An AI-as-a-Service Platform for an Artificial Intelligence of Things (AIoT)

Ali Nadar, Jérôme Härri
EURECOM, 450 route des Chappes, 06904 Sophia-Antipolis, France
ali.nadar, haerri@eurecom.fr

***Abstract*—With decentralized Machine Learning (ML) strategies and modern edge Tensor Processing Unit (TPUs), smart devices are no longer only consumers but also producers of Artificial Intelligence (AI), transforming an Internet of Things (IoT) into a global and decentralized Artificial Intelligence of Things (AIoT). With the availability of a large amount of AI actors comes not only the challenge to discover and network with them, but also the potential to use their AI capabilities as a service. However, the heterogeneity of the AI actors, their AI capabilities, AI contextual environment, mobility or even the AI characteristic available or sought requires not only a robust IoT architecture but also flexible AI semantics. In this paper, we present an AI-as-a-Service platform assisting AI consumers to identify existing AI tailored to their needs among the AIoT. We describe the architecture, the APIs, the message flows and AI semantics to identify the most appropriate AI workers when and where needed to efficiently generate AI models from distributed vehicles. As a proof-of-concept, we select an application scenario demonstrating the trainability/changeability of AI models between vehicles according to their context using the CARLA driving simulator.**

***Index Terms*—AI-as-a-Service, AI-of-Things, Machine learning, Ontology, Semantic, Driving Simulator, CARLA.**

I. Introduction

With advances in sensing, communication, and networking, autonomous vehicles and unmanned aerial vehicles (UAVs) are expected to play a vital role in a variety of Internet-of-Things (IoT) areas, including Industry 4.0, logistics, transportation, and public safety. Embedded with their multitude of sensors, these IoT actors are expected to be a significant source of data, which can be digested by state-of-art AI models to either improve the operation of the IoT or as functions supporting the IoT actors themselves. The inefficiency to share large amount of data in the IoT coupled with the significant energy requirement to train AI models on IoT infrastructures and the availability of modern Tensor Processing Units (TPUs) enabled an alternative strategy to distribute the AI tasks directly to the IoT actors.

Providing AI-as-a-Service (AIaaS) yet faces various challenges. First, coordination would be required for decentralized AI mechanisms to identify and cherry-pick the most appropriate AIoT entities to cooperatively perform AI tasks. Second, AI knowledge needs to be named and compared for a match between demand and supply, and AI ontologies should be defined for AI knowledge apparently named differently to be inferred actually as similar. Finally, trust and traceability should be enforced to guarantee the service outcome. In this paper, we present an AIaaS platform that allows users to play as AI actors by accessing and utilizing ML models without the need for heavy investment in infrastructure and expertise in producing such models.

Due to the diversity of user's environment, the available contexts may change over time. It is challenging to anticipate a complete set of contexts while we design a context aware system. In this paper, we propose a context modeling approach which can dynamically handle various context types and values. More specifically, our middleware platform is context-aware embedded system designed and implemented to deploy context management system for providing services to vehicles depending to their contextual information. We use ontologies to enhance the meaning of a user's context values and automatically identify the relations among different context values. Our contributions are fourfold: first we provide an overview of potential challenges to address AIaaS in AIoT; second we propose a centralized AI-as-a-Service (AIaaS) architecture called *Infrastructure Assisted Knowledge Management (IAKM)*; third we propose a Map-based ML ontology for the metadata of AI-models and our approach for proximity metrics; finally, we conduct a tangible experiment showcasing the integration of our IAKM platform with CARLA driving simulator. The platform is available as open-source under an Apache 2.0 license and is developed as a multi-docker container platform.

The rest of this paper is organized as follows: Section III introduces AIoT, whereas Section IV, we present the IAKM platform and its components. We describe in Section V our proposed ontology and the similarity search mechanism, whereas in Section VI, we demonstrate an experiment for IAKM-CARLA implementation and integration scenario, in Section VII we summarize and conclude the paper.

II. Literature Survey

A literature survey on the benefits and challenges of AI-driven applications in IoT domains has been performed, and the means of creating/consuming knowledge on decentralised learning for distributing knowledge, as well as on Map-based ML ontology for vehicular systems and AI proximity metric have been proposed. In AIoT architectures literature, in [1] *Zhang and Tao* made a comprehensive survey of AIoT, covering AIoT computing architectures; AI technologies for empowering IoT with perceiving, learning, reasoning, and behaving abilities; promising AIoT applications; and the challenges and opportunities facing AIoT research. In [2], an architecture for an envisioned Knowledge IoT Platform has been proposed, such proposal needs more elaboration for components architecture, technology selection as well as the deception of the followed methodology to manage the messages processing between different components of Edge/Client. Nevertheless, we noticed based on our survey that the concept of AI-driven applications and AI knowledge interchangeability remains on the conceptual/survey level and

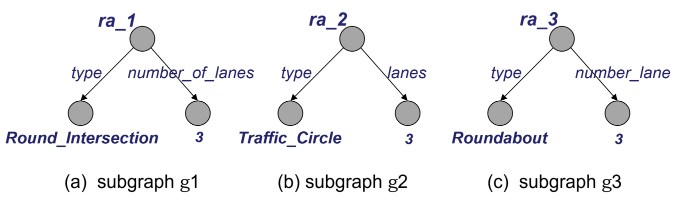


Figure 1: Diverse semantic graph structures

its implementation or format is left for future work. This paper, thus, provides an actual proposed an AIaaS platform and open API enabling AI actors to produce, train or consume AI models based in their capabilities and contexts. In Core map-based machine learning ontology literature, *Braga et al.* [3] proposed an ontology to represent the knowledge around the Machine Learning discipline, they considered seven main classes, which provides a natural structuring of ML parameters *Algorithms, Applications, Dependencies, Dictionary, Framework, Involved, MLTypes*. In [4] *Lihua Zhao et al.* proposed a core map ontology for vehicle situation understanding. The proposed ontology introduces the main classes of the map ontology. A road consists of junctions and road segments, where a road segment consists of an arbitrary number of lanes. In semantic relatedness and similarity in map features, [5] *José Paulo et al.* propose an algorithm to measure the relatedness of two terms using the knowledge base of BDpedia by computing the semantic relatedness of two terms as proximity rather than distance, as in similar ontology based approaches. Searching for data using a pure SPARQL query requires a full schematic knowledge about the knowledge graph, including nodes, properties and their relationships. Although lots of efforts have been devoted to the graph similarity search, [6]–[11], they suffer from various drawbacks, where most of them focus on the structure similarity, such as SAPPER [7], kGPM [8] and Ness [7]. SAPPER [7] investigates the problem of approximate subgraph search allowing some edges unmatched. It does not support the vertex/edge label substitution. kGPM [8] proposes a graph pattern query, which allows a path to match an edge. However, it restricts that the vertex/edge labels specified in the query graph q should be exactly matched. Exploiting the neighborhood-based similarity measure, Ness [7] and NeMa [9] try to identify the top-k approximate matches of a query graph q. Generally speaking, most of these methods concentrate on the graph structure similarity without considering the semantic similarity. However, in RDF graphs, two graph patterns may have large structural dissimilarity, such as Fig. 1 (a), (b) and (c), but they describe the identical semantic meaning for *roundabout*.

III. Artificial Intelligence of Things (AIoT)

Artificial Intelligence of Things (AIoT) relies on a service-oriented IoT architecture to support decentralized large-scale dynamic AI management. In an AIoT, every component produces or consumes AI-based knowledge. It is therefore likely that an AI required by a particular service already exists in the AIoT and can be transferred rather than being recreated if it may be discovered.

Applying decentralized AI functions to IoT is however not straightforward due to peculiar AI requirements. Considering a risk assessment AI model for the roundabout scenario as described by *Duncan et al.* in [12] and summarily illustrated in Fig. 2, AIoT resources are defined not only in the spatial domain (where) but also in the temporal domain (when). Red vehicles train as long as they remain in the roundabout, blue vehicles may train as soon as they enter the roundabout, and other grey vehicles are simply not suited for training, either because they never access the roundabout or do not have the required capabilities. AIoT capabilities must therefore be identified in space and time and match with the requirement of AI models.

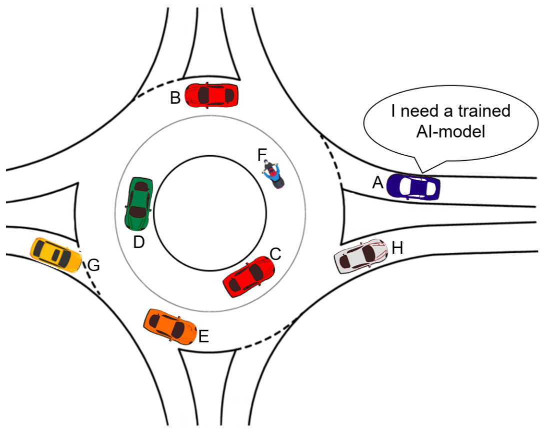


Figure 2: Space & time AIoT resources for AI functions.

	capability	status(t)	AI role(t)
A	able to use	asking to use AI-model	AI consumer
B,C	able to train	training AI-model	AI producer
D	able to train	ready for training AI-model	AI producer
E	not-able to train	No computation resources	-
F	not-able to train	unfitted vehicle class	-
G	able to train	can train in roundabout	-
H	able to train	exited roundabout	-

Table I: AIoT resources for AI functions

Accordingly, the success of AIoT will depend on services functions capable, among others, to solve the following challenges:

- Naming - what kind of AI model is it ?
- Context - where shall AI models be inferred/trained ?
- Timing - when shall AI models be inferred or trained?
- Processing - how can AI models be trained ?

Would these challenges be solved, an AIoT has the potential to move the AI paradigm from traditional AI-as-a-Duty (i.e. I create the AI I need) toward AI-as-a-Service (i.e. I create AI for someone else).

IV. Infrastructure-Assisted Knowledge Management (IAKM)

We present in this section the Infrastructure-Assisted Knowledge Management (IAKM) platform, as a centralized implementation of AIaaS. The IAKM architecture consists of a server and a client entity as depicted on Fig. 3. IAKM server is accessed by An AI actor (consumer/producer) through an IAKM agent over a RESTful AIaaS-APIs. The IAKM agent may be located either on a vehicle or integrated in an external entity requiring IAKM services.

A. *IAKM server*

The IAKM server corresponds to the backend processing of the AIaaS framework, it consists of a message passing

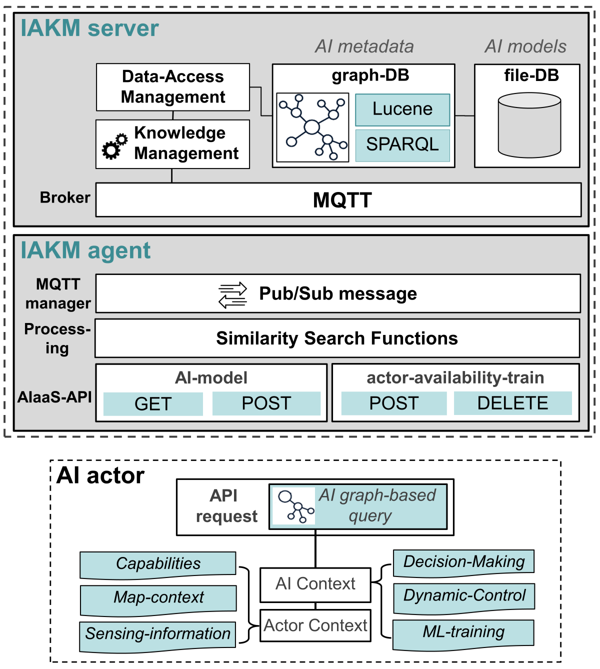


Figure 3: IAKM actor/server architecture for AIaaS

component (MQTT), a data access management engine and a Knowlege Management (KM) engine. For data storage, it has a RDF knowledge graph database to store the metadata of AI models as well as a file database to store byte-codes AI models. Their roles are described below:

- Message Passing - provides publish/subscribe mechanisms to exchange information either between the client and the server or between different servers.
- Databases - provides a dual-storage approach, utilizing file format storage database like '*MongoDB*' for AI models and graph format storage database like '*GraphDB*' for context storage in RDF triples
- Knowledge Management - identifies AI models available locally according to the AI engine proximity metric, or coordinate the localization and predictive caching of most appropriate AI models.
- Data Access Management - maps the AI model metadata to its byte-codes based on the KM requests, also handles the transforms of graph-based queries into SPARQL/Lucene query.

The IAKM server may be located either in the Cloud or at a 5G MEC or even on edge devices, and will adopt the semantic context corresponding to its location or scope (i.e. to optimize AI model caching).

B. *IAKM Agent*

IAKM agent could communicate with many AI actors, it manages user request in threefold; first it exposes AIaaS-API to receive request content. Second, using the semantic functions described in section V-C, it interprets the graph-based query to validate, analyse and align the query with the structure of IAKM ontology. Finally, it communicates with IAKM server microservices using publish/subscribe messaging pattern. The described AIaaS-API as depicted on Fig. 3 is an HTTP RESTful service, all requests must be in JSON/RDF-format that describes the Actor/AI context. The proposed endpoints are as following:

- *Register* - a client (AI actor) announces his availability to train AI model for a given context. It sends a registration request indicated by PUSH method and *http://agent-host:agent-port/actor_available_train/*$\{$*context*$\}$ as *url*
- *Unregister* - a client (AI actor) removes his availability for a given context, either for not more fitting his previous context or for saving resources. It sends a remove registration request indicated by DELETE method and *http://agent-host:agent-port/actor_available_train/*$\{$*context*$\}$ as *url*
- *Find* - while approaching a complex situation, an AI actor (i.e. Consumer) asks for an AI model according to a giving context. It sends a search request indicated by GET method and *http://agent-host:agent-port/ai_model/*$\{$*context*$\}$ as *url*.
- *Store* - a client (Producer) already owning a trained AI model for a given context, it sends a save request indicated by PUSH method using *http://agent-host:agent-port/ai_model/*$\{$*context*$\}$ as *url* and the AI model byte-codes in the message body.

C. *AI actor*

In our proposed architecture, AI actor acts as a client for IAKM platform, it could be a consumer, producer or even as an AI worker with ML model training task. The actor communicates with IAKM agent using RESTful API requests, the request should include a graph-based query that contextually describes the needed AI, the map context and the capabilities of an AI actor.

V. Map-based ML Ontology and Semantic

In fact, AIaaS framework enables AI actors to experiment with AI for various purposes without a large initial investment, these actors could belong to a various distributed and heterogeneous systems. In vehicular systems domain, every stakeholder has its own representations of knowledge, concepts and proprieties. Therefore the data interpretation, interoperability/interchangeability between the different AI actors becomes challenging due the semantics ambiguity and non-shared vocabularies. Therefore, a common knowledge representation method is necessary to harmonize this diverse data by providing a common framework for interpretation. In this section, we focus on ontology/semantic activities; first we define our domain of consideration; second we design an ontology for IAKM platform; finally we describe our approach to implement the AI semantic similarity search.

A. *Knowledge Domains*

Without loss of generalities, we define in this work an IAKM ontology that includes terms and naming related to the application of machine learning in vehicular systems. For this purpose, we describe the main domains/concepts required to create such ontology: "*Model*" is the conceptual class for all instance of AI models,"*Framework*" is the machine learning framework required to manage AI models. "*Algorithm*" is a set of mathematical processes or techniques by which an AI system conducts its tasks. "*Dependency*" describes the required inputs and the expected output for AI model. "*Dictionary*" is the acronyms, nicknames and phrases for which an interpretation is required. "*MLType*" identifies the type of machine learning (supervised, unsupervised, reinforcement learning ...). "*Actor*" (respresents the IAKM client, it could be) is all types of AI actors who created and improved the available resources to facilitate the use and application of Machine Learning the information of actors managing AI models. "*Map*" describes the contextual topology on which an AI application (model) could be used.

B. Ontology

Braga et al. [3] proposed an ontology adapted to Machine Learning, which provides a natural structuring of ML parameter. *Zhao et al.* [4] presented a core ontology for autonomous driving. Inspiring by their contributions, in our study as depicted on Fig. 4 we build an IAKM ontology that combines the two ontological aspects. On top of it, we define 3 additional entities which are directly related to our IAKM platform; *Actor* to store actor capabilities and information as well as to register/unregister the availability of AI workers according to their context, *Model* to bind entities and store model metadata and *Map* to describe the map context correspond to the AI model.

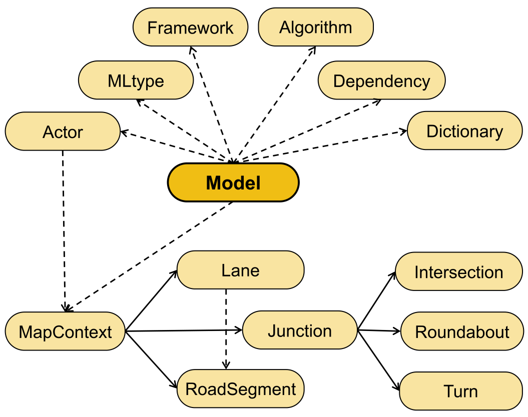


Figure 4: Proposed IAKM Ontology

C. Semantic Search

AI semantics represent a critical function of AIaaS, and needs to remain flexible enough to identify the most proximate AI, when the request is either incomplete or an exact AI model is not available. Compared to IoT or Named Data Networking (NDN) semantics, it is highly unlikely that a globally agreed way to describe an AI model or its usability context could be defined, considering that any AIoT component may produce any model under any condition. However, ontologies and knowledge graphs are designed not only to describe content but also to capture and infer relationships between data elements. In the IAKM, semantic search mechanism is a data search technique which uses the intent and contextual meaning behind a search query to deliver more relevant results. In this work, we present our methodology for searching the best fitted AI model according to its metadata stored in graph database and therefore find the unique identifier of AI data as stored in file database. In Fig. 5, we describe our approach for AI semantic search in IAKM platform

According to Fig. 5, the graph-based query passes through multiple functions across IAKM component before being landed at the SPARQL query endpoint. we describe below the processing functions as the following:

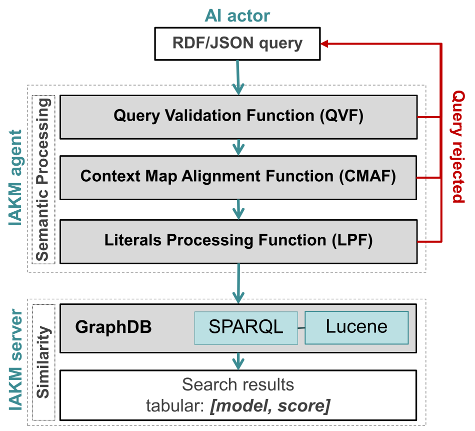


Figure 5: Semantic processing & Similarity search

1) Query Validation Function (QVF): - In our approach, for storing/retrieving AI metadata using query-based graph, every node in the graph should be an instance of a class included in IAKM ontology concepts, in other RDF term every instance should has "*rdf:type*" predicate to one class within Model, Algorithm, Framework, Dependency, MLType and Actor, otherwise query graph must be rejected. We except the Dictionary class as it is not instantiable and considered as a static class that contains all acronyms definitions.

2) Context Map Alignment Function (CMAF): - In automotive industries, car manufacturers provide HD-maps, which may have their own structure, describing the road map, road structure, and spatial composition based on a existing road centerline concept. In such heterogeneous map description vocabularies context, the IAKM platform uses semantic matching to find the structural commonalities between the user's map context expression and the IAKM ontology definitions. To this end, IAKM platform presents a semantic matching function called Context-Map Alignment Function (CMAF) that uses Natural Language Processing (NLP) to align large language expressions with IAKM ontology. For this purpose, we use NLP-as-a-Service powered by OpenAI API with the large model *gpt-3.5-turbo*, this model could semantically select the closest defined ontology concept to the unknown requested map context expression.

$$semMatch(gpt, x, nodes) = \begin{cases} node & \text{if } x \in Roads \\ \emptyset & \text{if } x \notin Roads \end{cases}$$

The function "*semMatch*" takes as parameters; '*gpt*' for openAI model, '*x*' for user unknown expression and '*nodes*' for list of all map context sub-classes of IAKM ontology. For instance; CMAF could ask openAI with this message: "*in one word, in roads context, which one is closer and could mean 'circular intersection': Lane, Junction, Intersection, Roundabout, Turn, RoadSegment*. In this case the function matches the term "*circular intersection*" with the ontology class "*Roundabout*", which has semantically the same meaning. In contrast, for tricky expressions in road context, such as 'circular computer', the function catchs the unrelatedness by answering: "*None, it's not a perfect match for a 'circular computer' in a roads context*"

3) Literals Processing Function (LPF): - This function helps to make inference-like process by finding the common representation between the RDF global-KG attributes with query graph attributes (RDF literals). In our study, we use the exact match for attributes similarity as we keep LPF in conceptual level.

4) SPARQL & Lucene: - In IAKM server,we propose a similarity search technique using SPARQL query with Lucene connector to determine how relevant a given stored AI model is to a consumer query. Lucene uses the Boolean model to first narrow down the documents that need to be scored based on the use of Boolean logic in the query specification. GraphDB Lucene engine is a Full-Text Search (FTS) using native Lucene queries. Technically, our proposed similarity search is in 2 dimensions; first, transforming the graph triplets into SPARQL query constraints in order to find a similar representation in global IAKM RDF graph. Second, transforming the query literal attributes into FTS terms in order to find the most relevant AI metadata available in IAKM RDF graph by leveraging graphDB Lucene connector.

VI. IAKM IMPLEMENTATION WITH CARLA

Having established the theoretical outlines for IAKM architecture, map-based ML ontology and semantic search mechanism, our focus now shifts to presenting a clear experiment that highlights the IAKM platform's capabilities. Our approach involves leveraging the capabilities of the CARLA driving simulator to animate the interaction between IAKM and AI actors within the virtual environment when an AI actor is symbolically represented using a CARLA vehicle. CARLA is an open-source simulator for autonomous driving research, it has been developed from the ground up to support development, training, and validation of autonomous driving systems. In the following subsections we describe the implementation steps for the experiment; First, we devise a ML control model that caters to different types of road junctions, then we delve into the requests processing among CARLA vehicles acting as AI actors, adopting consumer/producer roles and the IAKM server, and finally we explain the results of experimenting with the proposed traffic scenarios.

A. ML model for dynamic decision control

To ensure the realism of our experiment in simulating AI exchange between CARLA vehicles and the IAKM platform, it is imperative to utilize an authentic machine learning model. Without loss of generalities, in this work we introduce a simplified ML model designed to make decision controls for vehicles while nearing a complex traffic situations like intersection, roundabout, etc.. Thus, we opted for the Random Forest Classifier (RFC) due to its robust ensemble learning capabilities. Ensemble methods, such as RFC, are known for their ability to combine multiple weak learners to enhance overall model performance. This is particularly advantageous in our context of vehicle decision control, where the system can benefit from the collective intelligence of diverse decision trees within the random forest.

The proposed ML model aims to avoid collision, as shown in Fig. 6, we identify a Collision-Spot in road way-points where the paths of the two vehicles intersect, and then we define a control area (in red) where the ego vehicle must exercise caution, specifically by according priority to incoming vehicles. Building upon this identification, we can now precisely define four key features that form the foundation of our model:

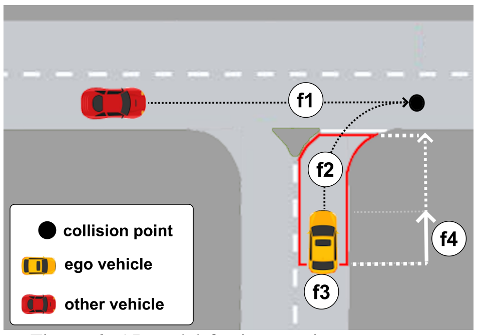


Figure 6: AI model for intersection management

- **f1**: Time-To-Collision for '*ego*' vehicle. Using speed and distance metrics, we determine the time interval before the ego vehicle reaches the identified collision point.
- **f2**: Time-To-Collision of '*target*' vehicle. Using speed and distance metrics, we determine the time interval before the ego vehicle reaches the identified collision point.
- **f3**: speed of the ego vehicle
- **f4**: normalized float distance in [0,1] interval for '*ego*' vehicle within the red control area.

While ego is in control area, data collection is performed at each time step to capture the 4 input features and the control tag as output label for RFC model, we pick up the label according to the mapping table using the pair of (throttle & break) of actual dynamic control. Once ego vehicle leaves the control area, ego vehicle trains its ML model using the collected dataset where the combination of features helps the model learn patterns and make accurate predictions. For the categorical output label, we identify 15 dynamic control labels as showing in Table.II

Dynamic Control		
Throttle	**Break**	**Label**
$0.0 < t \leq 0.1$	0	t1
$0.1 < \mathrm{t} \leq 0.2$	0	t2
.....	.....	.....
$0.6 < \mathrm{t} \leq 0.7$	0	t7
0	0	n
0	$0.0 < \mathrm{b} \leq 0.1$	b1
0	$0.1 < \mathrm{b} \leq 0.2$	b2
.....	.....	.....
0	$0.6 < \mathrm{b} \leq 0.7$	b7

*RFC model has 15 labels: 7/7 levels for throttle/break and neutral

Table II: Mapping control(throttle,break) to model target(label)

B. IAKM integration with CARLA driving simulator

The designed ML model serves as a key integration for the systematic exchange process between the AI actors (CARLA vehicles) and the backend server (IAKM). In Fig. 9a, [veh-A] is considered as human-driven control and already has a non trained RFC ML model for driving control, while approaching a roundabout inside the control area, it collects the dataset (4 features, 1 label), then by exiting the control area it trains the model with the dataset, and as AI producer it pushes the model to IAKM server according to its context as depicted in (Fig. 7). At another point in time, [veh-B] upon nearing a road junction, it recognizes the complexity and lacks an ML model to aid in decision-making, it acts as AI consumer and asks IAKM server for AI model according to its context.

At IAKM server, upon request receipt, it activates a similarity search mechanism to locate the closest model, noticing that in our experiment we consider 70% as the similarity threshold for SPARQL/Lucene search in graphDB for context closeness of consumer's graph-based query. Upon receiving a response from the server, the vehicle dynamically adapts its behavior: if the server provides a high-scored (bigger than 70%) machine learning model, the vehicle incorporates it for decision-making; otherwise, it employs a waiting strategy as a benchmark for control, wherein it remains stationary until the surrounding environment is devoid of other vehicles, ensuring a safe and opportune moment to cross the junction, as shown in Fig. 9b [veh-C] remains in a halted state until the blue zone is clear of incoming vehicles, ensuring a safe and obstruction-free environment before proceeding, and consequently leading to a traffic queue formation behind it.

Semantically speaking and according to our defined ontology, as a producer [veh-A] produces his trained ML model along with its context as a graph-based query as illustrated in Fig. 7, this graph/model should be stored in IAKM server at graphDB/fileDB respectively, the graph describes the semantic context that a vehicle as a consumer should fit in order to receive and consume the relevant ML model. To construct the graph, the producer instantiates all entities in IAKM ontology as following; for [MapContext], according to the producer vocabulary, he describes his map context as *round intersection*, for [algorithm] the ML type is "*RandomForest*", for [framework] the package is "*sklearn*", and for [dependency] context the model has four features and one output label as shown in Fig. 6, and finally it connects all instances to the core model instance, the latter has "*control*" attribute for its [type] and "*NIL*" for its [unique-ID] (UID). We note at this stage that model UID is marked as "*NIL*" in order to be assigned to graph in IAKM server level after the model storage in fileDB. Once producer sends his model/graph-based query, the query should be successfully processed in semantic functions of IAKM agent. For the case of [veh-A], the MapContext is described with *round intersection* which is not defined in IAKM ontology. For this reason, CMAF makes a processing in map alignment with IAKM ontology by replacing the unknown *round intersection* entity with the defined *roundabout* entity in IAKM ontology. Finally, IAKM server receives the producer message, and save consequently its ML model in fileDB and query graph at its RDF graphDB.

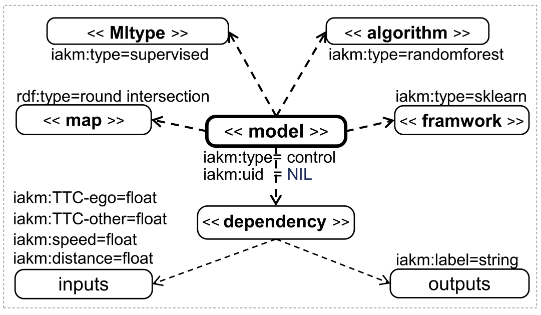


Figure 7: graph-based query to store AI metadata for veh-A (Producer)

At a later stage, an AI consumer [veh-B] while approaching the T-intersection, it realises the complexity of the situation where no local AI model is available to make a control decision. Therefore, it generates graph-based query by defining the map-context as "*junction*", and in similar way for other entities like in producer case but with more generic to simplify the request and to emphasis on similarity search process to find best fitted AI model. Likewise, at the IAKM agent level, the query graph passes through the semantic functions where QVF validates its structure, CMAF finds the structure conformity of "*junction*" with IAKM ontology and for LPF function we skip it as we keep the module at a conceptual level and we will define and implement it in future work.

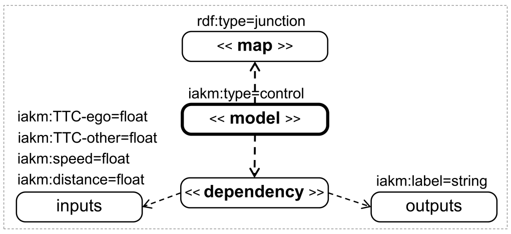


Figure 8: graph-based query to find AI metadata for veh-B (Consumer)

Once IAKM server receives the query, the DataAccess manager activates the similarity search process using SPARQL engine with Lucene embedded to determine how relevant a stored AI models is to a consumer graph query. Lucene uses the Boolean model to first narrow down the documents that need to be scored based on the use of Boolean logic in the query specification. As illustrated in Table. III, we apply the similarity search in 2 phases; in first we make sure that query structure is aligned with the proposed ontology, and second we applied Lucene connector functionality to find the most relevant AI metadata. we see more detailed description in section VI-C.

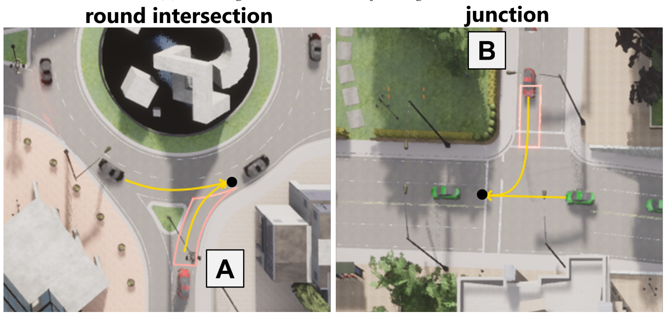
C. Experimentation

As we previously mentioned, we use CARLA simulator to incorporate a traffic scenario integrated with IAKM platform where a producer [veh-A] and a consumer [veh-B] as illustrated in Fig.9a could precept their surrounding environments. We start the simulation by spawning a considerable number of auto-piloted vehicles, then we spawn [veh-A] in autopilot mode before the red control area of the "*round intersection*" in order to collect a full featured dataset for its dynamic control during his existence in the red control area, upon leaving the designated area [veh-A] initiates ML model training and subsequently pushes the trained model to the IAKM server specifying "*round intersection*" as the map context in a graph-based query. In a later step, we spawn [veh-B] without any AI model, also in autopilot mode at a random position before the control area of "*junction*", then by approaching this "*junction*", it detects the complexity of such situation and sends request to IAKM server to get the best fitted AI model and indicating "*junction*" as map context in its graph-based query.

On IAKM server, the DataAccess service activates the query transformation from RDF graph format to SPARQL query format. As depicted in table.III the similarity search approach is bi-dimensional:

- SPARQL constraints; by adding graph instances and its relations as SPARQL constraints to ensure that all resulting models adhere to the graph structure of the query.
- Lucene FTS; by adding all attributes of query graph into the Lucene FTS field as a chain of (*key*:*value*) pairs

(a) *training* in [veh-A] & *inferring* in [veh-B]



(b) AI model NOT found in IAKM server according to [veh-C] context

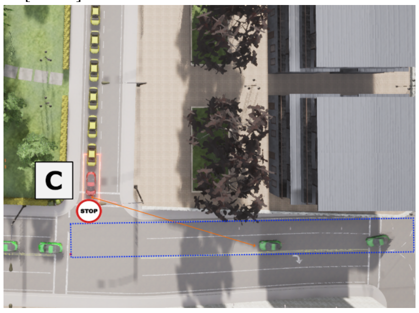


Figure 9: CARLA-IAKM scenarios

in order to identify the best proximity (highest scores) withing the retrieved models.

Table III: Similarity Search SPARQL/Lucene

Query Level	Expression
E1 SPARQL Search (instances & relationships)	?model rdf:type iakm:model . ?map rdf:type iakm:junction . ?dependency rdf:type iakm:dependency . ?inputs rdf:type iakm:inputs . ?outputs rdf:type iakm:outputs . ?model iakm:hasMap ?map . ?model iakm:hasDependency ?dependency . ?dependency iakm:hasInputs ?inputs . ?dependency iakm:hasOutputs ?outputs .
E2 Lucene Search (attributes)	type : control TTC-ego : float TTC-other : float speed : float distance : float label : float

Full query
PREFIX luc: http://ontotext.com/connectors/lucene PREFIX luc-index: http://ontotext.com/connectors/lucene/instance PREFIX iakm: http://example.org/IAKM.owl SELECT ?model ?score { ?search a luc-index:iakm-lucene-connector; luc:query "E2"; luc:entities ?model . { E1 } }

In SPARQL constraints, logically speaking and as we defined in our ontology for MapContext, every *roundabout* is a *junction*, but not every *junction* is a *roundabout*. Accordingly, inference might be very complicated in SQL/NoSQL database. However being a semantic database, graphDB leverages OWL2-RL rulesets for advanced inferences, allowing it to implicitly identify relations through graph reasoning. In the proposed IAKM platform, the similarity result may yield zero or more pairs of "*model-id*" and "*scores*", if bigger than zero, IAKM server selects the best scored model-id and finds its associated AI model in fileDB in order to send it to the consumer, otherwise if zero, it means the query context does not match, then it replies with *NIL* response to the consumer as shown in Fig.9b.

VII. From Theory to Application: IAKM key contributions and Conclusions

We introduced in this paper the potentials of AI-as-a-Service for an AI-of-Things (AIoT) and presented an architecture for the Infrastructure Assisted Knowledge Management (IAKM) platform as a centralized implementation of an AIaaS framework, wherein we designed an integrated map-based ML ontology tailored to accommodate diverse AI-driven applications within vehicular domains. To address semantic nuances, our research introduced a framework outlining the methodology for the storage and retrieval of AI models based on contextual information and leveraging an integrated strategy, we employed the SPARQL/Lucene engine for similarity search mechanisms while relying on the semantic alignment prowess of the OpenAI large language model to harmonize diverse context vocabularies. Additionally, we proposed the implementation of a RESTful API to enable seamless and decoupled communication between AI actors and IAKM agents. Furthermore, to validate the effectiveness of our approach, we conducted a real experiment using the CARLA driving simulator, specifically focusing on machine learning support in an AI-driven junction risk assessment scenario. The platform is provided as open-source, the full code and video for the IAKM implementation and its integration with CARLA in a concrete junction management scenario is available on EURECOM gitlab: *gitlab.eurecom.fr/cats/carla-iakm*

References

[1] J. Zhang and D. Tao, "Empowering things with intelligence: A survey of the progress, challenges, and opportunities in artificial intelligence of things," 2020.

[2] D. Deveaux, "On the networking of knowledge in vehicular networks," Ph.D. dissertation, 2021, © EURECOM. Personal use of this material is permitted. Systems and Control[cs.SY]. Sorbonne Université, 2021. English. ffNNT : 2021SORUS294ff. fftel-03592855.

[3] J. Braga, F. Regateiro, J. Dias, and I. Stiubiener, "A machine learning domain ontology to populate knowledge base to support intelligent agents working in autonomous systems domains of the internet infrastructure," 07 2021, pp. 1–12.

[4] L. Zhao, R. Ichise, S. Mita, and Y. Sasaki, "Core ontologies for safe autonomous driving," 10 2015.

[5] J. P. Leal, V. Rodrigues, and R. Queirós, "Computing Semantic Relatedness using DBPedia," vol. 21, pp. 133–147, 2012. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2012/3519

[6] S. Zhang, J. Yang, and W. Jin, "Sapper: Subgraph indexing and approximate matching in large graphs," *Proc. VLDB Endow.*, vol. 3, pp. 1185–1194, 2010.

[7] A. Khan, X. Yan, and K.-L. Wu, "Towards proximity pattern mining in large graphs," 06 2010, pp. 867–878.

[8] J. Cheng, X. Zeng, and J. X. Yu, "Top-k graph pattern matching over large graphs," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 2013, pp. 1033–1044.

[9] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan, "Nema: Fast graph search with label similarity," *Proc. VLDB Endow.*, vol. 6, pp. 181–192, 2013.

[10] Y. Wu, S. Yang, and X. Yan, "Ontology-based subgraph querying," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 2013, pp. 697–708.

[11] S. Yang, Y. Wu, H. Sun, and X. Yan, "Schemaless and structureless graph querying," *Proc. VLDB Endow.*, vol. 7, pp. 565–576, 2014.

[12] D. Deveaux, T. Higuchi, S. Uçar, J. Härri, and O. Altintas, "A knowledge networking approach for ai-driven roundabout risk assessment," in *2022 17th Wireless On-Demand Network Systems and Services Conference (WONS)*, 2022, pp. 1–8.