SAMM Copilot: Bootstrapping Semantic Models with the Eclipse Semantic Modeling Framework from Domain Data in JSON Using Large Language Models

Nazanin Mashhaditafreshi[1,†], Andreas Textor[2,*], Pascal Rübel[3], Nastaran Moarefvand[3] and Achim Wagner[3]

[1] *metaphacts GmbH. Walldorf, Germany.*
[2] *Bosch Connected Industry. Stuttgart, Germany.*
[3] *German Research Center for Artificial Intelligence (DFKI). Kaiserslautern, Germany.*

Abstract
The Semantic Aspect Meta Model (SAMM) is a modeling formalism for describing semantic models of parts of a Digital Twin – so-called Aspect Models. It is an open specification developed as part of the Eclipse Semantic Modeling Framework (ESMF). With SAMM being based on the Resource Description Framework (RDF) and Shapes Constraint Language (SHACL), Aspect Models are usually created and edited manually using a suitable textual editor or the graphical Aspect Model Editor. A well-defined mapping exists between Aspect Models and the JSON data they describe, enabling new bottom-up modeling approaches. In this way, instead of having a manual process for semantic modeling, Aspect Models can be automatically or semi-automatically derived from existing domain data in JSON format, making the modeling process more accessible and reducing manual effort. The proposed workflow translates JSON data into Aspect Models automatically using Large Language Models (LLMs). Our results demonstrate that LLMs can effectively bootstrap semantic models, and preliminary human evaluation suggests the feasibility and usefulness of this method in practice.

Keywords
Text-to-Knowledge Graph, Semantic Aspect Meta Model, Large Language Models

1. Introduction

In recent years, the concept of digital twin [1] has emerged as a transformative approach to connect the physical and digital worlds. A digital twin is a virtual representation of a physical asset, enhanced by real-time data flow between the two realms. This technology supports a wide range of applications across industries, including manufacturing, healthcare, urban planning, and automotive, enabling optimization of processes, resource management, and personalized services [2, 3, 4, 5].

Digital twins and semantic models enable smarter, more connected systems by providing a common language that aids integration with other tools and systems. This is crucial in industries like manufacturing, healthcare, and logistics, where clear communication and data sharing are key. Semantic models ensure interoperability across systems by offering a shared framework for information exchange, allowing seamless communication. They are typically based on frameworks like RDF and OWL, defining relationships and concepts within specific domains.

LLMs have gained popularity for generating human-like text, but they sometimes produce factually incorrect outputs. Semantic models and ontologies help by linking LLM outputs to structured knowledge, ensuring accuracy and domain-specific alignment. This integration mitigates hallucinations and enhances reliability in specialized domains. While creating and maintaining semantic models can be time-consuming, LLMs can automate repetitive tasks in semantic modeling, speeding up the process and allowing experts to focus on higher-level tasks.

LLM-TEXT2KG 2025: 4th International Workshop on LLM-Integrated Knowledge Graph Generation from Text (Text2KG), June 1 - Jun 5, 2025, co-located with Extended Semantic Web Conference (ESWC), Portoroz, Slovenia

[†] The author was affiliated with Bosch Connected Industry as part of her master's thesis at RPTU while conducting this work.

nm@metaphacts.com (N. Mashhaditafreshi); Andreas.Textor@de.bosch.com (A. Textor); Pascal.Ruebel@dfki.de (P. Rübel); Nastaran.Moarefvand@dfki.de (N. Moarefvand); Achim.Wagner@dfki.de (A. Wagner)

This work explores the intersection of digital twins, semantic modeling, and LLMs, aiming to minimize effort on repetitive tasks and allowing experts to focus on creative and intellectual work. The Semantic Aspect Meta Model (SAMM) is a metamodel designed for the semantic modeling of domain data. SAMM is a critical component of the Eclipse Semantic Modeling Framework (ESMF)[1], a sub-project of the broader Eclipse Digital Twin initiative. Historically, SAMM originated within the Semantic Data Structuring Working Group of the Open Manufacturing Platform[2], which was officially launched in September 2020 under the name BAMM [6].

This work examines how Aspect Models' structure and semantic descriptions can be automatically or semi-automatically derived from domain data in JSON format for the first time. All artifacts are available for reproducibility[3]. The research aims to address the following questions:

- **RQ1**: How can existing domain data in JSON format be leveraged to automatically or semi-automatically derive the basic structure of Aspect Models within SAMM?
- **RQ2**: How do open-source models compare to commercial solutions, such as OpenAI's models, in generating Aspect Models?
- **RQ3**: What automated methods can be developed to evaluate the Aspect Models generated by LLMs?
- **RQ4**: To what extent can data augmentation techniques improve the accuracy of LLMs when creating Aspect Models from domain data?
- **RQ5**: How can an LLM be integrated into various end-user tools and workflows, including but not limited to an Aspect Model Editor, to help domain experts?

2. Related Work

While related studies have explored the use of LLMs in ontology learning, semantic modeling, and schema generation, to the best of our knowledge, this is the first work to generate SAMM Aspect Models directly from structured JSON data using LLMs.

Babaei et al. [7] explored the use of LLMs for Ontology Learning (OL). In this work, the authors developed the LLMs4OL framework, which evaluates the potential of LLMs to automatically create complex ontologies, focusing on tasks such as term typing, taxonomy discovery, and relationship extraction. The authors found that while LLMs show promising potential for OL, they require fine-tuning for specific tasks to be practical and effective solutions.

The paper by [8] presents a semi-automatic approach to ontology learning based on the NeOn methodology framework [9] using LLMs. The authors guide LLMs through the NeOn methodology to build a structured ontology step by step using a prompt pipeline, while utilizing the Stanford wine ontology as a benchmark for their experiments. A prompt pipeline provided to GPT-3.5 was used to generate the output, which was then refined through syntax verification, consistency checks, and error resolution, using various tools and APIs to improve the initially created ontology. The authors determined that combining proper prompt engineering strategies with well-established ontology development practices can significantly enhance the consistency of ontologies generated by LLMs.

The study in [10] investigates the capability of LLMs to generate capability ontologies from natural language descriptions, focusing on GPT-4 Turbo, Claude 3, and Gemini Pro (which was excluded due to token limitations). The authors experimented with zero-shot, one-shot, and few-shot prompting techniques to generate ontologies of varying complexity, using a structured prompt template with the CaSk ontology as context. They evaluated the outputs through syntax validation in Protégé, consistency checks via Pellet OWL reasoner, and hallucination detection using SHACL shapes. Results showed that LLMs performed well, with Claude outperforming GPT-4 Turbo, and few-shot prompting yielding the best results. Limitations include the fixed examples in few-shot prompting and the high token cost of providing ontology context, suggesting future improvements through embedding techniques.

[1]https://projects.eclipse.org/projects/dt.esmf
[2]https://openmanufacturingplatform.github.io/
[3]https://github.com/NazaninTafreshi/sammcopilot

Mior et al. [11] examined the challenges associated with analyzing JSON documents, which, unlike relational databases, lack predefined schemas. This absence necessitates a trial-and-error approach, wherein analysts inspect a subset of documents, formulate assumptions, and subsequently validate them across the dataset. To mitigate these inefficiencies, prior research has explored automatic schema discovery. Notably, Baazizi et al. [12] proposed a method that derives individual document schemas and subsequently merges them into a unified representation. Building on this, the authors investigate the enhancement of JSON schema generation using large language models (LLMs), specifically comparing Code Llama and T5. Furthermore, they fine-tuned Code Llama using LoRA for a single epoch, demonstrating that the fine-tuned model outperformed others in generating more useful schemas. These findings suggest that integrating LLM-derived schemas into existing schema discovery tools can facilitate the generation of schemas that closely align with those produced by domain experts.

Ghanem et al. [13] investigates Text-to-Knowledge Graph (T2KG) construction using Large Language Models (LLMs), evaluating Zero-Shot Prompting, Few-Shot Prompting, and Fine-Tuning. The study compares Llama2, Mistral, and Starling, highlighting Fine-Tuning's superior performance, particularly when dataset size increases. Results indicate that fine-tuned models, particularly Mistral and Starling, outperform Zero-Shot and Few-Shot prompting in T2KG tasks, with Fine-Tuning leading to more accurate and structured knowledge graphs.

3. Methodology

The goal is to create a semantic model based on the domain data in JSON format. An example of JSON data from a device (in this example, a Bosch Smart Plug) is shown in Snippet 1. Normally, a domain expert would start from scratch and manually pick the correct modeling elements from SAMM to create an Aspect Model.

Similar to how the ESMF SDK generates JSON payloads from Aspect Models, it is possible to write a heuristic-based approach to evaluate a JSON structure and generate the corresponding Aspect Model, without using machine learning or LLMs. However, this heuristic approach has several limitations. For example, it cannot generate meaningful descriptions for elements like `'energyConsumption'`. In contrast, LLMs can produce relevant and useful descriptions, especially when given the right context. Furthermore, domain experts could simply provide requirements such as *"the phone number should include a country code and follow a specific regular expression,"* and LLMs could handle these instructions effectively. This makes LLMs a more capable and generalizable solution compared to other approaches.

For the above-mentioned reasons, our goal is to explore how LLMs can create the first draft of an Aspect Model based on the input data. This automated approach saves time and reduces manual effort for domain experts.

```json
{ "powerConsumption": 3.0,
  "energyConsumption": 7590.0,
  "energyConsumptionStartDate": "2024-10-25T15:25:24Z" }
```

Snippet 1: Example of domain data in JSON format.

SAMM is used to describe the structure of the data, but it does not store runtime data. The data can stay in its current format as provided by the device. If compatibility with Asset Administration Shell (AAS) is needed, AAS Submodel Template can be generated from Aspect Model. This AAS Submodel Template can later be instantiated and filled with runtime data, yielding an AAS Submodel Instance. Also, ESMF SDK supports the generation of other artifacts like Java Code, OpenAPI, and documentation pages. Since both SAMM and AAS can be represented in RDF format, they can be described using an RDF-based graph. This creates an interoperable knowledge graph. Figure 1 shows this workflow.

Figure 2 depicts the main steps in our research. First, we need a dataset to fine-tune the LLMs. We evaluate the output of the models using automatic methods and compare open-source and commercial LLMs. We also test different fine-tuning and prompting techniques. Finally, we deliver the solution to target users and collect feedback from domain experts through human evaluation.

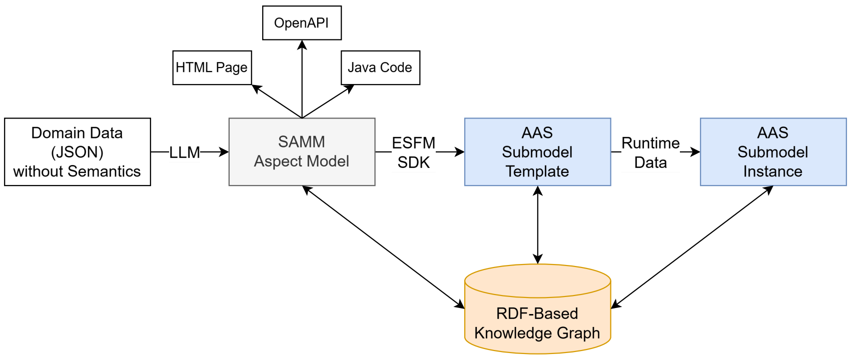


Figure 1: General workflow of the proposed solution.

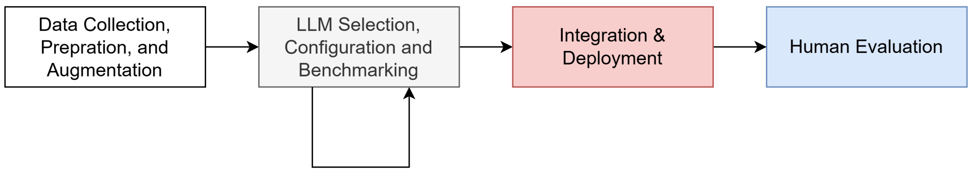

Data Collection, Prepration, and Augmentation
LLM Selection, Configuration and Benchmarking
Integration & Deployment
Human Evaluation

Figure 2: High-level steps of this research.

3.1. Data Collection, Preparation, and Augmentation

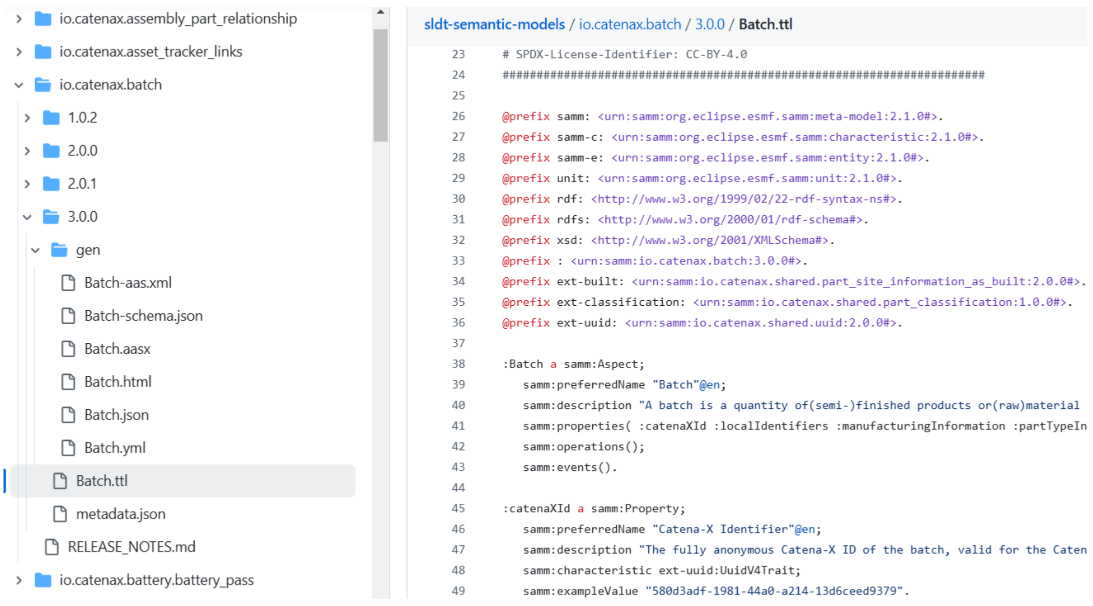
Semantic models created in Eclipse Tractus-X[4] were used as a data source for training and fine-tuning models. This is the only publicly available high-quality dataset with an open license, containing both complex and simple Aspect Models. Additionally, other data sources, such as the examples provided on the ESMF modeling guideline page[5] and the BatteryPassport model[6], can be considered for evaluation purposes, but they do not have a sufficient number of models, quality, and complexity to be considered for training.

In Figure 3, the structure of semantic models used in the Tractus-X project is shown. Each model is given a unique identifier, such as `'io.catenax.batch'`, and can have multiple versions. These versions are displayed in the left sidebar of Figure 3. For each version, a Uniform Resource Name (URN) identifier is generated following the SAMM schema, for example, `'urn:samm:io.catenax.batch:3.0.0#'`. The model is saved in a `.ttl` file, organized within a structured folder hierarchy.

Generated artifacts, such as sample JSON payloads (used as input for our LLMs) or AAS Submodel files, are stored in the `'gen'` folder. Some cleanup steps were necessary to prepare the data. This involved removing unnecessary information, such as extra comments, and fixing inconsistencies, modeling errors, and outdated artifacts. Since the number of collected models was not sufficient for training, we applied data augmentation techniques to increase the number of samples.

After completing the data cleanup, we retained a total of 155 Aspect Models. By applying various data augmentation techniques, we modified the original Aspect Models to create new ones, resulting in a total of 364 models. For data augmentation, Property names in the SAMM model were modified, altering the resulting JSON keys. Moreover, the `'samm:exampleValue'` attribute was removed to make the values random. Lastly, elements were randomly eliminated from the Aspect Models to simplify the model. The dataset was then divided into training, validation, and test sets. The training and validation sets were utilized during the model training process, while the test set was reserved exclusively for evaluation purposes. Table 1 summarizes the dataset sizes for both the original and augmented datasets. In this research, 20% of the data was allocated as the test set, 10% of the remaining data was used for validation, and the rest was utilized for training.

[4]https://github.com/eclipse-tractusx/sldt-semantic-models (retrieved on 18 September 2024)
[5]https://eclipse-esmf.github.io/samm-specification/snapshot/modeling-guidelines.html
[6]https://github.com/batterypass/BatteryPassDataModel



```
> io.catenax.assembly_part_relationship
> io.catenax.asset_tracker_links
v io.catenax.batch
  > 1.0.2
  > 2.0.0
  > 2.0.1
  v 3.0.0
    v gen
        Batch-aas.xml
        Batch-schema.json
        Batch.aasx
        Batch.html
        Batch.json
        Batch.yml
      Batch.ttl
      metadata.json
    RELEASE_NOTES.md
> io.catenax.battery.battery_pass
```

sldt-semantic-models / io.catenax.batch / 3.0.0 / Batch.ttl

```
# SPDX-License-Identifier: CC-BY-4.0
#######################################################################

@prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#>.
@prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#>.
@prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#>.
@prefix unit: <urn:samm:org.eclipse.esmf.samm:unit:2.1.0#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <urn:samm:io.catenax.batch:3.0.0#>.
@prefix ext-built: <urn:samm:io.catenax.shared.part_site_information_as_built:2.0.0#>.
@prefix ext-classification: <urn:samm:io.catenax.shared.part_classification:1.0.0#>.
@prefix ext-uuid: <urn:samm:io.catenax.shared.uuid:2.0.0#>.

:Batch a samm:Aspect;
   samm:preferredName "Batch"@en;
   samm:description "A batch is a quantity of(semi-)finished products or(raw)material
   samm:properties( :catenaXId :localIdentifiers :manufacturingInformation :partTypeIn
   samm:operations();
   samm:events().

:catenaXId a samm:Property;
   samm:preferredName "Catena-X Identifier"@en;
   samm:description "The fully anonymous Catena-X ID of the batch, valid for the Caten
   samm:characteristic ext-uuid:UuidV4Trait;
   samm:exampleValue "580d3adf-1981-44a0-a214-13d6ceed9379".
```

Figure 3: Semantic models in Tractus-x project.

Dataset	Train	Validation	Test	Total
Original Data	110	13	32	155
Augmented Data	241	27	96	364

Table 1
The size of the training, validation, and test sets.

3.2. LLM Selection, Configuration, and Benchmarking

In this study, we evaluate a range of LLMs, including both commercial and open-source options, to identify suitable candidates for fine-tuning. Our primary goal is to assess models that can operate efficiently on consumer-grade laptops equipped with GPUs, ensuring that the process remains accessible and practical for a wider range of users. This consideration reflects our intention to balance computational feasibility with model performance, enabling experimentation without requiring expensive high-end server infrastructure.

For commercial models, we focus on OpenAI's GPT series, specifically GPT-4o-mini and GPT-4o. These models are well-regarded for their robust performance across diverse tasks and provide a strong benchmark for comparison with open-source alternatives. Leveraging commercial models allows us to evaluate cutting-edge capabilities while considering trade-offs such as accessibility, cost, and scalability.

In the open-source category, we prioritize smaller models that are compatible with our hardware constraints based on benchmarks and the availability of models for fine-tuning in the Unsloth library[7] [14]. We limited our options to Unsloth, because it provides faster and more efficient fine-tuning compared to other available libraries. We selected Llama 3.1[15], Llama 3.2, Qwen2.5-Coder[16], and CodeLlama[17] for this study. To ensure compatibility with laptops equipped with GPUs, we limit our selection to smaller versions of these models, which range from 3 billion to 8 billion parameters. This size range keeps a practical balance between computational feasibility and performance, making these models suitable for experimentation on such hardware.

To identify the most promising models for fine-tuning, we evaluate them using zero-shot, one-shot, and two-shot prompting techniques. These methods test the models' ability to generate accurate and coherent responses with minimal task-specific data. Since we do not have access to a large dataset for fine-tuning, it is particularly important to focus on models that perform well in few-shot scenarios. Models that perform poorly in few-shot evaluations are unlikely to benefit significantly from fine-

[7]https://github.com/unslothai/unsloth

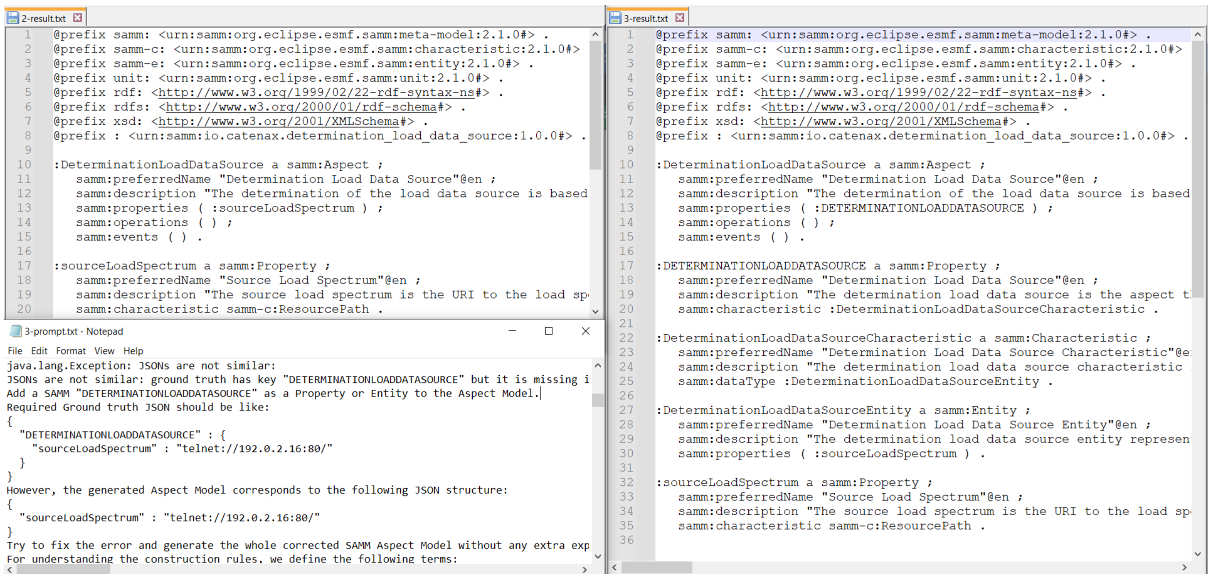
tuning. This is because few-shot performance often serves as an indicator of a model's inherent ability to generalize to new tasks. If a model struggles to generate meaningful outputs with limited examples, it suggests that the underlying pre-trained representations are either weak or not well-suited to the target tasks. Fine-tuning such models on a small dataset may fail to overcome these shortcomings and could lead to overfitting, where the model memorizes the small dataset instead of learning generalizable patterns. Consequently, investing resources in fine-tuning these models would not be promising, as the expected improvements in performance are minimal. By focusing on models that already demonstrate competence in few-shot prompting, we increase the likelihood of achieving meaningful gains through fine-tuning while making efficient use of our limited data and computational resources.

The fine-tuning approach included leveraging cloud services such as Azure OpenAI and OpenAI's API. We also performed local fine-tuning of open-source models, specifically Qwen2.5-Coder, using the Unsloth library on Google Colab with T4 GPU instances. The local fine-tuning utilized PEFT with LoRA configuration and the Supervised Fine-Tuning (SFT) Trainer.

Prompting Techniques

Zero-shot and few-shot prompting templates are shown in Snippet 2, 3 and 4. Iterative prompting is done by passing the previous output along with the discovered exception in the next iteration, depicted in Snippet 5. Moreover, extra hints are provided to help the model solve the issue, which contain documentation details or some generic instructions.

Figure 4 provides an example of the iterative approach. On the left side, at the top, the result of the second attempt is shown, which is stored in *'2-result.txt'*. At the bottom left, it can be seen that there is a missing element in the JSON, and some guidelines are provided in the prompt. For example, the model is instructed to add a *Property* or *Entity* with a specific name. On the right side, the addition of this top-level entity to the model is shown. This highlights the effectiveness of this approach.



```
@prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#> .
@prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#>
@prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#> .
@prefix unit: <urn:samm:org.eclipse.esmf.samm:unit:2.1.0#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix : <urn:samm:io.catenax.determination_load_data_source:1.0.0#> .

:DeterminationLoadDataSource a samm:Aspect ;
   samm:preferredName "Determination Load Data Source"@en ;
   samm:description "The determination of the load data source is based
   samm:properties ( :sourceLoadSpectrum ) ;
   samm:operations ( ) ;
   samm:events ( ) .

:sourceLoadSpectrum a samm:Property ;
   samm:preferredName "Source Load Spectrum"@en ;
   samm:description "The source load spectrum is the URI to the load sp
   samm:characteristic samm-c:ResourcePath .
```

```
java.lang.Exception: JSONs are not similar:
JSONs are not similar: ground truth has key "DETERMINATIONLOADDATASOURCE" but it is missing i
Add a SAMM "DETERMINATIONLOADDATASOURCE" as a Property or Entity to the Aspect Model.
Required Ground truth JSON should be like:
{
  "DETERMINATIONLOADDATASOURCE" : {
    "sourceLoadSpectrum" : "telnet://192.0.2.16:80/"
  }
}
However, the generated Aspect Model corresponds to the following JSON structure:
{
  "sourceLoadSpectrum" : "telnet://192.0.2.16:80/"
}
Try to fix the error and generate the whole corrected SAMM Aspect Model without any extra exp
For understanding the construction rules, we define the following terms:
```

```
@prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#> .
@prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#>
@prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#> .
@prefix unit: <urn:samm:org.eclipse.esmf.samm:unit:2.1.0#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix : <urn:samm:io.catenax.determination_load_data_source:1.0.0#> .

:DeterminationLoadDataSource a samm:Aspect ;
   samm:preferredName "Determination Load Data Source"@en ;
   samm:description "The determination of the load data source is based
   samm:properties ( :DETERMINATIONLOADDATASOURCE ) ;
   samm:operations ( ) ;
   samm:events ( ) .

:DETERMINATIONLOADDATASOURCE a samm:Property ;
   samm:preferredName "Determination Load Data Source"@en ;
   samm:description "The determination load data source is the aspect t
   samm:characteristic :DeterminationLoadDataSourceCharacteristic .

:DeterminationLoadDataSourceCharacteristic a samm:Characteristic ;
   samm:preferredName "Determination Load Data Source Characteristic"@e
   samm:description "The determination load data source characteristic
   samm:dataType :DeterminationLoadDataSourceEntity .

:DeterminationLoadDataSourceEntity a samm:Entity ;
   samm:preferredName "Determination Load Data Source Entity"@en ;
   samm:description "The determination load data source entity represen
   samm:properties ( :sourceLoadSpectrum ) .

:sourceLoadSpectrum a samm:Property ;
   samm:preferredName "Source Load Spectrum"@en ;
   samm:description "The source load spectrum is the URI to the load sp
   samm:characteristic samm-c:ResourcePath .
```

Figure 4: An example of iterative prompting.

```
You are a bot to help people create Semantic Aspect Meta Model (SAMM) from given JSON data. Create
    ↪ SAMM model based on the following JSON:
JSON:
<JSON EXAMPLE>
Provide only the SAMM model without any extra explanation. Make sure that the output is a valid RDF
    ↪ Turtle format.
```

Snippet 2: Zero-shot prompt template.

```
This is an example SAMM model:
<EXAMPLE SAMM ASPECT MODEL>
This is its corresponding JSON example:
<JSON PAYLOAD OF EXAMPLE SAMM ASPECT MODEL>
Your task is to create a SAMM model from a JSON Example.
Json Example:
<JSON EXAMPLE>
Provide only the SAMM model without any extra explanation. Make sure that the output is a valid RDF
    ↪ Turtle format.
```

Snippet 3: One-shot prompt template.

```
This is an example SAMM model:
<EXAMPLE SAMM ASPECT MODEL 1>
This is its corresponding JSON example:
<JSON PAYLOAD OF EXAMPLE SAMM ASPECT MODEL 1>
This is an example SAMM model:
<EXAMPLE SAMM ASPECT MODEL 2>
This is its corresponding JSON example:
<JSON PAYLOAD OF EXAMPLE SAMM ASPECT MODEL 2>
Your task is to create a SAMM model from a JSON Example.
Json Example:
<JSON EXAMPLE>
Provide only the SAMM model without any extra explanation. Make sure that the output is a valid RDF
    ↪ Turtle format.
```

Snippet 4: Two-shot prompt template.

```
In your previous attempt you created this Semantic Aspect Meta Model SAMM Aspect Model
<PREVIOUS SAMM ASPECT MODEL OUTPUT>
But it has the following error:
<PREVIOUS EXCEPTION>
Try to fix the error and generate the whole corrected SAMM Aspect Model without any extra
    ↪ explanation.
<EXTRA HINTS>
```

Snippet 5: Prompt template with feedbacks.

LLM Evaluation

To evaluate the generated output of the LLM, we introduce three automated mechanisms. In the first step, the generated model must be a valid RDF Turtle format. Apache Jena[8] was used for RDF Turtle validation. In the second step, we ensure that the model is a valid Aspect Model. In the third step, we utilize the ESMF SDK to generate a sample JSON payload from the Aspect Model generated with LLM. The generated JSON payload should structurally be similar to the original input provided. To determine if two JSON objects are similar, it is first checked that both have the same number of keys. For keys with primitive values, such as integers, the values are not considered. However, for keys with complex objects or arrays, their similarity is checked recursively. For arrays, their sizes must be equal, and all elements within the arrays must also be similar.

The three evaluation methods are referred to as *'Valid Turtle'*, *'Valid SAMM'*, and *'Correct'*, respectively. For each input, these methods produce three boolean values. Ideally, all three values would be *'true'*.

However, it is challenging to measure the completeness of models. SAMM contains various modeling elements. In its most basic form, an Aspect Model consists of one Aspect and multiple *Properties* with *Characteristics*. *Characteristics* can include elements such as *Measurement*, *Enumeration*, and *Duration*, among others. A model is considered more complex if it contains diverse elements such as *Characteristics*, *Entities*, *Traits*, *Constraints*, and so on. Another indicator of complexity is the length of the model, measured by the number of triples it contains. A complete model typically includes more

[8]https://github.com/apache/jena

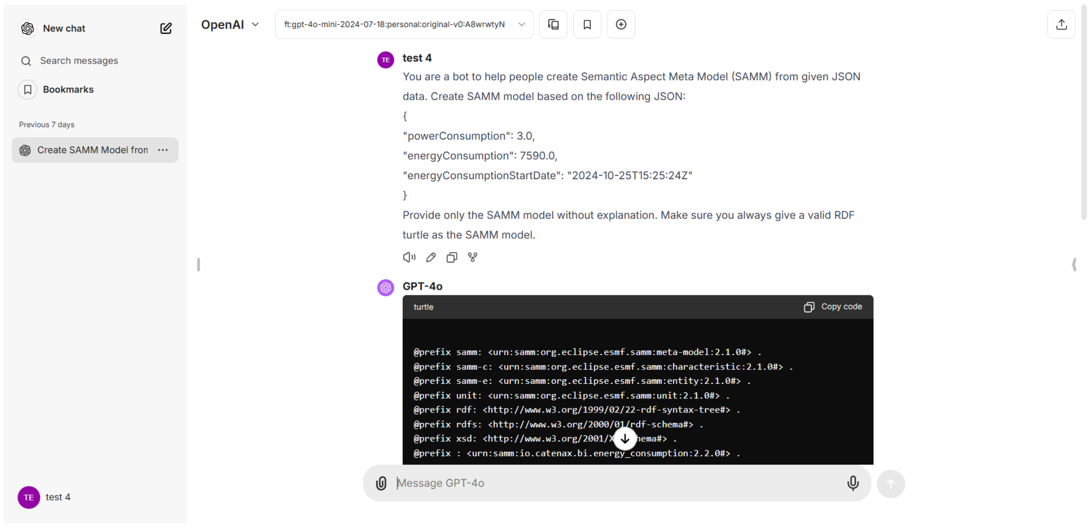
constraints, unit information, detailed descriptions, and precise components. If the LLM generates a more complex model for the same task, it may indicate a higher degree of completeness. Nevertheless, since the input provided to the model is only a JSON file, it is not expected to produce a fully complete model, as our goal is to create only the first draft.

3.3. Integration and Deployment

There are several ways to integrate LLMs to assist experts in modeling Aspect Models. One approach is to provide a user interface similar to ChatGPT, where the user can simply input their data. However, this method does not allow us to automatically verify whether the model generates an Aspect Model that actually corresponds to the provided input. Unless we use tool-calling functionalities to invoke a custom application.

A custom application takes user input and performs all necessary steps in the background. This approach ensures that the generated output is always a valid SAMM Aspect Model, and that the correct Aspect Model is produced. If an error occurs, it can be fed back into the LLM for correction using appropriate prompt engineering methods. Although this approach is less interactive, it guarantees higher quality responses. The generated model can then be manually or automatically transferred to other tools like Aspect Model Editor.

As shown in Figure 6, LibreChat can serve as a user interface similar to ChatGPT. A screenshot of this platform, called *SAMM Copilot*[9], with an example input is depicted in Figure 5. The Aspect Model Editor can use a library such as LangChain4j[10] to connect to various LLM models, including commercial models from Azure, OpenAI, and AWS, as well as locally hosted models via Ollama[11].



test 4

You are a bot to help people create Semantic Aspect Meta Model (SAMM) from given JSON data. Create SAMM model based on the following JSON:

```
{
"powerConsumption": 3.0,
"energyConsumption": 7590.0,
"energyConsumptionStartDate": "2024-10-25T15:25:24Z"
}
```

Provide only the SAMM model without explanation. Make sure you always give a valid RDF turtle as the SAMM model.

GPT-4o

```turtle
@prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#> .
@prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#> .
@prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#> .
@prefix unit: <urn:samm:org.eclipse.esmf.samm:unit:2.1.0#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-tree#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/X hema#> .
@prefix : <urn:samm:io.catenax.bi.energy_consumption:2.2.0#> .
```

Figure 5: SAMM Copilot platform user interface.

In order to further enhance the user experience, the solution can be integrated directly into the Aspect Model Editor. As shown in Figure 7, user can provide an example domain data directly in the Aspect Model Editor and then get the first version of the Aspect Model. In this way, the domain expert would not need to start from scratch and perform repetitive tasks.

In future scenarios, users will also be able to provide their requirements in natural language, as the process is depicted in Figure 8. As shown in Figure 8a, when the user clicks on the magic wand icon, a dialog box will open. The user can describe the required changes in natural language, as shown in Figure 8b. For example, they could ask the AI to add a regular expression constraint to a property. In

[9]https://sammcopilot.studio/
[10]https://github.com/langchain4j
[11]https://ollama.com/

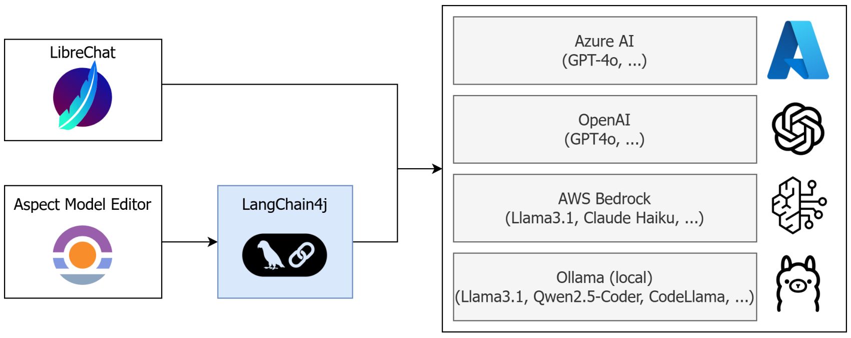


Figure 6: Overall architecture and components of the end-solution.

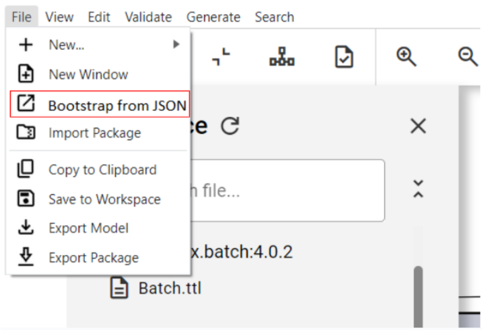


Figure 7: Bootstrap Aspect Model from JSON example in the Aspect Model Editor.

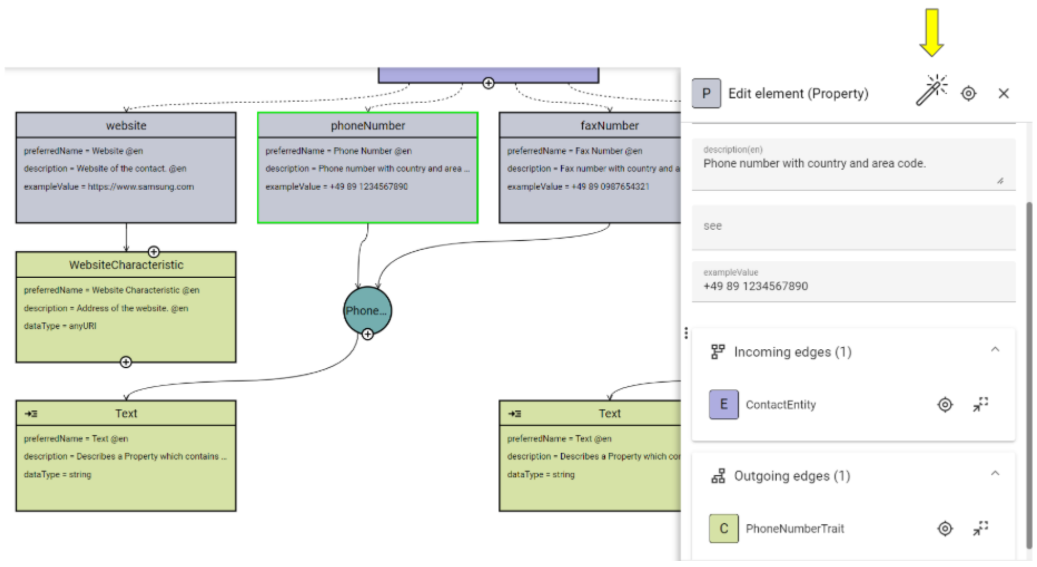
the background, the AI will apply the necessary changes to further enhance the Aspect Model, such as adding descriptions or performing more complex operations like adding constraints.

3.4. Human Evaluation

We conducted human evaluations to gather feedback on our tool, which is a user interface like ChatGPT. Participants were asked to create a model for the Bosch Smart Plug manually, and an example JSON payload was provided to them. Then, they were instructed to perform the same task using SAMM Copilot. After performing both tasks, they were asked to complete a short evaluation form. This would give a better indication of the real-world applicability of the developed solution.

The evaluation form consisted of the following sections and questions:

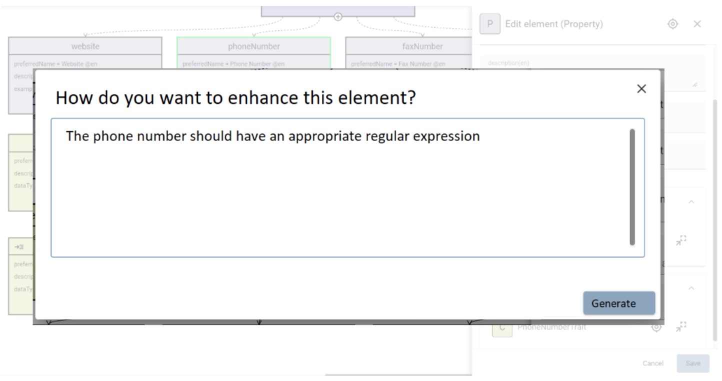
1. Introduction: Participants were welcomed to the evaluation with the following note: *"Thank you for participating in this user evaluation. Filling this form only takes 5 minutes."*
2. User Familiarity with SAMM: Participants were asked to indicate their familiarity with the SAMM by choosing one of the following options:
   a) No Experience
   b) I have created a few models (1-5 models).
   c) I am an experienced modeler (more than 5 models).
3. Manual Modeling Duration: Participants were asked: *"How long did it take to manually model?"*
4. Use of External Sources: Participants were requested: *"Did you use any external sources? If yes, please name them."* (An open-ended response is provided to the participants.)
5. SAMM Copilot Usage Duration: Participants were questioned: *"How long did it take to use SAMM Copilot?"*
6. Satisfaction with SAMM Copilot Results: Participants were inquired to evaluate their satisfaction with the model produced by SAMM Copilot by selecting one of the following options:
   a) Yes, it is a valid and complete model.


website
preferredName = Website @en
description = Website of the contact. @en
exampleValue = https://www.samsung.com
phoneNumber
preferredName = Phone Number @en
description = Phone number with country and area ...
exampleValue = +49 89 1234567890
faxNumber
preferredName = Fax Number @en
description = Fax number with country and a
exampleValue = +49 89 0987654321
WebsiteCharacteristic
preferredName = Website Characteristic @en
description = Address of the website. @en
dataType = anyURI
Phone...
Text
preferredName = Text @en
description = Describes a Property which contains ...
dataType = string
Text
preferredName = Text @en
description = Describes a Property which co
dataType = string
P
Edit element (Property)
see
Incoming edges (1)
E
ContactEntity
Outgoing edges (1)
C
PhoneNumberTrait

description(en): Phone number with country and area code.

exampleValue: +49 89 1234567890

(a) User can select the desired element for enhancement and click on magic wand icon.



How do you want to enhance this element?

The phone number should have an appropriate regular expression

(b) In the input field user described the requirement.

Figure 8: Enhance an element of the Aspect Model by AI using natural language specification.

b) Yes, it is a valid model, but it is not complete.
c) No, it is a valid model, but it is wrong.
d) No, it is not a valid model.

4. Experiments and Results

Experiment 1: Effect of Temperature

The aim of this experiment is to examine the effect of the temperature on the quality of the output in the inference mode. Temperature introduces randomness in the generated results and may lead to hallucinations when higher values are used. For each model, the optimal temperature value varies. Typically, lower temperature values are preferred when deterministic output is required. In this experiment, we focus exclusively on the GPT-4o-mini model.

Table 2 presents the results, where at a temperature of $T = 0.7$, the fine-tuned GPT-4o-mini model produced 41 ”Valid SAMM” outputs, and 32 ”Correct” outputs. With a temperature of $T = 0.0$, the model generated the same results, with 41 ”Valid SAMM” outputs, and 32 ”Correct” outputs.

The fact that the temperature setting has minimal impact on the model’s performance suggests that the structural aspects of the output are well-learned and remain stable. Changes in descriptive outputs

Table 2
Effect of temperature on the performance of fine-tuned GPT-4o-mini model (one-shot prompting) trained on the original data.

Model	Valid Turtle	Valid SAMM	Correct
Fine-tuned GPT-4o-mini (temperature 0.7)	85 (88%)	41 (42%)	32 (33%)
Fine-tuned GPT-4o-mini (temperature 0.0)	79 (82%)	41 (42%)	32 (33%)

such as `samm:description` might become more apparent with varied temperature settings, as they are more sensitive to next-token probability distributions.

Experiment 2: Effect of Examples and Number of Attempts

For few-shot prompting, we provide the model with one or two example SAMM Aspect Models along with their corresponding JSON payloads in the prompt. This helps the model learn from the provided examples. A well-chosen example that encapsulates the essence of the data and is sufficiently complex can enhance the model's performance. Therefore, we select one complex Aspect Model that includes most of the elements of SAMM and one simpler Aspect Model.

Additionally, since the output of an LLM is non-deterministic and varies with each execution, we aim to investigate the effect of repeating the same prompt multiple times. Specifically, we examine whether providing the same prompt repeatedly influences the results and if this repetition improves the model's performance. Both experiments are conducted using the GPT-4o-mini model.

Table 3 shows the results of this experiment. Using `'Waste'`[12] as a simpler Aspect Model example led to 34 "Valid SAMM", and 28 "Correct" outputs. The model produced 41 "Valid SAMM", and 32 "Correct" outputs using `'SecondaryMaterialContent'`[13] as a more complete Aspect Model example. A more complex Aspect Model example, which contains more elements, increased the number of valid and correct outputs.

Table 3
Effect of example complexity on the performance of fine-tuned GPT-4o-mini in one-shot prompting after three attempts (pass@3).

Model	Valid Turtle	Valid SAMM	Correct
`Waste` as a simple example	84 (87%)	34 (35%)	28 (29%)
`SecondaryMaterialContent` as a complex example	85 (88%)	41 (42%)	32 (33%)

The effect of multiple attempts is shown in Table 4, where with `'Waste'` as an example, "Correct" outputs increased from 21 in the first attempt to 25 in the second and 28 in the third. When `'SecondaryMaterialContent'` was used, the "Correct" outputs increased from 27 in the first attempt to 31 in the second and 32 in the third.

Table 4
Effect of multiple attempts on the performance of the fine-tuned GPT-4o-mini (pass@3 metric) in one-shot prompting.

Model	Attempt 1	Attempt 2	Attempt 3
`Waste` as a simple example	21 (21%)	25 (26%) (+4)	28 (29%) (+3)
`SecondaryMaterialContent` as a complex example	27 (28%)	31 (32%) (+4)	32 (33%) (+1)

The better performance observed when using a more complex example `'SecondaryMaterialContent'` highlights the advantage of using examples with richer structures and more elements for one-shot inference. The incremental improvements with repeated attempts

[12]https://github.com/eclipse-tractusx/sldt-semantic-models/blob/main/io.catenax.waste/2.0.0/Waste.ttl
[13]https://github.com/eclipse-tractusx/sldt-semantic-models/blob/main/io.catenax.shared.secondary_material_content/1.0.0/SecondaryMaterialContent.ttl

demonstrate that the model can refine its outputs with multiple tries. However, after a certain point, the performance seems to plateau.

Experiment 3: Comparison of Llama3.1-8b, Qwen2.5-coder-7b, Llama3.2-3b, and CodeLlama-7b without Fine-Tuning

Four open-source LLMs were selected and evaluated without any fine-tuning. Using our one-shot prompt template, which includes an example in the prompt, we compared the performance of these models. The model with the best performance will be considered for the next experiments.

As seen in Table 5, Qwen2.5-Coder 7B generated 39 ”Valid Turtle” outputs. CodeLlama 7B yielded 31 ”Valid Turtle” outputs. Also, Llama 3.1 8B produced 29 ”Valid Turtle” outputs. Lastly, Llama 3.2 3B resulted in 5 ”Valid Turtle” outputs. None of the models were capable of generating a valid Aspect Model using one example of an Aspect Model in the prompt.

Table 5
Performance of open-source models without fine-tuning (one-shot prompting).

Model	Valid Turtle	Valid SAMM	Correct
Qwen2.5-Coder 7B	39 (40%)	0	0
CodeLlama 7B	31 (32%)	0	0
Llama 3.1 8B	29 (30%)	0	0
Llama 3.2 3B	5 (5%)	0	0

The results indicate that smaller LLMs perform poorly, underscoring the limitations of smaller models in tasks requiring detailed and complex output generation. While code-specific models show some improvement, they still do not produce ”Valid SAMM” outputs. This highlights the gap between smaller and larger LLMs in handling complex tasks, and the need for fine-tuning these models.

Experiment 4: Effect of More Shots on Qwen2.5-coder-7b and Llama3.1-8b

We also aim to investigate whether two-shot prompts improve the performance of LLMs. This experiment is conducted only on the top two open-source models. The model that demonstrates the best performance will be fine-tuned in the subsequent experiments.

The impact of using more shots on Qwen2.5-Coder-7b and Llama 3.1 8B is presented in Table 6. For Qwen2.5-Coder 7B, the two-shot setting resulted in 3 ”Correct” outputs. For Llama, using two-shot prompt produced 4 ”Valid SAMM”. For other cases, no ”Correct” SAMM was generated.

Table 6
Comparison of one-shot and two-shot prompting on the Qwen2.5-Coder and Llama 3.1 base models.

Model	Valid Turtle	Valid SAMM	Correct
Qwen2.5-Coder 7B (two-shot)	38 (39%)	12 (12%)	3
Qwen2.5-Coder 7B (one-shot)	39 (40%)	0	0
Llama 3.1 8B (two-shot)	24 (25%)	4 (4%)	0
Llama 3.1 8B (one-shot)	29 (30%)	0	0

The introduction of a second example in the prompt improves the models' ability to produce valid SAMM Aspect Models. This suggests that more examples can help LLM to learn the general rules of creating an Aspect Model, but the lack of improvements in the ”Correct” output indicates that models are not able to capture the complex relation between Aspect Model and its JSON mapping. The Qwen2.5-Coder model demonstrates a stronger ability to generalize in this regard.

Experiment 5: Batch Size Effect on Fine-Tuning GPT-4o-mini

While fine-tuning GPT-4o-mini, we aim to investigate how batch size affects the quality of the model. Typically, larger datasets require larger batch sizes. In one scenario, we use a batch size of 8, a learning rate of 0.2, and train for 3 epochs. In another scenario, we use a batch size of 2, a learning rate of 0.2, and also train for 3 epochs.

As shown in Table 7, with a batch size of 8, the model had 13 ”Valid SAMM” outputs, and 5 ”Correct” outputs. With a batch size of 2, the model had better performance with 14 ”Valid SAMM” outputs, and 12 ”Correct” outputs.

Table 7
Effect of batch size on model performance with one-shot prompting trained on original data.

Model	Valid Turtle	Valid SAMM	Correct
GPT-4o-mini (batch size = 8)	84 (87%)	13 (13%)	5 (5%)
GPT-4o-mini (batch size = 2)	79 (82%)	14 (14%)	12 (12%)

The results indicate that smaller batch sizes improve the quality of outputs. This highlights the importance of tailoring batch sizes to the task and dataset size to achieve optimal performance. Small batch sizes enable the model to focus on individual examples more effectively.

Experiment 6 : Comparison of GPT-4o-mini Base, Fine-Tuned on Original Data, and Fine-Tuned on Augmented Data

In this experiment, different aspects were considered. One of the aims of this experiment is to evaluate the performance of the model without any fine-tuning. We do not expect to obtain any valid Aspect Models using a zero-shot prompt, as this domain was not part of the LLM’s pretraining. However, GPT-4o or GPT-4o-mini might be capable of learning from examples in few-shot prompting. We use one-shot and two-shot prompts and compare the performance of the models.

The second goal of this experiment is to compare the performance of GPT-4o and GPT-4o-mini on our task to determine if the larger GPT-4o model outperforms the mini version. Typically, larger models are expected to perform better.

The third aspect is to observe the effect of data augmentation. We have two datasets. The augmented dataset contains more samples. However, we are interested in seeing if our approach in data augmentation helps the LLM or not.

Table 8 shows the results of the GPT-4o-mini model under different fine-tuning and prompting strategies. The base model showed low performance in zero-shot, with improvements in one-shot and two-shot prompting. Fine-tuning on the original dataset improved results, especially in ”Valid SAMM” and ”Correct.” Fine-tuning on the augmented dataset led to mixed results, with a drop in zero-shot performance but better outcomes in one-shot and two-shot scenarios.

Table 8
Performance of GPT-4o-mini under various prompting and fine-tuning scenarios.

Model	Valid Turtle	Valid SAMM	Correct
GPT-4o-mini (zero-shot)	75 (78%)	0	0
GPT-4o-mini (one-shot)	88 (91%)	24 (25%)	4 (4%)
GPT-4o-mini (two-shot)	83 (86%)	34 (35%)	9 (9%)
GPT-4o-mini fine-tuned on original data (zero-shot)	79 (82%)	20 (20%)	21 (21%)
GPT-4o-mini fine-tuned on original data (one-shot)	85 (88%)	41 (42%)	32 (33%)
GPT-4o-mini fine-tuned on original data (two-shot)	80 (83%)	39 (40%)	31 (32%)
GPT-4o-mini fine-tuned on augmented data (zero-shot)	67 (69%)	25 (26%)	22 (22%)
GPT-4o-mini fine-tuned on augmented data (one-shot)	82 (85%)	43 (44%)	35 (36%)
GPT-4o-mini fine-tuned on augmented data (two-shot)	75 (78%)	36 (37%)	28 (29%)

The results demonstrate that fine-tuning significantly improves the performance of the GPT-4o-mini model over its base configuration. The augmented dataset does not lead to substantial improvements, indicating that the data augmentation methods are not sufficient to improve the fine-tuned model. This emphasizes the importance of dataset quality and diversity, which might have been negatively impacted by the augmentation process.

Experiment 7: Iterative Prompting on Fine-Tuned GPT-4o-mini

Using error messages and general guidelines after the first attempt may help the model correct itself. An effective exception message is crucial for success, as it guides the model in the right direction. Since the previous output is part of the prompt, the context window plays an important role. The aim of this experiment is to investigate whether iterative prompting improves the performance of the LLM or not.

The comparison of iterative prompting on fine-tuned GPT-4o-mini is shown in Table 9. With simple retries using one-shot prompting on the model, fine-tuned on the original data, the correct answers increased from 27 in the first attempt to 31 in the second and 32 in the third one. With iterative feedback prompts, the correct answers increased from 33 in the first attempt to 41 in the second and 42 in the third one.

Table 9
Performance of fine-tuned GPT-4o-mini using simple retries versus iterative feedback.

Model	Attempt 1	Attempt 2	Attempt 3
GPT-4o-mini (one-shot) using simple retry	27 (28%)	31 (32%) (+4)	32 (33%) (+1)
GPT-4o-mini (one-shot) using iterative feedback	33 (34%)	41 (42%) (+8)	42 (43%) (+1)

The iterative feedback with updated prompts based on LLM previous errors demonstrates a more substantial improvement over simple retries. This indicates that dynamic prompt adjustment is key in maximizing the model's potential for complex tasks. Using simple retries only resulted in adding 4 more ”Correct” Aspect Models, however, with iterative prompting, it increased to 8 in the second attempt. The final result has 10 more ”Correct” Aspect Models. This indicates that investing time on generating proper exceptions and guidelines for the model would be a rewarding path.

Experiment 8: Comparison of Qwen2.5-Coder Base and Fine-Tuned Models

This experiment compares the performance of the Qwen2.5-Coder 7B base model and its fine-tuned counterpart across zero-shot, one-shot, and two-shot prompting. The fine-tuning process was expected to enhance the model's ability to align with SAMM constraints and generate structurally correct outputs.

The results for one-shot and two-shot for fine-tuned Qwen2.5-Coder are described in Table 10.

Table 10
Performance comparison of Qwen2.5-Coder 7B base and fine-tuned models.

Model	Valid Turtle	Valid SAMM	Correct
Base Qwen2.5-Coder 7B (zero-shot)	40 (41%)	0	0
Fine-tuned Qwen2.5-Coder 7B (zero-shot)	53 (55%)	5 (5%)	1 (1%)
Base Qwen2.5-Coder 7B (one-shot)	39 (40%)	0	0
Fine-tuned Qwen2.5-Coder 7B (one-shot)	43 (44%)	10 (10%)	6 (6%)
Base Qwen2.5-Coder 7B (two-shot)	38 (39%)	12 (12%)	3 (3%)
Fine-tuned Qwen2.5-Coder 7B (two-shot)	38 (39%)	6 (6%)	2 (2%)

The results in Table 10 indicate that fine-tuning improves the model's performance across most metrics but shows varying degrees of effectiveness depending on the prompting strategy. For zero-shot prompting, the fine-tuned model is able to create 1 ”Correct” Aspect Model. These results suggest that fine-tuning helps the model better adhere to the SAMM constraints even without example guidance. Similarly, in one-shot prompting, the fine-tuned model shows improvement over the base model. This highlights that fine-tuning enhances the model's ability to generalize from a single example. For two-shot prompting, the fine-tuned model performs similarly to the base model. This suggests that additional examples in the prompt may offer limited benefit as the fine-tuned model is already trained on task-specific patterns. Too many examples can introduce redundancy or complexity, leading to overfitting and reduced performance on new inputs.

Human Evaluation

The evaluation was conducted with six experienced modelers who provided feedback on the SAMM Copilot tool. Table 11 summarizes the quantitative and qualitative responses collected.

Participant	Manual Modeling Time	SAMM Copilot Time	Satisfaction with SAMM Copilot
1	11 minutes	2 minutes	Yes, valid and complete model
2	7 minutes	>10 minutes	No, invalid model
3	5 minutes	8 minutes	Yes, valid but incomplete model
4	7 minutes	4 minutes	Yes, valid but incomplete model
5	60 minutes	45 minutes	No, valid but incorrect model
6	10 minutes	Less than 3 minutes	No, invalid model

Table 11
Summary of human evaluation results.

The evaluation results reveal valuable insights into the strengths and limitations of SAMM Copilot. One of the most significant advantages of SAMM Copilot is its ability to reduce the time required for modeling compared to manual methods. For most participants, the tool enabled faster model generation, with times ranging from 2 to 8 minutes, as opposed to 5 to 60 minutes for manual modeling. Despite its potential for efficiency, the validity and completeness of the generated Aspect Models were inconsistent. While one participant was satisfied with the generated Aspect Models, finding them both valid and complete, others identified significant shortcomings. Two participants noted that the models were valid but incomplete, requiring additional manual effort to meet their expectations. Furthermore, two participants considered the models either invalid or incorrect, highlighting a critical limitation in SAMM Copilot's ability to generate reliable outputs consistently. Participants also provided constructive feedback for future improvements. Many appreciated the detailed descriptions generated by SAMM Copilot, which they found useful compared to the brief descriptions typically added during manual modeling. Overall, while SAMM Copilot demonstrates clear potential to streamline the modeling process and reduce manual effort, its current limitations hinder its reliability and usability. Addressing the issues of model validity, error handling, and integration will be critical to improving the tool and ensuring broader user satisfaction and adoption.

5. Conclusion and Future Work

Our work examined the potential of automating the creation of semantic models by transforming JSON data into SAMM Aspect Models using LLMs. Both commercial and open-source LLMs were analyzed, and evaluation methods were introduced to measure their effectiveness in various setups. In addition to automated evaluations, human evaluations of the generated outputs were conducted. These evaluations demonstrated a significant improvement in efficiency, with the process achieving speeds up to four times faster than manual modeling. One of the main contributions of this work was the collection and curation of a dataset for fine-tuning, which is openly available. This work also outlined how LLMs can be used by end-users and integrated with existing tools for generating semantic models.

We addressed several research questions through experiments and analyses, as outlined below:

- **RQ1: The ability to automatically derive basic Aspect Model structures from JSON data.** The capability of LLMs to generate basic Aspect Model structures from JSON data was successfully demonstrated through multiple experiments. Human evaluation further confirmed the LLMs' success in fulfilling this task, providing evidence that bottom-up semantic modeling with LLMs is both feasible and efficient.
- **RQ2: Comparative analysis of open-source and commercial LLMs.** A comparative analysis revealed key differences in performance between open-source and commercial LLMs. Open-source models, while advantageous due to lower costs, enhanced privacy, and local usage, struggled

with the complexity of generating semantic models without fine-tuning. On the other hand, commercial models like GPT-4o-mini demonstrated superior performance, even in their baseline state, effectively handling complex tasks. This analysis underscored the importance of model size and sophistication in achieving high-quality results.

- **RQ3: Implementation of automatic evaluation methods.** An automatic evaluation pipeline was developed to ensure the accuracy and consistency of LLM outputs. This pipeline included validation of RDF Turtle syntax using Apache Jena, checking SAMM Aspect Model conformity with the ESMF SDK, and assessing JSON structural similarity between input and generated Aspect Models. These methods provided a robust framework for evaluating correctness, although assessing completeness remains a subjective task requiring expert judgment. This framework ensures that outputs align with expected standards, facilitating reliable automation of semantic modeling.
- **RQ4: Techniques to improve accuracy of LLMs in deriving Aspect Models.** Data augmentation techniques were employed to simplify Aspect Models and make minor modifications to JSON payload values, enabling easier processing by LLMs. While these augmentations did not significantly enhance the LLMs' learning capabilities, they proved to be valuable for evaluation. Simplified models were easier to process and served as effective inputs for testing iterative feedback mechanisms. Refining model outputs through successive prompts improved accuracy, showing their value in generating correct Aspect Models.
- **RQ5: Integration of trained LLM models in practical tools.** The integration of LLMs into practical tools is crucial for real-world adoption. This research showcased how LLMs can be accessed through diverse interfaces, including chat-based systems and visual modeling tools. Prototypes and approaches for the integration into tools such as the Aspect Model Editor, illustrating how LLMs can be seamlessly integrated into user workflows.

It was indicated by this work that the performance of small open-source models was not promising for the tasks at hand. One aspect is due to the small amount of available training data. To address this limitation, model distillation techniques could be explored. These techniques involve generating synthetic data using larger, more capable LLMs, such as GPT-4o, and then using this synthetic data to train smaller models. Qwen2.5-Coder was identified as the most capable open-source model among those tested.

Models like DeepSeekCoder were not evaluated, as they were unavailable for fine-tuning within the Unsloth library. In the future, as more advanced models become compatible with libraries like Unsloth, they can be included in similar studies to enhance outcomes and broaden the scope of comparisons. Due to a lack of resources and time, open-source models with more parameters were not fine-tuned.

In addition, the human evaluation did not focus on an in-depth comparison of the quality of outputs generated by different LLMs. In future studies, once more stable models are available, a comprehensive evaluation of the results across various models could yield valuable insights.

Another limitation of the proposed workflow is that, in some cases, a single JSON example may not be sufficient to represent the full range of data, as the structure of the data can vary. In such scenarios, multiple examples may be required to capture the diversity of the data. More targeted and focused evaluations were not conducted to determine whether the approach would work effectively in these cases. However, in general, LLMs are capable of understanding and processing the structure of various JSON formats. For the dataset, it would be necessary to develop a method for generating multiple variants of JSON payloads to account for different data structures. This would help to ensure that the LLMs are exposed to a broader range of examples, improving their ability to generate accurate Aspect Models for more diverse datasets.

The scope of commercial LLMs in this study was limited to OpenAI's GPT models due to budget constraints. Future research could extend the analysis to other providers, such as Claude (Anthropic) and Google Gemini. Additionally, a comprehensive exploration of hyperparameters, such as the learning rate multiplier, could be addressed in subsequent studies.

References

[1] M. Grieves, Digital twin: manufacturing excellence through virtual factory replication, White paper 1 (2014) 1–7.

[2] S. Z. Phua, M. Hofmeister, Y.-K. Tsai, O. Peppard, K. F. Lee, S. Courtney, S. Mosbach, J. Akroyd, M. Kraft, Fostering urban resilience and accessibility in cities: A dynamic knowledge graph approach, Sustainable Cities and Society 113 (2024) 105708.

[3] F. Sarani Rad, R. Hendawi, X. Yang, J. Li, Personalized diabetes management with digital twins: A patient-centric knowledge graph approach, Journal of Personalized Medicine 14 (2024) 359.

[4] M. Kümpel, C. A. Mueller, M. Beetz, Semantic digital twins for retail logistics, in: Dynamics in logistics: Twenty-five years of interdisciplinary logistics research in Bremen, Germany, Springer International Publishing Cham, 2021, pp. 129–153.

[5] D. Piromalis, A. Kantaros, Digital twins in the automotive industry: The road toward physical-digital convergence, Applied System Innovation 5 (2022) 65.

[6] A. Textor, S. Stadtmüller, B. Boss, J. Kristan, Bamm aspect meta model., in: ISWC (Posters/Demos/Industry), 2021.

[7] H. Babaei Giglou, J. D'Souza, S. Auer, Llms4ol: Large language models for ontology learning, in: International Semantic Web Conference, Springer, 2023, pp. 408–427.

[8] N. Fathallah, A. Das, S. De Giorgis, A. Poltronieri, P. Haase, L. Kovriguina, Neon-gpt: A large language model-powered pipeline for ontology learning, The Semantic Web: ESWC Satellite Events 2024 (2024).

[9] M. C. Suárez-Figueroa, A. Gómez-Pérez, M. Fernandez-Lopez, The neon methodology framework: A scenario-based methodology for ontology development, Applied ontology 10 (2015) 107–145.

[10] L. M. V. da Silva, A. Kocher, F. Gehlhoff, A. Fay, On the use of large language models to generate capability ontologies, in: 2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2024, pp. 1–8.

[11] M. J. Mior, Large language models for json schema discovery, arXiv preprint arXiv:2407.03286 (2024).

[12] M.-A. Baazizi, H. B. Lahmar, D. Colazzo, G. Ghelli, C. Sartiani, Schema inference for massive json datasets, in: Extending Database Technology (EDBT), 2017.

[13] H. Ghanem, C. Cruz, Fine-tuning vs. prompting: evaluating the knowledge graph construction with llms, in: 3rd International Workshop on Knowledge Graph Generation from Text (Text2KG) Co-located with the Extended Semantic Web Conference (ESWC 2024), volume 3747, 2024, p. 7.

[14] M. H. Daniel Han, U. team, Unsloth, 2023. URL: http://github.com/unslothai/unsloth.

[15] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., Llama: Open and efficient foundation language models, arXiv preprint arXiv:2302.13971 (2023).

[16] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, et al., Qwen2. 5-coder technical report, arXiv preprint arXiv:2409.12186 (2024).

[17] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, et al., Code llama: Open foundation models for code, arXiv preprint arXiv:2308.12950 (2023).