A Deep Learning Model for Natural Language Querying in Cyber-Physical Systems

Juan Alberto Llopis[a], Antonio Jesús Fernández-García[b], Javier Criado[a], Luis Iribarne[a], Rosa Ayala[a], James Z. Wang[c]

[a] *Applied Computing Group, Department of Informatics, University of Almería, Spain*
[b] *Universidad Internacional de La Rioja, Spain*
[c] *Data Science and Artificial Intelligence Area, College of Information Sciences and Technology, The Pennsylvania State University, USA*

Abstract

As a result of technological advancements, the number of IoT devices and services is rapidly increasing. Due to the increasing complexity of IoT devices and the various ways they can operate and communicate, finding a specific device can be challenging because of the complex tasks they can perform. To help find devices in a timely and efficient manner, in environments where the user may not know what devices are available or how to access them, we propose a recommender system using deep learning for matching user queries in the form of a natural language sentence with Web of Things (WoT) devices or services. The Transformer, a recent attention-based algorithm that gets superior results for natural language problems, is used for the deep learning model. Our study shows that the Transformer can be a recommendation tool for finding relevant WoT devices in Cyber-Physical Systems (CPSs). With hashing as an encoding technique, the proposed model returns the relevant devices with a high grade of confidence. After experimentation, the proposed model is validated by comparing it with our current search system, and the results are discussed. The work can potentially impact real-world application scenarios when many different devices are involved.

Keywords: Deep Learning, Recommender System, Transformer, Web of Things, Natural Language

1. Introduction

Throughout the history of humanity, the quality of life has constantly been improving due to advancements in technology. The human being, already evolving with inventions such as the wheel and metallurgy, developed further with the advent of electricity and the industrial revolution. This continues to be the case today with new technologies [1]. Nowadays, evolution aims at digitalizing or making processes intelligent, as well as automating and improving the quality of life in each area as follows: Smart Home to automate household tasks, Smart City to automate city management and control tasks, Industry 4.0 to automate manufacturing and distribution in industrial processes, and finally Smart Buildings, Smart Grid and Smart Healthcare, among others [2].

This digitalization process is being driven by more and more universities, governments, companies, and other institutions, through the development of products and services. As more companies participate, a greater variety of devices that provide solutions for more complex problems are developed, thus streamlining the digitalization process. However, as the heterogeneity of devices increases, each company designs a different device to solve the same problem, making integrating devices difficult [3]. Furthermore, the digitalization process increases the number of devices deployed and the complexity of the devices' operations. Due to the growth in the number of devices, their complexity, and their heterogeneity, a quick and efficient way of finding devices or services that meet users' requests is desired. Finding specific IoT devices or services in environments where the user may not know what devices are available or how to access them becomes even more challenging. In these scenarios, the use of natural language sentences can be desirable.

This article solves the matching problem between users' sentences and devices by creating a decision-making aid model using the Transformer and Web of Things (WoT) technology. We propose a deep learning model capable of matching devices and services with a user query in natural language, returning a list of devices that matches the query. Our proposed method will help to integrate Artificial Intelligence (AI) and the Internet of Things (IoT) into industrial Cyber-Physical Systems (CPSs) and Smart ecosystems such as Smart Homes and Smart Cities. In doing so, more complex and precise searches could be made possible than those performed by traditional search systems. Figure 1 shows a simplified scenario, where a user sends a query as a natural language sentence and gets a list of recommended CPSs deployed

in a Smart Home building. The recommended list is sorted in descending order, and the four first devices best suit the user's request are returned.

Regarding communication and interoperability between devices and services, an existing problem is the heterogeneity of IoT devices. When IoT devices want to communicate, they need a standard way of doing so. The WoT initiative was created to resolve this problem, supported by the World Wide Web Consortium (W3C) through a series of recommendations. When using WoT, devices are searched for in a common way using the Thing Description (TD). This document defines the features and properties of the device, providing more information than just the device itself. Another method of finding devices is using the services described in the TD (properties, actions, and events), providing a more precise result than returning the entire device information.

Other problems when searching for devices are (a) the location of sleeping devices, devices that, following a period of time without activity, reduce their energy consumption by deactivating features; (b) excessive use of energy due to device response calls, for instance, through broadcasts; every time a device search is performed, all network devices will be called, using more energy, which in some cases may be limited; and (c) network overhead due to device scanning. In our paper, we solve these problems by searching through the repository of a discovery service, which contains the TDs of the available

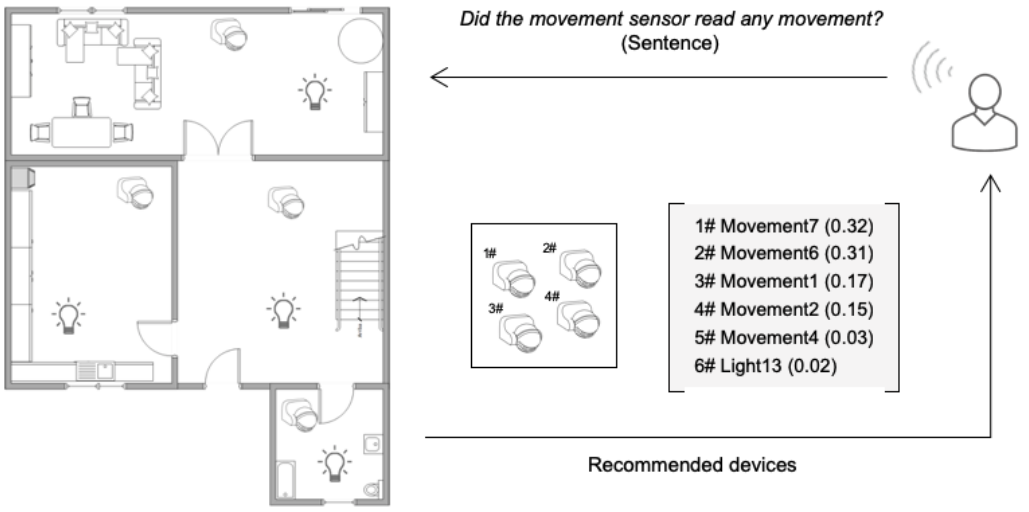


Figure 1: Proposed scenario.

devices, and out of these TDs, the recommender finds the device that best suits the user's request.

When a user sends a query in the form of a natural language sentence, the output of the recommender returns a device or a list of devices. This transformation or classification is made from a list of available devices. Accordingly, when the user looks for a device capable of meeting their requirements, the system will translate the sentence into a device available in the repository, similar to translating from one language to another. The recommender proposed uses the Transformer [4], an attention-based sequence transduction model. The Transformer is a novel solution that delivers better results regarding the problems it is used for, mainly in natural language problems, like translation [5]. For this reason, the recommender uses the Transformer to aid a decision-making system. It can be seen then that the **main contributions** of this paper are:

(a) A recommender for IoT systems based on a Transformer model.

(b) The support of natural language sentences to find relevant IoT devices or services.

(c) The ability of the recommender to generate its own natural language sentences.

(d) The use of the Web of Things paradigm to define IoT devices and services.

From a scientific point of view, the novelty is the application of Smart Sensing (SS) and Artificial Intelligence for CPSs to aid in smart decision-making, explicitly using the Transformer as a recommendation tool to match WoT devices. It is important to remark that users may not be familiar with the available Web of Things (WoT) devices or services. Its description is auto-generated by using the Thing Description (TD) to describe its features, properties, and capabilities, as well as its nature by applying query expansion. This technique enhances information retrieval effectiveness by extending or reformulating user queries [6, 7]. The user introduces a natural language sentence matching WoT devices in the discovery service repository. The sentence the user enters goes through a hashing process, and an Embedding layer is performed on that result. Finally, a Transformer layer is applied to the result obtained in the Embedding layer. The following research questions are addressed to identify the objectives to approach the aforementioned facts:

RQ1 *Is it possible to use deep learning to find relevant devices or services that users may require (i.e., recommendations) in CPSs through natural language?*

RQ2 *Can Transformers be used in industrial CPSs to aid the decision-making process for smart industrial scenarios?*

RQ3 *Can auto-generated descriptions for WoT devices or services be enough to be found by unfamiliar users that may not know their existence through queries in natural language?*

RQ4 *Is it apt to return a ranked list of WoT devices or services instead of providing a fixed query-component binding?*

This article is structured as follows. Section 2 offers an overview of various related projects about searching and applying the Transformer, and explains important terms. Section 3 presents a general overview of the proposed model and briefly describes the steps from the model. Section 4 provides an in-depth explanation regarding the steps of the proposed model before going on to deal with the experimental setup, from fetching data to the created model. Section 5 describes the results obtained from the experimentation process and the validation scenario. Section 6 discusses the results of the validation scenario. Finally, we discuss future directions in Section 7.

2. Literature Review

This section offers an overview of projects related to our proposal. Furthermore, a background of the related work is outlined to describe some of the terms used in the paper.

2.1. Background

The introduction of some of the terms used in the paper can be helpful before explaining our proposal. As it has been introduced, our work tries to find an equivalence or a proximity relation between a user query described in natural language, and the devices or services available in a CPS or a Smart ecosystem that have some kind of link with the query made. In this sense, devices often correspond to elements of an IoT-based solution, and the operations and interaction capabilities of those devices provide services.

Nevertheless, a current issue in the Internet of Things (IoT) is the heterogeneity of devices. Devices with the same features but made by different providers may have different ways of operating and communicating, making it harder for developers to create IoT solutions. For this reason, our work is based on the Web of Things (WoT) technology, a W3C initiative created in 2010 [8] with the idea of adapting application layer technology to IoT, thus solving the problem explained in the previous sentence. With this initiative, IoT fragmentation and the cost of IoT solutions are reduced, as each device has a common way of describing itself through the web, improving the interoperability and heterogeneity of the devices.

To extract the services to match the users' queries, the Thing Description (TD) is used, a JSON-LD document that contains basic information about the device: name, brief description, security information, and InteractionAffordance [9]. InteractionAffordance is a field, composed of properties, actions, and events that describes a way of interacting with a thing using hypermedia controls [10], *i.e.*, the services offered by the thing. Using the TD, machines and humans can understand the device and use the services that the device offers, *i.e.*, using the TD document, we can extract the services offered by the device that the user needs.

Discovery Service shapes the other part of our solution. A Discovery Service discovers devices, and their TDs are stored in the repository used by the recommender to respond to the user's request. Discovery Services facilitate the search for services that meet the user's request for many services offered. When a user has a request, the Discovery Service must be able to search for and return a service that matches the user's request [11]. It can unregister services on a repository or directory, allowing access to the offered services. In our architecture, we search through TDs stored in the repository associated with the Discovery Service.

These discovery services, which can help users and applications to identify, connect and interact with devices and services in the WoT, can be implemented from some straightforward techniques to others that are much more complex such as retrieval systems or recommender systems. In our research in the context of WoT, we choose an implementation based on Deep Learning recommender systems to provide a list of recommendations of devices and services that users might be interested in, so they can then choose from this ranked list according to which ones of them are likely to be more relevant and/or interesting.

In our proposal, with the list of TDs and a sentence sent by the user,

the proposed solution matches the user's query into the available devices or services. The available services are encoded using the label encoding technique, while the sentences are encoded using the hashing technique. Label encoding is an encoding technique where an integer number is assigned for each word. In our solution, it is employed to transform the services used for the recommender into numbers to train and evaluate the model.

Hashing is an encoding technique used to vectorize the features, creating a vector of nxn hash integer features for the n available features [12]. In our solution, hashing is used to vectorize the sentences obtained from the user's requests, using a large vocabulary size to avoid collisions when hashing the sentences. Finally, the model is trained using Embedding and Transformer layers. Embedding is a process that compacts high-dimensional vectors into a low-dimensional space to ease the operation process through the available data [13]. Once the space is reduced, a Transformer layer trains the model. The Transformer is a novel sequence transduction model based on multi-head self-attention [4], on which it relies to pay attention to multiple words of a sequence in parallel, considering all the possible relations between them. It is used on translation tasks and performs better than convolution and recurrent-based solutions.

2.2. Related Work

In IoT environments, information can be gathered from devices or external sources. In addition, the information returned to the user relates to the search techniques used when collecting the information. Our research is focused on returning the device information or the services that best suit the user's requirements. As such, our work is compared with published work about searching device information and services, information retrieval and recommender systems, mainly in WoT, because of its similarity to our solution. Non-WoT work that focuses on similar search techniques is also included in the comparison.

There are many research efforts on WoT technology [14, 15, 16]. In [14], the authors designed and developed a recommendation system for WoT APIs using user preferences and device similarity between the queries and the information stored about the available APIs. Instead of using the whole API for the recommendation system, our research uses the different services offered by the discovery service API. In [15, 16], the main topic concerns discovering WoT devices. In [15], WoT Store is proposed, a platform to manage and discover WoT Things. It also has a Thing Manager and an

Application Manager. The Thing Manager allows users to search for Things by filling in a form with a list of predefined fields. A set of Things is returned as a result of the search operation. For the query, semantic data is used. The work of *Bovent and Hennebert* [16] centers on finding devices over the network. Deployed devices are searched for through semantic queries using SPARQL. Devices in the network with the resources that match the queries will respond to the user's request.

Some researchers explored techniques to find or discover devices [17, 18, 19]. They proposed an IoT framework capable of discovering IoT devices. In [17], the framework can discover, index, and search for IoT devices (IoTCrawler). The search process is performed through GraphQL and NGSI-LD. When a GraphQL query is built, a set of NGSI-LD queries resolve the GraphQL query. IoTCrawler facilitates the searching and ranking streams of data, sensors, platforms, and observable properties using different ontologies. In [18], apart from the framework for the IoT search engine, the authors proposed a naming service and used Long-Short-Term Memory (LSTM) for query prediction. The proposed framework can search IoT devices by location, function, and service type. Devices returned by the search engine can be ranked. The query prediction module predicts the volume of queries to aggregate similar queries and reduce system usage. Finally, in [19], the IoT framework is called COBASEN, a software framework to discover and interact with IoT devices. The Context Module discovers IoT devices connected to the middleware. After the devices are discovered, stored, and indexed, the search engine allows users to perform queries that return a ranked list of IoT devices.

In the topic of retrieval systems, authors focus on discovering and searching for devices by using their metadata and context information. In [20], the authors propose a retrieval system for IoT that extracts metadata from the internet to make the discovery process context-aware. In addition, the authors propose the usage of metadata to calculate the relevant rank of contents in a given topic. In contrast, our proposal uses the sentence given by the user to return a list of ranked devices. In our proposal, the rank is performed using a Softmax function, sorting the result by the confidence the ML model has in each solution. Furthermore, context information can be extracted from the Thing Description to make the search process context-aware, *e.g.*, using the location of the device. In [21], the authors review the current trends in IoT, proposing as future work the use of ML to enhance data retrieval in IoT to solve the challenge of managing large amounts of data. Our work

proposes using ML to enhance the retrieval of IoT information by matching users' queries with natural language sentences, thus solving the challenge of managing large amounts of data related to the description of the device.

Regarding classification, the Transformer is an attention-based sequence transduction model used mainly for natural language problems [4, 22]. During the search process for IoT devices, natural language is used to search the available devices or services. However, machine learning is not used in the field of WoT, and in the field of IoT, it is used but not to search for devices or services. Consequently, the papers referenced in the related work are focused on natural language or, in the case of IoT papers, predicting the measurement value of devices.

In [23, 24, 25], the Transformer is used to create a machine-learning model which can solve natural language problems. In [23], the authors proposed using the Transformer for speech recognition. Causal convolution is used for context modeling and frame rate reduction, and Transformer and VGGNet for encoding. Results from the experiment with LibriSpeech data show that the Transformer performs better for encoding. Conversely, for prediction, LSTM is a better alternative. In [24], the authors suggested using the Transformer as an encoder with a CNN-based feature extractor and a position-wise classifier to perform sound event detection. The authors evaluated the proposal using the Transformer with the DCASE2019 Task 4 dataset, where the Transformer outperforms the CRNN-based model. Finally, in [25], the authors used Embedding and the Transformer to analyze the sentiment of tweets written in Spanish. The proposal is evaluated using a dataset provided by Task 1 of the 2019 edition of the TASS workshop. Using tweets from five Spanish language variants, the Transformer ranked first in analyzing two and second in analyzing the other three.

Another study regarding the use of the Transformer is [26], where the authors used it in an IoT scenario. In such an approach, the authors proposed DeepHealth, a framework for instant intelligent predictive maintenance. Using the Transformer, DeepHealth can evaluate industrial facilities' current and future health conditions. For future condition evaluation, the Transformer predicts the next sequence of sensor signals. The output of the prediction is used again by the Transformer to assess the aforementioned health conditions of the industrial facilities.

Finally, recommender systems are widely used in many fields and applications [27, 28]. It includes research focused on recommender systems in IoT environments. In [29], the authors proposed a system to recommend services

related to IoT devices. The user queries for services that improve or support the CPSs associated with the user profile. After a list of services is retrieved, the recommender filter and score the services using the information of the user's profile and similar profiles. In [30], the authors proposed a voice recognition system integrated with a recommender for controlling IoT devices in a Smart Home. The system has an authorization system using image recognition. After the user is authorized, the user can send commands via a chatbot or voice commands. In the case of using voice commands, the voice is transformed into text and tokenized to extract the desired action. Finally, devices are recommended using the nearest neighbour algorithm with information about past choices and choices made by users with a similar profile. The last paper about recommender systems in IoT environments is RecRules [31]. In RecRules [31], the authors proposed a recommender based on IF-THEN rules. The system recommends rules to connect a pair of devices, causing an action on the latter device when a trigger is detected on the other device. The model gets better results than similar proposals when trained using collaborative, technology, and functionality paths using Random Forest as the main algorithm.

Table 1 shows the comparison between our proposal and the related work papers, describing the following aspects:

(a) Objective: Indicates the main objective of the paper;

(b) Syntactic: Indicates whether the paper uses syntactic information (including queries) to accomplish the objective;

(c) Semantic: Indicates whether the paper uses context information from the user's queries (including semantic queries);

(d) Natural Language: Indicates whether the natural language (including syntactic and semantic data) is used to accomplish the objective;

(e) Transformer: Indicates whether the Transformer algorithm or any variant of the algorithm is used;

(f) IoT: Indicates whether the scope of the problem is related to IoT;

(g) WoT: Indicates whether the scope of the problem is related to WoT.

Research Work	Objective	SY	SE	NL	TR	IoT	WoT
Meissa et al. (2021) [14]	Recommender	○	○	○	○	○	●
Sciullo et al. (2020) [15]	Manage & discover	●	●	○	○	○	●
Bovet and Hennebert (2014) [16]	Discover	○	●	○	○	○	●
Iggena et al. (2021) [17]	Discover & index	●	●	○	○	●	○
Hatcher et al. (2021) [18]	Discover & prediction	●	○	○	○	●	○
Lunardi et al. (2015) [19]	Discover	●	●	○	○	●	○
Zhao et al. (2015) [20]	Retrieval System	●	●	○	○	●	○
Younan et al. (2020) [21]	Survey IoT	●	●	○	○	●	●
Yeh et al. (2019) [23]	Classification	●	●	●	●	○	○
Miyazaki et al. (2020) [24]	Classification	●	●	●	●	○	○
González-Barba et al. (2020) [25]	Classification	●	●	●	●	○	○
Zhang et al. (2021) [26]	Classification & pred.	●	●	●	●	●	○
Mashal et al. (2016) [29]	Recommender	●	●	○	○	●	○
Torad et al. (2022) [30]	Recommender	●	●	●	○	●	○
Corno et al. (2019) [31]	Recommender	●	●	●	○	●	○
Our Work	Recommender	●	●	●	●	●	●

Table 1: Summary of related work (SY: Syntactic, SE: Semantic, NL: Natural Language, TR: Transformer) (● included, ○ not included).

As shown, our approach combines the search and recommendation process in WoT with classification techniques using deep learning. We have previously carried out a small proof of concept [32] using a small dataset with 645 instances to evaluate the usage of Transformer as a recommendation tool for finding relevant WoT devices in CPSs. In [18], the authors used machine learning to predict query aggregation and improve query performance. However, machine learning is not used in the search process. In our proposal, machine learning and natural language are used in the search process for devices and services. Furthermore, our proposal also uses the Transformer as a novel technique to accomplish the search objective and recommend devices and services. Deephealth [26] uses the Transformer for classification and prediction in IoT environments, although it is used to classify and predict the sensor signals. In our proposed method, the Transformer matches the sentence sent by the user with one of the devices or services available, being the first time using the Transformer as a recommender in IoT and WoT.

3. Overview

This section provides a general overview of the proposed Transformer-based recommender model for matching WoT devices and queries in the form

of natural language sentences introduced by the user. Figure 2 represents the proposed model, which is not a formal methodology but defines the pipeline of steps and processes used to achieve the paper's objective. The processes represented in Figure 2 range from data fetching to ranking results from the Softmax function. As Figure 2 shown, our approach consists of seven steps, two of them related to Fetching and Pre-processing data, the next two steps related to the data processing to adapt the data to machine learning, and the last steps related to training and applying the models. At the end of step six, a Softmax function is performed through the result of the Transformer layer. Furthermore, the Softmax output is sorted in descending order to return a list of recommended devices to the user, instead of only one possible device. If the first device on the list has a high enough value, in our case higher than 0.75, then only the first device on the list is returned to the user.

As the proposal aims to aid the decision-making process, this solution returns a device if the system is sure that the returned device matches the user's request, keeping the user from having to make a decision. However, if the system is not sure about the returned device, it returns a list of top-k devices, forcing the user to select one device among them, but from a smaller list than using the full list of devices. The main steps of the approach are:

(a) **Fetching Data**. This step involves getting compatible datasets and fetching them for the proposed model. In our approach, we extract the data from a dataset about anomaly detection algorithms in IoT. In-depth details of this step can be found in Subsection 4.1.

(b) **Feature Engineering**. This step consists of pre-processing the data before using it. Unused features are removed, needed features are added, and features are combined. This section is required to enhance the model and get better results. In-depth details of this step can be found in Subsection 4.2.

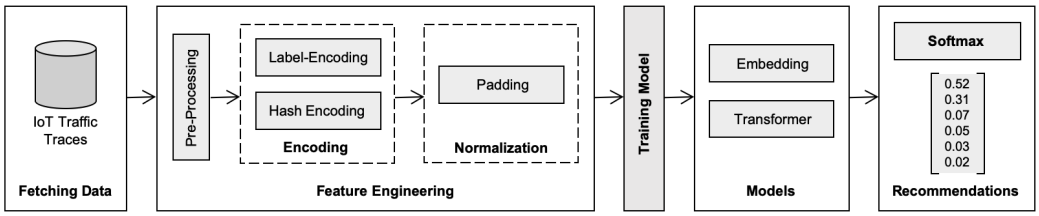


Figure 2: Pipeline of steps and processes for smart decision-making.

(c) **Encoding**. This step consists of encoding the features to make them interpretable by machine learning algorithms. This step uses two feature encoding strategies: label encoding for the classification features and hashing for the users' sentences. Additionally, tokenization was used instead of hashing to evaluate the performance of using individual words in the Embedding process instead of the whole sentence. In-depth details of this step can be found in Subsection 4.3.

(d) **Normalization**. This step consists of normalizing the data encoded to homogenize it. Padding transforms each vector for vectors of the same size, making them usable by the machine learning algorithm. In-depth details of this step can be found in Subsection 4.4.

(e) **Training Model**. This step consists of randomly splitting the dataset into two groups, the data used to train the model and the data used to validate the model. In-depth details of this step can be found in Subsection 4.5.

(f) **Model**. This step involves creating a model using machine learning algorithms to match the devices with the users' sentences. Two layers are used in this step: the Embedding Layer to compact the vector and the Transformer to operate over the Embedding Layer result and classify the sentences. In-depth details of this step can be found in Subsection 4.6.

(g) **Recommendations**. After the model is created, a Softmax function is performed through the result of the Model to aid in the decision-making of the device. The Softmax function obtains a list of the available devices and their match score with the user's sentence in descending order. In-depth details of this step can be found in Subsection 4.7.

4. Creation of the Recommender Model

This section explains the rationale for implementing each proposed step to help the reader understand why the process is split into the proposed seven steps. Furthermore, this section explains the proposed model and the experimentation performed. Finally, how each step has been applied is described in detail, explaining the type of experimentation carried out with the dataset for each step.

4.1. Fetching Data

The dataset used for the research, *mainSimulationAccessTraces.csv* (IoT Traffic Traces), is a dataset created to evaluate anomaly detection algorithms. Devices deployed in a Smart Home call other services through HTTP endpoints. The dataset has IoT devices, services, locations in the Smart Home, and operations. However, although a feature representing the user's sentence is missing, the dataset is still studied due to its similarity to our problem and its large scale of observations. The dataset has a set of services for each device, a problem that simulates a TD of the WoT. It also has 357952 observations of 13 features so that the model can learn and result in a good recommender for this large amount of observations. Figure 3 shows the distribution of the most frequent services used for the recommendation process. The most common services used in the dataset are movement devices, temperature devices, and battery devices. These devices are shown in Table 2. They represent 98.74% of the total services available in the dataset. As the recommender model uses natural language sentences and a vocabulary of words for the training process, the non-homogeneous distribution between different devices is not a problem. Completely different services can be easily matched correctly by the recommender. However, the non-homogeneous distribution between the same type of device in different places can make the recommender model mismatch the correct device. For instance, each device has a service to register the device in the discovery service. This service is called once in the whole dataset, and the name is very similar to the rest of the device's services. For that reason, that service will be mismatched with other services of the same device. This behavior is analyzed in Section 5. Table 3 shows the set of features from the dataset. The latter was obtained from Kaggle[1] in the form of a CSV file.

The reason for selecting this dataset is that it is the only one available with a large number of observations to train and validate the machine learning model. In addition, this dataset fits our research problem, a service-centric dataset that allowed us to match user queries with CPS services. To create the sentences associated with each observation of the dataset, we need information related to the devices. Using WoT technology, the required information is described in the TD, while in this dataset, we have the information defined in each observation. Each observation defines a call to a

[1]Dataset – `https://www.kaggle.com/francoisxa/ds2ostraffictraces`

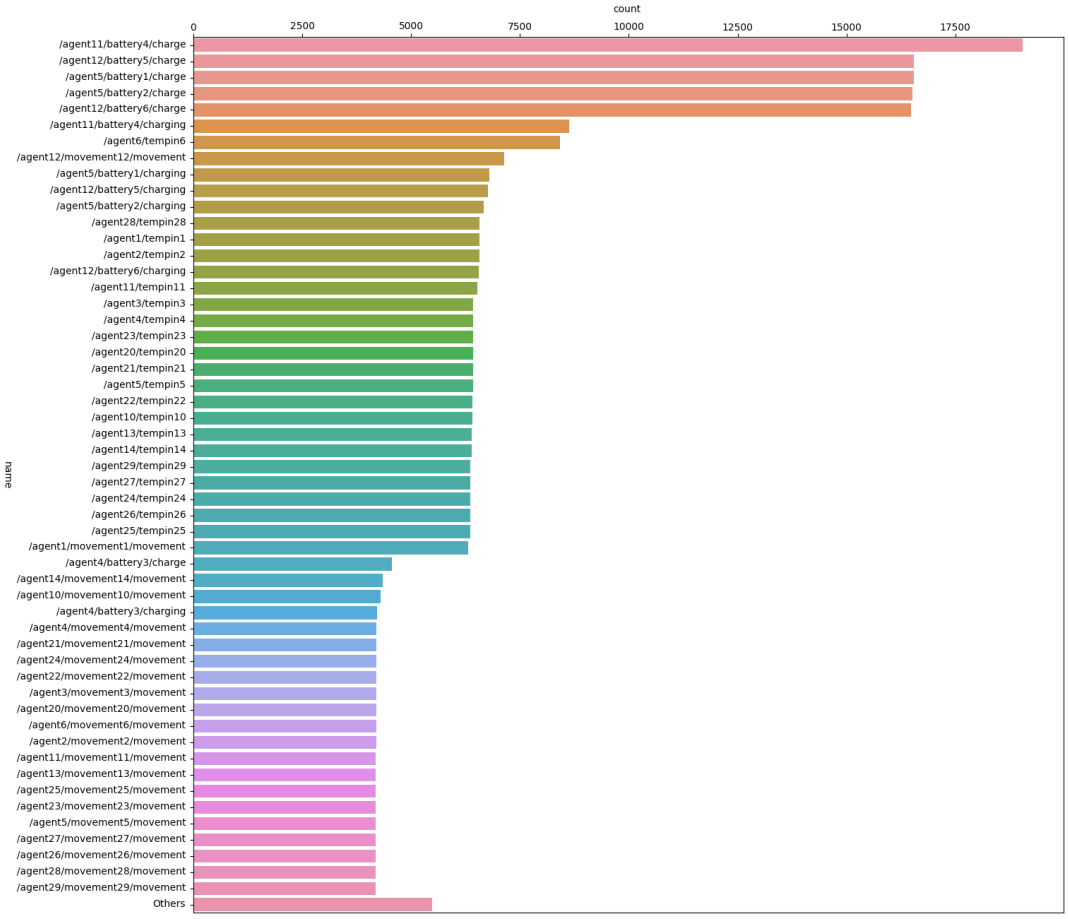


Figure 3: Dataset distribution of the most frequent device services.

#	device	count	#	device	count
1	/agent11/battery4/charge	19032	28	/agent27/tempin27	6350
2	/agent12/battery5/charge	16531	29	/agent24/tempin24	6349
3	/agent5/battery1/charge	16526	30	/agent26/tempin26	6347
4	/agent5/battery2/charge	16494	31	/agent25/tempin25	6347
5	/agent12/battery6/charge	16460	32	/agent1/movement1/movement	6293
6	/agent11/battery4/charging	8611	33	/agent4/battery3/charge	4540
7	/agent6/tempin6	8407	34	/agent14/movement14/movement	4336
8	/agent12/movement12/movement	7124	35	/agent10/movement10/movement	4277
9	/agent5/battery1/charging	6784	36	/agent4/battery3/charging	4204
10	/agent12/battery5/charging	6748	37	/agent4/movement4/movement	4189
11	/agent5/battery2/charging	6649	38	/agent21/movement21/movement	4186
12	/agent28/tempin28	6552	39	/agent24/movement24/movement	4184
13	/agent1/tempin1	6551	40	/agent22/movement22/movement	4183
14	/agent2/tempin2	6550	41	/agent3/movement3/movement	4183
15	/agent12/battery6/charging	6540	42	/agent20/movement20/movement	4183
16	/agent11/tempin11	6512	43	/agent6/movement6/movement	4182
17	/agent3/tempin3	6416	44	/agent2/movement2/movement	4179
18	/agent4/tempin4	6414	45	/agent11/movement11/movement	4178
19	/agent23/tempin23	6406	46	/agent13/movement13/movement	4174
20	/agent20/tempin20	6405	47	/agent25/movement25/movement	4174
21	/agent21/tempin21	6404	48	/agent23/movement23/movement	4172
22	/agent5/tempin5	6401	49	/agent5/movement5/movement	4172
23	/agent22/tempin22	6397	50	/agent27/movement27/movement	4171
24	/agent10/tempin10	6397	51	/agent26/movement26/movement	4170
25	/agent13/tempin13	6384	52	/agent28/movement28/movement	4169
26	/agent14/tempin14	6370	53	/agent29/movement29/movement	4169
27	/agent29/tempin29	6351	54	Others	5475

Table 2: Most frequent device services.

service available in a Smart Home scenario; in other words, each observation is a request sent by a user to a device to execute one of the services the device has available. Therefore, we complete each request with a natural language sentence built using the information of each service to adapt the dataset to our study case.

4.2. Feature Engineering

To enhance the model and get better results a basic feature engineering process is applied. Feature engineering transforms the dataset, creating a new one with features that better suit the proposed model. To transform the dataset using feature engineering, a wide range of techniques are used [33]. This paper uses feature creation and feature deletion techniques to pre-process the data before creating the model. Feature creation is applied to create new derived features that complete the dataset and are useful in the recommender model. In contrast, feature deletion is applied to remove features that do not contribute to the recommender model. To apply feature engineering techniques to our problem, we have to consider that the features selected must allow the system to classify users' sentences with the highest

Table 3: Set of features describing the IoT Traffic Traces datasets.

Feature	Description	Class description	#Class
sourceID	Device ID that performs the query.	Categorical: lightcontrol1, movement2, tempin2, tempin4, etc.	84
sourceAddress	Device service address of the source device.	Categorical: /agent2/lightcontrol2, agent4/movement4, etc.	89
sourceType	Device service type of the source device.	Categorical: /lightControler, /movementSensor, /sensorService, etc.	8
sourceLocation	Device location of the source device	Categorical: BedroomParents, Kitchen, Garage, Watterroom, etc.	21
destinationServiceAddress	Device service address that receives the query.	Categorical: /agent2/lightcontrol2, /agent4/tempin4, etc.	85
destinationServiceType	Device type of the destination device.	Categorical: /lightControler, /movementSensor, /sensorService, etc.	8
destinationLocation	Device location of the destination device.	Categorical: BedroomParents, Kitchen, Garage, Watterroom, etc.	21
accesedNodeAddress	Operation of the destination service called.	Categorical: /agent3/movement3/ lastChange, /agent12/battery5/charge, etc.	170
accesedNodeType	Type of operation of the destination service.	Categorical: /derived/boolean, /basic/number, /sensorService, etc.	12
operation	Classification of the performed operation.	Categorical: registerService, write, read, subscribe, etc.	5
value	Value sent to the service.	Numerical and Binary	-
timestamp	Time when the operation was performed.	Timestamp	-
normality	Indicates if the connection is normal or anomalous.	Categorical: normal, anomalous(DoSattack), anomalous(scan), etc.	8

accuracy. That means the features must be able to represent a user's sentence and features that represent the device. The deleted features are:

(a) Source features: All the source features are removed from the dataset (`sourceID`, `sourceAddress`, `sourceType`, and `sourceLocation`). For our solution, only the information about the available devices is required to classify the queries, not the user.

(b) `destinationServiceAddress`: Looking at Table 3, this feature seems similar to the feature `accesedNodeAddress` but with a lower number of classes. This similarity is due to this class being included in the feature `accesedNodeAddress`. As such, it is deleted from the dataset.

(c) `accesedNodeType`: Looking at Table 3, this feature is similar to the feature `destinationServiceType` but with a higher number of classes. This similarity is due to the inclusion of `destinationServiceType` in this feature. However, `destinationServiceType` was selected because it is more accurate for our problem. For instance, an operation on a light is specified as */derived/boolean* instead of being represented as */lightControler*.

(d) `value`, `timestamp`, and `normality`: In our solution, these features do not add relevant information to infer knowledge. Accordingly, they are deleted.

The remaining features (`destinationServiceType`, `destinationLocation`, and `operation`) are merged into one new feature (1). Following this structure, users' sentences can be auto-generated by using the information of the Thing Description (TD) to describe their features, properties, and capabilities as well as their nature. Furthermore, the auto-generation structure can improve users' sentences by extending user queries with additional information that can help in the recommendation process; for instance, by adding the location of the desired device in the query.

This new feature represents the three merged features and creates a feature that simulates a user's sentence. As the sentences are created using the same schema for each service, they are not very diverse, with a homogeneous language structure and a medium complexity. This solution can help initially train the recommender model to adapt it to new devices before storing real natural language sentences. An example of a sentence from this feature is defined in (2).

I need to + `op` + `destinationServiceType` + *in* + `destinationLocation`. (1)

I need to write a light controller in the bedroom. (2)

After applying feature engineering, the dataset is described by two features, by `accessedNodeAddress`, a categorical type with 170 classes; and by `Sentence`, a text type feature defined in (1).

4.3. Encoding

In machine learning, algorithms may need to transform categorical data into numerical data to make the data usable. The transformation of these features by the machine learning algorithms is called encoding [34]. To apply encoding, a wide range of techniques are used [35]. This paper uses two encoding techniques to transform categorical data into numerical data: label encoding and hashing. These techniques were explained in Section 2.1. Label encoding is used to keep the order of the features used for the recommender. In contrast, hashing creates a numerical vector of the feature representing the user's sentences. Another technique used is tokenization, which represents, for each sentence, a vector with the number of times each word is repeated in the sentence, creating a mxn vector where m is the number of different words (the vocabulary) in the largest sentence, V_m and n is the number of sentences (3).

$$m = |V_m|; n = |V_T|; V_m \in V_T \implies \exists V_m | \forall V_n \in V_T : |V_m| \geq |V_n|. \quad (3)$$

For the dataset, the features selected are categorical or text. Label encoding is used on the device features and hashing on the sentence features. For the second experiment, Tokenizer was used instead of hashing on the sentence features. The transformation operations are described below:

(a) Label encoding: *accesedNodeAddress* feature is encoded into numerical values. All the classes are encoded in the $\{0, 1, \ldots, \#Class\}$ range of values. After a sentence is matched with a device, an inverse transformation can be performed to decode the result and get the matched device.

(b) Hashing: The *Sentence* feature is encoded into vectors using the function *one_hot* from Keras, which performs a hash encoding of each sentence using md5. This is done using a vocabulary size higher than the number of different words from the sentences. Using a number lower than that of the different words may result in collisions in the encoding process. The vocabulary size is 250. Each sentence, V_T, is defined as a subset, V, of values between 0 and the vocabulary size, where each value represents a word, p, from the available vocabulary, P (4).

(c) Tokenizer: In the second experiment, the *Sentence* feature is encoded using Tokenizer, which transforms the *Sentence* feature into vectors. Once the *Sentence* feature is represented as a vector, Tokenizer transforms the vector into a matrix representing each sentence as the repetition of each available word in the vocabulary.

$$V_T = \{V_0, V_1 \dots V_n\}; V \subseteq P; p \in P; P = \{0, 1 \dots vocabularySize\} \tag{4}$$

4.4. Normalization

After applying feature engineering techniques or encoding techniques, features may need a transformation process to homogenize the data, *i.e.*, to represent the features commonly. Padding is used on our features to transform each vector created in the encoding step for vectors of the same size. Padding is a technique usually used in image processing when processed by Convolutional Neural Networks. It adds new values to the vector to increase its size [36]. In our model, it is used to add new values to each vector sentence, improving the model's accuracy. Figure 4 shows an example of a natural language sentence's encoding and normalization process.

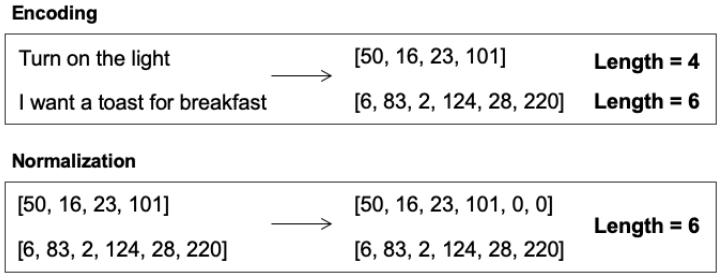


Figure 4: Encoding and normalization example.

4.5. Training Model

Before applying the proposed model and after the data have been processed and adapted to the machine learning algorithms, the dataset is split into two sets of data: (a) the training dataset, which represents 75% of the instances; and (b) the validation dataset, which makes up the remaining 25% of the instances. The splitting process is performed randomly, distributing the classes homogeneously across the training and test sets. Therefore, as they would be computationally expensive and would not give useful information about the model, there is no need to use techniques to evaluate the distribution of the datasets or to homogenize the model in the validation process.

4.6. Model

The model is the main step of the machine learning process. Many machine learning algorithms exist to create a model for performing tasks such as recommending. Each algorithm displays better results depending on the type of problem. For instance, the Transformer is a novel algorithm used mainly for natural language classification problems. Embedding and the Transformer are used in our work to build the recommender. An Embedding layer is applied to compact the vectors into low-dimensional space, and the Transformer is used on the dataset as the main algorithm of our solution. Using the Transformer is to study whether this novel algorithm can resolve IoT and WoT classification problems. This paper also studies whether the Transformer can improve previous IoT and WoT research results.

(a) Embedding: Natural language features are represented as high-dimensional vectors. To ease the operation process and obtain better results when applying the Transformer layer, an Embedding layer for both tokens and the token index is created before using the Transformer layer, compacting the high-dimensional vectors into a low-dimensional space. The selected dataset has an Embedding layer of *output dimension* equal to 128, *input dimension* of the tokens equal to the vocabulary size used, that is 250; and *input dimension* of the position equal to the maximum length of the vectors, that is 9, for the experiment using hashing; and 40 for the experiment using Tokenizer.

(b) Transformer: The Transformer is an attention-based sequence transduction model and has been used for natural language problems (5).

Our proposal is based on matching between WoT devices and sentences, a natural language problem. In that regard, the Transformer is used as the main layer of the model. The low-dimensional vector obtained from the Embedding layer is used for the encoder-decoder structure of the Transformer architecture. For the Transformer encoder and decoder, 2 attention heads are used. Furthermore, the hidden layer size of the feedforward network is 128, and the size of each attention head for the query and the key is the *output dimension* of the Embedding layer (128).

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \ . \tag{5}$$

4.7. Recommendations

The model learns from the input data using the algorithms selected. However, the result for the model must be adapted to extract useful information, and there are transformation techniques for this, two of which are Sigmoid and Softmax. In this paper, ReLU (6) and Softmax (7) are used as activation functions in fully connected layers (Dense) to extract useful information from the result provided by the model. ReLU is used at intermediate layers, giving outputs of 0 if the input is negative and returning the input value if it is positive. In addition, the softmax function takes an input vector and returns a probability distribution over the classes, it has the effect of scaling the values such that they all fall within the range [0, 1] and sum to 1. The highest value is returned in the Softmax function as the device best suits the user's request. However, the Softmax function is modified to create a list providing all the results instead of just the best result. The modification of the Softmax function allows a list of the top-k results to be returned. This modification prevents the return of a non-precise value because if the first result is under 0.75, the user gets a sorted list in descending order of the top-k results to select the device that best suits the user's request.

$$f(x) = max(0, x). \tag{6}$$

$$\sigma(\overrightarrow{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \ . \tag{7}$$

The Softmax function is applied to 170 neurons, representing each possible device that can match the user's sentence. Finally, the value or vector of values is returned, decoding the value to get the matched device. In the case of a single device, the service obtained is executed automatically. On returning a vector of values, the top-k values are returned, allowing the user to select the device among the recommended devices.

5. Results

This section shows the results of the proposed model using different configurations. The experiments aim to study whether the model could return relevant recommendations of devices or services to the users by processing natural language queries. The number of neurons, the dropout, and variables such as the batch size or epochs were modified for the experiments. Furthermore, two configurations were carried out, one using hashing to encode the user's sentences and another using tokenization to pre-process the user's sentences. Finally, a validation scenario was created using the proposed models and compared with the current search system to validate the results.

5.1. Experimentation

Table 4 shows the layers and the accuracy of the models trained for the experimentation of the Hash Encoding model. The first four layers used of the models are the same for each model: an Embedding layer, the Transformer layer, a GlobalAveragePooling1D layer, and a Dropout of 0.1. The embedding layer is required to use the Transformer algorithm. The model uses these layers because of the increased accuracy and to compact the high-dimensional vectors into a low-dimensional space. The last two columns describe the validation and final accuracy of the models.

Figure 5 shows the final configuration selected from the experimentation, marked in Table 4, in which validation accuracy was higher than the testing accuracy, and the value loss was lower than the testing loss. The reason for this result is the use of *Dropout*, a regularization method that drops or ignores neurons from the neural network (*e.g.*, *Dropout(0.5)*, which means that 50% of the neurons are dropped). Dropout avoids overfitting in the training process, leading to a more robust model and higher accuracy.

Regarding the hyperparameters of the selected model, Table 5, shows the hyperparameters used in the training process. For selecting the best configuration, different hyperparameters were used to keep a high value of accuracy

while avoiding overfitting in the model. After not getting improvements in the results, the model with the highest accuracy was selected.

Our best result using hashing reaches a value of 79.62%, which is a good result bearing in mind that the sentence is created using the available features from the original dataset. Figure 6 and Table 5 show the accuracy results in the training and validation process. The results with different hyperparameters show similar results. Using a relatively low batch size (5000) to train the model delivers better and faster results than a high batch size, reaching the peak accuracy value using less number of epochs.

A problem that may appear is overfitting due to the feature creation being applied to the dataset, meaning the model might not be able to classify new sentences. Regarding this possible problem, we have to take into account that sentences can be created following the proposed schema using the information available from the request: (a) operation, (b) service, and (c)

5.Layer	6.Layer	7.Layer	8.Layer	9.Layer	10.Layer	Output Layer	AV	ACC
Dense(512, ReLU)	Dropout (0.5)	Dense(256, ReLU)	Dropout (0.5)	-	-	Dense(170, Softmax)	78.37%	78.24%
Dense(512, ReLU)	Dropout (0.8)	Dense(256, ReLU)	Dropout (0.5)	-	-	Dense(170, Softmax)	76.08%	76.18%
Dense(128, ReLU)	Dropout (0.8)	Dense(64, ReLU)	Dropout (0.5)	Dense(32, ReLU)	Dropout (0.1)	Dense(170, Softmax)	75.94%	75.96%
Dense(256, ReLU)	Dropout (0.8)	Dense(128, ReLU)	Dropout (0.5)	-	-	Dense(170, Softmax)	75.85%	75.96%
Dense(128, ReLU)	Dropout (0.8)	Dense(64, ReLU)	Dropout (0.5)	-	-	Dense(170, Softmax)	72.79%	72.86%
Dense(128, ReLU)	Dropout (0.8)	Dense(64, ReLU)	Dropout (0.5)	-	-	Dense(170, Softmax)	58.27%	58.05%

Table 4: Layer configurations of the Hash Encoding model experimentation (AV: Validation Accuracy; ACC: Accuracy)

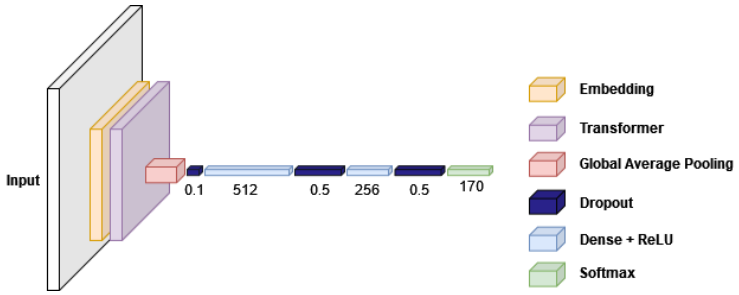


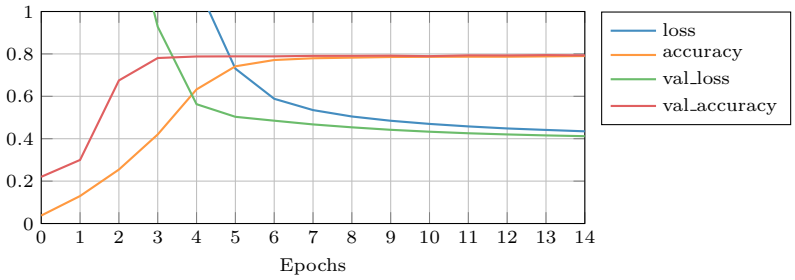
Figure 5: Diagram of the proposed model.

location. The information can be extracted from the original user's sentence to create sentences matching the devices. A validation scenario is explained in the next subsection to validate the model and check whether an overfitting problem exists.

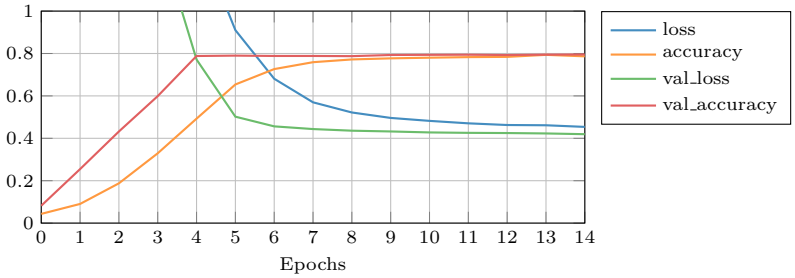
The first experiment was performed using the whole sentence for the model creation. Each sentence is transformed to numerical values using hashing and vocabulary size. Another solution is to create the model using each word from each sentence, similar to One-Hot encoding. To split each sentence into words, tokenization is used. With a tokenizer, each sentence is represented as a vector of numerical values, where each value represents the number of times a word is used in the sentence. The tokenization configuration's accuracy value is higher than the experiment using hashing, 80%. The configuration used is the same as that used in the hashing experiment. In the dataset, the vector's length is lower than the number of observations (40) because the sentences are created following a schema. Accordingly, the same words are repeated in all the sentences, *i.e., smaller vocabulary size*, the four words used to build the schema: (a) I, (b) need, (c) to, and (d) in; and the three features used: (i) operation (5 classes), (ii) `destinationServiceType` (8 classes) and (iii) `destinationLocation` (21 classes). This result shows that encoding each word instead of each sentence in datasets with a small vocabulary may increase the accuracy value. However, the accuracy may be compromised for datasets with an extensive vocabulary. Finally, the model is used in the validation scenario, and the result is compared to the model created using hashing to check whether overfitting exists in the experiments.

EO	**FFS**	**B**	**E**	**AV**	**ACC**
128	128	5000	10	79.83%	79.62%
128	128	5000	50	79.31%	79.51%
128	128	2000	5	79.71%	79.52%
128	128	10000	20	79.49%	79.59%
128	128	30000	20	79.52%	79.39%
64	64	5000	10	79.50%	79.58%
64	64	2000	5	79.44%	79.57%
64	64	10000	20	79.44%	79.57%

Table 5: Results for the selected layer of the Hash Encoding model (EO: Embedding Output; FFS: Feed Forward Size; B: Batch Size; E: Epochs; AV: Validation Accuracy; ACC: Accuracy)



(a) Accuracy in the hashing model.



(b) Accuracy in the tokenization model.

Figure 6: Accuracy results in training and validation process using a batch size of 5000.

5.2. Validation

In the previous subsection, a configuration for each experiment was selected, bearing in mind the accuracy. The validation was considered to select the best configuration in the case of configurations with the same accuracy. In this subsection, a validation scenario is described for the validation process. When evaluating models, the accuracy has to be evaluated using another metric, the reason being is because the accuracy can mislead the researcher, for instance, having a high level of accuracy in the creation of the model, but the opposite in real-life situations or classifying new data.

The Softmax function's output value is used as a metric to evaluate the recommender, indicating the confidence level that the model is about the recommended devices. However, this metric is not a valid evaluation metric for the performance of the recommender. Thus, in addition to this metric, *Precision@n*, *Recall@n*, and *Coverage* are calculated to measure the quality of the suggestions. For *Precision@n* and *Recall@n*, the evaluation can be around the relevant recommendations [37] or around the number of positives and negatives returned [38, 39]. In the case of using positives and negatives,

Precision@n and *Recall@n* measure the relevant recommendations returned (true positive), the nonrelevant recommendations returned (false positive), and the relevant recommendations not returned (false negative). As in our scenario, the number of recommendations returned is preordained; we used *Precision@n* [40], where the set of recommendations returned can be relevant recommendations for the user, defined as $N = \{N_1, N_2, .., N_i\}$, that is a subset of the possible relevant recommendations that the user can receive, $R = \{R_1, R_2, .., R_r\}$, defined as $N \subseteq R$.

The *Precision@n* is a metric that measures the number of relevant recommendations produced (8) that can be defined as follows:

$$Precision@n = \frac{i}{t}\,. \tag{8}$$

where,

- i is the number of relevant recommendations $|N|$.
- r is the number of possible relevant recommendations $|R|$.
- t is the number of devices returned in the recommendation $|T|$.
- T is a set of devices returned in the recommendation $T = \{T_1, T_2, .., T_t\}$.
- D is a set of available devices $D = \{D_1, D_2, .., D_d\}$.
- d is the number of available devices $|D|$, being d in our case of study 170.

In our case study, t is 4. T is a subset of the available devices ($T \subseteq D$). The *Recall@n* is a metric that evaluates the number of recommended relevant devices (9) that can be defined as follows:

$$Recall@n = \frac{i}{r}\,. \tag{9}$$

Finally, *Coverage* is the percentage of devices that are recommended (10). With a low value of *Coverage*, it is harder for the recommender to have high values for *Precision@n* and *Recall@n*. *Coverage* can be defined as follows:

$$Coverage = \frac{t}{d}\,. \tag{10}$$

For the validation, a pertinent scenario is created, in which the user can select the model that will process the request. After the user introduces the request, the system matches the request using the selected model with the available devices. In the validation scenario, the matching process returns a

sorted list in descending order of the four devices that best suit the user's request. The idea is only to return the first and execute the service attached to the device when the accuracy value for the returned device is over 75%. However, for the validation, despite the high accuracy value of the first device, the four first devices and their accuracy are returned to study the matching process for each configuration. That means the *Coverage* of the recommender is 0.024 (4/170), because four devices are recommended out of 170 available devices. With this low *Coverage* value, it will be difficult for the *Precision@n* and *Recall@n* to have high values.

New sentences and sentences from the dataset are used to validate the recommender for new and known environments. Furthermore, sentences adapted to the dataset and simple sentences are used to validate the models in every scenario. While simple sentences measure the model's performance for entirely new situations, sentences adapted to the dataset measure the model's performance in situations similar to those used to train the model.

All these sentences were used to validate the proposal for different scenarios. For instance, in a Smart Home, a user can send complex sentences to the system completely different from those used in the training model. Furthermore, in an industrial environment, a machine can send requests to the system following a common schema used by the system to create the model, but requesting different information to that known by the system, for instance, requesting a new type of device that was included in the manufacturing system.

Tables 6 and 7 show some of the sentences used for the validation in the hashing model and in the Tokenization model, respectively, considering the dataset, the matching result, the accuracy of the recommender, the *Precision@n* and *Recall@n* values; and whether the sentence is new for the model. The same sentences are used for the hashing and the tokenization models.

For the hashing model, sentences from the dataset are matched with the correct device at an accuracy rate close to 100%. Furthermore, for new sentences, if the sentence includes the device name used to train the model, the model can match the sentence to the correct device with an accuracy close to 100%. However, if the device name differs from that used to train the model, or if a complex sentence is used, the model cannot always match the correct device in the top-4 returned list. For instance, the system can't return the correct device in abstract instructions where the desired device is not defined, *e.g.*, *Did you sense any movement?*. In addition, multiple devices perform the same actions, even when they can be located in different

Table 6: Sentences used for validation in the hashing model.

Sentence	First device	Second device	Third device	Fourth device	Precisior	Recall.	New
lightcontroler in bedroomchildren	/agent1/lightcontrol1 /lightOn(44.09%)	/agent1 /tempin1(11.75%)	/agent12/doorlock3 /open(10.24%)	/agent4/battery3 /charging(9.93%)	1/4(25%)	1/1(100%)	✓
Turn on the light in the bedroom	/agent13/movement13 /lastChange(26.69%)	/agent13/lightcontrol13 /lightOn(25.80%)	/agent13 /tempin13(11.99%)	/agent13/movement13 /movement(10.30%)	1/4(25%)	1/1(100%)	✓
Did you sense any movement?	/agent2/tempin2(45.69%)	/agent1 /tempin1(19.68%)	/agent12/doorlock3 /open(18.94%)	/agent4 /tempin4(14.06%)	0/4(0%)	0/21(0%)	✓
Did the movement sensor read any movement?	/agent14/movement14 /movement(91.21%)	/agent4/movement4 /movement(4.73%)	/agent12/movement12 /movement(1.19%)	/agent1/movement1 /movement(1.03%)	4/4(100%)	4/21(19%)	✓
Stop the washing machine	/agent2 /tempin2(81.36%)	/agent1 /tempin1(3.97%)	/agent6 /tempin6(3.75%)	/agent6/movement6 /movement(3.23%)	0/4(0%)	0/4(0%)	✓
Stop the washingservice in the Showerroom	/agent14/washingmachine3 /washing(99.26%)	/agent14/movement14 /movement(0.24%)	/agent14 /tempin14(0.19%)	/agent14/lightcontrol14 /lightOn(0.07%)	1/4(25%)	1/1(100%)	✓
Close the entrance doorlock	/agent12/movement12 /movement(58.55%)	/agent12/movement12 /lastChange(17.49%)	/agent12/lightcontrol12 /lightOn(7.24%)	/agent12/doorlock3 /open(4.94%)	1/4(25%)	1/1(100%)	✓
Close the doorlockservice entrance	/agent12/doorlock3 /open(99.99%)	/agent4/tempin4(-)	/agent2/tempin2(-)	/agent1/tempin1(-)	1/4(25%)	1/1(100%)	✓
I need to write lightcontroler in bedroom	/agent13/lightcontrol13 /lightOn(99.14%)	/agent13/movement13 /lastChange(0.56%)	/agent13 /tempin13(0.07%)	/agent20/lightcontrol20 /lightOn(0.06%)	1/4(25%)	1/1(100%)	
I need to read movementSensor in BedroomParents	/agent2/movement2 /movement(99.99%)	/agent2/movement2 /lastChange(-)	/agent2/tempin2(-)	/agent6/movement6 /movement(-)	1/4(25%)	1/1(100%)	
I need to write washingservice in Showerroom	/agent14/washingmachine3 /washing(99.99%)	/agent14 /tempin14(0.01%)	/agent3/tempin3(-)	/agent14/lightcontrol14 /lightOn(-)	1/4(25%)	1/1(100%)	
I need to open doorlockservice in entrance	/agent12/doorlock3 /open(99.99%)	/agent10/doorlock2 /open(-)	/agent4/tempin4(-)	/agent5/tempin5(-)	1/4(25%)	1/1(100%)	

Table 7: Sentences used for validation in the Tokenization model.

Sentence	First device	Second device	Third device	Fourth device	Prec.	Recall.	New
lightcontroler in bedroomchildren	/agent4/battery3 /charge (55.73%)	/agent4/battery3 /charging (43.29%)	/agent6 /tempin6 (0.96%)	/agent2/lightcontrol2 /lightOn (0.01%)	0/4 (0%)	0/1 (0%)	✓
Turn on the light in the bedroom	/agent6/tempin6 (94.18%)	/agent13/lightcontrol13 /lightOn (3.45%)	/agent2/lightcontrol2 /lightOn (1.13%)	/agent4/battery3 /charging (0.43%)	1/4 (25%)	1/1 (100%)	✓
Did you sense any movement?	/agent6/tempin6 (43.65%)	/agent4/battery3 /charge (28.92%)	/agent4/battery3 /charging (25.06%)	/agent13/lightcontrol13 /lightOn (1.69%)	0/4 (0%)	0/21 (0%)	✓
Did the movementsensor read any movement?	/agent13/lightcontrol13 /lightOn (31.90%)	/agent6/tempin6 (30.16%)	/agent12/movement12 /movement (15.44%)	/agent4/battery3 /charge (9.72%)	1/4 (25%)	1/21 (4.76%)	✓
Stop the washing machine	/agent4/battery3 /charge(40.73%)	/agent4/battery3 /charging (35.13%)	/agent6/tempin6 (23.87%)	/agent13/lightcontrol13 /lightOn (0.17%)	0/4 (0%)	0/4 (0%)	✓
Stop the washingservice in the Showerroom	/agent6/tempin6 (69.47%)	/agent13/lightcontrol13 /lightOn (16.64%)	/agent4/battery3 /charging (4.39%)	/agent4/battery3 /charge (4.29%)	0/4 (0%)	0/1 (0%)	✓
Close the entrance doorlock	/agent4/battery3 /charge(55.34%)	agent4/battery3 /charging (43.71%)	/agent6/tempin6 (0.94%)	/agent12/movement12 /movement(-)	0/4 (0%)	0/1 (0%)	✓
Close the doorlockservice entrance	/agent4/battery3 /charge (55.95%)	/agent4/battery3 /charging (43.47%)	/agent6/tempin6 (0.57%)	/agent12/movement12 /movement(-)	0/4 (0%)	0/1 (100%)	✓
I need to write lightcontroler in bedroom	/agent6/tempin6 (69.84%)	/agent4/battery3 /charge (11.79%)	/agent4/battery3 /charging (10.75%)	/agent13/lightcontrol13 /lightOn (5.07%)	1/4 (25%)	1/1 (100%)	
I need to read movementSensor in BedroomParents	/agent6/tempin6 (81.43%)	/agent13/lightcontrol13 /lightOn (10.86%)	/agent4/battery3 /charge (2.32%)	/agent4/battery3 /charging (2.21%)	0/4 (0%)	0/1 (0%)	
I need to write washingservice in Showerroom	/agent6/tempin6 (76.37%)	/agent13/lightcontrol13 /lightOn (17.16%)	/agent2/lightcontrol2 /lightOn (2.78%)	/agent2/movement2 /movement (2.18%)	0/4 (0%)	0/1 (0%)	
I need to open doorlockservice in entrance	/agent6/tempin6 (85.97%)	/agent4/battery3 /charge (5.31%)	/agent4/battery3 /charging (5.15%)	/agent13/lightcontrol13 /lightOn (1.85%)	0/4 (0%)	0/1 (0%)	

places around the house, which makes finding a suitable device even harder.

However, new sentences using the complete device name (although this is not the way we colloquially refer to the devices) are matched correctly with high accuracy; for instance, *Close the doorlockservice entrance*, is matched correctly with the entrance door at an accuracy rate of 99.9%. Conversely, new sentences with a different device name, are rarely matched correctly; for instance, *Close the entrance doorlock* returns the correct device in the fourth position with an accuracy of 4.9%.

The problem of matching sentences that have different device names can be achieved by pre-processing the sentence and simplifying the names, using names like *door lock* or *doorlock* instead of *doorlockservice*. Another solution is to increase the *Coverage*, *i.e.*, returning more devices in the recommended list. Currently, 170 possible devices are to be matched, and only 4 devices are returned. However, the accuracy value for these devices is very low, and returning many devices may not be helpful for a recommendation.

For the *Precision@n* and *Recall@n* measurement, most of the sentences have a *Precision@n* of 25% with a *Recall@n* of 100%, which means that when only one device is relevant to the user, that device is returned in the recommended list of devices. Returning the first device when the accuracy is higher than 75% can increase the *Precision@n* value in these scenarios. When just one device is available, it is directly returned, increasing the *Precision@n* to 100%. However, when requesting multiple devices, the recommender has a *Precision@n* of 100% but a low *Recall@n* value, because the *Coverage* of the recommender is low. To increase the *Recall@n* value in these scenarios, a possible solution could be to increase the *Coverage*; if increasing, it will cause the *Recall@n* to increase too. For instance, if returning four devices, the recommender has a *Recall@n* value of 25%. Still, if returning 10 devices, the recommender has a *Recall@n* value of 75%, the *Coverage* value will be increased to return ten devices for that request, increasing the *Recall@n* value from 25% to 75%.

Regarding the distribution of services shown in Figure 3, in most recommendations, the system returns, in the top positions, services related to the battery, the temperature, or the movement. For new sentences like *Did you sense any movement?* or *Stop the washing machine*, unknown by the model, it returns one of the most frequent services in the dataset. Furthermore, for services with a low frequency in the dataset compared to other services with the same name, the model cannot match them correctly. For instance, the sentence *I need to registerService lightcontroler in Bedroom-*

Children has to return the base service `/agent1/lightcontrol1` that appears only once in the dataset. However, the model, instead of returning the base service, returns the service related to turning the light on, `/agent1/lightcontrol1/lightOn`. The same applies to other services with a low frequency in the dataset, like `/agent21/doorlock5`.

Figure 7 represents the confusion matrix of the most frequent classes, which account for 98.74% of the recommendations. We have selected the most frequent classes because the visualization of the confusion matrix using the 170 classes would be very large and difficult to interpret. The color indicates the number of times the model recommends in the first position the service in the x-axis, being the darker the most frequent and the white the less frequent. As shown in the confusion matrix, when the model returns a service that is not the requested one, the service returned is from the same device or a similar one. In the context of a Smart Home, it would be returning other light from the same room. For instance, the model returns `/agent12/battery6/charge` instead of returning the services `/agent12/battery5/charge`, `/agent12/battery5/charging` or `/agent12/battery6/charging`. Another thing to remember is that the model is a recommender, thus it will return a list of recommendations to help when searching for devices and not a single device. Therefore, a list of similar devices would be useful in the result of the recommendation.

Using the tokenization model, despite it having a higher degree of accuracy, the results are worse than with the hashing model. For known sentences, the tokenization model cannot match the correct device in the top 1 position; in the best case, a relevant device is in the top 4 positions. Furthermore, the tokenization model can't correctly classify new sentences. This is due to an incorrect implementation of the tokenizer or because a model that works with complete sentences is better than models that work with each word of the sentence for this problem.

Finally, the model with better results is compared with the current search system. The search system is based on syntactic queries; the user builds a query to retrieve the available CPS devices. To compare both approaches, as the current system is unable to use natural language sentences, each sentence is divided into two words, the name of the CPS and the location of the device, *i.e., lightcontroler and bedroomchildren*; and these sentences are used in the validation of the current system and represented in Table 8.

Figure 8 shows the comparison between the *Precision@n* and *Recall@n* results of the hashing model and the current search system. The hashing

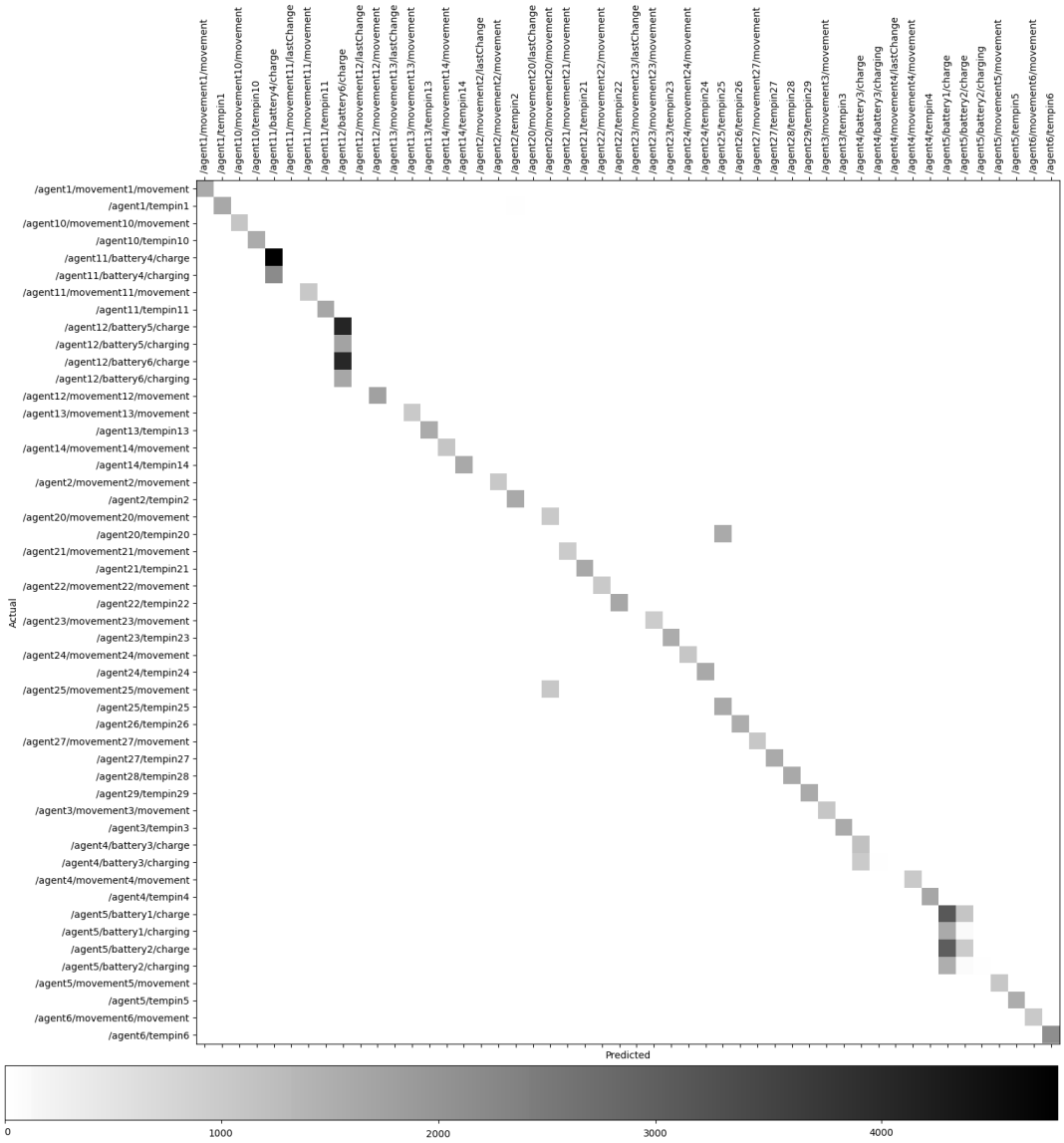


Figure 7: Confusion matrix of the most frequent devices.

model has better results than the current search system in two of the sentences used (S2 and S9) and worse results for one of the sentences (S3). For sentences 2 and 9 hashing model has better results because the name of the location (*bedroom*) is similar to the name of other locations (*bedroomchildren* and *bedroomparents*), causing the current search system to return devices in *bedroomchildren* and *bedroomparents* instead of *bedroom*. However, the hashing model has worse results in Sentence 3 because the model cannot understand the sentence. At the same time, the current search system can query the word *movement* to retrieve CPSs with a similar name.

Both systems have similar results, with the same *Precision@n* and *Recall@n* in most of the sentences. However, the current system is unable to process and understand natural language sentences. To get the results presented in this section, a system capable of dividing a sentence into the location and the device used has to be implemented, thus searching for devices with a location name similar to the requested location or with a device name similar to the requested device. Furthermore, the current system cannot learn synonyms for searching for devices. For instance, the proposed model can search for a light using words such as bulb, light, and ikea, while the current system can only search for devices with similar names to those stored in the database.

6. Discussion

The previous section presented a validation scenario to compare the recommender system based on Transformer with our current search system. For the validation, the system returned the top-4 devices instead of using a threshold to manage the number of returned devices. The top-4 devices were returned because, in the experimentation, returning the top-4 devices bearer in good results. The experimental scenario has 170 possible devices. Therefore, returning a fixed number of devices avoids overloading the user with too much information. For instance, using a threshold to return devices whose combination is 80% or higher can bear in returning lots of devices with a low percentage value (returning 100 devices with the value of 0.08%).

6.1. Limitations

The results of the validation scenario showed that both approaches have similar results for recommending CPSs using natural language sentences. However, our current search system has some limitations. For our current

Table 8: Sentences used for validation in our current system.

Sentence	First device	Second device	Third device	Fourth device	Prec.	Recall.	New
lightcontroler in bedroomchildren	/agent1/lightcontrol1	/agent1/lightcontrol1 /lightOn	-	-	1/4 (25%)	1/1 (100%)	✓
Turn on the light in the bedroom	/agent2/lightcontrol2	/agent1/lightcontrol1	/agent1/lightcontrol1 /lightOn	/agent2/lightcontrol2 /lightOn	0/4 (0%)	0/1 (0%)	✓
Did you sense any movement?	/agent4/movement4	/agent2/movement2	/agent1/movement1	/agent3/movement3	4/4 (100%)	4/21 (19%)	✓
Did the movement sensor read any movement?	/agent4/movement4	/agent2/movement2	/agent1/movement1	/agent3/movement3	4/4 (100%)	4/21 (19%)	✓
Stop the washing machine	-	-	-	-	0/4 (0%)	0/4 (0%)	✓
Stop the washingservice in the Showerroom	/agent14/ washingmachine3/washing	/agent14/washingmachine3		-	1/4 (25%)	1/1 (100%)	✓
Close the entrance doorlock	/agent12/doorlock3/open	/agent12/doorlock3	-	-	1/4 (25%)	1/1 (100%)	✓
Close the doorlockservice entrance	/agent12/doorlock3/open	/agent12/doorlock3	-	-	1/4 (25%)	1/1 (100%)	✓
I need to write lightcontroler in bedroom	/agent2/lightcontrol2	/agent1/lightcontrol1	/agent1/lightcontrol1 /lightOn	/agent2/lightcontrol2 /lightOn	0/4 (0%)	0/1 (0%)	
I need to read movementSensor in BedroomParents	/agent2/movement2	/agent2/movement2 /movement	/agent2/movement2 /lastChange	-	1/4 (25%)	1/1 (100%)	
I need to write washingservice in Showerroom	/agent14/ washingmachine3/washing	/agent14/ washingmachine3	-	-	1/4 (25%)	1/1 (100%)	
I need to open doorlockservice in entrance	/agent12/doorlock3/open	/agent12/doorlock3	-	-	1/4 (25%)	1/1 (100%)	

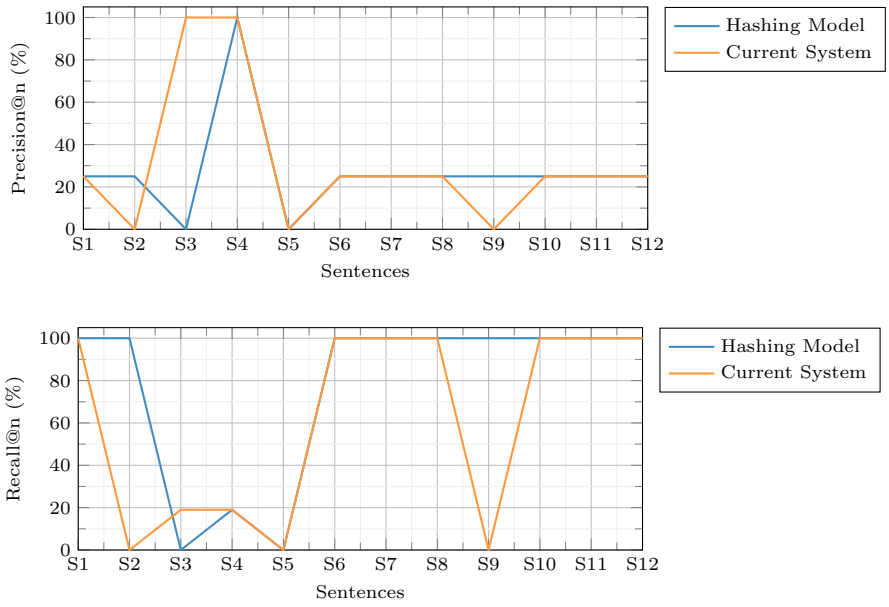


Figure 8: Comparison between Precision@n and Recall@n for the validation.

search system to work as presented in the validation scenario, the system has to get the word that defines the CPS and the word that defines the location where the CPSs are deployed. Developing a system that extracts that information from predefined sentences is possible, but the research is about natural language sentences; each sentence can be structured differently, increasing the complexity of the development. Furthermore, the current search system cannot search CPSs using synonyms, while the Transformer-based recommender system can learn the synonyms to retrieve the correct CPS. Another difference between the current search system and the proposed approach is that the Transformer-based recommender system can be trained to work with different languages. In contrast, the current search system can only work with the language of the device information. For instance, if the device's information is stored in English, the current search system can only search using sentences written in English.

From our point of view, with the results of the validation scenario, the proposed approach improves our current search system. The Transformer-based recommender system is more dynamic than the current search system; it can understand sentences with different structures and in different languages. However, the proposed approach lacks a generalization feature; it

has to be trained with sentences related to the new CPSs added to the system. Therefore, new CPSs cannot be recommended when they are deployed. The recommender system can support the current search system to solve this problem. When a new device is added, the current search system is used, storing the natural language sentences used by the user to train the recommender system. After the recommender system has enough natural language sentences, it is trained to recommend the new CPS. Furthermore, as shown in the paper, the proposed approach can also generate sentences to train the recommender system until it gets enough natural language sentences used by the user. With both approaches, we can ensure that the system can adapt to dynamic environments where devices are frequently added.

One limitation regarding the generation of sentences is that the sentences have been generated using the description of IoT devices or services in the WoT. This is a good way to save time and resources. It also facilitates the deployment of new facilities but can also introduce bias into the system. This bias can be mitigated by learning how users interact with the devices and services. The system can learn from the user's feedback and adjust its recommendations accordingly in future deployments.

Another limitation is that the recommender system may suffer from Contextual Bias, and it should be considered if the readers want to reproduce the model. Our model might inadvertently learn biases present in the text or context it is trained on. For example, our model is trained based on a dataset where the use of temperature devices is very frequent, but others, such as light bulbs, are not frequently used, so it may develop a skewed understanding of certain kinds of devices. When reproducing the model with another dataset, the practitioners should be mindful of their data and context to minimize this bias.

Finally, to create the recommender, we use the Transformer architecture and the attention mechanisms. It has a complex and highly parallel structure. The lack of interpretability in the Transformer architecture is a limitation because of understanding and explaining the reasoning behind the model predictions (or, in this case, recommendations). It frequently happens when using Deep Learning algorithms. There are some recent visualization techniques based on the attention weights in the transformer, but they are still under development, and it remains an ongoing research challenge

6.2. Threats to validity

The proposed validation scenario suffers from a series of **threats to validity**, such as:

(a) *Real scenario.* The proposed recommender system is validated using a simulated dataset from Kaggle. A set of new and known natural language sentences are used to evaluate the performance of the proposed approach using *Precision@n*. However, the validation lacks a deployment in a real scenario. Deploying the recommender system in a real scenario can help validate the proposed approach better; the use of simulated data differs from the use of subjects participating in the experiment. In addition, the proposed recommender must be validated by including additional datasets different from the one selected to prevent the recommender from being effective only in a single scenario.

(b) *Reliable dataset.* A reliable dataset is required to train the model. Finding or creating valuable datasets is a difficult task. For this paper, a dataset from Kaggle was adapted to fit the problem presented, linking natural language sentences with devices deployed in a Smart Home. As an initial scenario, this dataset can be used for evaluation and validation to answer the proposed research questions. Still, future research must create a dataset with real-world information to ensure a reliable dataset that can be used for real-world problems.

(c) *Comparison with other techniques.* To validate research, it must be compared with similar research in the same environment using the same features. Comparing the proposal with other techniques helps to identify the improvements and limitations of the proposed model. In the validation scenario, the proposed recommender system is compared with the search system used currently by our research group. This comparison helps to identify the improvements of the proposed recommender system over the current search system and the proposal's limitations over a traditional search system. However, as related work about recommenders in IoT scenarios, the proposed recommender system must be compared with current research about recommender systems in IoT environments. The reason for not comparing the proposed recommender system with these works is that there is not enough information in the papers to replicate the research using the same scenario and features of our validation scenario.

To mitigate threats (a) and (b), validating the recommender system in a real scenario would be necessary, creating a dataset using natural language sentences used by the subjects participating in the scenario. A possible Smart scenario to validate the recommender system would be garbage collection, where garbage trucks and containers must be located and monitored.

To mitigate threat (c), it would be necessary to replicate the experiment, using other algorithms to train the model, such as TF-IDF, and comparing the results with the results obtained using Transformer.

7. Conclusion and future work

This paper addressed the problem of using deep learning to match user queries in natural language with Web of Things devices or services. Following the steps and processes, we created a recommender system using the Transformer. We used a dataset of devices deployed in a Smart Home with features representing the services of these devices. With this dataset, we could study how our recommender would work with auto-generated descriptions by using the Thing Description despite not having sufficient ones to create a deep learning model. The dataset was pre-processed using feature engineering techniques to add and remove features, encoding techniques to transform categorical data into numerical data, and normalization to homogenize the data before creating the proposed model. After pre-processing the data, the model was created using an embedding and a Transformer layer. Finally, Softmax was used as an activation function to extract useful information from the result provided by the model. Creating the model resulted in a list of recommended devices sorted in descending order.

To validate the model, a scenario was established in which the selected models from the experimentation were used with a set of sentences introduced by the user. For the evaluation, three metrics were used: *Coverage*, *Precision@n*, and *Recall@n*. The hashing model delivered good results; if the relevant device was returned as the first solution, it was returned with a high grade of confidence (high accuracy). Furthermore, in most of the cases, it had the highest possible value of *Precision@n* and *Recall@n* (*e.g.*, 1/4 of *Precision@n* with only one relevant device available). However, the Tokenization model showed worse results in the validation than the experimentation results. This poor accuracy in the validation could have been due to a bad configuration in the model creation.

As such, we created a Transformer-based Recommender model to match user queries in natural language sentences with WoT devices or services. The proposed model is compared with the current search system, giving similar results in the *Precision@n* and *Recall@n* metrics when the traditional search system can extract information from natural language sentences to search for devices. However, with sufficient training, the proposed model can understand synonyms and different languages, giving better results than traditional search systems capable of extracting information from sentences. On top of this, the first research question addressed in Section 1 has been answered affirmatively. Regarding the second research question, the proposal can aid the decision-making process for different smart scenarios, affirmatively answering the second research question due to the validation results. Furthermore, using the auto-generated information extracted from the Thing Description, the system can recommend the correct device when users send a non-precise sentence that includes multiple devices, *e.g.*, *Did the movement sensor read any movement?*. However, the system can't correctly match abstract sentences sent by the user, *e.g.*, *Did you sense any movement?*, thus partially answering the third research question affirmatively. Finally, in the validation, returning a list of devices instead of providing a fixed query component resulted in both a higher *Precision@n* and a higher *Recall@n* value. Some scenarios had the relevant service returned in the second or fourth position of the list, and in other scenarios, there was more than one relevant device available. Therefore, the fourth research question is answered affirmatively.

In future work, the recommender system could be improved by applying the recommender system, along with the services, to other sections of the Thing Description, focusing on aiding decision-making. Another future work we would like to explore is comparing the proposed model using the Transformer with models using other algorithms for natural language problems, such as TF-IDF. Finally, we intend to develop a system that uses the proposed model for device recommendations in a real Smart scenario of the Web of Things, for instance, in Industry 4.0, for a garbage collection scenario.

Additionally, as the number and complexity of IoT devices and services continue to increase, users may need to use more complex and detailed natural language queries to find the devices and services they need. To provide better recommendations, the system could leverage contextual information to understand better the intent behind a user's query, such as geolocation or timestamp information, making it easier for users to identify the devices they need in complex environments.

Acknowledgments

Anonymous

References

[1] V. Sima, I. G. Gheorghe, J. Subić, D. Nancu, Influences of the Industry 4.0 Revolution on the Human Capital Development and Consumer Behavior: A Systematic Review, Sustainability 12 (10) (2020) 4035. doi:10.3390/su12104035.

[2] M. Hartmann, B. Halecker, Management of Innovation in the Industrial Internet of Things, in: Conference Proceedings, The International Society for Professional Innovation Management (ISPIM), 2015, pp. 1–17.

[3] G. Fortino, C. Savaglio, C. E. Palau, J. S. d. Puga, M. Ganzha, M. Paprzycki, M. Montesinos, A. Liotta, M. Llop, Towards multi-layer interoperability of heterogeneous IoT platforms: The INTER-IoT approach, in: Integration, interconnection, and interoperability of IoT systems, Springer, Cham, 2018, pp. 199–232. doi:10.1007/978-3-319-61300-0_10.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention Is All You Need, in: Proc. of the NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, p. 6000–6010. arXiv:1706.03762.

[5] S. Chaudhari, V. Mithal, G. Polatkan, R. Ramanath, An Attentive Survey of Attention Models, ACM Trans. Intell. Syst. Technol. 12 (5) (oct 2021). doi:10.1145/3465055.

[6] H. K. Azad, A. Deepak, Query expansion techniques for information retrieval: A survey, Information Processing & Management 56 (5) (2019) 1698–1735. doi:10.1016/j.ipm.2019.05.009.

[7] C. Carpineto, G. Romano, A Survey of Automatic Query Expansion in Information Retrieval, ACM Computing Surveys 44 (1) (jan 2012). doi:10.1145/2071389.2071390.

[8] D. Guinard, V. Trifa, E. Wilde, A resource oriented architecture for the Web of Things, 2010, pp. 1–8. doi:10.1109/IOT.2010.5678452.

[9] V. Charpenay, S. Käbisch, H. Kosch, Introducing Thing Descriptions and Interactions: An Ontology for the Web of Things, in: Proceedings of the 1st Semantic Web Technologies for the Internet of Things Workshop, Vol. 1783, 2016, pp. 55–66.
URL http://ceur-ws.org/Vol-1783/\#paper-06

[10] V. Charpenay, S. Käbisch, On Modeling the Physical World as a Collection of Things: The W3C Thing Description Ontology, in: The Semantic Web, Vol. 12123, Springer International Publishing, 2020, pp. 599–615. doi:10.1007/978-3-030-49461-2_35.

[11] B. Pourghebleh, V. Hayyolalam, A. Aghaei Anvigh, Service discovery in the Internet of Things: review of current trends and research challenges, Wireless Networks 26 (7) (2020) 5371–5391. doi:10.1007/s11276-020-02405-0.

[12] I. Lopez-Arevalo, E. Aldana-Bobadilla, A. Molina-Villegas, H. Galeana-Zapién, V. Muñiz-Sanchez, S. Gausin-Valle, A Memory-Efficient Encoding Method for Processing Mixed-Type Data on Machine Learning, Entropy 22 (12) (2020). doi:10.3390/e22121391.

[13] J. Weston, F. Ratle, R. Collobert, Deep Learning via Semi-Supervised Embedding, in: Proceedings of the 25th International Conference on Machine Learning, ICML'08, Association for Computing Machinery, New York, NY, USA, 2008, p. 1168–1175. doi:10.1145/1390156.1390303.

[14] M. Meissa, S. Benharzallah, O. Kazar, A Personalized Recommendation for Web API Discovery in Social Web of Things, The International Arab Journal of Information Technology 18 (2021) 438–445. doi:10.34028/iajit/18/3A/7.

[15] L. Sciullo, L. Gigli, A. Trotta, M. D. Felice, WoT Store: Managing resources and applications on the web of things, Internet of Things 9 (2020) 100164. doi:10.1016/J.IOT.2020.100164.

[16] G. Bovet, J. Hennebert, Distributed semantic discovery for web-of-things enabled smart buildings, in: 2014 6th International Conference on New Technologies, Mobility and Security (NTMS), IEEE Computer Society, 2014, pp. 1–5. doi:10.1109/NTMS.2014.6814015.

[17] T. Iggena, E. B. Ilyas, M. Fischer, *et al.*, IoTCrawler: Challenges and Solutions for Searching the Internet of Things, Sensors 21 (5) (2021) 1559. doi:10.3390/S21051559.

[18] W. G. Hatcher, C. Qian, W. Gao, F. Liang, K. Hua, W. Yu, Towards Efficient and Intelligent Internet of Things Search Engine, IEEE Access 9 (2021) 15778–15795. doi:10.1109/ACCESS.2021.3052759.

[19] W. T. Lunardi, E. D. Matos, R. Tiburski, L. A. Amaral, S. Marczak, F. Hessel, Context-based search engine for industrial IoT: Discovery, search, selection, and usage of devices, IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2015-October (2015) 1–8. doi:10.1109/ETFA.2015.7301477.

[20] F. Zhao, Z. Sun, H. Jin, Topic-centric and semantic-aware retrieval system for Internet of Things, Information Fusion 23 (2015) 33–42. doi:10.1016/j.inffus.2014.01.001.

[21] M. Younan, E. H. Houssein, M. Elhoseny, A. A. Ali, Challenges and recommended technologies for the industrial Internet of Things: A comprehensive review, Measurement 151 (2020) 1–16. doi:10.1016/j.measurement.2019.107198.

[22] D. Cai, W. Lam, Graph transformer for graph-to-sequence learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, 2020, pp. 7464–7471. doi:10.1609/aaai.v34i05.6243.

[23] C.-F. Yeh, J. Mahadeokar, K. Kalgaonkar, Y. Wang, D. Le, M. Jain, K. Schubert, C. Fuegen, M. L. Seltzer, Transformer-Transducer: End-to-End Speech Recognition with Self-Attention (2019). arXiv:1910.12977, doi:10.48550/arXiv.1910.12977.

[24] K. Miyazaki, T. Komatsu, T. Hayashi, S. Watanabe, T. Toda, K. Takeda, Weakly-Supervised Sound Event Detection with Self-Attention, in: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Institute of Electrical and Electronics Engineers Inc., 2020, pp. 66–70. doi:10.1109/ICASSP40776.2020.9053609.

[25] J. Ángel González, L. F. Hurtado, F. Pla, Self-attention for Twitter sentiment analysis in Spanish, Journal of Intelligent & Fuzzy Systems 39 (2020) 2165–2175. doi:10.3233/JIFS-179881.

[26] W. Zhang, D. Yang, Y. Xu, X. Huang, J. Zhang, M. Gidlund, DeepHealth: A Self-Attention Based Method for Instant Intelligent Predictive Maintenance in Industrial Internet of Things, IEEE Transactions on Industrial Informatics 17 (2021) 5461–5473. doi:10.1109/TII.2020.3029551.

[27] A. J. Fernández-García, L. Iribarne, A. Corral, J. Criado, J. Z. Wang, A recommender system for component-based applications using machine learning techniques, Knowledge-Based Systems 164 (2019) 68–84. doi:10.1016/j.knosys.2018.10.019.

[28] A. J. Fernández-García, R. Rodríguez-Echeverría, J. C. Preciado, J. M. C. Manzano, F. Sánchez-Figueroa, Creating a Recommender System to Support Higher Education Students in the Subject Enrollment Decision, IEEE Access 8 (2020) 189069–189088. doi:10.1109/ACCESS.2020.3031572.

[29] I. Mashal, O. Alsaryrah, T.-Y. Chung, Performance evaluation of recommendation algorithms on Internet of Things services, Physica A: Statistical Mechanics and its Applications 451 (2016) 646–656. doi:10.1016/j.physa.2016.01.051.

[30] M. A. Torad, B. Bouallegue, A. M. Ahmed, A voice controlled smart home automation system using artificial intelligent and Internet of Things, TELKOMNIKA Telecommunication, Computing, Electronics and Control 20 (4) (2022). doi:10.12928/telkomnika.v20i4.23763.

[31] F. Corno, L. De Russis, A. Monge Roffarello, RecRules: Recommending IF-THEN Rules for End-User Development, ACM Trans. Intell. Syst. Technol. 10 (5) (sep 2019). doi:10.1145/3344211.

[32] J. A. Llopis, A. J. Fernández-García, J. Criado, L. Iribarne, Matching user queries in natural language with Cyber-Physical Systems using deep learning through a Transformer approach, in: 2022 International Conference on INnovations in Intelligent SysTems and Applications (INISTA), 2022, pp. 1–6. doi:10.1109/INISTA55318.2022.9894230.

[33] H. El-Amir, M. Hamdy, Feature selection and feature engineering, in: Deep Learning Pipeline: Building a Deep Learning Model with TensorFlow, Apress, Berkeley, CA, 2020, pp. 233–276. doi:10.1007/978-1-4842-5349-6_8.

[34] J. T. Hancock, T. M. Khoshgoftaar, Survey on categorical data for neural networks, Journal of Big Data 7 (28) (2020) 1–41. doi:10.1186/s40537-020-00305-w.

[35] K. Potdar, T. S. Pardawala, C. D. Pai, A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers, International Journal of Computer Applications 175 (2017) 7–9. doi:10.5120/ijca2017915495.

[36] N. T. Nam, P. D. Hung, Padding Methods in Convolutional Sequence Model: An Application in Japanese Handwriting Recognition, in: Proceedings of the 3rd International Conference on Machine Learning and Soft Computing, ICMLSC 2019, Association for Computing Machinery, New York, NY, USA, 2019, p. 138–142. doi:10.1145/3310986.3310998.

[37] M. Junker, R. Hoch, A. Dengel, On the evaluation of document analysis components by recall, precision, and accuracy, in: Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR '99, 1999, pp. 713–716. doi:10.1109/ICDAR.1999.791887.

[38] E. Mena-Maldonado, R. Cañamares, P. Castells, Y. Ren, M. Sanderson, Agreement and Disagreement between True and False-Positive Metrics in Recommender Systems Evaluation, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Association for Computing Machinery, New York, NY, USA, 2020, p. 841–850. doi:10.1145/3397271.3401096.

[39] A. Gunawardana, G. Shani, A Survey of Accuracy Evaluation Metrics of Recommendation Tasks, J. Mach. Learn. Res. 10 (2009) 2935–2962.

[40] G. Shani, A. Gunawardana, Evaluating recommendation systems, in: F. Ricci, L. Rokach, B. Shapira, P. B. Kantor (Eds.), Recommender Systems Handbook, Springer US, Boston, MA, 2011, pp. 257–297. doi:10.1007/978-0-387-85820-3_8.