC software is a classic representation

Figure 4. The structure of the low-speed wireless sensor network gateway based on IPv6

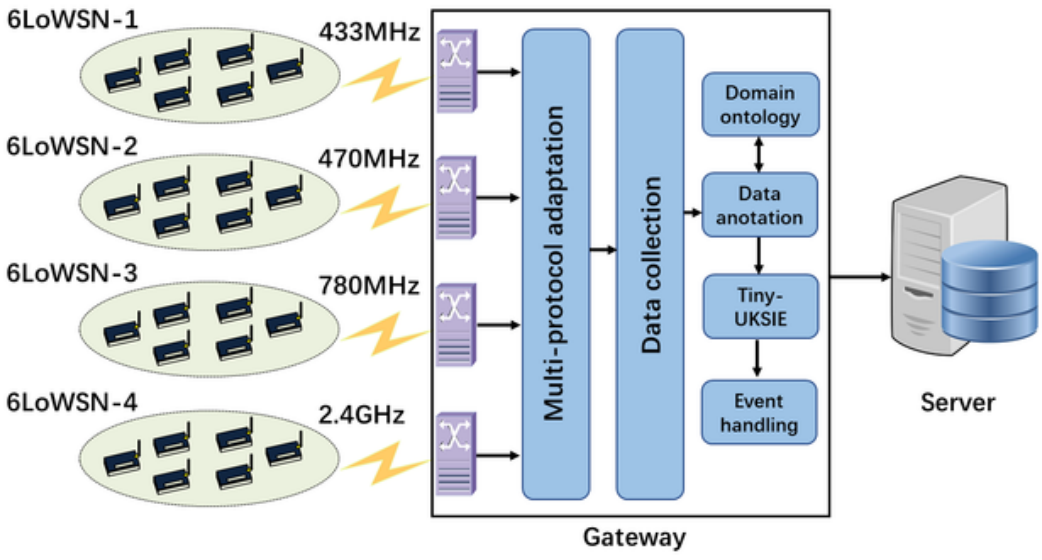
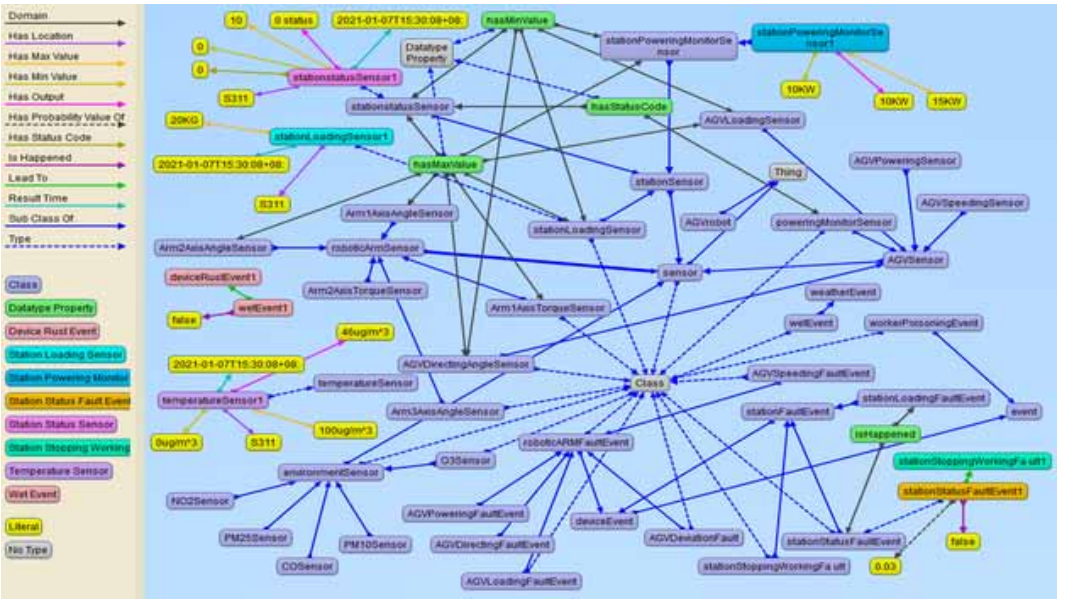


Figure 5. Ontology-based on experimental environment



of triples, so it is necessary to convert the triple ontology into the quadruple ontology. The conversion principle is to set the probability factor in the possible knowledge to its true probability value, while the probability factor in the non-possibility knowledge is set to a constant "1". Tab. 3. Shows the example for the conversion from triples to four-tuples. Specifically, choosing one of them from the table, such as $A_1, leadTo, B$ and $A_1, leadTo, B, 0.005$, the probability factor in the non-possibility knowledge has been set to "0.005".

Data Annotation and Rule Set Settings

The gateway receives data transmitted from the underlying sensor network every 100 milliseconds and performs data annotation and inference once, which in turn uses the inference results as a reference for policy execution. Data annotation is the process of combining sensor data with the corresponding

Table 3. Example for the conversion from Triples to Four-tuples

Triples	Four-tuples	Element type
tempSenor1,instanceOf,tmepSensor	***tempSenor1,instanceOf,tmepSensor,1***	**instance**
tempSensor,subClassOf,Sensor	*tempSensor,subClassOf,Sensor,1*	class
hasOutput,type,DatatypeProperty	*hasOutput,type, DatatypeProperty,1*	property
tempValue,owl:maxCardinality,125	*tempValue,owl:maxCardinality,125,1*	relation
rdf:type,rdfs:range,rdfs:Class	*rdf:type,rdfs:range,rdfs:Class,1*	constraint
rdfs:domain,rdfs:range,rdfs:Class	*rdfs:domain,rdfs:range,rdfs:Class,1*	axiom
$A_1, leadTo, B$	$A_1, leadTo, B, 0.005$	instance
$A_1, ishappened, true$	$A_1, ishappened, true, 1$	instance
B,hasprobabilityValueOf,0.005	*B,hasprobabilityValueOf,0.005,1*	instance

data variables of four-tuple knowledge. In the implementation, the hash map is used to store the sensor ID and the corresponding value. When the ontology knowledge needs to be updated, the value (current sensor data) is obtained by looking up the corresponding key in the hash map. The setting of rule set follows the semantic Standard Specification of W3C, uses the relevant basic reasoning rules stipulated by W3C, adds custom probability rules, and finally converts it into an expression suitable for four-tuple reasoning.

Genetic Algorithm Settings

Before each execution of rule inference, Tiny-UKSIE first determines whether there is an unoptimized rule set, and if not, executes rule inference directly. Otherwise, the GA algorithm optimizer will be started to optimize the Alpha network. During the execution of the GA algorithm, to reduce the impact of computer resource allocation on the inference time, 10 inferences will be executed for each fitness evaluation, and the number of iterations is set to 200. The parameters of the related GA algorithm are set as follows: there are 8 individuals in the population, the selection rate is set to 0.25, the crossover rate is set to 0.25, and the mutation rate is set to 0.125.

Program File Preparation and Execution

During the experiment, programs such as Tiny-UKSIE inference engine and data annotation will be executed, and these programs will run in the gateway shown in Fig. 4. First, the inference engine and other programs will be compiled by the cross compiler to generate executable programs suitable for running on the ARM-32-bit gateway. At the same time, the ontology files and rule files required for the inference engine execution will be copied to the gateway. In addition, when the Tiny-UKSIE inference engine runs, it also needs to record the time and memory it consumes. In the experiment, two mainstream reasoning engines, Jena[9] and RDF4J[10], are selected for comparison. The same functions need to be implemented when these reasoning engines are running, including data annotation, rule-based reasoning, and calculating their time and memory consumption.

Tiny-UKSIE Evaluation

Optimization Effect Evaluation of GA Algorithm

The GA algorithm is used to find the optimal Alpha network structure to optimize the RETE algorithm. The experimental conditions were set as follows: the authors formulated three rule sets, and the number of rules contained in the three rule sets was 450 (RuleSet1), 210 (RuleSet2), and 49 (RuleSet3), respectively. Moreover, the number of $EQsubseq$ in the three rule sets was 82 (RuleSet1), 35 (RuleSet2), and 7 (RuleSet3), respectively. Furthermore, the number of iterations was set to 200 for all three experiments, and in each experiment, the best individual was counted once every 10 iterations.

Fig. 6 shows the variation curve of the inference time with the number of iterations for the three rule sets. It can seen that the change of reasoning time tends to be stable after about 150 iterations. Specifically, the GAvalgorithm has an obvious optimization effect on the RuleSet1, and the reasoning time varies greatly between the beginning and end of optimization. Especially in the first 50 iterations, the reasoning time decreases significantly. However, when the number of iterations reaches about 120, the reasoning time decreases slowly. Compared with RuleSet1, the optimization effect of the GA algorithm on RuleSet2 is less significant. After the first 30 iterations, the reasoning time begins to change slowly. Unfortunately, the GA algorithm has the worst optimization effect on RuleSet3. After 200 optimization iterations, the change of inference time is still not significant.

The authors counted the time before and after optimization of the GA algorithm for the three rule sets, and the statistical results are shown in Tab.4. To reduce the impact of dynamic allocation of computing resources on the inference time and memory consumption of the inference engine, each time in the experiment is the sum of the time to perform 100 inferences. From the table, it can be seen that the more the number of patterns with the same degree of sharing in a rule set, the more

Figure 6. Optimization results of GA algorithm for inference time

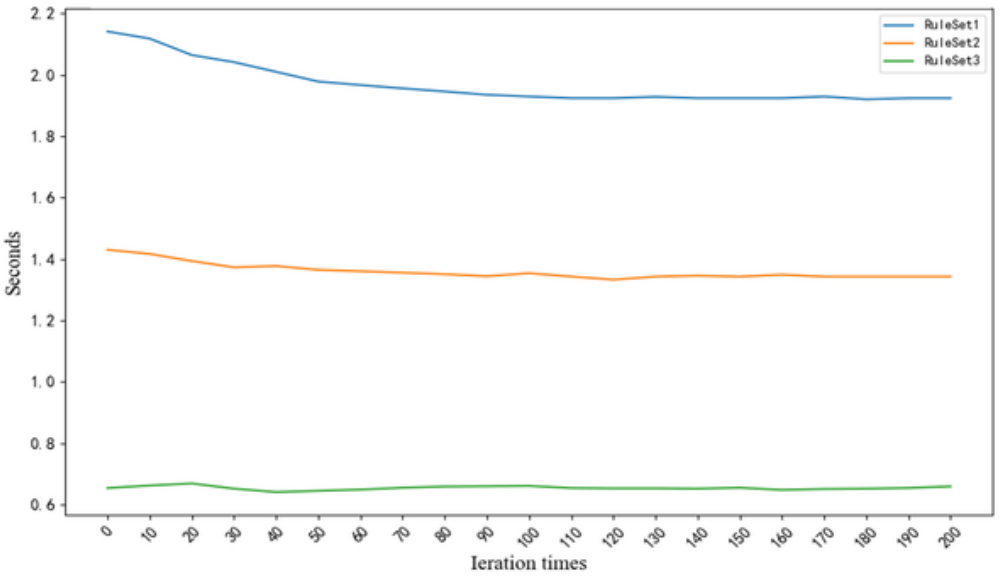


Table 4. Comparison of time optimization of three rule sets

Ruler set	The number of rules	The number of EQsubseq	Inference time at the beginning (ms)	Inference time at the ending (ms)	Optimization ratio (%)
Rule set 1	450	82	2141 ms	1954 ms	8.73%
Rule set 2	210	35	1436 ms	1342 ms	6.55%
Rule set 3	49	7	657 ms	644 ms	1.98%

effective the GA algorithm is in optimizing the rule set. On the contrary, the fewer the number of patterns with the same degree of sharing, the less effective the GA algorithm is in optimizing the rule set. After statistical analysis, the authors can conclude that when the rule set contains a large number of four-tuples with the same degree of sharing, such as RuleSet1, the inference time is reduced by up to 8.73% when the rule set is optimized by GA algorithms.

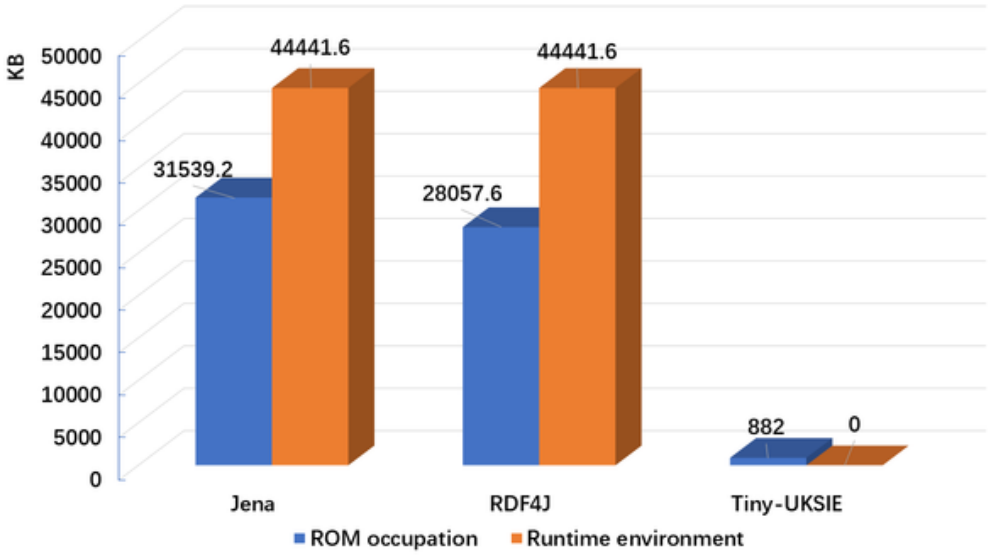
Comparison of ROM Occupation

In current IoT applications, most IoT nodes are resource-constrained. Especially in terms of storage capacity, there is a big gap with cloud servers. Therefore, to realize the semantic IoT application, the resource occupation of the inference engine, especially the storage resource occupation, is the focus of attention. The authors compare the designed Tiny-UKSIE inference engine with Jena and RDF4J, two mainstream inference engines, and the ROM occupation is shown in Tab. 5. The comparison of ROM occupation of the three inference engines is shown in Fig. 7. It can be seen that the ROM occupation of the Tiny-UKSIE inference engine is extremely small, only 882KB. Compared with the other two mainstream inference engines, the storage resource occupation is reduced by as much as 97.2%.This is due to two reasons: first, the authors cut some non-essential functions for the target usage scenarios; second, Tiny-UKSIE is implemented in C++ programming language, which can

Table 5. Comparison of ROM occupation of three inference engines

Inference Engine	ROM occupation	Runtime environment
Jena	30.8 MB	Corresponding JRE environment (43.4MB)
RDF4J	27.4 MB	Corresponding JRE environment (43.4MB)
Tiny-UKSIE	882 KB	No need for the additional runtime environment

Figure 7. Comparison of ROM occupation of three inference engines

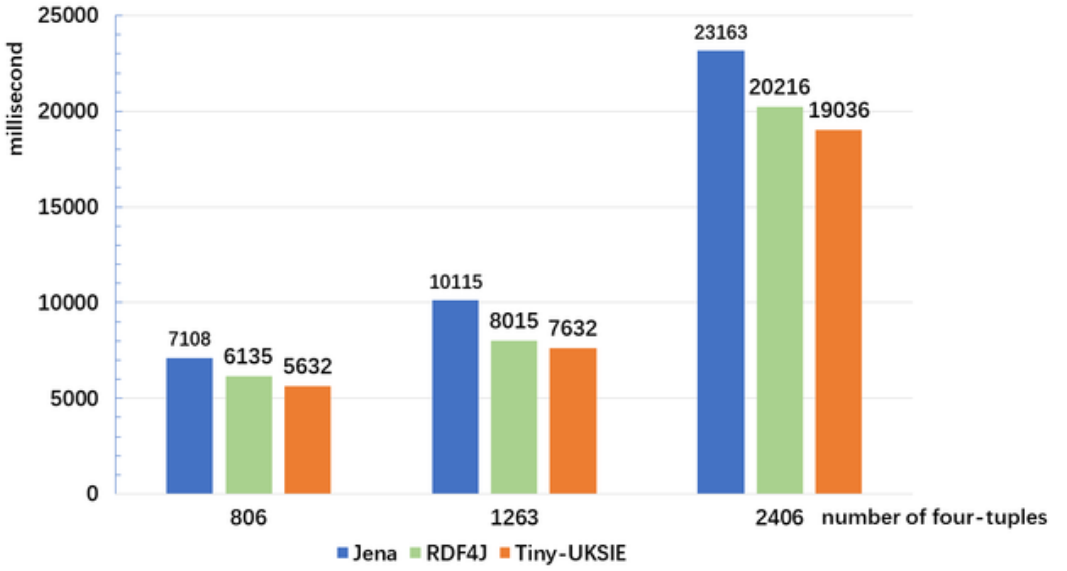


be directly run in Linux OS or IoT devices without OS, reducing the additional storage resource occupation due to the running environment configuration. However, both Jena and RDF4J inference engines are implemented in Java language programming, which requires the configuration of the running environment. This will result in more storage resource usage, such as Jena and RDF4J, which both require 43.4MB of additional storage space. In practical applications, when the device ROM resources are fixed, the smaller the program code of the inference engine, the lower its occupancy of the device ROM resources. That is, Tiny-UKSIE has a low ROM resource occupancy rate. The device will have more ROM resources to store other functional program code, which is more conducive to the function expansion of the application device. Therefore, overall, Tiny-UKSIE is more suitable to run on IoT edge computing devices with limited ROM resources.

Comparison of Inference Time

Inference time is a key metric for evaluating inference engines, which reflects the execution efficiency of inference engines. Especially in application scenarios with dynamic data changes and high real-time requirements, which have high requirements on inference efficiency. For this experiment, the authors use three numbers of ontology instances, and the number of four-tuples is 806, 1263, and 2406, respectively. Since the single inference time of each inference engine is small, to reduce the impact of resource allocation on the single inference time in the gateway system, the authors set the number of inferences for each ontology by the three inference engines to 100 and record the total time of 100

Figure 8. Comparison of the inference time of three inference engines



inferences. The statistical results are shown in Fig.8. It can be seen from the figure that Jena has the longest inference time and tiny-UKSIE has the least inference time in all three experimental scenarios, and the inference time of tiny-UKSIE is reduced by 20.77%, 24.55%, and 17.82%, respectively. This reflects that due to the optimization effect of the GA algorithm on the RETE network and the lightweight design of the inference engine, the inference time of Tiny-UKSIE is reduced compared with both Jena and RDF4J under the same conditions. In other words, the inference efficiency of tiny-UKSIE is higher, which is more suitable for IoT application scenarios with dynamic data changes.

Comparison of RAM Consumption

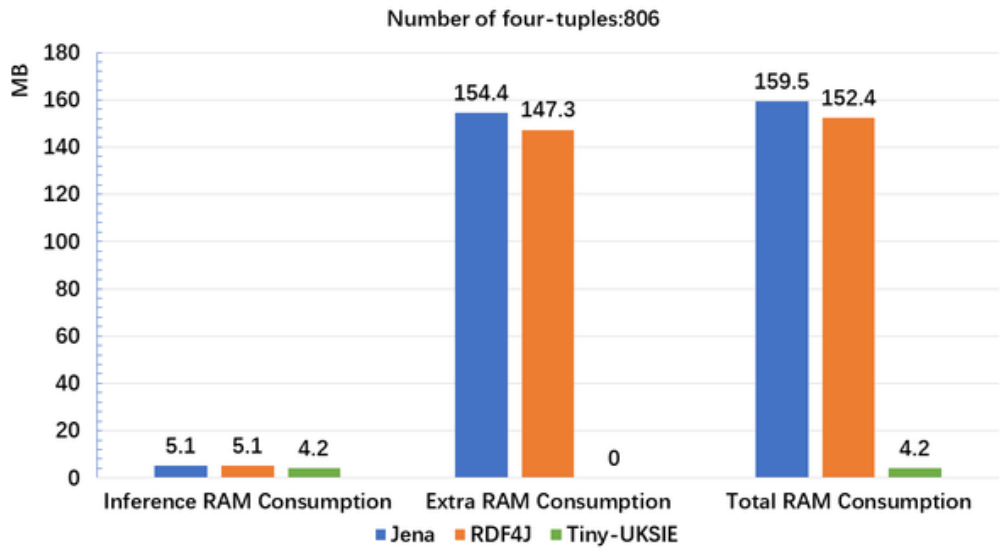
In terms of the storage resource requirements and usage of the inference engine, in addition to the ROM size occupied by itself, the runtime RAM consumption is a more critical indicator. During the experiment in Section "*Comparison of Inference Time*", the authors counted the RAM consumption of each inference engine during operation, including inference RAM consumption, extra RAM consumption, and total RAM consumption. Specifically, the inference RAM consumption refers to the RAM occupied by the inference algorithm when it is running; and the additional RAM consumption refers to the RAM occupied by the operating environment on which the inference engine depends, such as the RAM occupied by the Java Virtual Machine (JVM), here the JVM is the running environment on which the Jena and RDF4J inference engines depend. Total RAM consumption refers to the sum of the first two types of memory consumption.

Table 6 shows the statistical results of RAM consumption when the three inference engines are running with different numbers of four-tuples. Moreover, the RAM consumption comparisons under the three experimental conditions are shown in Fig. 9, Fig. 10, and Fig. 11, respectively. It can be seen that under the three conditions, Tiny-UKSIE consumes less RAM than the other two inference engines, with minimum consumption of 4.2MB, and the inference RAM consumption is reduced by up to 17.65%; moreover, in terms of total RAM consumption, Tiny-UKSIE is even more advantageous, the total RAM consumption is reduced by up to 97.37%. In practical applications, There are two main reasons for the RAM consumption advantage of Tiny-UKSIE. One is that the programming languages used by the three inference engines are different. Tiny-UKSIE is written in C++ language, while

Table 6. Inference RAM consumption under different four-tuples

Inference Engine Type	Number of Four-tuple Instances	Inference RAM Consumption (MB)	Extra RAM Consumption (MB)	Total RAM Consumption (MB)
Jena	806	5.1	154.4	159.5
	1263	5.7	154.4	160.1
	2406	7.4	154.4	161.8
RDF4J	806	5.1	147.3	152.4
	1263	5.6	147.3	152.9
	2406	7.5	147.3	154.8
Tiny-UKSIE	806	4.2	0	4.2
	1263	4.9	0	4.9
	2406	6.5	0	6.5

Figure 9. Comparison of RAM consumption (Number of four-tuples:806)



the other two are written in Java language. In contrast, C++ has an advantage over Java in reducing RAM consumption. Another reason is that both Jena and RDF4J inference engines rely on JVM, which brings more RAM consumption, while Tiny-UKSIE does not. In practical applications, when the RAM resources of the device are fixed, the less RAM the inference engine occupies, the lower its RAM resource usage. That is, Tiny-UKSIE has a low RAM resource occupancy rate. This will free up more RAM resources of the hardware device for other functional calculations, which is beneficial to improve the computing performance of the hardware device. This is particularly important for the semantic application of the IoT, which can ensure that the hardware device can process and transmit monitoring data while performing semantic reasoning, which is beneficial to the real-time nature of network data transmission. Therefore, Tiny-UKSIE consumes less RAM when running, and is more suitable for IoT applications where device RAM resources are limited.

Figure 10. Comparison of RAM consumption (Number of four-tuples:1263)

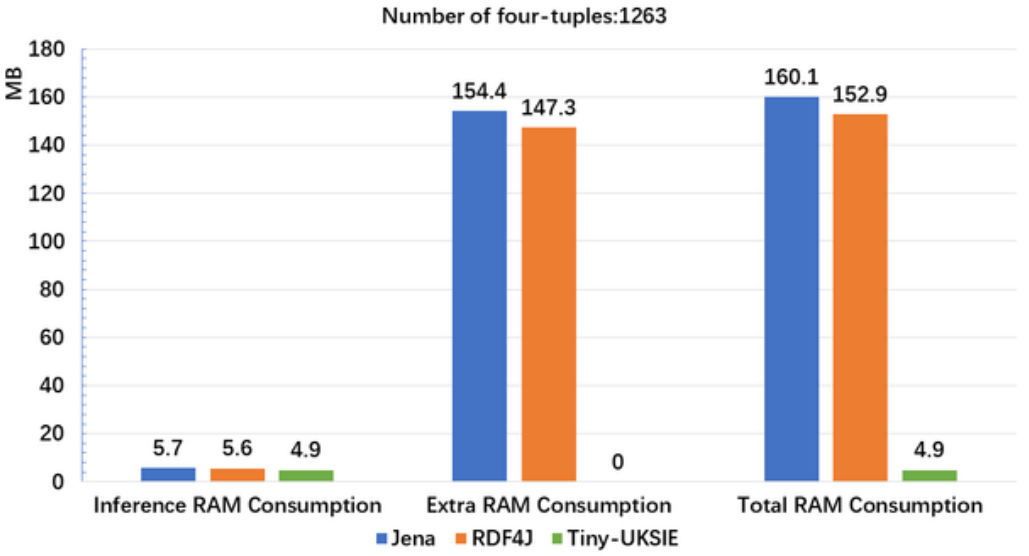
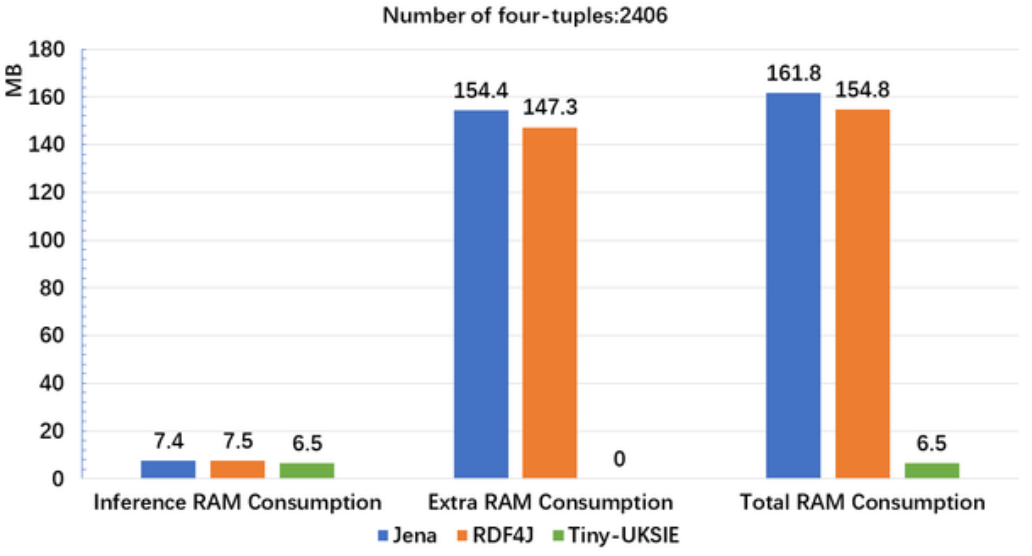


Figure 11. Comparison of RAM consumption (Number of four-tuples:2406)



CONCLUSION

Applying semantic reasoning technology to the IoT field can enhance the value of data and improve the intelligence of applications. The current mainstream inference engines cannot be applied to the IoT field due to high storage resource usage. Moreover, these inference engines, such as Jena, RDF4J, etc., do not support the reasoning of uncertain knowledge. Based on the RETE reasoning algorithm, this

paper proposes a lightweight reasoning engine Tiny-UKSIE. The GA algorithm is used to optimize the Alpha network of the RETE algorithm, which reduces the storage resource usage and the reasoning time, as well as improves the reasoning efficiency. Aiming at the problem that uncertain knowledge cannot be expressed and reasoned, this paper proposes a four-tuple knowledge representation method with probability factors, constructs four-tuple-based probabilistic inference rules, and realizes the adaptation to four-tuple patterns by improving the RETE network so that the inference engine can reason about uncertain knowledge. The results of multiple comparative experiments show that Tiny-UKSIE supports uncertain knowledge reasoning. Moreover, it has more advantages in terms of storage resource occupation and inference efficiency and is suitable for resource-constrained IoT application scenarios, which can drive the semantic applications and development of IoT.

In future work, the authors plan to conduct comparative experiments with more inference engines. In addition, the authors will perform more experiments on more hardware platforms and make optimizations to improve the generality of the inference engine. Moreover, to expand the feasibility of the uncertainty knowledge representation method, it would be of great significance to study the efficient conversion method from triples to four tuples. Furthermore, the authors are planning to study algorithm optimization methods in the case of the fewer number of patterns with the same degree of sharing, to further reduce storage resource usage and improve inference efficiency.

ACKNOWLEDGMENT

This work was sponsored in part by the National Key R&D Program of China (No.2020YFB1708801), and in part by the Natural Science Foundation of Chongqing, China(No.cstc2021jcyj-msxmX0330).

FUNDING AGENCY

The publisher has waived the Open Access Processing fee for this article.

REFERENCES

Ai, M., Su, X., & Riekki, J. (2017). Semantic Reasoning for Context-Aware Internet of Things Applications. *IEEE Internet of Things Journal*, *4*(2), 461–473. doi:10.1109/JIOT.2016.2587060

Al-Osta, M., Bali, A., & Gherbi, A. (2019). Event driven and semantic based approach for data processing on IoT gateway device. *Journal of Ambient Intelligence and Humanized Computing*, *10*(12), 4663–4678. doi:10.1007/s12652-018-0843-y

Ali, S., Khusro, S., Ullah, I., Khan, A., & Khan, I. (2017). SmartOntoSensor: Ontology for Semantic Interpretation of Smartphone Sensors Data for Context-Aware Applications. *Journal of Sensors*, *2017*, 1–26. doi:10.1155/2017/8790198

Bonatti, P. A., Ioffredo, L., Petrova, I., Sauro, L., & Siahaan, I. R. (2020). Real time reasoning in owl2 for gdpr compliance. *Artificial Intelligence*, *289*(12), 103–115.

Cantone, D., Nicolosi-Asmundo, M., & Santamaria, D. F. (2021). An improved set-based reasoner for the description logic d4,×. *Fundamenta Informaticae*, *178*(4), 315–346.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., & Wilkinson, K. (2004). Jena: Implementing the Semantic Web Recommendations. In *Proceedings of Thirteenth International World Wide Web Conference*. Association for Computing Machinery.

Fareh, M. (2019). Modeling Incomplete Knowledge of Semantic Web Using Bayesian Networks. *Applied Artificial Intelligence*, *33*(11), 1022–1034.

Forgy, C. L. (1982). RETE: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, *19*(1), 17–37.

GSMA. (2019). Key trends shaping the mobile industry. *The Mobile Economy*, *2019*, 32–48.

Hlel, E., Jamoussi, S., & Hamadou, A. B. (2018). A new method for building probabilistic ontology (Prob-Ont). *Information Retrieval and Management: Concepts, Methodologies, Tools, and Applications*, *3*, 1409–1434.

Horrocks, I., & Patel-Schneider, P. (2004). Reducing owl entailment to description logic satisfiability. *Journal of Web Semantics*, *1*(4), 345–357. doi:10.1016/j.websem.2004.06.003

Kalatzis, N., Routis, G., Marinellis, Y., Avgeris, M., Roussaki, I., Papavassiliou, S., & Anagnostou, M. (2019). Semantic Interoperability for IoT Platforms in Support of Decision Making: An Experiment on Early Wildfire Detection. *Sensors, 19*(3), 1-39.

Li, F., Du, J., He, Y., Song, H. Y., Mohcine, M., & Rao, G. (2020). Time event ontology (teo): To support semantic representation and reasoning of complex temporal relations of clinical events. *Journal of the American Medical Informatics Association: JAMIA*, *27*(7), 1046–1056. doi:10.1093/jamia/ocaa058 PMID:32626903

Li, S. M., Chen, S. H., & Liu, Y. (2019). A Method of Emergent Event Evolution Reasoning Based on Ontology Cluster and Bayesian Network. *IEEE Access: Practical Innovations, Open Solutions*, *7*, 15230–15238.

Meditskos, G., & Bassiliades, N. (2008). A rule-based object-oriented OWL reasoner. *IEEE Transactions on Knowledge and Data Engineering*, *20*(3), 397–410. doi:10.1109/TKDE.2007.190699

Meditskos, G., & Bassiliades, N. (2010). DLEJena: A practical forward-chaining OWL 2 RL reasoner combining Jena and Pellet. *Journal of Web Semantics*, *8*(1), 89–94. doi:10.1016/j.websem.2009.11.001

Motik, B., Shearer, R., & Horrocks, I. (2009). Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, *36*(1), 165–228. doi:10.1613/jair.2811

Mughal, M. H., Shaikh, Z. A., Wagan, A. I., Khand, Z. H., & Hassan, S. (2021). Orffm: An ontology-based semantic model of river flow and flood mitigation. *IEEE Access: Practical Innovations, Open Solutions*, *9*, 44005–44031.

Radhika, R., Bhuvaneswari, A., & Kalpana, G. (2022). An intelligent semanticification rules enabled user-specific healthcare framework using IoT and deep learning techniques. *Wireless Personal Communications*, *122*(3), 2859–2883. doi:10.1007/s11277-021-09033-7

Rahmani, A. M., Gia, T. N., Negash, B., Anzanpour, A., Azimi, I., Jiang, M. Z., & Liljeberg, P. (2018). Exploiting smart e-Health gateways at the edge of health care Internet-of-Things: A fog computing approach. *Future Generation Computer Systems*, *78*(2), 641–658. doi:10.1016/j.future.2017.02.014

Riali, I., Fareh, M., & Bouarfa, H. (2019). Fuzzy Probabilistic Ontology Approach: A Hybrid Model for Handling Uncertain Knowledge in Ontologies. *Information Journal on Semantic Web and Information System*, *15*(4), 1–20.

Setiawan, F. A., Budiardjo, E. K., & Wibowo, W. C. (2019). ByNowLife: A novel framework for OWL and Bayesian network integration. Information, 10(3), 1-21.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, *5*(2), 51–53. doi:10.1016/j.websem.2007.03.004

Tai, W., Keeney, J., & O'Sullivan, D. (2015). Resource-constrained reasoning using a reasoner composition approach. *Semantic Web*, *6*(1), 35–39. doi:10.3233/SW-140142

Tsarkov, D., & Horrocks, I. (2006). FaCT++ Description Logic Reasoner: System Description. *3rd International Joint conference on Automated Reasoning (IJCAR-2006)*, 292-297.

Urkude, G., & Pandey, M. (2021). Contextual triple inference using a semantic reasoner rule to reduce the weight of semantically annotated data on fail–safe gateway for wsn. *Journal of Ambient Intelligence and Humanized Computing*. Advance online publication. doi:10.1007/s12652-020-02836-9

Van Woensel, W., & Abidi, S. S. R. (2019). Benchmarking semantic reasoning on mobile platforms: Towards optimization using OWL2 RL. *Semantic Web*, *10*(4), 637–663.

Xin, S., Yan, X. M., Shang, Y. M., Ouyang, T., & Dong, K. (2017). An Improved Rete Algorithm Using Shared Degree Model. *Acta Automatica Sinica*, *43*(9), 1571–1579.

Zhou, J., Ma, L., Liu, Q. L., Zhang, L., Yu, Y., & Pan, Y. (2006). Minerva: A scalable OWL ontology storage and inference system. *Proceedings of 1st Asian Semantic Web Conference*, 429-443. doi:10.1007/11836025_42

ENDNOTES

1 http://www.cplusplus.com/
2 https://github.com/Feoss1013648658/Tiny-UKSIE/tree/master/RETE -master
3 https://franz.com/agraph/tbc/
4 https://www.w3.org/
5 http://www.w3.org/TR/owl2-syntax
6 https://www.w3.org/TR/2017/REC-vocab-ssn-20171019/
7 https://www.w3.org/2015/spatial/wiki/SOSA_Ontology
8 https://www.w3.org/TR/owl-time/
9 https://jena.apache.org/
10 https://rdf4j.org/

Daoqu Geng received the Ph.D. degree in physical electronics from Graduate University of Chinese Academy of Sciences, Beijing, China, in 2011. He is currently an Associate Professor with the College of Automation, Chongqing University of Posts and Telecommunications, Chongqing, China. His current research interests include Semantic IoT, industrial IoT, and industrial big data.

Haiyang Li was born Luoyang, Henan, China, in 1997.He received the B.S degree in electrical engineering and automation from the Luoyang Normal University, in 2019. He is currently pursuing the M.S degree in control engineering in Chongqing University of Posts and Telecommunications, Chongqing, China. His research interests include semantic web and the industrial Internet of Things.

Chang Liu was born Longquanyi, ChengDu, China, in 1994.He received the B.S degree in automation from the Sichuan University of Science & Engineering, in 2018, where he is currently pursuing the M.S degree in control sicence and engineering in Chongqing University of Posts and Telecommunications, China. His research interests include semantic web and the industrial Internet of Things. He currently studies the semantic web of things and big data.