

Utilizing Large Language Models for Automated Log-Based Thing Description Generation¹

Max THOMA^{a,2}, Leonhard ESTERBAUER^a Thomas PREINDL^a and
Gernot STEINDL^b

^aAutomation Systems, TU Wien, Vienna, Austria

^bUniversity of Applied Sciences Burgenland, Eisenstadt, Austria

ORCID ID: Max Thoma <https://orcid.org/0009-0000-0142-6394>, Leonhard Esterbauer
<https://orcid.org/0009-0007-6459-4185>, Thomas Preindl
<https://orcid.org/0000-0001-7268-5393>, Gernot Steindl
<https://orcid.org/0000-0002-9035-9206>

Abstract. Purpose: Semantic device descriptions, such as the [Web of Things \(WoT\)](#) [Thing Description \(TD\)](#), are a great tool to improve the level of interoperability in [Internet of Things \(IoT\)](#) systems. However, a majority of new and existing [IoT](#) devices do not ship with a [TD](#), and developers often need to create them by hand. This makes it tedious for anyone who wishes to integrate, migrate, or modernize devices of their existing infrastructure into a [WoT](#) ecosystem. Therefore, an automated approach for [TD](#) generation that facilitates this process is needed. Methodology: We propose a [Large Language Model \(LLM\)](#)-based approach to automate [TD](#) generation. By utilizing message logs and conformance checks, we introduce an iterative process that leverages [LLM](#) technologies to generate [TDs](#). The proposed methodology is evaluated in a case study of 76 [IoT](#) devices communicating over MQTT. Findings: Our results show that with the proposed methodology, an [LLM](#) can generate [TDs](#) from MQTT message logs with an average functional accuracy of up to 91%, and a descriptive accuracy of around 85%, demonstrating strong overall performance. Value: All generated [TDs](#) and the prototypical Python implementation of the methodology can be found in our repository. The proposed methodology helps the adoption of the [WoT](#) by offering an automated generation of [TDs](#) in environments where MQTT message logs are available.

Keywords. Web of Things, Thing Description, Large Language Models, Device Integration

1. Introduction

In recent years, the rapid adoption of [Internet of Things \(IoT\)](#) technology has led to a diverse landscape of communication protocols, causing significant interoperability issues

¹This project is supported with the funds from the Climate and Energy Fund and implemented in the framework Program “Smart Cities Demo – Boosting Urban Innovation 2020”. FFG Project Number: 884792.

²Corresponding Author: Max Thoma {firstname}.{lastname}@tuwien.ac.at

in various domains and processes [1,2,3,4]. Semantic descriptions are treated as a savior for identifying these issues while additionally enabling automatic integration processes of IoT devices [1,5,3].

However, the importance of semantic descriptions goes far beyond the identification of interoperability issues. Jara et al. even claim that a common semantic device description is vital to reach global interoperability in the IoT realm [6]. A promising candidate for closing this gap is the [Web of Things \(WoT\)](#) standard developed by the [World Wide Web Consortium \(W3C\)](#) [7].

The primary part of the [WoT](#) standard specifies a [JSON for Linking Data \(JSON-LD\)](#)-based structure that represents IoT devices and abstracts capabilities into three interaction affordances, namely *properties*, *actions*, and *events* [8]. Additionally, [Thing Descriptions \(TDs\)](#) can be enriched with semantic annotations, enabling humans and machines to put information in context. These features are well received in the IoT community and consequently, the standard gains momentum in industry [9] or even medical projects [10].

However, currently most of the IoT devices are not shipped with a TD, and it is unlikely that a majority of hardware manufacturers will provide TDs for their products in the near future. Brownfield and legacy systems specifically struggle with the adoption of semantic models [11] such as the [WoT TD](#). This significantly slows down digitalization processes and hinders technology penetration [12]. Necessarily, companies that are already utilizing the [WoT](#) standard have to create the TDs by hand. This task requires knowledge about the devices and protocols, while additionally understanding the syntax of the [WoT](#) standard. Consequently, crafting a TD by hand is a time-consuming and tedious task that could prevent companies from broadly adopting semantic descriptions in their setup. A web-based tool called the [Thing Description Generator and Validator \(TDGV\)](#) has been developed by Rahamneh et al. that addressed these issues by using templates, and schemas for validation [13]. However, TDGV still requires manual modeling steps and knowledge about the device.

AI technologies such as [Large Language Models \(LLMs\)](#) have been proven to be a good fit for generating syntax-based content in a variety of domains [14,15]. However, their application extends beyond code generation as they also fulfill Software Engineering tasks such as documentation, and testing [15]. Creating a TD requires many skills related to these kinds of tasks. Hence, we assumed that it should be feasible for an LLM to create a valid TD, given that it is properly instructed and provided with enough context information. To gather this context information, we decided to use message logs. Especially for simpler protocols such as MQTT, this way of supporting automatic TD generation is intriguing. However, to the best of our knowledge, it is unknown how to set up a log-based auto-generator for semantic descriptions that also validates the results. Therefore, we formulated the following research question: *How can message logs be used to generate accurate TDs using LLM technology?*

The main contribution of the paper is a novel process for generating semantic descriptions based on message logs. The process is exemplary implemented in a [Smart Local Energy Community \(SLEC\)](#) real-world setting using MQTT-based message logs and the LLM models *Cogito*, *Gemini*, and *GPT-4o*. For evaluation purposes, the auto-generated TDs are compared to 76 hand-crafted TDs of typical IoT devices with accuracy scores. The results show a high level of attribute recognition for [WoT affordances](#) from simple MQTT message logs.

The remainder of this paper is structured as follows: Chapter 2 provides an overview of related work for auto-generating code with LLMs. Thereafter, Chapter 3 proposes our novel methodology, and Chapter 4 showcases the methodology in a case study for MQTT devices. Following that, Chapter 5 describes the evaluation steps of the case study, and Chapter 6 describes the gathered results. Lastly, Chapter 7 discusses the outcomes before Chapter 8 summarizes the paper and outlines directions for future work.

2. Related Work

LLM technology has been used and researched in many fields, including software engineering [15] and systems engineering [16]. Notably, LLMs are used to generate artifacts which are often referred to as **Artificial Intelligence Generated Content (AIGC)**. In the software engineering sector, this is most prominently source code, but it can also be comments, unit tests, and many more [15]. A common critique and concern with LLMs is that their output is not reliable enough and may lead to problems. Especially, in the industrial automation domain, issues caused by erroneous LLM artifacts may lead to serious problems, as high availability and safety are essential requirements. Chen et al. developed an extensive framework for **Large Language Model-Systems Engineering (LLM-SE)** in [16]. The goal of their framework is to satisfy the high requirements of industrial applications through Systems Engineering approaches. The authors were able to identify various critical characteristics of AIGC in this context, ranging from reliability, certainty, stability, and many more. Moreover, the authors also provide tools and concepts to improve these characteristics. Most notable for our use case is the *Knowledge Engineering* for LLM-SE approach. The authors propose to provide structured data (such as knowledge graphs) to the LLM to improve the AIGC's quality. We followed a similar approach in our proposed methodology, where we embed the **WoT TD** specification as a JSON schema into the LLM prompt.

Another example from the industrial domain is LLM4PLC introduced by Fakihi et al. [17]. The authors propose a methodology using LLMs for reliable **Programmable Logic Controller (PLC)** code generation. They suggest combining the generative capabilities of LLM technology with the formal verification capabilities of model checking. An expert user interacts with the LLM in an iterative process to define and describe an automation scenario. The LLM then creates PLC code, which runs through syntax and semantic model checks. The result of these tests is fed back to the LLM, which tries to fix any errors. With this iterative methodology, the authors were able to improve the quality of the generated source code significantly. Moreover, using fine-tuned models further improved their results. Our proposed LLM pipeline was inspired by this paper from Fakihi et al. [17]. However, for our particular use case, it was a requirement that no user interaction was necessary. Thus, we do not follow the *expert-in-the-loop* approach. But, similar to our use case, Fakihi et al.'s PLC code generation falls into the category of so-called low-resource languages. This means that **WoT TDs** are far less common as a resource on the internet than, for example, *Python*. As an illustrative example, there are less than 100 projects on GitHub that are tagged with “web-of-things”³, whereas there are over 518,000 projects that are tagged with “Python”⁴ (as of May 2025). The poten-

³<https://github.com/topics/web-of-things>

⁴<https://github.com/topics/python>

tial and opportunities of LLM technology for this kind of task are also highlighted by Tarassow [18].

We are not the first to combine LLM technology with WoT TDs. In 2024, Kinder et al. [19] presented an approach for task planning utilizing LLMs and WoT TDs. The running example of task planning in their paper is preparing a cup of coffee. Compared to our approach, the LLM does not generate TDs, but consumes them. They begin by instructing an LLM to generate Python classes from TDs. These LLM generated classes, together with a task description, are then used to prompt the LLM again. The final LLM artifact is the *plan* in the form of a Python script. Their proposed methodology outperformed other state-of-the-art LLM task planners by up to 12%. Moreover, in 2025, Marcinkowski et al. have proposed to use LLMs to translate existing structured device descriptions to WoT TDs [20]. In their evaluation, they used preexisting descriptions for a thermostat and an RGB light bulb encoded as a JavaScript Object Notation (JSON) object. Compared to our approach, the TDs were generated in a single shot, and only the functional correctness of the artefacts was evaluated by the authors. To summarize, LLMs have been successfully applied to industrial and software engineering applications. These promising results lead us to the following proposed methodology, described in the next section.

3. Proposed Methodology

Instead of relying on an (existing) IoT device to provide WoT TDs, we propose to use LLM technologies to generate TDs for IoT devices from their message logs. The basic idea behind this approach is that many real-world device integration scenarios revolve around the integration of information producers such as sensors into IoT platforms. The capabilities of these devices are often fully contained in a message log. Collecting these messages is easier than creating TDs from scratch. Thus, the idea is to utilize a capable and well-instructed LLM that is able to understand the context of the message logs and reliably create TDs. Since TDs are well defined and can be described schematically, this task becomes feasible for an LLM. Moreover, it also opens the opportunity to analyze the LLM's artifacts statically and report errors back to the LLM in an iterative cycle.

Figure 1 illustrates the five steps of our proposed methodology. First ❶, a *Message Logger* samples messages of an IoT device that is connected to a *Message Broker*. Second ❷, the LLM generates text output based on a prompt that consists of three parts: the *Message Logs* (A), a *Task Instruction* (B), and the *WoT TD Schema* (C). In the third step ❸, a check ensures that the LLM output conforms to the *WoT TD Specification*. If the *Conformance Check* failed ❹, an error report is sent back to the LLM, which is prompted to retry. Finally ❺, when the *Conformance Check* succeeds, the TD is returned in step five.

The generation of the TD does not happen in a *single-shot*. Rather, the artifact is generated *peu à peu*. Figure 2, depicts how the generation process from step ❷ was implemented. After each step where the LLM is used, the *Conformance Check* from step ❸ is performed. Simplified, a device's capability is called an *affordance* in the WoT TD, and can be one of three things: a *property*, an *action*, or an *event*. Each TD *affordance* is generated by the LLM in a separate step. The LLM has access to a TD schema, the message logs, as well as a list of possible enumeration values. This list of possible enumeration values is generated automatically from the message log. It is the *set* of all sent

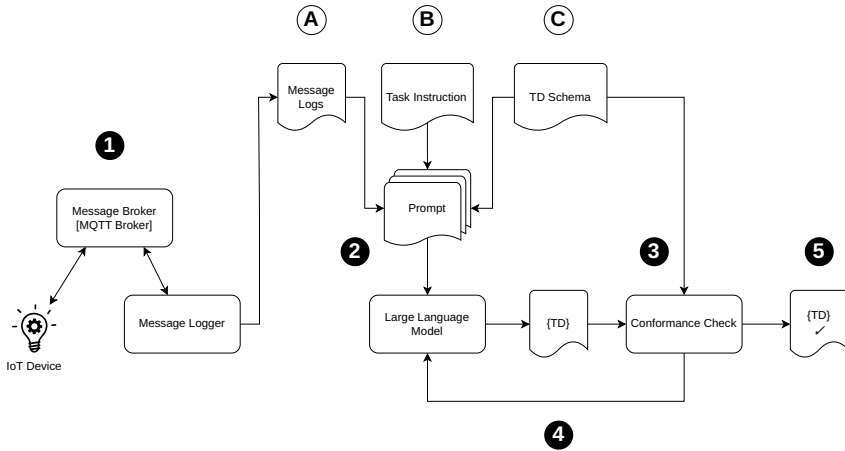


Figure 1. Graphical representation of proposed methodology

payloads from the *affordance* under consideration. In case the **LLM** classifies an *affordance* to be of *object* type, an additional classification step is performed to determine the sub-properties. Once all affordances are generated by the **LLM**, they are *assembled* to a coherent *TD candidate*. Finally, the **LLM** is instructed to add a *title* and *description* to the *TD*, finishing the *TD candidate* to a proper *TD*.

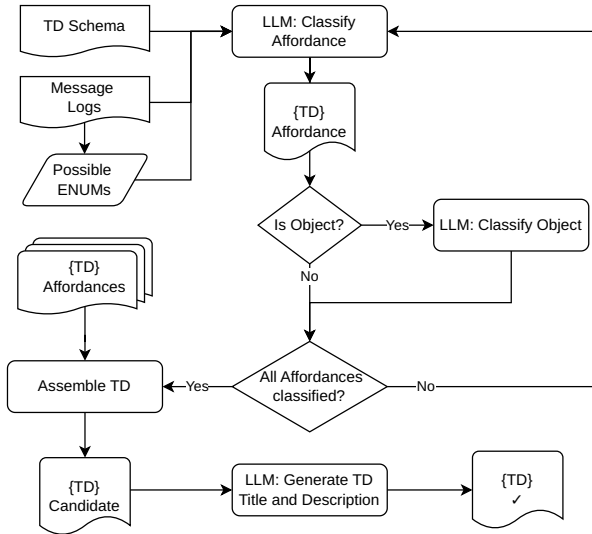


Figure 2. Flow Chart of the **TD** generation procedure

The prompt for *classifying the affordances* of an RGB light bulb can be seen in Listing 1. It consists of three parts. The first part is a context dump. The **LLM** gets the whole message log of the light bulb. The idea behind this is to give the **LLM** the full picture of the device that it is supposed to describe. The second part highlights the task's focus, namely, extracting a concrete affordance. In our example, this would be the *status*

```

1  Given is the following MQTT message log of an IoT device:
2  {device message log:
3  topic: application/bulb/status; payload: On; retain: true
4  topic: application/bulb/status; payload: Off; retain: true
5  topic: application/bulb/status; payload: Failed; retain: true
6  topic: application/bulb/power; payload: true; retain: false
7  topic: application/bulb/power; payload: false; retain: false}
8
9  For this the following part of the message log:
10 {affordance message log:
11 topic: application/bulb/status; payload: On; retain: true
12 topic: application/bulb/status; payload: Off; retain: true
13 topic: application/bulb/status; payload: Failed; retain: true}
14
15 Determine what kind of affordance this is!

```

Listing 1: Prompt used for affordance classification

```

1  class ClassificationAffordance(BaseModel):
2      title: str
3      description: str
4      name: str
5      # property, event or action
6      affordance_type: AffordanceType
7      # string, boolean, integer, object, null, number
8      type: AttributeType
9      is_enum: bool

```

Listing 2: Pydantic Class that is used for the classification of the affordances

property of the light bulb. The *affordance message log* is all messages that were sent by that specific affordance. In MQTT, this device message log would be all messages from all topics that the device publishes and is subscribed to, whereas the affordance message log would be all messages from a specific topic. We assumed the MQTT topic structure to be device-centric and non-overlapping. Hence, it is trivial to automatically filter the message log per device and per affordance. Finally, the third part gives a specific instruction. In addition to the prompt in Listing 1, the LLM also receives a so-called *system* prompt, which entails the schema of the *Classification* class that is shown in Listing 2. All answers from the LLM have to adhere to this class. This ensures that only affordances are generated that conform to the WoT TD specification. For brevity, the class attributes in Listing 2 have their description removed. This description gives the LLM additional information to understand its task. When assembling all affordances to a complete TD, the LLM is given a schema that encompasses all relevant parts of the WoT TD. The goal

of the described methodology is to reach a high level of automation. Each step can be done autonomously without a developer's intervention. This can be useful in many scenarios, such as the following discovery scenario inspired by McCool's **WoT** discovery use case description [21], which is depicted in Figure 3. With the proposed approach, the discovery of a new **IoT Device** can be fully automated. First, the **LLM Pipeline** samples the device and generates the corresponding **TD**. Second, the resulting **TD** is published at a central device repository such as a **WoT TD Directory**. Finally, third parties such as a **Cloud Provider** or simply a **User** are able to semantically understand the **IoT Device** and integrate it into any application of choice. This could be useful for integrating brownfield or legacy systems into modern infrastructure, as creating a message log is less complex than writing **TDs** by hand.

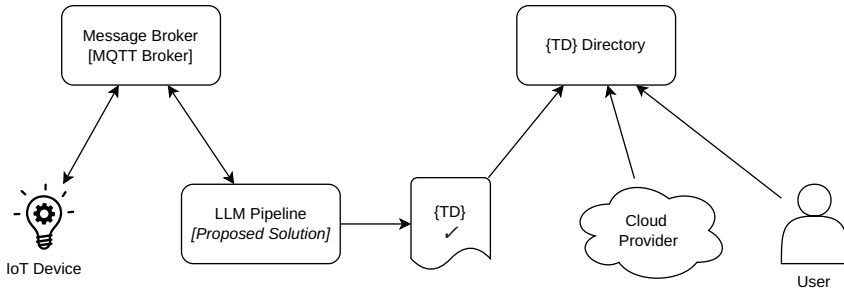


Figure 3. Discovery use case example, inspired by [21]

4. Case Study: Thing Description for MQTT Devices

The proposed methodology was evaluated using a case study of devices that communicate via MQTT in a **Local Energy Community (LEC)**. Many different definitions for **LECs** exist in the literature [22]. Thus, for clarity, we use the following definitions: **LECs** deploy **Renewable Energy Source (RES)** and share them among members. This is different from **SLECs** where **Information and Communication Technology (ICT)** is used in order to achieve more complex goals such as grid friendliness or higher levels of energy autarky. In general, **IoT** systems experience a high level of device and technology heterogeneity [23]. This is also true, and arguably even worse, for **SLECs**, as devices from diverse domains are expected to work together. Our goal is to automatically and semantically describe devices with **WoT TDs**. This will improve semantic interoperability and help with the device integration process into **IoT** platforms.

The case study focuses on MQTT, because it is a popular **IoT** protocol based on the publisher-subscriber principle [24]. Creating message logs is as simple as either subscribing to the wildcard topic “#” or the device-specific topics. Messages are ephemeral unless the retain flag is set in the message header. Then, the latest message is stored by the MQTT Broker and is made available immediately to new subscribers. The **WoT TD** supports the MQTT protocol via an official binding template [25]. However, the usual three affordances *actions*, *events* and *properties* do not match naturally with the publish-subscribe principle of MQTT. There is no natural way in MQTT to “request” a *property*. Thus, the distinction between what should be considered an event and what a property

is up for debate. One may argue that the retain flag should be used as an indicator if an affordance should be modeled as an event or a property. However, no guidance is given from the binding template. It is allowed for properties to not be retained under the condition that they are not marked as “readproperty” [25]. Therefore, we argue that it is up to individual taste to make this modeling distinction.

5. Evaluation

There are three parts to our evaluation strategy: First, based on reference IoT devices, we generate mock MQTT message logs that capture all affordances of the IoT device. Second, we conduct the experiment. That is, for each message log, we generate five TDs using an LLM. The LLM has ten tries for each prompt to produce an artifact that is conformant with the WoT TD specification. Third, we calculate metrics that measure the performance of the generated artifacts.

Our proposed methodology from Section 3 was implemented using Python. The code, the results, and a full generation example can be found on our GitHub repository⁵. For the original dataset, refer to Zenodo⁶. The two key technologies for the pipeline are *Pydantic*⁷ and *Instructor*⁸. *Pydantic* was used to model the relevant subset of the WoT TD description as Python classes. *Instructor* was issued to generate structured output, i.e., TD JSON-LD files with the LLM. They work in tandem as the *Pydantic* Python Classes are used as input for *Instructor*. The *Instructor* library has two main tasks in our use case. First, it generates the prompt for the LLM using the *Pydantic* classes, the message logs, and a task instruction. This corresponds to step ② of our proposed methodology (see Figure 1). Second, it evaluates whether the LLM artifact is conforming to the defined classes (step ③). If any error is encountered, it is reported back to the LLM (step ④). For our implementation, we capped this iterative loop to ten iterations. Finally, the TD is saved (step ⑤) for later analysis.

5.1. Evaluation Dataset

Our evaluation data set consists of a total of 76 devices that can be categorized into 54 different device types that are related to the SLEC domain. For a full list, see our full dataset linked in the *Results* section. For 23 of the 76 devices, we manually composed TDs that describe their affordances. The inspiration and examples for these devices stem from multiple open-source projects. Notably, Home Assistant’s MQTT integrations⁹, EVCC’s documentation¹⁰, SmartDataModels¹¹, WoT specification [8], and the WoT MQTT binding template [25] being the main sources. Additionally, we adapted the TDs used by Kinder et al. in [19] to MQTT. For this, we converted all HTTP references to MQTT, added missing descriptions and titles, removed TDs that did not meet our use case, and made the MQTT topic unique per affordance. This gave us 27 additional TDs.

⁵<https://github.com/max-thoma/log-based-td-generation>

⁶<https://doi.org/10.5281/zenodo.15526223>

⁷<https://docs.pydantic.dev/latest/>

⁸<https://python.useinstructor.com>

⁹<https://www.home-assistant.io/integrations/#search/MQTT>

¹⁰<https://docs.evcc.io/docs/Home>

¹¹<https://smartdatamodels.org>

Finally, we ported 26 of the WebThings TDs from the W3C testing repository ¹². We performed the same changes as with the previous dataset, removed some redundant devices, deleted any reference to being virtual devices, and added enumeration values to some of the string-type affordances. The provenance distribution of TDs and affordances in our dataset can be seen in Table 1.

Table 1. The provenance distribution of the Dataset

Dataset	No. of Things	No. of Affordances	Percent of Total Affordances
Custom	23	104	31.04%
W3C	26	50	14.93%
Kinder et al.	27	181	54.03%
Combined	76	335	100.00%

5.2. Evaluation Metrics

There exist multiple possible strategies to evaluate LLM artifacts [15]. For our particular use case, we decided to assess the generated TDs from two perspectives. On the one hand, we want to make sure that the TDs are functionally correct, that is, all affordances are technically sound and point to the right resource (i.e., the MQTT topic). On the other hand, we are interested in the descriptive nature of the TDs being sound. By descriptive nature, we mean that the description attribute, the affordance title, and the TD type correspond to the actual IoT device from the sampled message log. Both perspectives, functional and descriptive, need different metrics to be evaluated.

Table 2. Functional Correctness Requirements

	Affordance Type	Data Type	Subproperties	Enum Values	Topic
Strong Correctness	✓	✓	✓	✓	✓
Weak Correctness	✗	✓	✓	✓	✓

For the functional correctness, we defined two classes: *strong correctness* and *weak correctness*. The requirements for *strong* and *weak* correctness can be seen in Table 2. A ✓, indicates that the correctness criteria require that the generated affordance must have the same property as in the base dataset. The only difference between strong and weak correctness is that strong correctness requires that the affordance type must be equal, too. The intuition between the two correctness criteria is that the notion of *properties*, *events* and *actions* does not map naturally. As we have mentioned before, it is up to the taste of an engineer to decide how to model a certain affordance in MQTT. For instance, setting a target temperature of an air conditioning unit could be modeled as a changeable property or as an action. In a similar fashion, properties and events may be indistinguishable. Henceforth, we decided to introduce both *strong* and *weak* correctness metrics.

¹²https://github.com/w3c/wot-testing/tree/037c8d686ad8c145dd4fe01c2123ad26eca0e185/data/input_2022/TD/WebThings/TDs

```

1  { "@context": "https://www.w3.org/2022/wot/td/v1.1",
2    "@type": "Thing",
3    "title": "...",
4    "description": "...",
5    "properties": {
6      "propertyName": {
7        "title": "...",
8        "description": "...",
9      }
10   }
11 }

```

Listing 3: The descriptive aspects of the TD

We explicitly do not check the reference to the MQTT Broker, missing or wrong minimum and maximum values, missing or wrong content types, or missing or wrong *op* codes. While they help in the semantic integration of the IoT device, they are not essential. If an affordance is completely missing, it is counted as failed. Similarly, if the LLM was unable to generate a proper TD after ten tries, it is counted as failed. There was no obvious way within the *Instructor* library to track how many retries on average the LLM needed. However, in our final test runs, it never occurred that the LLM surpassed the retry limit. The MQTT topics in the TDs are taken from the message logs. Thus, it cannot happen that the LLM generates an affordance with a non-existent topic. Although it might hallucinate in the titles and descriptions of the affordances.

The descriptive correctness cannot be evaluated using the *strong* or *weak* correctness metric. To target this problem, we first need to specify what exactly we want to evaluate. The short snippet from a TD in Listing 3 is used for reference. There, we can find two main descriptive aspects:

1. The general device information:
 - The type of the TD i.e., the “@type” of the JSON-LD file (line 3)
 - The title of the TD (line 4)
 - The description of the TD (line 5)
2. The descriptive attributes of the affordances:
 - The title of the affordance (line 8)
 - The description affordance (line 9)

We evaluated the descriptive accuracy manually on a global level. The descriptions from the LLM frequently had more detailed information than the manually created ones. For instance, sometimes the enumeration values of a string type affordance were mentioned in the description attribute. In such cases, we gave the score *identical* if all the information was correct. Otherwise, a lower score was issued. Moreover, additional information that was gathered or interpreted from the MQTT topic was also counted as *identical* given that the interpretation was correct in the context of the TD. Additional information, which might be correct, but was not specified in the original TD, was in-

terpreted as *minor* or *moderate* difference, depending on the gravity. For instance, measurement units that are highly likely to be correct were interpreted as *minor* differences, whereas unlikely measurement units were considered *moderate* differences. Another example is boolean *actions*, for instance, the boolean action *pause* implies that the value *true* pauses the devices, and the value *false* resumes the device. However, if this was not always explicitly stated in the original TDs, we scored *minor* to *moderate* differences depending on how likely this implied functionality was. Features that do not exist in the original TD, but were advertised in the LLM TD were issued a *major* difference score. Table 3 summarizes our manual scoring system and assigns accuracy percentages to the score.

Table 3. Descriptive accuracy score

Description	Score	Percent
Identical or Equivalent: the descriptions are identical or have equivalent meaning	5	100%
Minor differences: there are minor inaccuracies in the description	4	75%
Moderate differences: the descriptions have moderate inaccuracies or omissions of information	3	50%
Major differences: there are major misrepresentations or omissions in the descriptions	2	25%
Unusable or Incorrect: the descriptions are incorrect and may be misleading	1	0%

6. Results

The experiments were conducted on three LLM models: Google’s gemini-flash-2.0 (hereafter referred to as *Gemini*), OpenAI’s gpt-4o-2024-08-06 (*GPT-4o*), and the open-source model cogito:70b-v1-preview-llama-q4_K_M¹³ (*Cogito*). We used a temperature of $t = 0.4$ for GPT-4o and Cogito, and left the default value for Gemini. The temperature values for GPT-4o and Cogito were chosen by consolidating relevant literature such as [26] for baseline values and then adjusting them through manual experiments, where we optimized for functional correctness. Google’s Gemini and OpenAI’s GPT-4o are very popular and well-researched models. Cogito, on the other hand, is a newer open-source model. It was chosen as it outperformed the popular open-source model Meta’s Llama-3x in our testing. While Cogito can be run locally, GPT-4o and Gemini need to be run on OpenAI’s and Google’s servers, respectively. Our selected TD data set contains 335 different affordances. Each affordance can either be an *action*, *event*, or *property*.

Table 4 shows the distribution of the data types per affordance type. For each reference TD, the LLM generated five candidates. This gives us a total of 1675 affordances to analyze. The message logs used for the generation were automatically derived from the

¹³<https://www.deepcogito.com/research/cogito-v1-preview>

Table 4. Frequency of affordances in reference data set

	Percent	Total	String	Boolean	Integer	Object	Null	Number
Property	48.66%	163	40	54	21	9	3	36
Event	7.16%	24	9	2	4	1	8	0
Action	44.18%	148	11	8	11	4	112	2
Total	100.00%	335	60	64	36	14	123	38
Percent		100.00%	17.91%	19.10%	10.75%	4.18%	36.72%	11.34%

TDs in our dataset. All results, including the exact prompts and mocked message logs, used in the experiment, can be found in our repository¹⁴.

The accuracy results are displayed in Figure 4 and Figure 5. We used the *weak correctness* constraint when comparing the accuracy of the data types, as the affordance type in this scenario does not matter. The results of our manual descriptive accuracy check can be seen in Table 5 and Table 6. The descriptive elements of the TD were split into two categories: first, the general description of the device, and second, the descriptive elements of each affordance. Then, we manually reviewed them and assigned a score from one to five, as described in the previous section (see Table 3). The score was then converted back to a percent value. Due to time constraints, not all five generated TDs were reviewed, but only the first set of each run. We split the results into the three datasets used.

Table 5. Comparison of the descriptive accuracy on the Thing Level

Dataset/Model	gemini	gpt-4o	cogito
Custom	91.30%	92.39%	96.74%
W3C	63.46%	67.31%	71.15%
Kinder et al.	87.96%	93.52%	83.33%
Combined	80.59%	84.21%	83.22%

Table 6. Comparison of the descriptive accuracy on the Affordance Level

Dataset/Model	gemini	gpt-4o	cogito
Custom	86.06%	88.22%	85.10%
W3C	77.50%	77.00%	77.00%
Kinder et al.	89.64%	85.36%	86.60%
Combined	86.72%	85.00%	84.70%

7. Discussion

It took Gemini roughly 23 minutes to generate the 380 TDs, GPT-4o took a bit over four hours, and Cogito around three and a half hours. The local LLM, Cogito, was run on

¹⁴<https://doi.org/10.5281/zenodo.15526223>

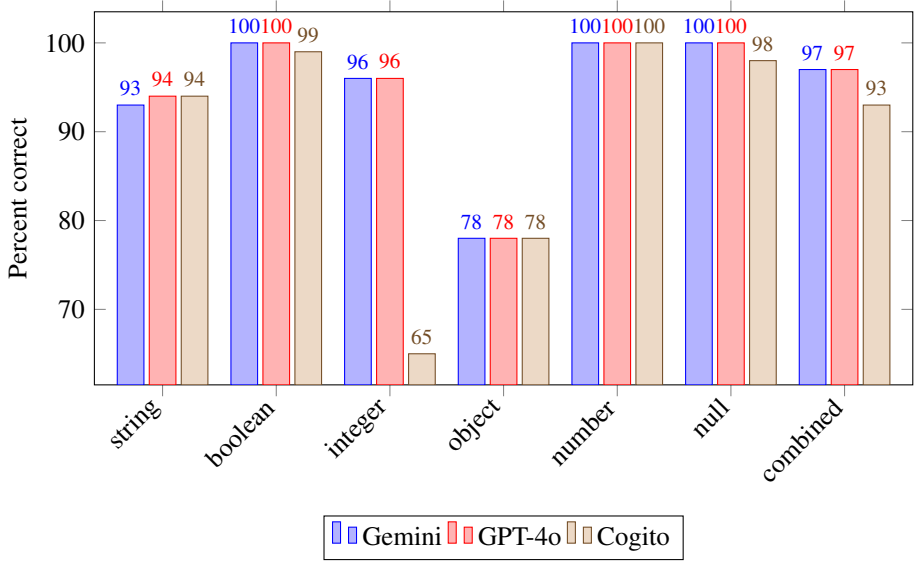


Figure 4. Accuracy per data type (weak correctness)

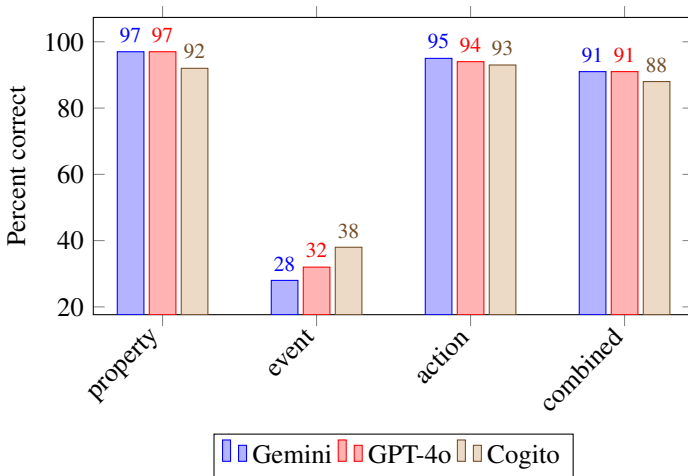


Figure 5. Accuracy per affordance type (strong correctness)

an NVIDIA A6000 GPU, an Intel Xeon Silver 4310 CPU with 24 Cores, and 128 GB of DDR4 RAM, whereas the proprietary models were run in the cloud. The differences in run-time are surprising, especially considering that all models had a similar accuracy and overall test results. These differences stayed consistent across multiple test runs. Thus, we believe that they are related to the specific models and the available compute power. However, as certain model parameters and the exact hardware are unknown for the commercial models, we can only speculate where the difference in runtime comes from.

7.1. Functional Accuracy

There is only a small noticeable difference in accuracy comparing the LLMs and they have a similar performance across the affordance and data types. For instance, all LLMs did great in affordances of all types, except for *object* types, which is the most complex type. The only outlier is Cogito, which had problems differentiating between *numbers* and *integers* (see Figure 4). However, if we do not distinguish between *integers* and *numbers*, the accuracy score increases to 98%. This problem could probably be fixed by providing examples or by performing additional static checks. For instance, instead of letting the LLMs decide what data type should be inferred from the MQTT payload, a parser could be deployed that uses a set of rules to determine the most likely data type. Alternatively, providing examples in the LLM prompt could reduce the inaccuracy.

The difference in accuracy between *events* and the other affordances is quite staggering (see Figure 5). This is especially interesting, since events and properties are very similar in the context of MQTT and TDs and thus we expected comparable performance. However, this similarity probably causes the issue. As we have discussed, the notion of an *event* and a *property* is indistinguishable in MQTT and is a design choice. Indeed, if we set the *events* and *properties* to be equal, we get an average accuracy score for *properties* and *events* of 85% for Cogito, 88% for GPT-4o, and Gemini.

7.2. Descriptive Accuracy

The results from the manual descriptive accuracy (Table 5 and Table 6) are promising. The difference in descriptive accuracy between the three datasets can be attributed to the varying MQTT topic structure. Our *custom* dataset follows the following structure: device type/affordance name, with many topics having additional location information. Both the device type as well as the affordance name often contain common abbreviations. For instance, *ev* for Electric Vehicle. The dataset of Kinder et al. uses a similar structure; however, not all of their topics contain information about the device type. W3C's testing dataset had a distinct topic structure: ID/affordance type/affordance name. Importantly, the ID is generic and does not reveal any information on the device type. Hence, making it challenging for the LLM to identify the devices reliably.

The overall performance of the descriptive accuracy was very stable across all three LLMs. When averaged out over all three data sets, they had very similar scores. One thing stood out, for unknown reasons, Cogito struggled with the *thing type* (see Listing 3 line three). It was the only LLM that frequently categorized the *@type* attribute of the TD as *ClassificationTD*. We assume that the LLM got confused, as this is the name of the Pydantic class that was given as the response schema. However, this had no significant impact on its overall score, since both *title* and *description* attributes contained the missing information. The *@type* attribute should be matched against some standardized vocabulary or an ontology to achieve the highest level of semantic interoperability.

Moreover, we investigated automatic evaluation scores for this task, specifically the cosine similarity of the text embedding vectors. A higher cosine similarity of a text embedding indicates a correlation in meaning of the original strings [27]. We calculated the text embedding of the descriptive attributes using OpenAI's text-embedding-3-large model and computed the correlation to the manual scores. However, only a moderate to

medium correlation between the two scores could be detected. Nevertheless, this does not invalidate the usefulness or effectiveness of the cosine similarity in general, but it shows that it may not be as expressive for this particular use case. The results of this experiment are in the full dataset.

7.3. Scoring Criteria and Limitations

While we strive to maintain objectivity in scoring, some bias cannot be ruled out, as the dataset was curated, scored, and edited by a single person. Although the **TD** dataset was assembled from various sources, many *descriptions* were added, since they were missing in the original sources. Moreover, what counts as *minor*, *moderate*, or *major* difference can be subjective and context-dependent. As discussed previously, assumed measurement units only received *minor* or *moderate* score deductions. Also, superfluous or redundant information had no point reductions. We believe this is justified, as the primary purpose of descriptive attributes in the **TD** is to give an intuition about the device and its affordances. In cases where this behavior would be harmful, stricter prompting and additional context about the device are needed.

7.4. Technology Readiness

We live in the age of generative AI and machine learning. The trend is quite clear, and it is safe to assume that **AIGC** will play an important role in the future, too [14]. With all the benefits of AI in mind, one should not underestimate its potential dangers. There exist numerous privacy concerns and attack vectors for **LLM** applications [28]. Our methodology does not incorporate security countermeasures against **LLM** targeted attacks. Therefore, we recommend deploying security and privacy countermeasures if our proposed methodology is deployed into production.

8. Conclusion and Future Work

In this paper, we introduced a novel approach to generating semantic device descriptions such as the **WoT TD** using message logs and **LLM** technology. We demonstrated the feasibility and competency of the approach in a case study of 76 **MQTT-IoT** devices with a total of 335 affordances, compiled from different sources. The evaluation showed that the **LLM** pipeline can generate **TDs** that, on average, have a high level of functional and descriptive accuracy to our reference data set. By automatically validating the **LLM** generated **TD** against our **WoT TD** schema, we could ensure that only **TDs** compliant with the specification were returned by the **LLM**.

There is still room for improvement in the performance of our approach, as was hinted in the discussion. A first next step would be to deploy a rule-based system to determine the data types, instead of relying on the **LLM**. Moreover, we would like to look into the feasibility of matching existing ontologies and semantic references to the “@context” of the **TD**, with the help of **LLMs**. Further, we are interested in enriching the generation process with more context information about the **IoT** device to enhance the descriptive accuracy. Also, fine-tuning of even smaller models might lead to additional improvements.

References

- [1] Jabbar S, Ullah F, Khalid S, Khan M, Han K. Semantic interoperability in heterogeneous IoT infrastructure for healthcare. *Wireless Communications and Mobile Computing*. 2017;2017. Publisher: Hindawi Limited.
- [2] Derhamy H, Eliasson J, Delsing J. IoT Interoperability—On-Demand and Low Latency Transparent Multiprotocol Translator. *IEEE Internet of Things Journal*. 2017 Oct;4(5):1754-63. Available from: <https://ieeexplore.ieee.org/document/7908961>.
- [3] Desai P, Sheth A, Anantharam P. Semantic Gateway as a Service Architecture for IoT Interoperability. In: 2015 IEEE International Conference on Mobile Services; 2015. p. 313-9. ISSN: 2329-6453. Available from: <https://ieeexplore.ieee.org/document/7226706>.
- [4] Amara FZ, Hemam M, Djezzar M, Maimor M. Semantic Web and Internet of Things: Challenges, Applications and Perspectives. *Journal of ICT Standardization*. 2022;10(2):261-92. Publisher: River Publishers.
- [5] Thakker D, Patel P, Ali MI, Shah T. Semantic Web of Things for Industry 4.0; 2020. ISSN: 22104968 Issue: 6 Publication Title: Semantic Web Volume: 11.
- [6] Jara AJ, Olivieri AC, Bocchi Y, Jung M, Kastner W, Skarmeta AF. Semantic Web of Things: an analysis of the application semantics for the IoT moving towards the IoT convergence. *Int J Web Grid Serv*. 2014 Apr;10(2/3):244-72. Available from: <https://doi.org/10.1504/IJWGS.2014.060260>.
- [7] Lagally M, Matsukura R, McCool M, Toumura K. Web of Things (WoT) Architecture 1.1. W3C;. Available from: <https://www.w3.org/TR/wot-architecture11/>.
- [8] Kaebisch S, McCool M, Korkan E, Kamiya T, Charpenay V, Kovatsch M. Web of Things (WoT) Thing Description 1.1. W3C; 2023. Available from: <https://www.w3.org/TR/wot-thing-description/>.
- [9] Salama F, Tsirkunen A, Korkan E, Käbisch S, Steinhorst S. WoT-Phyng-Sim: Integrating Physics Simulations with IoT Digital Twins using the Web of Things. In: 2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS); 2023. p. 1-8. Available from: <https://ieeexplore.ieee.org/document/10189326>.
- [10] Collins JT, Knapper J, McDermott SJ, Ayazi F, Bumke KE, Stirling J, et al. Simplifying the OpenFlexure microscope software with the web of things. *Royal Society Open Science*. 2021 Nov;8(11):211158. Available from: <https://royalsocietypublishing.org/doi/10.1098/rsos.211158>.
- [11] Modoni GE, Doukas M, Terkaj W, Sacco M, Mourtzis D. Enhancing factory data integration through the development of an ontology: from the reference models reuse to the semantic conversion of the legacy models. *International Journal of Computer Integrated Manufacturing*;30(10):1043-59. Publisher: Informa UK Limited. Available from: <https://www.tandfonline.com/doi/full/10.1080/0951192X.2016.1268720>.
- [12] Bamunuarachchi D, Georgakopoulos D, Banerjee A, Jayaraman PP. Digital Twins Supporting Efficient Digital Industrial Transformation. *Sensors*. 2021;21(20). Available from: <https://www.mdpi.com/1424-8220/21/20/6829>.
- [13] Rahamneh DS, Hanoun A, Jubair DF, Darabkh KA. WoTs-TDGV: Thing Description Generator and Validator. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*;15(3):281-97. Available from: <https://jowua.com/wp-content/uploads/2024/10/2024.I3.019.pdf>.
- [14] Wu J, Gan W, Chen Z, Wan S, Lin H. AI-Generated Content (AIGC): A Survey; 2023. Available from: <http://arxiv.org/abs/2304.06632>.
- [15] Hou X, Zhao Y, Liu Y, Yang Z, Wang K, Li L, et al. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans Softw Eng Methodol*. 2024 Dec;33(8). Available from: <https://doi.org/10.1145/3695988>.
- [16] Chen W, Yan-yi L, Tie-zheng G, Da-peng L, Tao H, Zhi L, et al. Systems engineering issues for industry applications of large language model. *Applied Soft Computing*. 2024 Jan;151.
- [17] Fakih M, Dharmaji R, Moghaddas Y, Quiros G, Ogundare O, Faruque MAA. LLM4PLC: Harnessing Large Language Models for Verifiable Programming of PLCs in Industrial Control Systems. In: ACM International Conference Proceeding Series. Association for Computing Machinery; 2024. p. 192-203.
- [18] Tarassow A. The potential of LLMs for coding with low-resource and domain-specific programming languages; 2023. Available from: <http://arxiv.org/abs/2307.13018>.
- [19] Kinder L, Käfer T. Towards Improving Large Language Models' Planning Capabilities on WoT Thing Descriptions by Generating Python Objects as Intermediary Representations. In: *CEUR Workshop Pro-*

- ceedings. vol. 3749; 2024. .
- [20] Marcinkowski D, Weis T. WoT-PL, Harnessing Large Language Models for IoT Schema Translation: A Conceptual Framework and Preliminary Findings. In: Proceedings of the 14th International Conference on the Internet of Things. IoT '24. New York, NY, USA: Association for Computing Machinery; 2025. p. 200–205. Available from: <https://doi.org/10.1145/3703790.3703833>.
 - [21] McCool M. Web of Things (WoT): Use Cases and Requirements; 2024. Available from: <https://w3c.github.io/wot-usecases/#UC-discovery-1>.
 - [22] Gruber L, Bachhiesl U, Wogrin S. The current state of research on energy communities. *Elektrotechnik und Informationstechnik*. 2021 Dec;138(8):515-24. Publisher: Springer.
 - [23] Rahman H, Hussain MI. A comprehensive survey on semantic interoperability for Internet of Things: State-of-the-art and research challenges. *Transactions on Emerging Telecommunications Technologies*. 2020 Dec;31(12). Publisher: Wiley Blackwell.
 - [24] Yassein MB, Shatnawi MQ, Aljwarneh S, Al-Hatmi R. Internet of Things: Survey and open issues of MQTT protocol. In: 2017 International Conference on Engineering & MIS (ICEMIS). IEEE; 2017. p. 1-6.
 - [25] Koster M, Korkan E, Aguzzi C. Web of Things (WoT) MQTT Binding Template. W3C; 2024. Available from: <https://w3c.github.io/wot-binding-templates/bindings/protocols/mqtt/index.html>.
 - [26] Chen M, Tworek J, Jun H, Yuan Q, Pinto HPdO, Kaplan J, et al. Evaluating Large Language Models Trained on Code; 2021. Available from: <http://arxiv.org/abs/2107.03374>.
 - [27] Goodfellow I, Bengio Y, Courville A. *Deep Learning*. MIT Press; 2016.
 - [28] Yao Y, Duan J, Xu K, Cai Y, Sun Z, Zhang Y. A Survey on Large Language Model (LLM) Security and Privacy: The Good, The Bad, and The Ugly. Shandong University; 2024. ISSN: 26672952 Issue: 2 Publication Title: High-Confidence Computing Volume: 4.