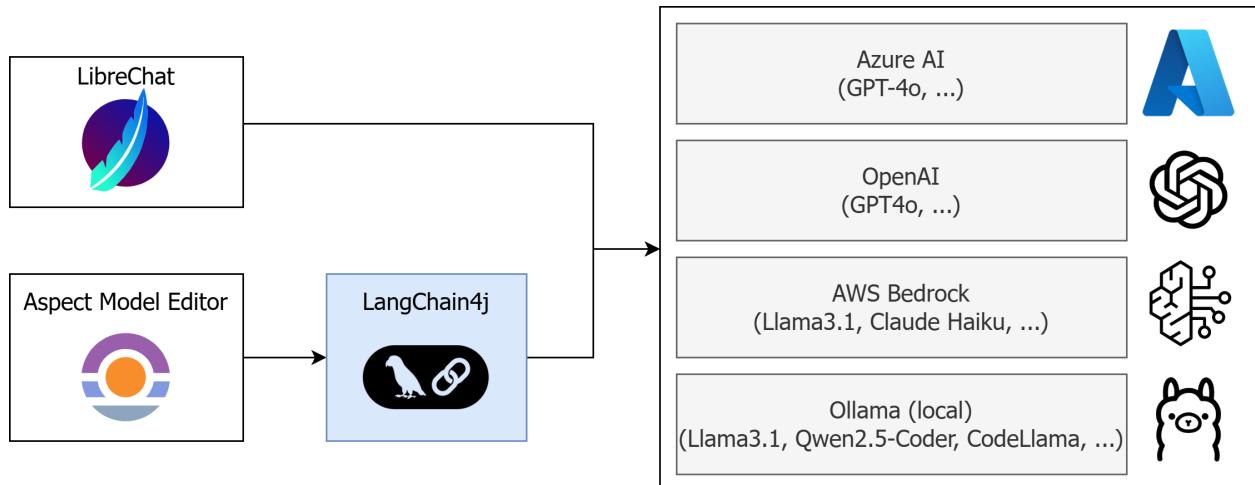


Master's Thesis

From Data to Semantics: Automated Aspect Model Creation and Knowledge Graph Integration for Digital Twins



Author: Nazanin Mashhaditafreshi
Matriculation Number: 419282
Reviewer 1: Dr.-Ing. habil. Achim Wagner
Reviewer 2: M.Sc. Pascal Rübel
Reviewer 3: Dr. Andreas Textor
Reviewer 4: M.Sc. Nastaran Moarefvand
Number of Thesis: 2024-36
Date: 03.02.2025

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und ohne unerlaubte, fremde Hilfe angefertigt habe. Alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß übernommen wurden, sind kenntlich gemacht. Alle verwendeten Quellen sind im Literaturverzeichnis zitiert.

Kaiserslautern, den 03.02.2025

Contents

Erklärung

Abstract	IV
-----------------	-----------

Kurzfassung	V
--------------------	----------

List of Abbreviations	VI
------------------------------	-----------

List of Figures	VII
------------------------	------------

List of Tables	IX
-----------------------	-----------

List of Code Snippets	X
------------------------------	----------

1 Introduction	1
-----------------------	----------

1.1 Motivation	2
1.2 General Objectives	3
1.3 Research Questions	3
1.4 Contributions	3
1.5 Structure of the Thesis	4

2 Background	5
---------------------	----------

2.1 Digitalization	5
2.2 Knowledge Graphs	6
2.3 Interoperability	7
2.4 Modeling Digital Twins with AAS and SAMM	10
2.5 Generative AI	14

3 Literature Review	19
----------------------------	-----------

3.1 LLMs for AAS	19
3.2 LLMs for Semantic Engineering	21
3.3 LLMs for Software Engineering	23
3.4 KGs for AAS and SAMM	24
3.5 Summary	25

4 Methodology	27
4.1 Data Collection, Preparation, and Augmentation	28
4.2 LLM Selection, Configuration, and Benchmarking	30
4.3 Experiments Setup	32
4.4 Integration and Deployment	34
4.5 Human Evaluation	35
4.6 Summary	35
5 Implementation	36
5.1 Data Collection and Preparation	36
5.2 Synthetic Data and Data Augmentation	38
5.3 Data Splitting	40
5.4 Input Data Format	41
5.5 Cloud Fine-Tuning	42
5.6 Local Fine-Tuning	44
5.7 LLM Evaluation	45
5.8 Prompting Techniques	45
5.9 Human Evaluation	47
5.10 Software Tools and Libraries	48
5.11 AAS and SAMM as RDF-based Graph Implementation	49
5.12 End-Solution Deployment	49
5.13 Summary	50
6 Results	53
6.1 Fine-Tuning Statistics	53
6.2 Experiments Result	60
6.3 Human Evaluation	64
7 Discussion	67
7.1 Fine-Tuning Insights	67
7.2 Experiments	67
7.3 Human Evaluation	70
7.4 Ethical Considerations	70
8 Conclusion and Future Work	72
A1 Prompt Guidelines	75
A2 Example Outputs	87
A2.1 GPT-4o-mini	88

<i>Contents</i>	III
A2.2 Fine-tuned Qwen2.5-Coder	89
A2.3 Fine-tuned GPT-4o-mini	92
Bibliography	96

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). Using existing tooling, Aspect Models can be translated into Asset Administration Shell (AAS) Submodel Templates and vice versa. Furthermore, it is possible to use Aspect Models as AAS Concept Descriptions to further describe the semantics of parts of (or whole) AAS Submodels or Submodel Templates.

With SAMM being based on 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), which are then used to generate AAS Submodel Templates. SAMM and AAS are represented within an RDF-based knowledge graph, ensuring interoperability and semantic richness in the creation of digital twins. This approach supports the seamless integration of data and enables the construction of models that are both interoperable and semantically well-defined.

This work lays the foundation for improving LLM-enhanced semantic modeling. A key contribution of this work is the creation and curation of a novel dataset, including synthetically generated Aspect Models, to support the training and evaluation of LLMs. By leveraging both commercial and open-source LLMs, baseline performance metrics are established, demonstrating significant improvements in modeling efficiency, achieving up to four times faster than manual workflows. The findings show that LLMs can be useful for semantic modeling in industrial digital twins and provide guidance on how to integrate them with semantic tools like the Aspect Model Editor.

Kurzfassung

Das Semantic Aspect Meta Model (SAMM) ist ein Modellierungsformalismus zur Beschreibung semantischer Modelle von Teilen eines Digitalen Zwillings – sogenannten Aspect Models. Es handelt sich um eine offene Spezifikation, die im Rahmen des Eclipse Semantic Modeling Framework (ESMF) entwickelt wurde. Mithilfe vorhandener Werkzeuge können Aspect Model in Asset Administration Shell (AAS) Submodel Templates und umgekehrt übersetzt werden. Darüber hinaus ist es möglich, Aspect Model als AAS Concept Descriptions zu verwenden, um die Semantik von Teilen (oder des gesamten) AAS Submodels oder der Submodel Templates weiter zu beschreiben.

Da SAMM auf dem Resource Description Framework (RDF) und der Shapes Constraint Language (SHACL) basiert, werden Aspect Model in der Regel manuell mit einem geeigneten Texteditor oder dem grafischen Aspect Model Editor erstellt und bearbeitet. Es besteht eine klar definierte Zuordnung zwischen Aspect Models und den JSON-Daten, die sie beschreiben, was neue Bottom-up-Modellierungsansätze ermöglicht. Auf diese Weise können Aspect Models anstelle eines manuellen Prozesses für die semantische Modellierung automatisch oder halbautomatisch aus vorhandenen Domain-Daten im JSON-Format abgeleitet werden, was den Modellierungsprozess leichter zugänglich macht und den manuellen Aufwand reduziert. Der vorgeschlagene Arbeitsablauf überetzt JSON-Daten automatisch in Aspect Model mithilfe von Large Language Models (LLMs), die dann zur Generierung von AAS Submodel Templates verwendet werden. SAMM und AAS werden innerhalb eines RDF-basierten Wissensgraphs dargestellt, was Interoperabilität und semantische Tiefe bei der Erstellung von digitalen Zwillingen gewährleistet. Dieser Ansatz unterstützt die nahtlose Integration von Daten und ermöglicht die Erstellung von Modellen, die sowohl interoperabel als auch semantisch klar definiert sind.

Diese Arbeit legt die Grundlage für die Verbesserung der durch LLMs unterstützten semantischen Modellierung. Ein wichtiger Beitrag dieser Arbeit ist die Erstellung und Pflege eines neuartigen Datensatzes, einschließlich synthetisch generierter Aspect Model, zur Unterstützung des Trainings und der Bewertung von LLMs. Durch die Nutzung sowohl kommerzieller als auch Open-Source-LLMs werden Baseline-Performance-Metriken festgelegt, die eine signifikante Verbesserung der Modellierungseffizienz zeigen und eine bis zu viermal schnellere Leistung als manuelle Arbeitsabläufe erreichen. Die Ergebnisse zeigen, dass LLMs für die semantische Modellierung in industriellen digitalen Zwillingen nützlich sein können und bieten eine Anleitung, wie sie mit semantischen Werkzeugen wie dem Aspektmodell-Editor integriert werden können.

List of Abbreviations

AAS Asset Administration Shell

ABox Assertion Box

API Application Programming Interface

AWS Amazon Web Services

CDD Common Data Dictionary

CRUD Create, Read, Update, and Delete

EU European Union

DT Digital Twin

ESMF Eclipse Semantic Modeling Framework

FAIR Findability, Accessibility, Interoperability, and Reusability

GPT Generative Pre-trained Transformers

HTTP Hypertext Transfer Protocol

IDTA Industrial Digital Twin Association

IRI Internationalized Resource Identifier

IRDI International Registration Data Identifier

JSON JavaScript Object Notation

KG Knowledge Graph

LLM Large Language Model

LoRA Low-Rank Adaptation

OMP Open Manufacturing Platform

OWL Web Ontology Language

PEFT Parameter-Efficient Fine-Tuning

RAG Retrieval Augmented Generation

RDB Relational Database

RDF Resource Description Framework

RDFS RDF Schema

RDBMS Relational Database Management System

REST API Representational State Transfer API

SAMM Semantic Aspect Meta Model

SDK Software Development Kit

SHACL Shapes Constraint Language

SMC Submodel Element Collection

SMT Submodel Template

SKOS Simple Knowledge Organization System

SPARQL Simple Protocol And RDF Query Language

SFT Supervised Fine-Tuning

TBox Terminology Box

TDB Triplestore Database

UI User Interface

URI Uniform Resource Identifier

URN Uniform Resource Name

XSD XML Schema Definition

XML Extensible Markup Language

List of Figures

2.1	Various kinds of digital representation of an asset, and how physical object and digital object interact with each other.	6
2.2	Representing an entity in RDF model in the left and PG model in the right.	7
2.3	W3C Technology Stack (source: [GÄNG11])	8
2.4	Layers of interoperability	8
2.5	Standardization within industries, and mediation as a way to provide interoperability between them.	10
2.6	Structure of Asset Administration Shell.	11
2.7	Building blocks of SAMM (source: [ECLI])	13
2.8	Data and Aspect Model for a Bosch Smart Plug.	14
2.9	Architecture of transformer-based models (source: [VASW23])	15
3.1	Structure of the semantic node as an intermediate unit in AASbyLLM application (source: [XIA24]).	20
3.2	The workflow of the NeOn-GPT ontology engineering (source: [FATH24]).	22
3.3	Sample of the one-shot prompt template for capability ontology generation (source: [SILV24]).	23
4.1	General workflow of the proposed solution.	28
4.2	High-level steps of this research.	28
4.3	Semantic models in Tractus-x project.	29
5.1	Data cleanup steps.	36
5.2	Data augmentation example reflected on JSON payload.	39
5.3	Cleanup Aspect Model when element B removed from the graph.	39

5.4	Data augmentation for generating new Aspect Models.	40
5.5	OpenAI's dashboard to create a fine-tuning job.	43
5.6	Folder structure to store all the attempts made.	46
5.7	An example of iterative prompting.	46
5.8	SAMM Copilot platform user interface.	50
5.9	Overall architecture and components of the end-solution.	51
5.10	Bootstrap Aspect Model from JSON example in the Aspect Model Editor.	51
5.11	Enhance an element of the Aspect Model by AI using natural language specification.	52
6.1	Job details for fine-tuning GPT-4o-mini on the original dataset.	54
6.2	Training and validation losses for the fine-tuned GPT-4o-mini on the original dataset.	54
6.3	Job details for fine-tuning GPT-4o-mini on the augmented dataset.	55
6.4	Training and validation losses for the fine-tuned GPT-4o-mini model trained on the augmented dataset.	55
6.5	Job details for fine-tuning GPT-4o on the original dataset.	56
6.6	Training and validation losses for the fine-tuned GPT-4o trained on the original dataset.	56
6.7	Details of fine-tuning GPT-4o-mini on the original dataset with batch size of 2. .	57
6.8	Training and validation losses for fine-tuning GPT-4o-mini using original dataset with batch size of 2.	57
6.9	Job details of fine-tuning GPT-4o-mini on the original dataset with batch size of 8. .	58
6.10	Training and validation losses for the fine-tuned GPT-4o-mini using original dataset with batch size of 8.	58
6.11	Training and validation losses for the fine-tuned Qwen2.5-Coder trained on the original dataset.	59
6.12	Comparison of time taken for manual modeling and using SAMM Copilot.	64

List of Tables

2.1	Inference Parameters Comparison for LLMs.	17
2.2	Top 25 LLMs on BigCodeBench Leaderboard [TEAM24].	18
3.1	Summary of related work.	26
4.1	The size of the training, validation, and test sets.	30
6.1	Effect of temperature on the performance of fine-tuned GPT-4o-mini model (one-shot prompting) trained on the original data.	60
6.2	Effect of example complexity on the performance of fine-tuned GPT-4o-mini in one-shot prompting after three attempts (pass@3).	61
6.3	Effect of multiple attempts on the performance of the fine-tuned GPT-4o-mini (pass@3 metric) in one-shot prompting.	61
6.4	Performance of open-source models without fine-tuning (one-shot prompting).	62
6.5	Comparison of one-shot and two-shot prompting on the Qwen2.5-Coder and Llama 3.1 base models.	62
6.6	Effect of batch size on model performance with one-shot prompting trained on original data.	63
6.7	Performance of GPT-4o-mini under various prompting and fine-tuning scenarios.	63
6.8	Performance of fine-tuned GPT-4o-mini using simple retries versus iterative feedback.	64
6.9	Performance comparison of Qwen2.5-Coder 7B base and fine-tuned models.	65
6.10	GPU memory consumption of Qwen2.5-Coder 7B for varying context window sizes.	65
6.11	Summary of the performance of different LLMs.	65
6.12	Summary of human evaluation results.	66

List of Code Snippets

4.1 Example of domain data in JSON format.	27
5.1 Example of an Aspect Model.	37
5.2 Example of a JSONL file format.	42
5.3 Zero-shot prompt template.	45
5.4 One-shot prompt template.	47
5.5 Two-shot prompt template.	47
5.6 Prompt template with feedbacks.	47
6.1 Comparison of the two generated elements using different temperatures.	60
A1.1 Hints inserted in the prompt when occurred exception in previous attempt was related to RDF.	75
A1.2 Hints inserted in the prompt when occurred exception in previous attempt was related to JSON.	77
A1.3 Generic hints inserted in the prompt to help the model correct the output.	81
A2.1 Zero-shot prompt template for one example	87
A2.2 One-shot prompt template for one example	87
A2.3 Output of GPT-4o-mini after applying zero-shot prompt template for one example	88
A2.4 Output of GPT-4o-mini after applying one-shot prompt template for one example	88
A2.5 Output of fine-tuned Qwen2.5-Coder after applying zero-shot prompt template for one example	89
A2.6 Output of fine-tuned Qwen2.5-Coder after applying one-shot prompt template for one example	91
A2.7 Output of fine-tuned GPT-4o-mini after applying zero-shot prompt template for one example	92
A2.8 Output of fine-tuned GPT-4o-mini after applying one-shot prompt template for one example	94

1 Introduction

In recent years, the concept of digital twin [GRIE14] 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 [PHUA24, SARA24, KÜMP21, PIRO22].

Digital twins and semantic models work together to enable smarter, more connected systems. By providing a common language, semantic models help digital twins integrate seamlessly with other tools and systems, facilitating collaboration. This connection is especially crucial in industries like manufacturing, healthcare, and logistics, where clear communication and consistent data sharing among partners, software vendors, and stakeholders are essential.

Semantic models serve a vital role in enabling interoperability across systems and domains. By providing a shared framework for understanding and exchanging information, they allow diverse systems to communicate seamlessly. These models are often based on frameworks like the Resource Description Framework (RDF) and Web Ontology Language (OWL), ensuring meaningful and machine-readable data exchange by defining the relationships, properties, and concepts within a specific domain.

Large Language Models (LLMs) have seen rapid adoption in recent years due to their ability to generate human-like text and assist with a variety of tasks. However, a persistent challenge with LLMs is their tendency to hallucinate outputs that are plausible-sounding but factually incorrect or inaccurate. To address this, semantic models and ontologies can play a critical role. By linking the outputs of LLMs to structured knowledge and predefined relationships, these models help ensure that generated information is accurate and aligned with domain-specific truths. Integrating semantics into LLMs not only helps mitigate hallucinations but also enhances their reliability in specialized domains.

However, the creation and maintenance of semantic models can be time-consuming, requiring manual effort from domain experts. To address this bottleneck, LLMs themselves can be employed to automate repetitive tasks in semantic modeling. By leveraging LLMs in this way, the process becomes faster and more accessible, allowing experts to concentrate on higher-level conceptual

work rather than spending time building a semantic model from scratch.

This thesis explores the intersection of digital twins, semantic modeling, and LLMs to explore how these technologies and concepts can be combined effectively. By leveraging the generative capabilities of LLMs, this work seeks to assist domain experts in minimizing the effort spent on repetitive tasks, enabling them to focus on the more creative and intellectual aspects of their work.

1.1 Motivation

The manual process of creating semantic models requires modeling knowledge as well as domain expertise. Apart from that, such a process is usually time-consuming. LLMs offer a promising opportunity to streamline this process. With their proven ability to generate structured text and artifacts, LLMs can assist in automating repetitive aspects of semantic modeling, reducing the burden on domain experts. This allows experts to concentrate on higher-level design, ensuring models are more robust, reusable, and aligned with best practices.

In addition to simplifying the creation process, integrating data and their semantic models based on established standards, such as the Asset Administration Shell (AAS) and the Semantic Aspect Meta Model (SAMM), creates interoperable knowledge graphs that unlock new possibilities for seamless data exchange and collaboration. SAMM is developed and specified within the Eclipse Semantic Modeling Framework (ESMF) project.

AAS and SAMM are considered as solutions to interoperability challenges. AAS, defined in IEC 63278-1:2023, is a standardized framework for creating digital twins of industrial assets. It provides a metamodel with key elements like *AssetAdministrationShell*, *Submodel*, and *ConceptDescription*, enabling assets to be uniquely identified and described. *Submodel* encapsulates aspect-specific data, while semantic annotations like *semanticId* and *ConceptDescription* ensure interoperability and machine-readable interpretations.

SAMM is a semantic modeling framework that describes the semantics and structure of data through core components such as *Aspect*, *Properties*, *Operations*, and *Events*. It provides mechanisms for contextualizing and constraining data using *Characteristics* and *Constraints*, facilitating precise and interoperable data exchange accompanied by robust tooling.

Both AAS and SAMM have RDF representations, enabling the creation of semantically rich and interoperable knowledge graphs. SAMM Aspect models can guide the generation of AAS Submodel Templates, while bidirectional mapping ensures compatibility between the two. This integrated approach provides a foundation for advanced applications, such as question answering powered by Graph-based Retrieval Augmented Generation (GraphRAG) techniques, as well as other future possibilities.

1.2 General Objectives

The general objective of this research is to explore how LLMs can be effectively used to facilitate the creation of semantic models. Specifically, this work aims to:

- Automate repetitive tasks in semantic modeling to ease the process for domain experts.
- Investigate methods to integrate LLMs with existing standards, such as SAMM and AAS, for creating interoperable and reusable semantic representations.
- Develop an interoperable knowledge graph based on established frameworks to improve data consistency and enable better collaboration across systems.

Through this approach, the research seeks to demonstrate the potential of LLMs in improving the efficiency, ease of use, and simplicity of semantic modeling in modern applications.

1.3 Research Questions

The goal of this thesis is to examine how the basic structure, and possibly more detailed semantic descriptions, of an Aspect Model can be automatically or semi-automatically derived from existing domain data in the JSON format. This thesis aims to address the following research 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?
- **RQ6:** How can SAMM and AAS be integrated to form a Knowledge Graph-based digital twin for enhanced data interoperability and decision-making?

1.4 Contributions

This thesis makes the following key contributions:

1. **Introduction of a Novel Dataset:** A new, curated dataset is developed specifically for

training and fine-tuning LLMs. This dataset also serves as a benchmark for evaluating LLMs performance, establishing a foundation for future research in this area.

2. **Development of Evaluation Methods:** Various evaluation methods are introduced, providing a framework to compare the performance of models comprehensively.
3. **Baseline Establishment:** Several state-of-the-art LLMs are fine-tuned and evaluated against the new test set. These evaluations establish baseline performance metrics for future research.
4. **Exploration of Advanced Prompting Techniques:** This work investigates various prompting methods, including role-playing, few-shot prompting, and iterative feedback techniques, to enhance the performance and usability of LLMs in deriving Aspect Models from JSON data.
5. **Integration Insights for LLMs-Enhanced Semantic Modeling:** This thesis explores how LLMs can be integrated into end-to-end solutions, such as the Aspect Model Editor used by semantic modelers, providing insights into practical applications.

All the code, datasets, and artifacts are available at *SAMMCopilot* GitHub repository¹, which can help reproduce the results of this work.

1.5 Structure of the Thesis

The rest of the thesis is organized into seven main chapters. Chapter 2 establishes the foundational context for the research, introducing key concepts and defining the scope of the study. Chapter 3 presents a comprehensive review of the existing literature, highlighting challenges and identifying gaps. Chapter 4 details the methods and approaches employed in the study, including the experimental setup, data collection, and analysis techniques. Chapter 5 focuses on the implementation aspects. In Chapter 6, the research findings are presented, followed by an analysis in Chapter 7 that explores their significance and implications. Finally, Chapter 8 concludes the thesis by summarizing the main contributions, reflecting on the outcomes, and suggesting potential directions for future research.

¹<https://github.com/NazaninTafreshi/sammcopilot>

2 Background

The goal of this chapter is to build a common understanding about the related topics in this thesis and it starts with an explanation of digital twins and a few related terms. Knowledge graphs and semantic models are then introduced, followed by a discussion on the significance of standardization in achieving interoperability. Key concepts like the Asset Administration Shell (AAS), Eclipse Semantic Modeling Framework (ESMF), and associated tools will be explored. Finally, the chapter introduces Generative AI and Large Language Models (LLMs).

2.1 Digitalization

Michael Grieves [GRIE14] defined digital twin (DT) as a digital representation of a physical product, connected by a flow of data between the real and virtual world. Over time, the terminology evolved and diverse definitions and applications of digital twins have led to its use as an umbrella term, including a range of related concepts and technologies, which are depicted in Figure 2.1.

- **Digital Model:** A foundational element of a digital twin, representing the structure, properties, and behavior of the physical asset in a digital format, which might also come with a realistic visualizations and simulations [PIRE19].
- **Digital Shadow:** A digital representation that receives a one-way data flow from the physical asset, and might incorporate various data sources, providing a real-time but passive view of the asset's state [BAUE16].
- **Digital Thread:** A concept that emphasizes the continuous flow of data and information throughout the lifecycle of a physical asset, connecting all stages from design and manufacturing to operation and maintenance [HELU15].

Muctadir et al. [MUCT24] conducted semi-structured interviews with 19 professionals from industry and academia to understand how they define, develop, and validate DTs. The interviews revealed a lack of uniformity in how practitioners understand DTs, and their components.

Nevertheless, DTs, as an umbrella term, provide real-time insights, support data-driven decision-making, enable virtual experimentation, enhance collaboration, and drive innovation. DTs applied in various domains and verticals. For example, in manufacturing, they are used to optimize production processes, monitor equipment performance, predict maintenance needs, and facilitate

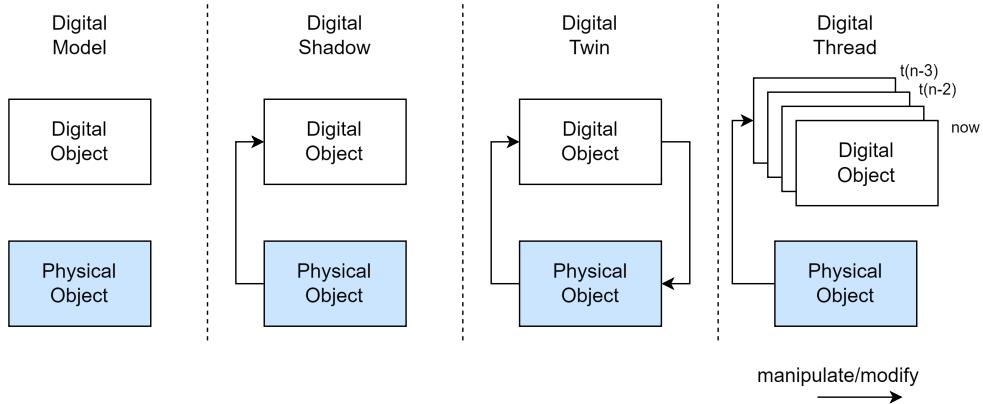


Figure 2.1: Various kinds of digital representation of an asset, and how physical object and digital object interact with each other.

remote troubleshooting and fault diagnosis [SOOR23, RÜBE23]. Digital twins of cities and their infrastructure help in planning, managing resources, and responding to events [PHUA24]. Also in healthcare, digital twins of patients and medical devices contribute to personalized treatments and remote monitoring [SARA24]. In retail and logistics, they aid in optimizing store layouts, managing inventory, and personalizing customer experiences [KÜMP21]. In automotive industry, DTs enhance the design, development, production, and maintenance of vehicles [PIRO22].

2.2 Knowledge Graphs

In the early 2000s, as the World Wide Web grew with mostly human-readable content, Tim Berners-Lee and his colleagues envisioned the so-called Semantic Web [LASS01]. In their vision, content on the internet would become machine-understandable, with data linked through Unique Resource Identifiers (URIs) to create a web of knowledge.

There are two data models to construct a knowledge graph: Resource Description Framework (RDF) and Property Graph (PG) which is also called Labeled Property Graph (LPG). In RDF, everything is represented by three components: Subject, Predicate, and Object. The Subject and Object are nodes in the graph, while the Predicate is the edge connecting them. To model a person with a name, two triples are needed: one to specify that the resource is a person, and another to link the person to a name with a specific literal. This can be visualized as a node labeled "person" with a "name" property, as shown in Figure 2.2.

Today, when Knowledge Graphs are discussed, Neo4j and other graph technologies, which are not necessarily based on RDF, are often mentioned. Specifically, in the context of Semantic Web literature, using PGs may not be considered a semantic approach [ALLE]. This might be due to the fact that RDF side is much more standardized and well-formed vocabularies and constructs, like Web Ontology Language (OWL), make it easier to create interoperable semantic models. However, the PG model is as expressive as RDF and often offers a more intuitive way to represent

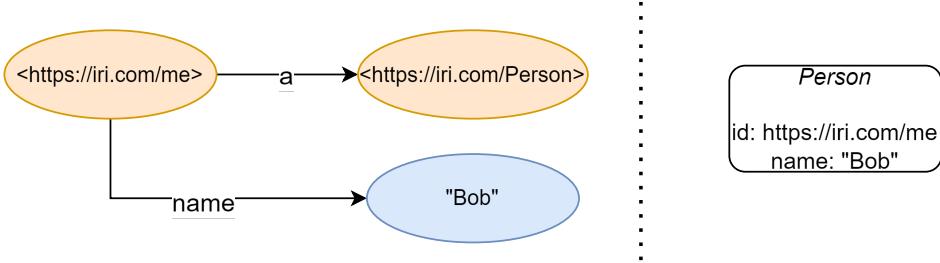


Figure 2.2: Representing an entity in RDF model in the left and PG model in the right.

graph data.

W3C as an standardization entity, plays a big role on the RDF side. Figure 2.3 depicts the Semantic Web Stack, with a lot of standardized components by W3C. Various standard serialization formats, such as Turtle, RDF/XML, N-Triples, etc., are used to ensure structural interoperability. Moreover, Shapes Constraint Language (SHACL) offers a flexible way to make sure that data is always in a valid state according to custom business logic. Also using RDFS and OWL construct, modelers are able to explicitly create complex and semantically rich ontologies that define relationships, classes, and properties within a domain. SPARQL Protocol and RDF Query Language (SPARQL) is the standardized query language across RDF databases, with full vendor support. Meanwhile, the PG community is advancing, with ISO GQL [INTE24] emerging as a standard query language. However, PGs still need more space to cover in terms of broader standardization.

2.3 Interoperability

Interoperability is the ability of two or more different systems, devices, or applications, to connect, communicate, and exchange services and data with one another, and be able to use the exchanged information meaningfully. Therefore, interoperability leads to an overall performance improvement and enables integrated solutions across various technological platforms.

Interoperability can be categorized into different aspects, as discussed by Santos et al. [SANT21]. In this work, it is viewed at three levels: technical, syntactic or structural, and semantic, as depicted in Figure 2.4.

Technical interoperability forms the base of the interoperability pyramid. It includes foundational standards, ranging from basic ones like IEEE 754 for floating-point representation to widely used communication protocols such as TCP, HTTP, and MQTT. For secure communication, protocols like TLS and OAuth2.0 are also essential. These fundamental aspects ensure that two systems can communicate reliably.

Syntactic or structural interoperability focuses on the basic exchange of information between systems. This is where formats like YAML, JSON, and XML come in, as well as standards like Open Packaging Conventions (OPC). At this level, systems can exchange and process data using

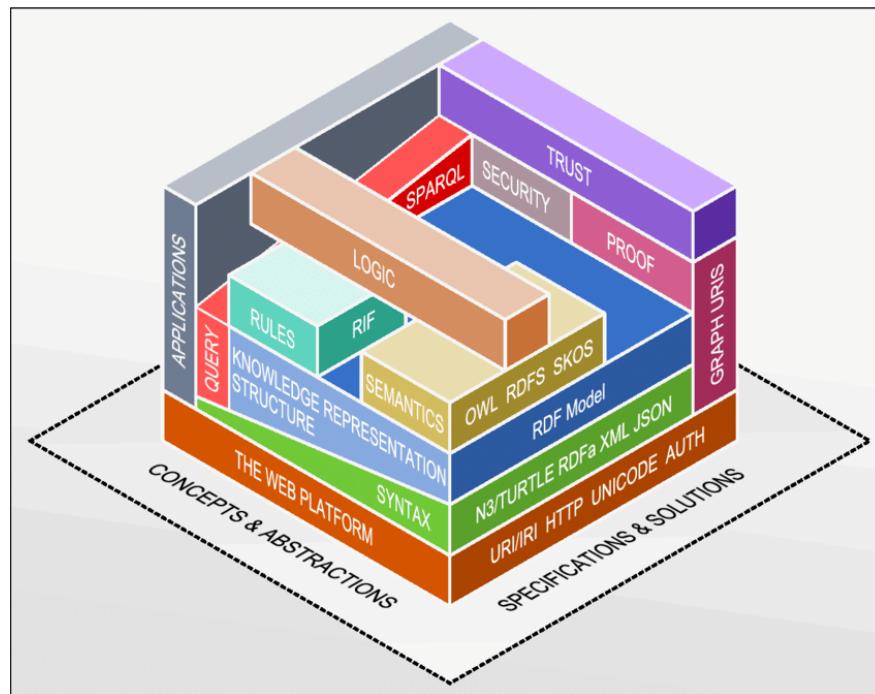


Figure 2.3: W3C Technology Stack (source: [GÄNG11])

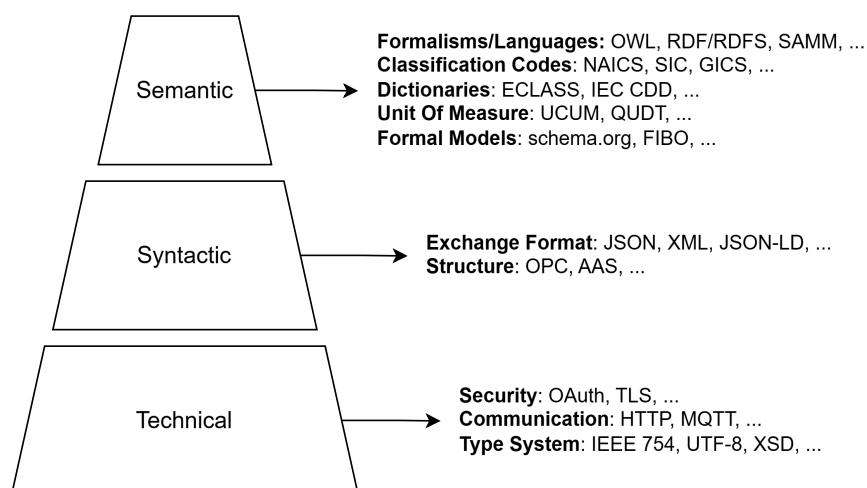


Figure 2.4: Layers of interoperability

a shared structure.

The next step, semantic interoperability, is concerned with the understanding of the meaning of the data being exchanged. For example, when a "fan" is referred to, it must be determined whether it refers to a person who follows a public figure or a device that generates airflow. Tools such as dictionaries and taxonomies help clarify these meanings. Semantic modeling takes this further, providing a structured way to represent data so that different systems can interpret it consistently. This often involves ontologies, which define the concepts, properties, and relationships within a specific domain, enabling seamless data exchange and understanding across systems. Technologies such as OWL and RDF support effective ontology creation, enhancing interoperability at this level.

Standardization aims to create a uniform framework through which systems can interoperate. Many well-established standardization bodies, such as IEEE, ISO, IEC, and VDMA, play pivotal roles in encouraging uniformity and compatibility across various industries. These organizations create standards that provide structured guidelines for technical and operational processes, helping industries coordinate effectively and enabling systems to work together seamlessly.

A primary challenge with standardization is that it can be difficult to apply effectively across a wide range of specialized fields, or verticals. Each industry has unique requirements, technologies, and operational workflows, which makes it challenging to design a single standard that accommodates all variations. Furthermore, the actual implementation of standards can be resource-intensive and complex, and often necessitating a shift in existing workflows, which can be challenging for many organizations.

Mediation provides a flexible approach to interoperability by acting as a translator between systems with differing standards or architectures. Mediation approaches are generally more adaptable in the short term and often realizable with lower initial costs, making them a more accessible option for companies with limited resources. Despite its flexibility, mediation also has limitations. While it can address immediate interoperability needs, it may not be a sustainable solution for large, evolving ecosystems, where scalability is critical.

As depicted in Figure 2.5, in many cases, a hybrid approach that leverages both standardization and mediation can provide a balanced solution. While standards establish a consistent framework for interoperability, mediation can fill the gaps where rigid standardization might not be achievable. To ensure compatibility between different standards, a mediator—such as software, a script, or a manual—acts as a translator, enabling systems using different standards to communicate effectively. Consequently, as the ecosystem grows and diversifies, mediation can be strategically phased out or restructured as standards become more broadly implemented and manageable.

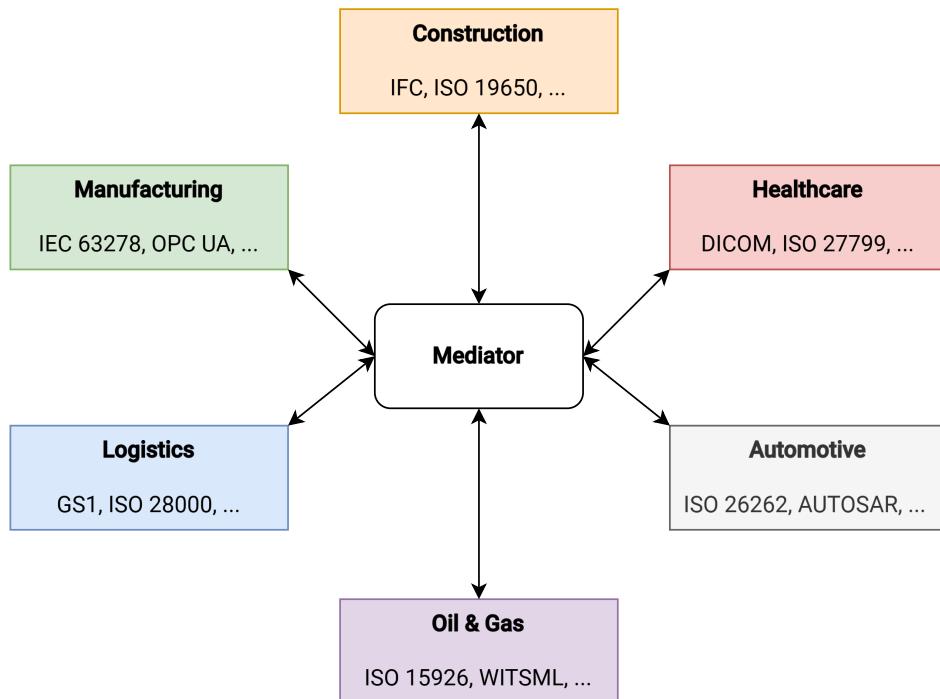


Figure 2.5: Standardization within industries, and mediation as a way to provide interoperability between them.

2.4 Modeling Digital Twins with AAS and SAMM

Asset Administration Shell

Asset Administration Shell (AAS) is a standardized framework for creating digital twins of industrial assets. The AAS specification, defined in IEC 63278-1:2023 [INTE23], is an International Electrotechnical Commission (IEC) standard promoted by the Industrial Digital Twin Association (IDTA), which supports the practical implementation of AAS and serves as its main user organization [PLATnd]. The latest version of the AAS specification, version 3, was published in April 2023 [ADMInd]. The AAS specification consists of five parts:

- Part 1 introduces the metamodel of AAS, outlining the building blocks for creating a digital twin.
- Part 2 defines the API and standard REST interfaces for interacting with and modifying the digital twin.
- Part 3 addresses data specification.
- Part 4, not yet released, will cover use cases and case studies.
- Part 5 focuses on the AASX package and its structure, designed as a self-contained container primarily for data exchange.

The metamodel of AAS comprises three main elements: *AssetAdministrationShell*, *Submodel*,

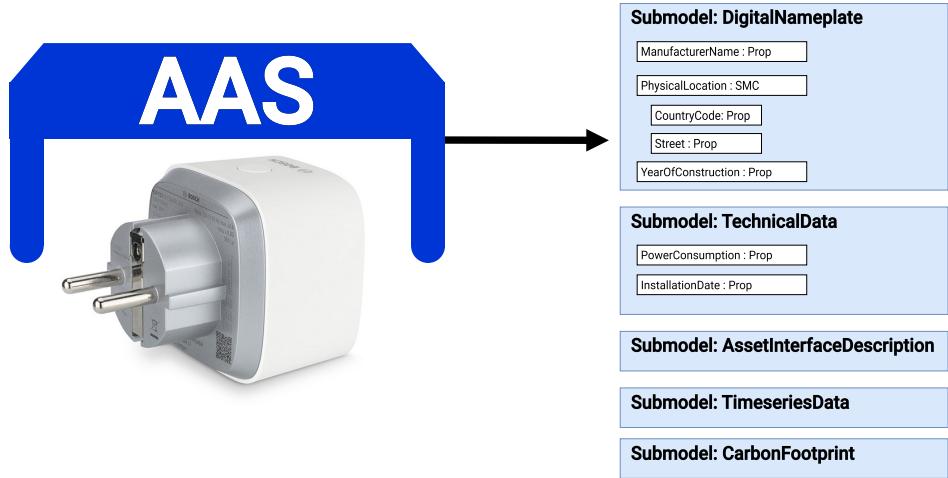


Figure 2.6: Structure of Asset Administration Shell.

and *ConceptDescription*. In this work, AAS is used to refer to the metamodel and standard as a whole, while *AssetAdministrationShell* refers to a part of it.

Each industrial asset is identified by an identifier or a set of identifiers, which enables linking to its *AssetAdministrationShell* via either a *globalAssetId* or *specificAssetId*—often represented by a QR code on the product. Each *AssetAdministrationShell* has a unique identifier, and each asset may have multiple *AssetAdministrationShells*, maintained by different stakeholders. The *AssetAdministrationShell* contains links to *Submodels*, each of which is self-contained and focused on a particular aspect of the asset, such as identification, location, or time-series data. The *Submodel* holds the actual information about the asset. For common use cases, IDTA publishes various pre-built and modeled *Submodel* Templates.

A *Submodel* is uniquely identified and consists of various *SubmodelElements*, such as *Property (Prop)*, *MultiLanguageProperty (MLP)*, *File*, and *SubmodelElementCollection (SMC)*. The metamodel provides mechanisms to assign meaning to each of these elements. Each element has a *semanticId*, enabling string-based semantic matching to identify similarities between attributes. Another method for providing context is the use of a *ConceptDescription (CD)*. The role of *ConceptDescription* is to describe the meaning of an attribute, define units, and offer links to additional sources of definitions. The *ConceptDescription* contains a *DataSpecification*, which acts as a template for this contextual and semantic information. This setup enhances readability for both human users and machines through the use of *semanticId*.

An example *AssetAdministrationShell* for a Bosch Smart Plug is shown in Figure 2.6. It contains various *Submodels*, such as *DigitalNameplate*, in which *ManufacturerName* is a *Property* and it can have a value such as '*Bosch*'.

Semantic Aspect Meta Model

The Semantic Aspect Meta Model (SAMM) is a metamodel designed for the semantic modeling of domain data. It is expressed in RDF, a machine-readable format that facilitates structured and interoperable data representation. SAMM is a critical component of the Eclipse Semantic Modeling Framework (ESMF)¹, a sub-project of the broader Eclipse Digital Twin initiative. ESMF offers various Software Development Kits (SDKs) that provide tools for generating artifacts such as diagrams, HTML documents, and Java code, facilitating seamless development and integration of digital twins. Historically, SAMM originated within the Semantic Data Structuring Working Group of the Open Manufacturing Platform², which was officially launched in September 2020 under the name BAMM [TEXT21].

SAMM consists of various building blocks, which are depicted in Figure 2.7, to describe and model asset information and behaviors. In SAMM, an *Aspect Model*, only describes the semantics and structure of the data, but it does not hold the instance and runtime data. In this metamodel, an *Aspect*, which is the core and root element of each *Aspect Model*, may include any number of *Properties*, *Operations*, and *Events*.

The Aspect is related to the digital twin of an asset and can logically also contain *Properties* that only exist on the digital twin but not on the asset itself. Each *Property* is associated with a *Characteristic* that provides context and meaning to the data it holds. For instance, if a *Property* represents power consumption, its *Characteristic* might specify that it is measured in Watt. *Characteristics* act as descriptors of *Properties*, helping to standardize the interpretation of asset data. They also hold the data type of an asset, which can be either an *Entity* or a specific subset of XSD data types allowed by SAMM. If the datatype of a *Property* is complex, the *Characteristic* associated with it can have an *Entity* as its datatype, enabling the representation of complex data structures within SAMM. *Constraints* add rules or limits to *Characteristics*, defining acceptable ranges, conditions, or patterns that the *Property* data must adhere. For example, a *Constraint* on a power consumption *Property* might set an allowable range from 0 Watt to 1000 Watt.

An example is shown in Figure 2.8, where an asset provides data through a REST endpoint associated with an *Aspect Model*. This setup demonstrates how SAMM allows complex, multi-dimensional data to be structured and made available in a standardized way, enhancing interoperability and data exchange within digital twin systems.

Such an *Aspect Model* can be cataloged in a dictionary, where both the endpoint location and the model itself are recorded and referenced together. This approach separates the data model from the instance data or runtime data. This separation enables the semantic model to evolve

¹<https://projects.eclipse.org/projects/dt.esmf>

²<https://openmanufacturingplatform.github.io/>

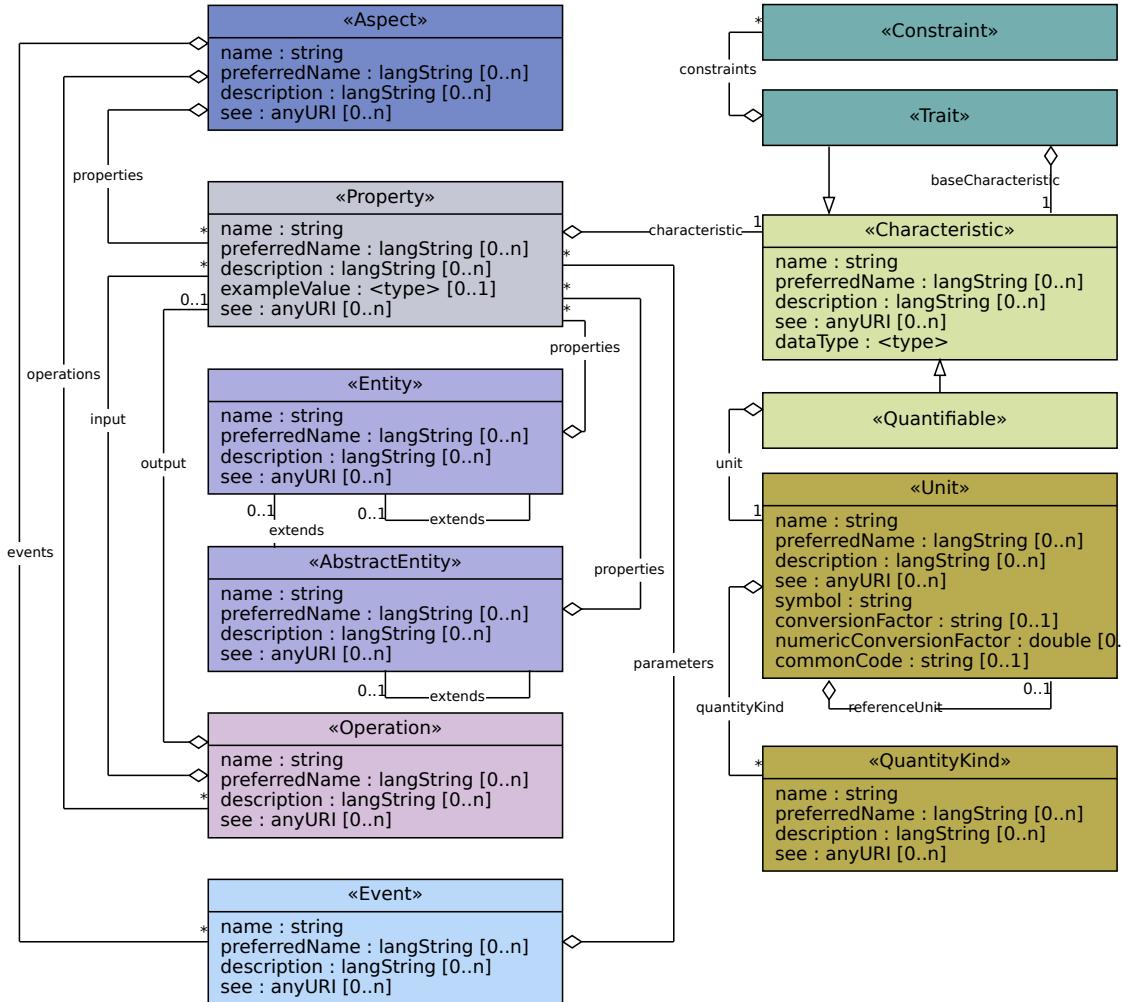


Figure 2.7: Building blocks of SAMM (source: [ECLI])

at its own pace, allowing it to be enriched and expanded as needed. This flexibility supports the ongoing development and refinement of the model, making it adaptable to future requirements without impacting the underlying asset.

In addition, ESMF SDKs provide the capability to generate *Submodel* Templates from Aspect Models and, conversely, to create Aspect Models from existing *Submodels*. For example, an *Aspect* would map to a *Submodel* Template. A *Property* in SAMM would map to a *Property* in AAS if its data type is not a complex *Entity*, otherwise it will map to a *SubmodelElementCollection*. Such mapping rules can be applied in the reverse direction to generate an *Aspect Model* from *Submodel* Template. This bidirectional compatibility allows for seamless integration between AAS and SAMM, enabling a smooth, gradual migration of assets to AAS standards with the support of SAMM and ESMF SDKs.

Additionally, ESMF SDKs can generate JSON schemas based on Aspect Models, allowing instance data to be validated against predefined ranges and constraints. This feature, currently unavailable in AAS, enhances interoperability by ensuring that data aligns precisely with specified model requirements.

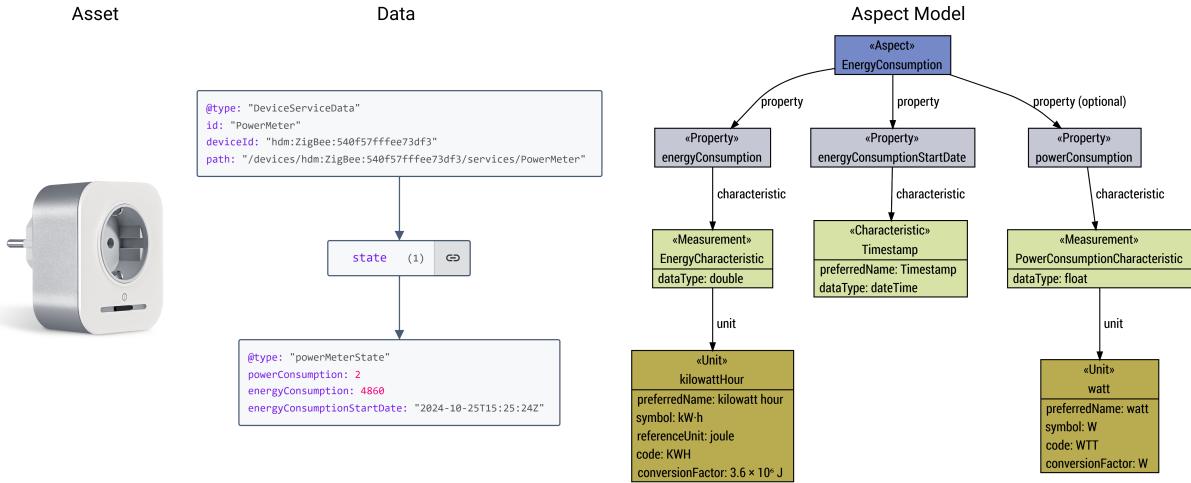


Figure 2.8: Data and Aspect Model for a Bosch Smart Plug.

2.5 Generative AI

A subfield of Artificial Intelligence (AI) is generative AI (GenAI), which involves creating new content, including text, audio, video, and more, based on the learned structures from existing data using generative models. The availability of large datasets and robust computational power has enabled developers to produce increasingly sophisticated generative models over time, which are capable of creating outputs that are difficult to distinguish from human-created content.

History

Nowadays, generative AI plays a crucial role in a wide range of applications, such as Natural Language Processing (NLP), Computer Vision, and more. Within NLP, generative AI contributes to various tasks, including text generation, summarization, question answering, named entity recognition, etc. Large Language Models (LLMs) are an important part of NLP and are based on transformer architectures. Transformer technology allows LLMs to rapidly grow by adding more parameters, enabling them to learn more complex relationships in the data. LLMs, like Generative Pre-trained Transformers (GPTs) developed by OpenAI, are trained on large amounts of data using powerful computing resources and consist of hundreds of millions to hundreds of billions of parameters. They can generate human-like text and are highly effective for different NLP tasks.

Traditionally, sequence-to-sequence (seq2seq) models formed the basis of many NLP tasks. These models are a type of neural network, designed to modify one data sequence into another using an encoder-decoder architecture. One of the major limitations of traditional seq2seq models is their difficulty in handling long sequences. However, transformer-based models address this issue with their self-attention mechanisms, which are able to manage long-range dependencies effectively.

Transformer architecture consists of an encoder and a decoder, each of which contains multiple

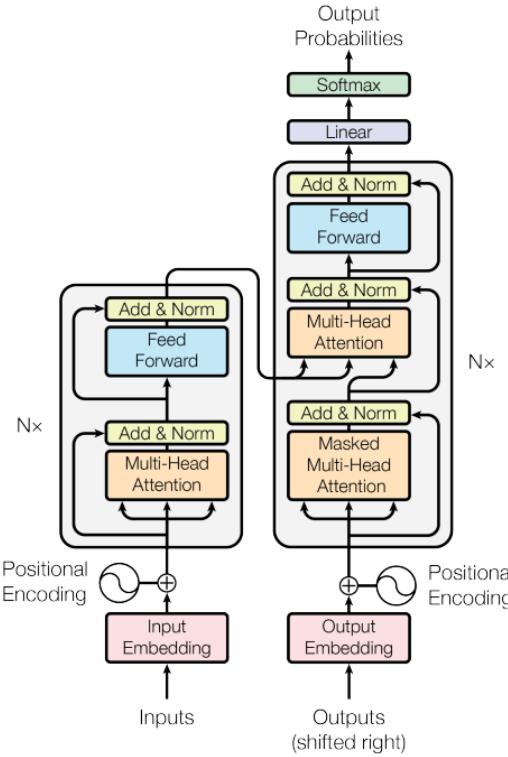


Figure 2.9: Architecture of transformer-based models (source: [VASW23])

layers. In each layer, there are two sub-layers: a multi-head self-attention layer, and a feed-forward neural network. The multi-head self-attention layer includes multiple sets of self-attention heads that are able to learn correlation in the whole input. Each self-attention head learns a different aspect.

Transformer models, as deep learning models, work with statistical numbers, not words, so a tokenizer is used to convert each word into a token to make it understandable for the Machine Learning (ML) model. Each token is then represented as a vector in the embedding layer. Additionally, positional encodings are added to these vectors to preserve the order of the words in the input sequence. The resulting vectors are passed to the self-attention layer. The self-attention weights are learned in parallel during training. The outputs of these layers are processed in the fully-connected feed-forward network. Residual connections and layer normalization are also used to improve the training process and make it more efficient. The architecture of transformer-based models is shown in Figure 2.9.

Adopting LLMs

Training LLMs from scratch requires substantial amounts of data, computational power, and resources. There are several pre-trained LLMs such as GPT [ACHI23], Llama [TOUV23], T5 [RAFF20], etc. which are open-source or can be accessed through purchasing an API and can be used for various tasks with less data and computational resources. There are different ways to

leverage the power of available pre-trained transformers for specific applications. Not only fine-tuning, but also serving LLMs requires extensive resources and high-performance GPU clusters. In practice, commercial providers like Azure, Amazon, and Google offer APIs to interact with generative models and LLMs. Depending on the available computational power and the task, one or a combination of these methods can be chosen:

1. Prompt Engineering

One approach to adapt an LLM is prompt engineering, where a prompt, including a set of instructions, is provided to an LLM to guide the pre-trained model in performing the target task using natural language.

There are two types of prompting techniques, including zero-shot and few-shot prompting:

- **Zero-shot prompting:** The pre-trained model can be used to perform a task without any additional training or examples in zero-shot prompting. In this case, the model makes prediction on the unseen data solely based on its existing knowledge.
- **Few-shot prompting:** In this method, one or more examples of labeled data can be provided to the model along with the input prompt. If only one example is provided, the approach is called one-shot prompting. If the model's performance is still not satisfactory, more examples can be employed. This involves providing multiple labeled data examples with the input prompt to improve the model's performance on the task.

2. Fine-Tuning the Model

When prompt engineering does not meet performance expectations and adding more examples is limited by the model's context window, fine-tuning on a specific dataset can be done to achieve better results. Fine-tuning is training the pre-trained model further on a smaller, similar dataset to adjust it to the particular task. It is one of the fundamental techniques used to improve the performance of an LLM on a certain task.

There are several techniques for fine-tuning:

- (a) **Full fine-tuning:** This approach retrains all model's parameters and a new version of the model with updated weights will be produced. It requires enough memory and computing resources to store and process all components that will be updated during training such as model's weights.
- (b) **Transfer learning:** In this method, most of the model's parameters, typically in the lower layers, are frozen. Only the final, task-specific layers (such as the classification head) are retrained, reducing the computational load. Parameter-Efficient Fine-Tuning (PEFT) is a transfer learning technique in which the model can be fine-tuned using a small set of parameters, reducing computational resources and preventing catastrophic forgetting, which can occur during full fine-tuning when all parameters are updated, potentially causing the model to forget previously learned information.

One popular PEFT method is Low-Rank Adaptation (LoRA) [Hu21], in which the pre-trained weight matrix is updated by being factorized into two smaller matrices in each layer of an LLM. The output matrix will have the same dimensions as the input pre-trained matrix.

3. Retrieval-Augmented Generation (RAG)

Another approach is RAG, which can be used as an alternative to fine-tuning or alongside with it, to enhance the model's performance. In RAG, relevant information related to the user's query is first extracted from external data sources and then passed to an LLM along with the query itself as a prompt to generate the final response. RAG helps the model access up-to-date information. This information can be updated at any time, as it is not hard-coded into the model architecture like fine-tuning.

Various hyperparameters are needed during the inference phase, and their impact on model performance is shown in Table 2.1.

Parameter	Higher Value Effect	Lower Value Effect
Temperature	Increases randomness; more diverse and creative output	Reduces randomness; more focused and deterministic output
Top_p	Expands token pool; allows diverse responses	Limits token pool; ensures predictable and high-likelihood responses
Top_k	Considers more tokens; increases diversity	Restricts token pool; yields more focused predictions
Presence Penalty	Reduces token repetition; promotes varied output	Allows repetition; may be useful for reiterative styles
Frequency Penalty	Discourages repetitive words; avoids loops in language	Allows repetition; keeps responses more predictable
Max Tokens	Allows longer responses; suitable for detailed answers	Limits response length; keeps output concise

Table 2.1: Inference Parameters Comparison for LLMs.

LLMs Landscape

In Table 2.2, a leaderboard of LLMs, including both commercial and open-source models, is presented, evaluated on code generation tasks using the BigCodeBench dataset [TEAM24]. The *Pass@1* metric measures the percentage of cases where the model generates a correct solution on the first attempt. As depicted in Table 2.2, most of the top performers are commercial models. The majority of the models have more than 32 billion parameters.

Rank	Model	Pass@1 Score (%)
1	GPT-4-Turbo-2024-04-09	32.1
2	GPT-4o-2024-08-06	30.8
3	Claude-3.5-Sonnet-20240620	29.4
3	DeepSeek-Coder-V2-Instruct (2024-07-24)	29.4
3	Gemini-1.5-Pro-Exp-0827	29.4
6	o1-Preview-2024-09-12 (temperature=1)	28.8
7	DeepSeek-V2-Chat (2024-06-28)	28.7
8	Gemini-1.5-Pro-Exp-0801	27.4
8	o1-Mini-2024-09-12 (temperature=1)	27.4
10	GPT-4o-2024-05-13	27.1
11	DeepSeek-Coder-V2-Instruct	27.0
12	Gemini-1.5-Pro-002	26.6
13	Llama-3.1-405B-Instruct	26.4
14	DeepSeek-V2.5	26.1
15	Claude-3-Opus-20240229	26.0
15	Mistral-Large-Instruct-2407	26.0
15	Llama-3.1-70B-Instruct	26.0
18	Gemini-1.5-Pro-API-0514	25.4
18	Qwen2.5-72B-Instruct	25.4
20	GPT-4o-mini-2024-07-18	25.3
21	Llama-3-70B-Instruct	24.6
21	Qwen2.5-32B-Instruct	24.6
21	Llama-3.1-Nemotron-70B-Instruct	24.6
24	Dracarys-Llama-3.1-70B-Instruct	24.3
25	Gemini-1.5-Flash-API-0514	23.6

Table 2.2: Top 25 LLMs on BigCodeBench Leaderboard [TEAM24].

3 Literature Review

In this section, relevant research and state-of-the-art methods are reviewed. No existing work was found that directly combines AI approaches with SAMM; however, the application of LLMs within the context of AAS is explored. The focus is then expanded to examine how LLMs are used in semantic engineering tasks, such as SPARQL query generation and ontology learning. Finally, a broader perspective is taken to discuss examples of LLMs in software engineering applications. A summary of related work, their key findings, and the associated challenges is provided in Table 3.1.

3.1 LLMs for AAS

The research paper by Xia et al. [XIA24] used LLMs for generating AAS models for digital twins within the context of Industry 4.0. The authors defined a semantic node (Figure 3.1), which will be processed by LLMs to generate the AAS models from technical data. They have developed software called AASbyLLM , which users can access via web-browser to convert their technical data to an AAS model. In order to generate AAS instance models from raw data, the following steps are involved: First, the user enters the raw technical data, which can be structured data such as JSON or unstructured text. Second, the data is enriched semantically via LLM agents, which convert it into semantic nodes. Third, using the RAG mechanism and an embedding LLM, the system searches for additional information that is semantically similar to the queried input through the ECLASS dictionary. Then, the relevance of the extracted output is tested by an LLM agent to filter out any unhelpful information, along with a brief explanation of why certain information was detected as irrelevant. Finally, the processed data is arranged into the format of an AAS model and presented to the user for feedback and review. If an expert verifies the system's output, it can be used to enhance the dictionary. Four LLMs were selected in this research paper: *GPT – 3.5* as a commercial model, and *Llama_2_70B_HF*, *Mixtral_8 × 7B_Instruct_v0.1*, and *WizardLM_70B_v1.0* as open-source models. The authors tested these four models using two approaches, with and without support of external library. They used human evaluation and two main metrics to determine the completeness and usefulness of the created AAS model. The effective generation rate and helpfulness score are two metrics were derived from the evaluators' ratings. The authors concluded that using RAG mechanism is beneficial for weaker models (*Llama_2*) while it does not improve results significantly for stronger

Semantic node	
Name	A concise and specific title assigned to the semantic node. It should be concise yet descriptive enough to convey the essence of the feature at a glance.
Conceptual definition	A definition of what the semantic node represents.
Usage of data (Affordance)	Describes the potential usage and application of the semantic node.
Value	The actual data that is described by the semantic node.
Value type <i>optional</i>	The type of data used for representing the actual data. (by default: String)
Unit <i>optional</i>	The measurement unit associated with the value. (if applicable)
Source description <i>optional</i>	This is an extended explanation that situates the semantic node within the specific context of its source, making the semantic node contextually traceable. (require extra information input)

Figure 3.1: Structure of the semantic node as an intermediate unit in AASbyLLM application (source: [XIA24]).

models (*GPT – 3.5*, *Mixtral*, *WizardLM*). Based on their research, LLMs can automate the process of creating AAS models by achieving 78% pass rate to generate helpful models without error in best case by *GPT – 3.5*. However, the current designed system is only limited to create a specific type of AAS *Submodel* for technical data, while other types could be supported in the future. Additionally, converting different data sources to a proper input format for LLMs is still challenging.

In the work by Hočvar et al. [HOČE24], the authors developed a system to query natural language in a knowledge graph using LLMs and RAG approach. They generated a dataset including hypothetical models for different assets in the factory, represented as AAS *Submodels*. These AAS *Submodels*, serialized as JSON, were converted into RDF triples as a knowledge graph and stored in Ontotext GraphDB. Using OpenAI’s text-embedding-ada-002 language model, these triples were transformed into vector embeddings and stored in a vector database. Users can enter AAS data or natural language queries through a user interface designed for this system. To manage natural language queries, the system follows two approaches: RAG and SPARQL query generation. In the RAG approach, the system analyzes the query and finds the matched vector embeddings within the knowledge graph. The resulting matched subgraph, along with examples, is then passed to an LLM (GPT-4) to generate the appropriate response. For SPARQL query generation, the user’s query is translated into a formal SPARQL query via LlamaIndex and executed on the knowledge graph. The results are then passed to an LLM (GPT-4) to generate the final answer. The researchers evaluated their system based on three methods: 1) examining the accuracy of data retrieval based on defined SPARQL parameters, 2) determining the accuracy of query results for specific information about assets, such as the number of dedicated machines, and 3) manually evaluating specific queries. The authors concluded that the overall performance of their system was satisfactory, though some inaccuracies remain in retrieving related data and handling large

datasets or certain queries due to LLM limitations. However, by adjusting parameters related to information retrieval, such as breadth and scope, and by utilizing SPARQL query generation, the system's performance improved.

In the paper by Tinnes et al. [TINN24], the authors used an LLM (ChatGPT GPT-3.5), to transform unstructured product descriptions into AAS. They created their own dataset for this purpose, using real data from the Siemens industrial product catalog. The approach begins by having the LLM categorize the product descriptions into various clusters. Then, the authors manually select random examples and convert them into AAS format. Finally, using ChatGPT, they generate AAS for the remaining product descriptions. To simplify the problem, they did not use the official AAS Submodel Template structure but instead defined a custom structured format; these examples later serve as ground truth for their system. The authors used RAG to generate the corresponding AAS of a product description. In this process, semantically related samples to the user's input are extracted from their dataset, which is stored in a vector database, and passed as a prompt to the LLM. At the end, the LLM generates the AAS JSON file for the input description. They found that LLMs are capable of clustering product descriptions, though refinements are still needed to achieve desired accuracy. The authors noted that while LLMs can generate AAS for unstructured product descriptions, in the worst cases, about 17% of the structure still required manual adjustment. Additionally, using random examples in prompts instead of semantically relevant examples appeared to worsen performance. Also, in their experiment, the number of examples in the prompt (tested up to six) did not significantly affect performance, with two-shot examples performing somehow as well as three-shot.

3.2 LLMs for Semantic Engineering

Babaei et al. [BABA23] 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. To conduct their evaluation, the authors selected models from different architecture categories: encoder-only, decoder-only, and encoder-decoder. They tested eleven LLMs (BERT-Large, PubMedBERT, BART-Large, Flan-T5-Large, Flan-T5-XL, BLOOM-1b7, BLOOM-3b, GPT-3, GPT-3.5, LLaMA-7B, and GPT-4) using zero-shot prompting. Additionally, they fine-tuned the Flan-T5-Large and Flan-T5-XL models using an instruction-tuning approach. These two models were chosen for fine-tuning because they are publicly available and performed better on all three tasks in zero-shot prompting compared to the other nine models. The authors found that while LLMs show promising potential for OL, they require fine-tuning for specific tasks to be practical and effective solution. Notably, the fine-tuned models, even with fewer parameters, performed better across all three tasks compared to zero-shot prompting.

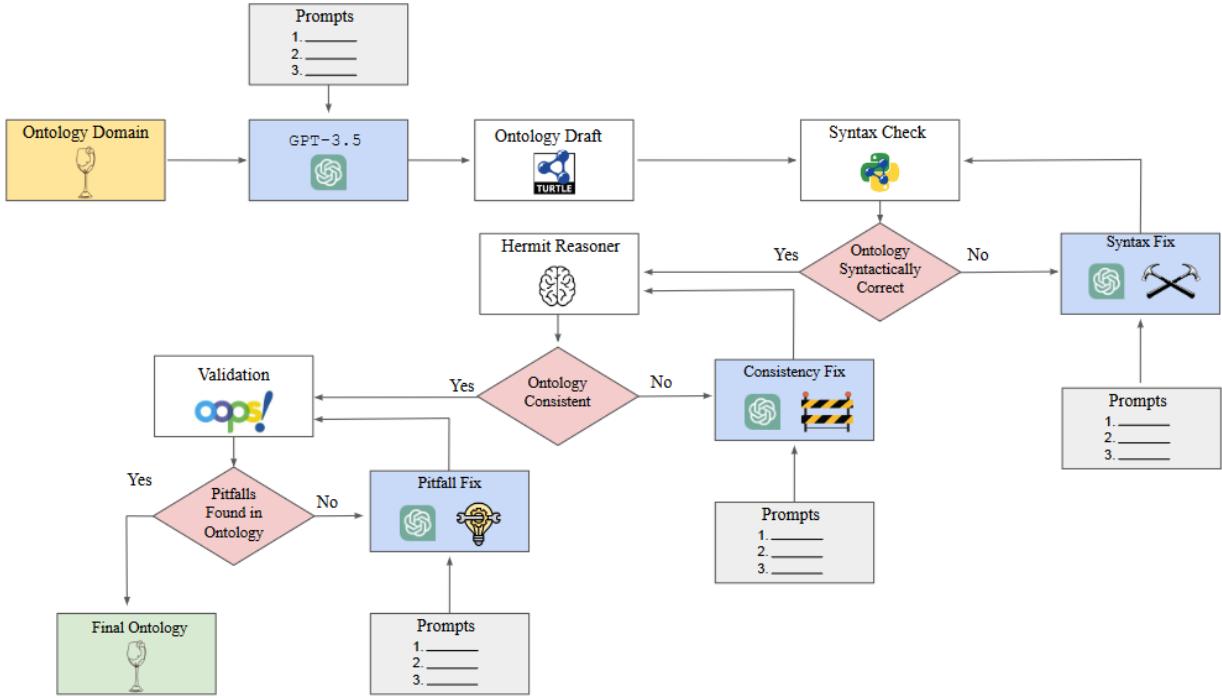


Figure 3.2: The workflow of the NeOn-GPT ontology engineering (source: [FATH24]).

The paper by [FATH24] presents a semi-automatic approach to ontology learning based on the NeOn methodology framework [SUÁR15] 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. The NeOn methodology focuses on thorough planning, making it ideal for developing ontologies in an iterative way that demands a deep understanding and an organized process, in contrast to Agile methodologies, which prioritize fast and simple planning. A prompt pipeline provided to GPT-3.5 was used to generate the Turtle 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 Figure 3.2 shows the proposed pipeline for NeOn-GPT in detail. They selected GPT-3.5 as their model, because it outperforms Llama, and PaLM on zero-shot prompting. They integrated the NeOn-GPT prompt pipeline with the metaphactory platform to provide structured guidance for ontology development. 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. They also deduced while LLMs are not fully capable of handling the entire ontology development process from natural language descriptions due to their lack of reasoning skills, they are useful for generating initial drafts, thereby reducing the time and effort required for experts, who no longer need to start from scratch.

In [SILV24], the authors considered the ability of LLMs to generate capability ontologies from natural language description. In this work, they chose three different LLMs: GPT-4 Turbo, Claude 3, Gemini Pro. However, they couldn't use Gemini Pro for this task due to the small

```

--- Instruction ---
Translate a natural language input describing a
capability into an OWL ontology ...

--- Context ---
${CaSk Ontology}

--- Input Data ---
${Coffeemaking Task Description}

--- Output ---
OWL-Ontology:
${Coffeemaking Result}

--- Input Data ---
${Task Description}

--- Output ---
OWL-Ontology:

```

Figure 3.3: Sample of the one-shot prompt template for capability ontology generation (source: [SILV24]).

token limitation. The temperature parameter was set to zero to reduce the randomness in the results during their experiments. They also tested three different prompting techniques (zero-shot, one-shot, and few-shot prompts) for generating seven various level of complexity in the capability ontologies. The structure of all the three prompt templates was similar with differences only in the sample examples used in the prompt. Figure 3.3 shows the sample of the one-shot prompt template used in this paper. They provided a straightforward natural language description, the CaSk ontology [KÖCH20] as context, and one example to help improve the models' performance. The authors created a semi-automated approach for evaluating the generated ontologies by checking syntax errors using Protégé editor¹, identifying inconsistencies through Pellet OWL reasoner², and addressing hallucinations and incompleteness using SHACL shapes. They argued that the results generated by LLMs were promising even with zero-shot prompting. However, utilizing few-shot prompting improved the results compared to zero-shot, and the models showed fewer significant errors in this case. Overall, Claude outperformed GPT for this task. One limitation of their work is that they did not vary the examples used in the few-shot prompting, which could be explored in future work. Additionally, providing the CaSk ontology with examples as input to the models increases token usage, thus raising costs, but future approaches could leverage embedding techniques to integrate context information more efficiently.

3.3 LLMs for Software Engineering

[KUNZ24] paper examines the potential of using LLMs to automatically generate Automation Markup Language (AML) models, a standardized XML-based format for exchanging data in industrial automation. They tested three various commercial LLM-based chat applications, such

¹<https://protege.stanford.edu/>

²<https://github.com/stardog-union/pellet>

as GPT-4o, GPT-4 provided with relevant information about AML, and Claude 3 Opus, for generating AML models from technical large documents. In their experiments, they provided a technical report about Tennessee Eastman Process (TEP), a well-known standard for automation tasks that includes publicly accessible technical documents. The challenge for this task was to only extract information for modeling and not unnecessary details like documentations for components that are not in the scope of modeling. In order to improve the performance of the models, they also gave an AML library with a small example of operational features of AML. They also evaluated if the models can perform well without providing technical report with using a different diagram for plant engineering. The researchers observed that tested LLMs are able to extract information for generating AML models from the provided report, and Claude outperformed the GPT-based models. However, as the LLMs processed more information, they tended to make more mistakes in generating a complete AML models. Therefore, although LLMs show promising potential for automatically generating AML models, additional research is needed to achieve consistency in the these models.

[MIOR24] discusses the challenges of working with JSON documents, which typically do not come with predefined schemas, unlike relational databases. This lack of schema makes it difficult to analyze data because users have to rely on a trial-and-error method to develop data pipelines. Analysts examine a few documents, make necessary assumptions, and then apply these assumptions to the entire dataset to check their accuracy. This method is inefficient and prone to errors. To address this issue, several researchers have proposed methods to automatically discover schemas from collections of JSON documents. Specifically, Baazizi et al. [BAAZ17] suggested an approach that involves identifying schemas for individual JSON documents and then merging them to create a comprehensive schema for the entire dataset. In this work, the authors explored how to improve generated JSON schemas automatically by using LLMs. They experimented with Code Llama and compared its performance to the T5 model. They also fine-tuned Code Llama using LoRA for a single epoch. Fine-tuned model outperformed other models in generating useful schemas. By learning from the derived schemas, existing schema discovery tools can be adapted to generate schemas that closely resemble those created by human experts.

3.4 KGs for AAS and SAMM

Rongen et al. [RONG23] proposed a framework for converting RDF-based models to AAS. This conversion preserves all information and maintains references to the original model in the AAS via *semanticIds*. As they argue, RDF-based models offer much higher semantic expressivity. Additionally, querying data is significantly easier compared to working with the AAS structure, where collecting various data points and manually traversing through them for queries can be cumbersome. However, the conversion mapping rules are limited and utilize only a few number of elements of AAS.

Rimaz et al. [RIMA24] argued that the RDF representation of AAS is more than just a serialization format; it can also be effectively utilized for additional functions. However, they note that a tool is required to convert other representations of AAS to RDF and vice versa. To address this, the authors developed a Python package to convert AAS/JSON to AAS/RDF and back. Their work highlighted several errors and issues in the RDF representation of AAS, including problems with handling ordered elements. Through a user interface, users can upload their AAS models to the platform. Data entries can be validated using SHACL, and users are also able to select predefined queries to query AAS. They showed how their proposed platform can be utilized in the context of sustainability for smart waste sorting system, as a use case.

3.5 Summary

A summary of related work is provided in Table 3.1. Most existing research uses LLMs to generate AAS Submodels, focusing on specific tasks such as creating Submodel Templates from product descriptions or catalogs. These methods are often limited in flexibility and mainly target narrow use cases, such as generating a digital nameplate for an asset or filling specific Submodels with data. While useful, they do not address the broader need for creating complete and adaptable models.

In this thesis, a more general solution is proposed, utilizing existing domain data in JSON format to generate Semantic Aspect Meta Model (SAMM) Aspect Models. By leveraging LLMs, the goal is to create an initial draft of the model that is as comprehensive as possible. This draft can then be refined by domain experts, significantly enhancing their productivity by reducing the manual effort required at the beginning of the modeling process.

Research	LLMs Used	Key Findings	Challenges
[XIA24]	GPT-3.5, Llama_2, Mixtral, WizardLM	LLMs can automate AAS model generation with 78% pass rate in best case by GPT-3.5.	Limited to specific Submodel structure.
[HOČE24]	GPT-4	LLMs can generate queries and responses in knowledge graphs using RAG and SPARQL.	Inaccuracies in large datasets, and challenges with complex queries.
[TINN24]	GPT-3.5 (ChatGPT)	LLMs can convert unstructured product descriptions to AAS with clustering. Prompt examples affect the results.	Manual adjustments still required.
[BABA23]	Flan-T5-Large, Flan-T5-XL	Fine-tuned LLMs show better performance for ontology learning tasks.	Fine-tuning needed for practical applications, especially for complicated and specialized tasks.
[FATH24]	GPT-3.5, Llama, PaLM	LLMs can assist in generating structured ontologies.	Need human involvement, LLMs lack reasoning skills for complete ontology development.
[SILV24]	GPT-4 Turbo, Claude 3	Few-shot prompting improves LLM results for capability ontology generation.	Increased token usage increases costs.
[KUNZ24]	GPT-4o, GPT-4, Claude 3 Opus	LLMs can generate AML models from technical reports, with Claude performing better.	Challenges in generating consistent AML models from complex data.
[MIOR24]	Code Llama, T5	LLMs can assist in schema discovery from JSON documents, with fine-tuning improving results.	Only focuses on completing and enhancing the JSON schema and not creating it from scratch.
[RONG23]	None	RDF models can be converted to AAS.	Conversion mapping is limited.
[RIMA24]	None	SPARQL, SHACL and reasoning directly on AAS/RDF using the official RDF representation of AAS.	Issues in the official AAS/RDF.

Table 3.1: Summary of related work.

4 Methodology

This chapter explains the methods we used to generate semantic models from JSON examples with LLMs. It includes steps for collecting and preparing data, selecting models, fine-tuning them, evaluating their performance, and delivering them to users.

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 4.1. 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.

```

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

Snippet 4.1: 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 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 4.1 shows this workflow.

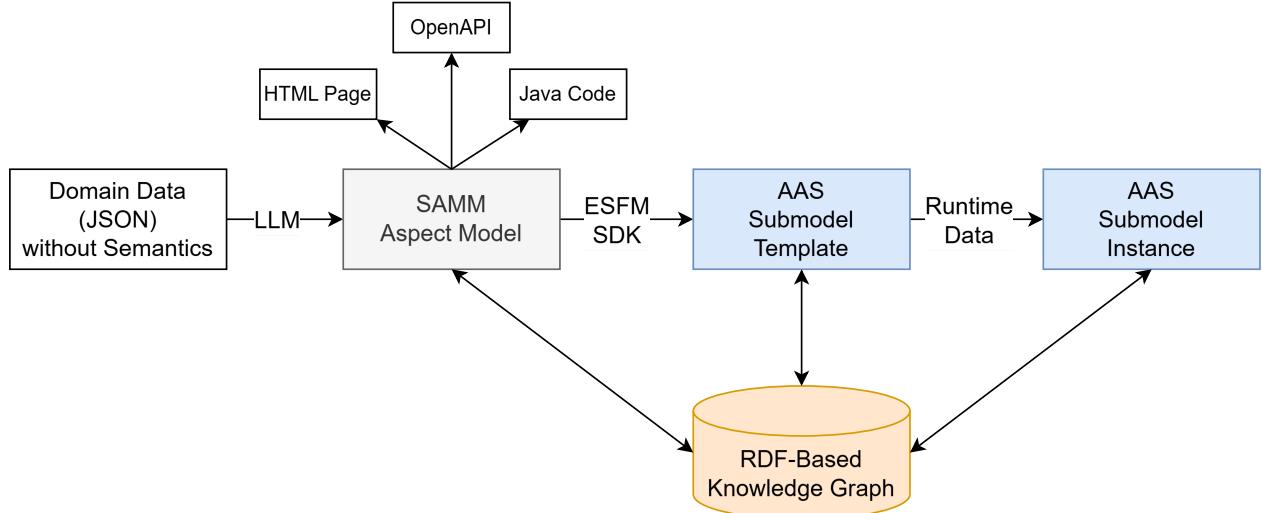


Figure 4.1: General workflow of the proposed solution.

Figure 4.2 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.

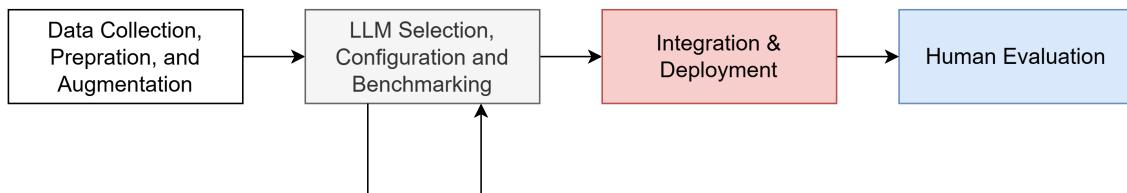


Figure 4.2: High-level steps of this research.

4.1 Data Collection, Preparation, and Augmentation

Semantic models created in Eclipse Tractus-X¹, collected on 18 September 2024, 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 page² and the BatteryPassport model³, can be considered for evaluation purposes, but they do not have a sufficient number of models, quality, and complexity to be considered for training.

¹<https://github.com/eclipse-tractusx/sldt-semantic-models>

²<https://eclipse-esmf.github.io/samm-specification/snapshot/modeling-guidelines.html>

³<https://github.com/batterypass/BatteryPassDataModel>

In Figure 4.3, 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 4.3. 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-XML 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. These steps are explained in detail in the next chapter.

```

sldt-semantic-models / io.catenax.batch / 3.0.0 / Batch.ttl
23     # SPDX-License-Identifier: CC-BY-4.0
24 #####
25
26 @prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#>.
27 @prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#>.
28 @prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#>.
29 @prefix unit: <urn:samm:org.eclipse.esmf.samm:unit:2.1.0#>.
30 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
31 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
32 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
33 @prefix : <urn:samm:io.catenax.batch:3.0.0#>.
34 @prefix ext-built: <urn:samm:io.catenax.shared.part_site_information_as_built:2.0.0#>.
35 @prefix ext-classification: <urn:samm:io.catenax.shared.part_classification:1.0.0#>.
36 @prefix ext-uuid: <urn:samm:io.catenax.shared.uuid:2.0.0#>.
37
38 :Batch a samm:Aspect;
39     samm:preferredName "Batch"@en;
40     samm:description "A batch is a quantity of (semi-)finished products or (raw) material";
41     samm:properties( :catenaXId :localIdentifiers :manufacturingInformation :partTypeIn
42     samm:operations();
43     samm:events();
44
45 :catenaXId a samm:Property;
46     samm:preferredName "Catena-X Identifier"@en;
47     samm:description "The fully anonymous Catena-X ID of the batch, valid for the Caten
48     samm:characteristic ext-uuid:UuidV4Trait;
49     samm:exampleValue "580d3adf-1981-44a0-a214-13d6ceed9379".
```

Figure 4.3: Semantic models in Tractus-x project.

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. 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 4.1 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.

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

Table 4.1: The size of the training, validation, and test sets.

4.2 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 Un-sloth library⁴. We limited our options to Un-sloth, because it provides faster and more efficient fine-tuning compared to other available libraries. We selected Llama 3.1[TOUV23], Llama 3.2, Qwen2.5-Coder [HUI24], and CodeLlama[ROZI23] 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-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

⁴<https://github.com/unslotha/unsloth>

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 Unslot library on Google Colab using T4 GPU instances. The local fine-tuning utilized PEFT with LoRA configuration and the Supervised Fine-Tuning (SFT) Trainer.

Effective prompting is critical to guide the LLM to generate correct outputs. We explored several prompting techniques as part of our experiments:

- Role-playing, where models were instructed to act as a specific persona.
- Few-shot prompting, where one, or two examples of the desired input-output patterns were provided in the prompt.
- Zero-shot prompting, in which no extra examples are included in the prompt.
- Iterative feedback, where the LLM is given its previous output along with errors for correction.

To determine the best-performing models, configurations, and prompting strategies, we designed several experiments, including:

- Testing the effect of different temperature settings on the output quality.
- Examining the influence of the number of examples (0, 1, or 2) in the prompts.
- Investigating the impact of repeating the same prompt multiple times.
- Exploring the contribution of iterative prompting.
- Evaluating the result of using different batch sizes on the fine-tuning process of GPT-4o-mini.
- Comparing the performance of different models.
- Identifying the impact of augmented data on the fine-tuning.

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. 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. 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 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.

4.3 Experiments Setup

In this section, we present the experiments conducted to evaluate different aspects of LLM performance and behavior. The experiments explored various factors affecting LLM performance, but they were not carried out in a strictly sequential manner. Instead, findings from earlier analyses guided subsequent decisions, enabling a focus on the most relevant LLMs and configurations.

Since GPT-4o-mini demonstrated the best overall performance in initial evaluations, it was prioritized for more in-depth studies, such as examining the effect of temperature and fine-tuning strategies. Consequently, not all LLMs were tested under every experimental condition. In Chapter 6, the results of these experiments are presented, with a summary provided in Table 6.11.

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.

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.

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.

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.

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.

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.

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.

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

4.4 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.

Another approach involves a custom application that 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.

Furthermore, we explore how both AAS and SAMM are represented in RDF, to store them in a knowledge graph. Leveraging this knowledge graph is not within the scope of this work. However, many question and answering systems based on LLMs proven to be more accurate when they are based on a well-formed knowledge graph, utilizing GraphRAG and other techniques. The generated SAMM Aspect Model is already in RDF-Turtle format. The ESMF SDK will generate AAS in XML or JSON. We will utilize various tools to convert these JSON files to RDF format.

4.5 Human Evaluation

We conducted human evaluations to gather feedback on our tool, called *SAMM Copilot*, 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.

4.6 Summary

This chapter has provided the methodology of the research conducted. An overview of the overall research goal and the general approach was provided. The next chapter will discuss these topics more in-depth and with more technical details and explanations.

5 Implementation

This chapter provides the specific details of how the methodology was implemented, including software tools, libraries, hardware, configuration parameters, and the details of the experiments. This chapter is crucial for reproducibility of the results.

5.1 Data Collection and Preparation

After collecting all the data from the Tractus-X semantic model repository, some data cleanup steps are necessary to ensure clean and uniform data for fine-tuning, as illustrated in Figure 5.1. First, license headers, lists of authors, and other comments at the beginning of an Aspect Model are not preserved due to token limitations of the LLMs input. This is depicted in the first lines of Snippet 5.1. Additionally, elements defined in 'samm:operations' and 'samm:events' do not influence the generated JSON payload and are primarily relevant for the API of digital twins and the generated OpenAPI specifications. To save tokens, these elements are removed from the Aspect Model.

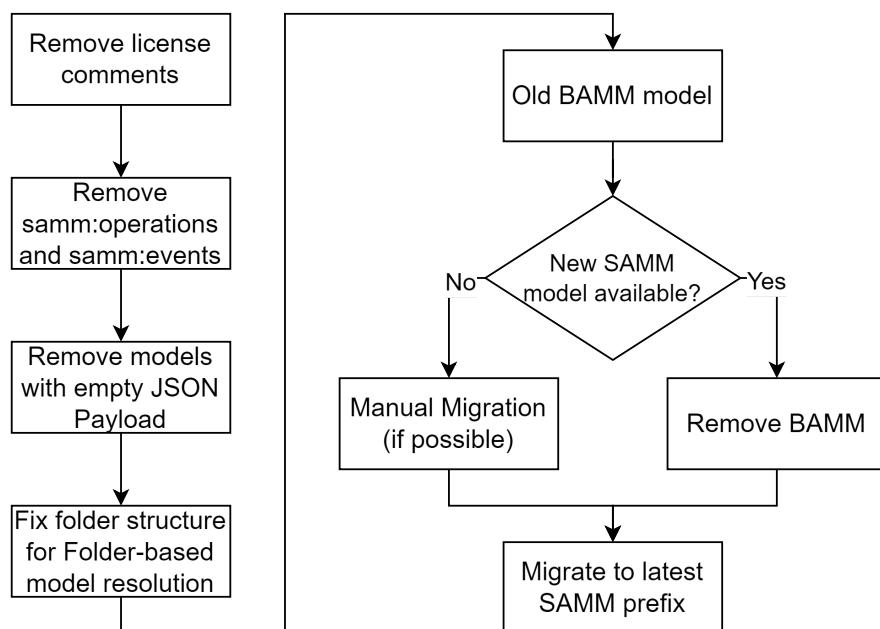


Figure 5.1: Data cleanup steps.

The ESMF SDK relies on the folder structure as a mechanism to resolve external entities that

are defined in other Aspect Models, known as the *ResolutionStrategy*. As it is shown in Snippet 5.1, the Aspect is defined in the file by the following statement on line 16 ':Batch a samm:Aspect ;'. Default or empty prefix indicated by colon (':') is defined as 'prefix : <urn:samm:io.catenax.batch:3.0.0#> .' in line 14. This model must be located in the following path 'io.catenax.batch/3.0.0/Batch.ttl'. In this context, an inconsistency was found in the 'BusinessPartnerCertificate' model, which caused an exception during resolution. This was manually corrected by changing the namespace to reflect the required folder structure.

```

1 ######
2 # Copyright(c) <Omitted to save space>
3 #
4 # Further comments and license information
5 #####
6
7 @prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#>.
8 @prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#>.
9 @prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#>.
10 @prefix unit: <urn:samm:org.eclipse.esmf.samm:unit:2.1.0#>.
11 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
12 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
13 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
14 @prefix : <urn:samm:io.catenax.batch:3.0.0#>.
15
16 :Batch a samm:Aspect;
17   samm:preferredName "Batch"@en;
18   samm:description "Omitted to save space"@en;
19   samm:properties ( :manufacturingInformation :partTypeInformation);
20   samm:operations ();
21   samm:events ().


```

Snippet 5.1: Example of an Aspect Model.

Within the Tractus-X semantic models repository, some older BAMM Aspect Models also exist. To identify these BAMM models, a simple search was conducted for files containing the BAMM namespace 'urn:bamm:io.openmanufacturing'. These BAMM Aspect Models are removed when newer and migrated versions are already available. If no alternative version exists, manual migration is performed using the Aspect Model Editor. Through this process, five models were migrated manually: 'io.catenax.part_site_information_as_planned', 'io.catenax.time_series_reference', 'io.catenax.certificate_of_dismantler', and 'io.catenax.battery.product_description'. One model, 'io.catenax.sealant.sealant_pass', was removed due to missing dependencies.

The final step in the cleanup process involved migrating all existing models from SAMM version 2.0.0 to SAMM version 2.1.0 to ensure that the LLM consistently generates the latest SAMM prefix. Figure 5.1 summarizes the data cleanup process.

After the data cleanup was completed, 155 models were retained. For data augmentation, the Aspect Models were modified, and new models were synthetically generated. However, such alterations can result in invalid models, so careful handling is required during this process, which is described in the next section.

5.2 Synthetic Data and Data Augmentation

Fine-tuning large language models (LLMs) often requires datasets specifically tailored to the task at hand. However, such data is not always readily available in the right quantity or format. To address this challenge, researchers create synthetic data—artificially generated datasets designed to mimic the characteristics of the data needed by the model. This approach is particularly useful when real-world data is scarce, contains sensitive information, or lacks the diversity required for effective training. Synthetic data bridges these gaps and provides a flexible way to prepare models for specialized tasks.

Creating synthetic data for LLM fine-tuning in this context is not a random process, as it must account for the complex nature of semantic models and the constraints they must satisfy. A practical approach is to start with existing semantic models and simplify them by removing or altering certain elements. Using simpler Aspect models has additional benefits, particularly during evaluation. Testing the LLM with smaller and less complex inputs, which often consist of fewer tokens, provides insights into its performance. If the model demonstrates strong performance on these smaller inputs, it suggests that a divide-and-conquer approach could be applied in the future, where large inputs are split into smaller parts. This strategy could facilitate fine-tuning LLMs for tasks involving very large or complex datasets.

Strategies for synthetic data generation are illustrated with an example JSON payload in Figure 5.2.

One approach to altering the Aspect Model is to change the names of properties, which directly affects the JSON serialization. The local part of the IRI of a *Property* or *Entity* is used as the key in the JSON output. For instance, if there is a property with the IRI '`urn:samm:io.catenax.shared.contact_information:3.0.0#phoneNumber`', its JSON serialization would appear as '`{"phoneNumber": "+49 89 1234567890"}`'. By modifying the IRI to '`urn:samm:io.catenax.shared.contact_information:3.0.0#PHONENUMBER`', the JSON serialization becomes '`{"PHONENUMBER": "+49 89 1234567890"}`'.

Alternatively, the `samm:payloadName` attribute can be employed to change the payload name

```

1  {
2    "contactProperty" : {
3      "faxNumber" : "+49 89 0987654321",
4      "website" : "https://www.samsung.com",
5      "phoneNumber" : "+49 89 1234567890",
6      "email" : "test.mail@example.com"
7    }
8

```



```

1  {
2    "contactProperty" : {
3      "FAXNUMBER" : "+49 89 0987654321",
4      "website" : "https://www.samsung.com",
5      "phoneNumber" : "+49 89 1234567890"
6    }
7

```



```

1  {
2    "CONTACTPROPERTY" : {
3      "PHONENUMBER" : "+82960901738437 795788802616337337126241155771"
4    }
5

```

Figure 5.2: Data augmentation example reflected on JSON payload.

without altering the IRI. This method is particularly useful when the desired payload name includes characters like "_" or "-".

If a property includes the 'samm:exampleValue' attribute, the sample JSON payload generator will always use the defined value. By removing this attribute, the generator defaults to producing a random value.

Dropping properties is another method for modifying the Aspect Model and its corresponding JSON payload. The 'samm:properties' attribute connects to a list of properties, where one property can be randomly removed to modify the structure and representation of the model.

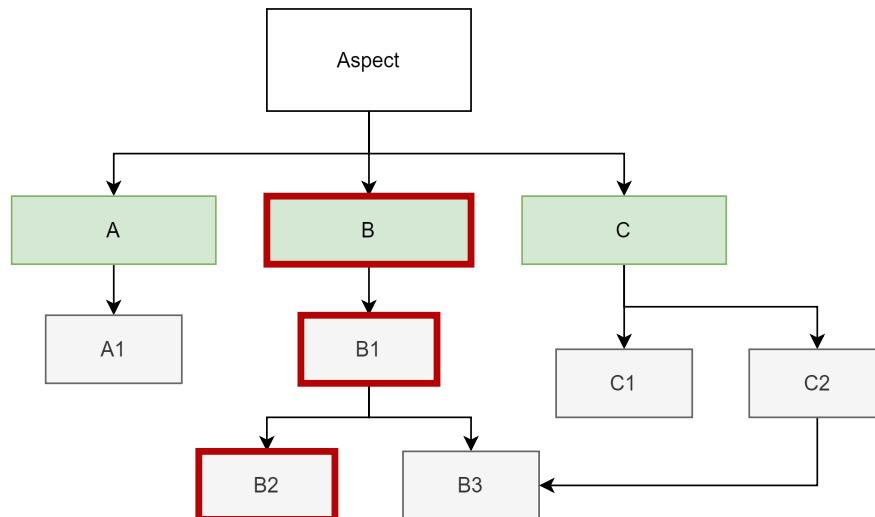


Figure 5.3: Cleanup Aspect Model when element B removed from the graph.

However, it is important to understand that such delete operations must be cascaded. Due to the graph structure, dependencies on the affected node or its related sub-nodes might be broken, as illustrated in Figure 5.3. Therefore, proper cleanup of the Aspect Model is essential.

To remove element 'B' in Figure 5.3, it is not possible to simply remove all its children due to the dependency from 'C2' to 'B3'. To address this, a Breadth-First Search (BFS) traversal is performed starting from the root Aspect element to identify the connected components. Unvisited nodes and their associated statements are then removed from the graph. In this case, 'B' is first removed, and the graph is then traversed. Since no path exists from the root element to 'B1' and 'B2', they can be safely deleted. The final Aspect Model and its JSON payload are saved as a new version and persisted to disk.

These modifications are applied with a random chance controlled by a user-defined parameter, enabling the generation of as many additional models as desired. However, generating too many models would increase the cost of training the model.

Figure 5.4 provides an overview of the data augmentation process.

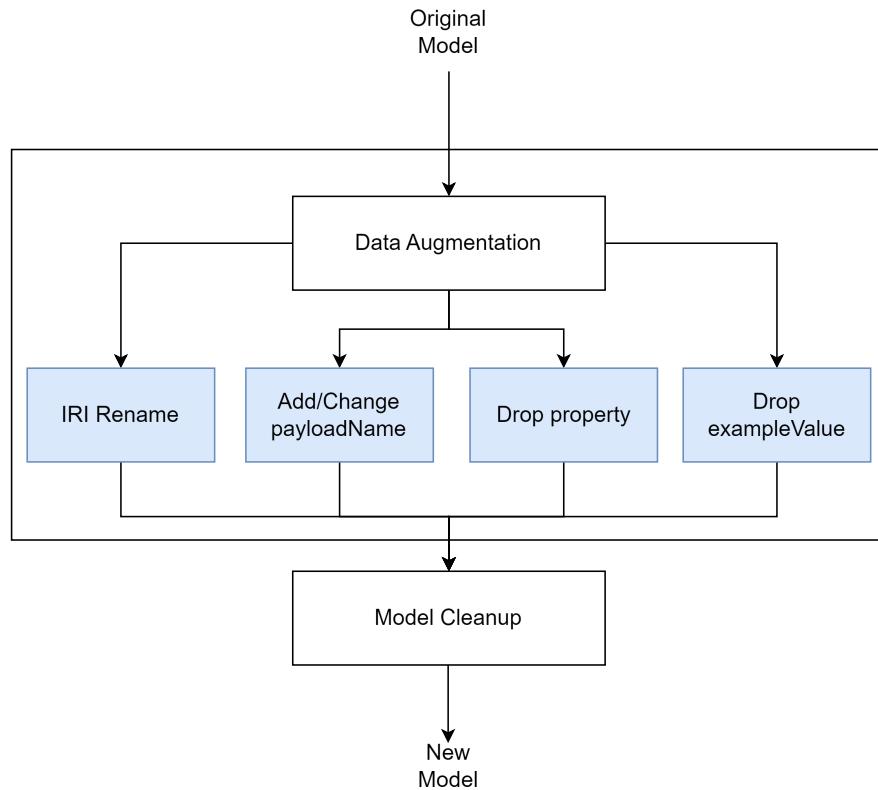


Figure 5.4: Data augmentation for generating new Aspect Models.

5.3 Data Splitting

After performing the data preparation and augmentation steps, the dataset was carefully split to ensure that no data leakage occurred during training and evaluation. The splitting strategy was

based on Aspect Model names rather than performing a completely random split. This approach was necessary to avoid scenarios where different versions of the same Aspect Model appeared in both the training and test datasets, which would lead to biased evaluations.

If the dataset were randomly split, it could result in cases where a newer version of a model (e.g., version 3) is included in the training set while an older version (e.g., version 2) is included in the test set or vice versa. For example, the model '`urn:samm:io.catenax.batch:3.0.0`' could be in the training set, and '`urn:samm:io.catenax.batch:2.0.0`' could be in the test set. While these versions may have only minor differences, the presence of shared data between training and testing would invalidate the evaluation.

To mitigate this issue, it was ensured that all shared Aspect Models, regardless of their version, were either entirely included in the training/validation set or entirely excluded from it as the test set. This strategy prevents data leakage and ensures that the test set is completely unseen during training, providing a fair evaluation of model performance.

5.4 Input Data Format

JSONL was used for fine-tuning the LLMs. Each line of JSONL contained either a simple prompt and completion or a ChatML format with system, user, and assistant roles. The structure of the JSONL is as follows:

- **One record per line:** Each line represents an individual data point.
- **Fields for fine-tuning:** A variety of formats can be used, starting with a simple prompt completion format that includes the following elements:
 - **prompt:** The input text provided to the model.
 - **completion:** The desired output text from the model.

This format is straightforward and often used for tasks like text generation, where the model is given a prompt and is expected to generate a completion based on it.

ChatML is a structured format used to represent conversations between a user and an AI. The main components of ChatML are:

- **role:** This element specifies who is sending the message. It can be one of the following:
 - * **user:** Represents a message from the user.
 - * **assistant:** Represents a message from the AI assistant.
 - * **system:** Represents system-generated messages, often used for instructions or context.
- **content:** This element contains the content of the message, which is the actual text sent by the sender.

Each message in a ChatML conversation is represented as a JSON object containing these two key elements: 'role' and 'content'. This structure helps in maintaining clarity and context within the conversation, allowing the AI to understand the flow of the interaction between the user, assistant, and system.

An example of a JSONL file is shown in Snippet 5.2:

```
1 {"messages": [{"role": "system", "content": "System Instruction"}, {"role": "user", "content": "Input 1"}, {"role": "assistant", "content": "Output 1"}]}
```

```
2 {"messages": [{"role": "system", "content": "System Instruction"}, {"role": "user", "content": "Input 2"}, {"role": "assistant", "content": "Output 2"}]}
```

Snippet 5.2: Example of a JSONL file format.

5.5 Cloud Fine-Tuning

Fine-tuning with Azure OpenAI or OpenAI is a scalable and efficient option that allows us to take advantage of pre-trained models while customizing them to fit specific use cases. Azure OpenAI allows more control on data residency and improved flexibility on deployment resource allocation. On the other hand, OpenAI tends to be cheaper alternative.

The fine-tuning process with both services involves several key steps:

- **Model Selection:** The process starts by selecting a pre-trained model from OpenAI's suite of available models, such as GPT-3, GPT-4o, or GPT-4o-mini. These models have been trained on vast datasets, and the fine-tuning process allows adaptation to specific needs without having to start from scratch.
- **Data Preparation:** To fine-tune a model, a dataset is required that aligns with the task the model is intended to perform. This dataset must be formatted according to OpenAI's guidelines, usually in JSONL (JSON Lines) format, with each line representing a training example that includes a prompt and the desired response.
- **Fine-Tuning through the API:** Once the dataset is ready, OpenAI's API is used to initiate the fine-tuning process. This is done by sending a fine-tuning request through the API with the prepared dataset. The API supports multiple fine-tuning options, including adjusting hyperparameters such as the number of training steps, batch size, and learning rate, which can be optimized based on the task.
- **Monitoring and Evaluation:** After fine-tuning begins, progress can be monitored via the API, and the model's performance can be evaluated using validation data. It is important to

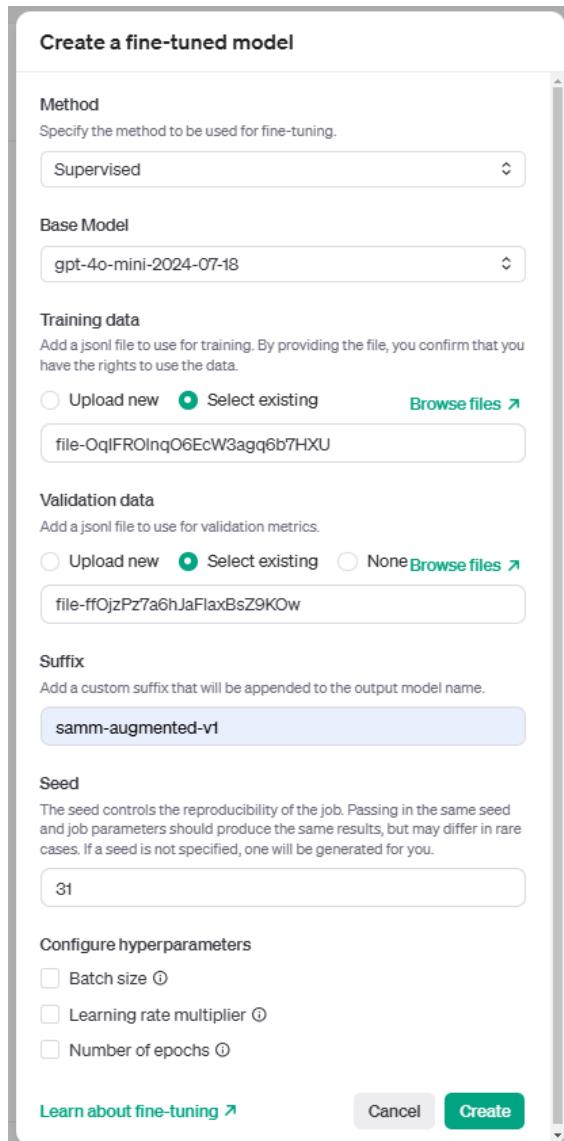


Figure 5.5: OpenAI's dashboard to create a fine-tuning job.

track metrics such as loss and accuracy to ensure the model is learning effectively. However, the default validation methods are not useful in this case because they are typically token-level loss metrics. At the time of fine-tuning, there was no option to add custom test criteria.

- **Deployment:** Once the model has been fine-tuned and validated, it can be deployed for inference through the OpenAI API. This allows to integrate the fine-tuned model into applications, whether it is for generating text, answering queries, or other specific tasks.

The steps to initiate fine-tuning through OpenAI's platform is depicted in Figure 5.5.

5.6 Local Fine-Tuning

The selection of the model for fine-tuning was driven by its performance in few-shot prompting. Models that demonstrated strong performance with the provided examples and successfully generated a SAMM Aspect Model were considered more suitable for fine-tuning. Among the candidates, Qwen2.5-Coder [HUI24] was chosen due to the superior performance of its base model compared to other open-source LLMs to generate more valid turtles.

The fine-tuning process for the Qwen2.5-Coder 7B [HUI24] parameter model was carried out on Google Colab T4 GPU using Unsloth Library. Unsloth was chosen due to its efficient assembly code and optimized performance. It allows larger models to run with a larger context window while using less VRAM and achieving faster inference speeds. Given the limitations of computational resources, Google Colab, which provides a T4 GPU, was utilized. For this reason, only models supported by Unsloth were selected.

For local fine-tuning, PEFT is used, utilizing a LoRA configuration with a rank of 16, optimized dropout settings, and memory-efficient gradient checkpointing. In PEFT, instead of updating all the parameters, only a small number of additional trainable parameters are introduced, such as low-rank adaptation (LoRA) layers. These layers are integrated into the model architecture.

The SFT Trainer configured to handle a small batch size of 1 per-device, with gradient accumulation steps set to 4. This allowed for an effective total batch size of 4, ensuring efficient utilization of GPU memory. The SFT Trainer is a specialized training framework used to fine-tune language models on labeled datasets. It handles various aspects of the training process, such as dataset preparation, model optimization, and evaluation.

Fine-tuning a base model typically requires more data. In this case, instruct models were found to be easier to fine-tune, though they are generally less creative. After fine-tuning, the model is saved in the GGUF format, which is produced by `llama.cpp`. In this process, q4_K_M quantization is applied, which, while not the most precise, allows for faster inference.

To import the model into Ollama, a ModelFile must be created. The ModelFile specifies the location of the model, parameters, templates, and other details necessary for Ollama to use it. By running '`ollama show --modelfile qwen2.5-coder:7b`', the ModelFile of a base model can be examined and modified for fine-tuned models. Full documentation is available on the Ollama GitHub page¹.

Once the model is loaded, it can be accessed via user interfaces such as LibreChat or connected to via Langchain4j to interact with Ollama models.

The experiments were conducted on a personal computing device with the following hardware specifications: an Intel(R) Core(TM) i7-9750H CPU operating at 2.60 GHz with six cores and

¹<https://github.com/ollama/ollama/blob/main/docs/modelfile.md>

twelve threads, 32.0 GB of installed RAM, and a 64-bit Windows 10 operating system. The device was equipped with an Nvidia GeForce RTX 2070 GPU with Max-Q design, featuring 8.0 GB of dedicated GPU memory and 15.9 GB of shared GPU memory, for a total of 23.9 GB of GPU memory.

5.7 LLM Evaluation

To evaluate the generated output, Apache Jena was used for RDF Turtle validation, and the ESMF SDK for SAMM and sample JSON payload validation. The JSON payload similarity check recursively compares nested objects and arrays. 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.

5.8 Prompting Techniques

Zero-shot and few-shot prompting templates are shown in Snippet 5.3, 5.4 and 5.5. Iterative prompting is done by passing the previous output along with the discovered exception in the next iteration, depicted in Snippet 5.6. Moreover, extra hints are provided to help the model to solve the issue, which are explained in Appendix A1. A folder structure as depicted in Figure 5.6 was used to keep track of the results and logs.

Figure 5.7 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.

- ¹ 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:
- ² JSON:
- ³ <JSON EXAMPLE>
- ⁴ Provide only the SAMM model without any extra explanation. Make sure that the ↳ output is a valid RDF Turtle format.

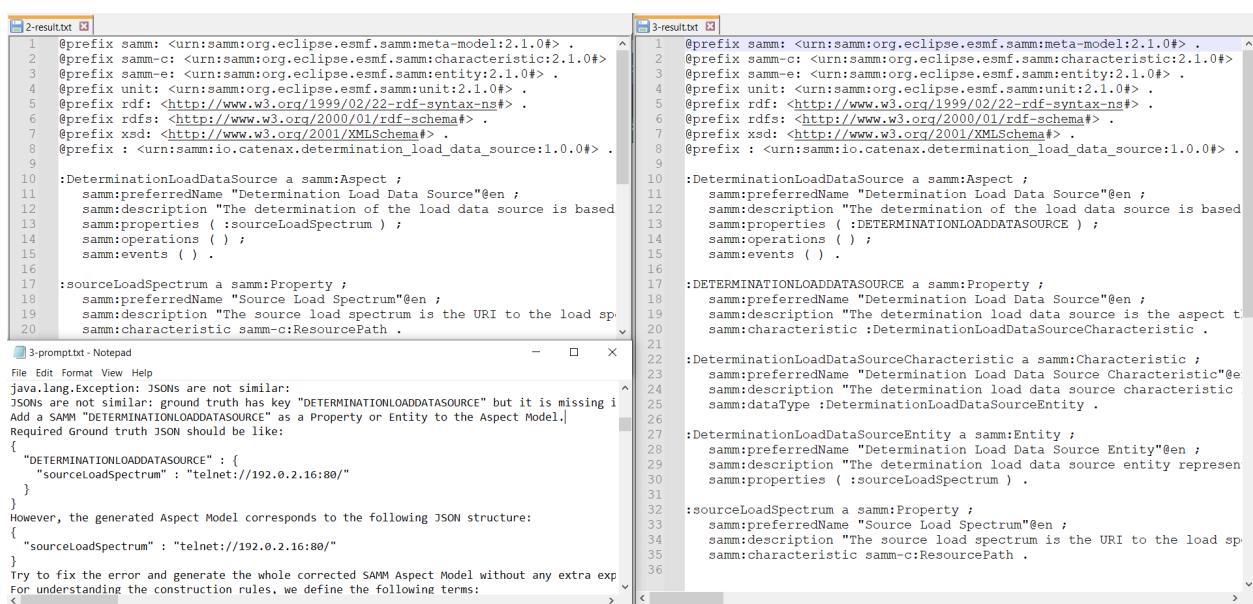
Snippet 5.3: Zero-shot prompt template.

```

|-io.catenax.eol_story
  |-2.0.0
    |-gen
      |-EndOfLife
        |-Experiment1
          |-azuregpt4omini20240718SemanticAspectMetaModelV1
            |-FewShotPromptTemplate1
              |-1-payload.txt
              |-1-prompt.txt
              |-1-result.txt
              |-1-summary.txt
              |-attempt.txt
              |-summary.txt
              |-validsamm.txt
        |-Experiment2
          |-qwen25codersammlatest
            |-FewShotPromptTemplate2
              |-1-exception.txt
              |-1-prompt.txt
              |-1-result.txt
              |-1-stacktrace.txt
              |-1-summary.txt
              |-2-exception.txt
              |-2-prompt.txt
              |-2-result.txt
              |-2-stacktrace.txt
              |-2-summary.txt
              |-3-exception.txt
              |-3-prompt.txt
              |-3-result.txt
              |-3-stacktrace.txt
              |-3-summary.txt
              |-attempt.txt
              |-summary.txt
        |-Experiment3

```

Figure 5.6: Folder structure to store all the attempts made.



The figure shows two side-by-side Notepad windows. The left window is titled '2-result.txt' and the right is '3-result.txt'. Both windows contain RDF triples in N-Triples format. The code is identical in both windows, showing the evolution of an Aspect Model over three iterations (Experiment 1, Experiment 2, Experiment 3). The code includes prefixes for namespaces like samm, rdf, rdfs, and xsd, and defines aspects such as 'DeterminationLoadDataSource' and 'sourceLoadSpectrum' with their properties and descriptions. The right window shows additional triples for 'DETERMINATIONLOADDATASOURCE' and 'DeterminationLoadDataSourceCharacteristic'.

```

@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 on the source load spectrum" ;
  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 spectrum" ;
  samm:characteristic samm-c:ResourcePath .

@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 on the source load spectrum" ;
  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 to be determined" ;
  samm:characteristic :DeterminationLoadDataSourceCharacteristic .

@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#> .

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

@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#> .

:DeterminationLoadDataSourceEntity a samm:Entity ;
  samm:preferredName "Determination Load Data Source Entity"@en ;
  samm:description "The determination load data source entity" ;
  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 spectrum" ;
  samm:characteristic samm-c:ResourcePath .

```

Figure 5.7: An example of iterative prompting.

```

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

```

Snippet 5.4: One-shot prompt template.

```

1 This is an example SAMM model:
2 <EXAMPLE SAMM ASPECT MODEL 1>
3 This is its corresponding JSON example:
4 <JSON PAYLOAD OF EXAMPLE SAMM ASPECT MODEL 1>
5 This is an example SAMM model:
6 <EXAMPLE SAMM ASPECT MODEL 2>
7 This is its corresponding JSON example:
8 <JSON PAYLOAD OF EXAMPLE SAMM ASPECT MODEL 2>
9 Your task is to create a SAMM model from a JSON Example.
10 Json Example:
11 <JSON EXAMPLE>
12 Provide only the SAMM model without any extra explanation. Make sure that the
  ↳ output is a valid RDF Turtle format.

```

Snippet 5.5: Two-shot prompt template.

```

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

```

Snippet 5.6: Prompt template with feedbacks.

5.9 Human Evaluation

The human evaluation form, for collecting feedback about the SAMM Copilot tool, is including familiarity with SAMM, manual modeling duration, usage of external sources, duration of modeling

with LLM, and overall satisfaction.

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.
 - (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.

5.10 Software Tools and Libraries

The software setup includes Ollama version 0.5.4 to deploy and host LLMs locally. A custom Java application is developed to use langchain4j for running inferences and performing the experiments. It also uses ESMF SDK to validate SAMM Aspect Models. In addition, LibreChat² is used as the web-based user interface to chat with the LLM.

²<https://www.librechat.ai/>

5.11 AAS and SAMM as RDF-based Graph Implementation

SAMM is natively represented in RDF, enabling direct storage in a triplestore. AAS supports multiple serialization formats, including XML, JSON, and RDF Turtle. The ESMF SDK generates AAS in AAS-XML and JSON format.

The AAS4j library, for instance, facilitates conversion from JSON to XML, though it does not officially support RDF serialization. While there is a draft pull request in the AAS4j repository to address RDF serialization, it remains under development and is not yet officially available. The BaSyx Python SDK³ can serialize data to JSON-to-RDF but lacks functionality for RDF-to-JSON conversion. On the other hand, the py-aas-rdf Python library⁴ can perform both JSON-to-RDF and RDF-to-JSON conversions.

Given these constraints, the solution involves a combination of tools and processes. The workflow invokes the SAMM CLI through Python code, creating the AAS-JSON file from the SAMM Aspect Model generated by LLM. Subsequently, the AAS-JSON is transformed into AAS-RDF using py-aas-rdf library.

This process yields two RDF datasets: the original SAMM data in RDF and the converted AAS-RDF representation. Both datasets can be inserted into the triplestore, which in this case is Ontotext GraphDB, ensuring comprehensive integration of SAMM and AAS data into the Knowledge Graph.

5.12 End-Solution Deployment

As shown in Figure 5.9, LibreChat can serve as a user interface similar to ChatGPT. A screenshot of this platform with an example input is depicted in Figure 5.8. The Aspect Model Editor can use a library such as LangChain4⁵ to connect to various LLM models, including commercial models from Azure, OpenAI, and AWS, as well as locally hosted models via Ollama⁶.

In order to further enhance the user experience, the solution can be integrated directly into the Aspect Model Editor. As shown in Figure 5.10, users 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

³<https://github.com/eclipse-basyx/basyx-python-sdk/tree/Experimental/Adapter/RDF/basyx/aas/adapter/rdf>

⁴<https://github.com/mhrimaz/py-aas-rdf>

⁵<https://github.com/langchain4j>

⁶<https://ollama.com/>

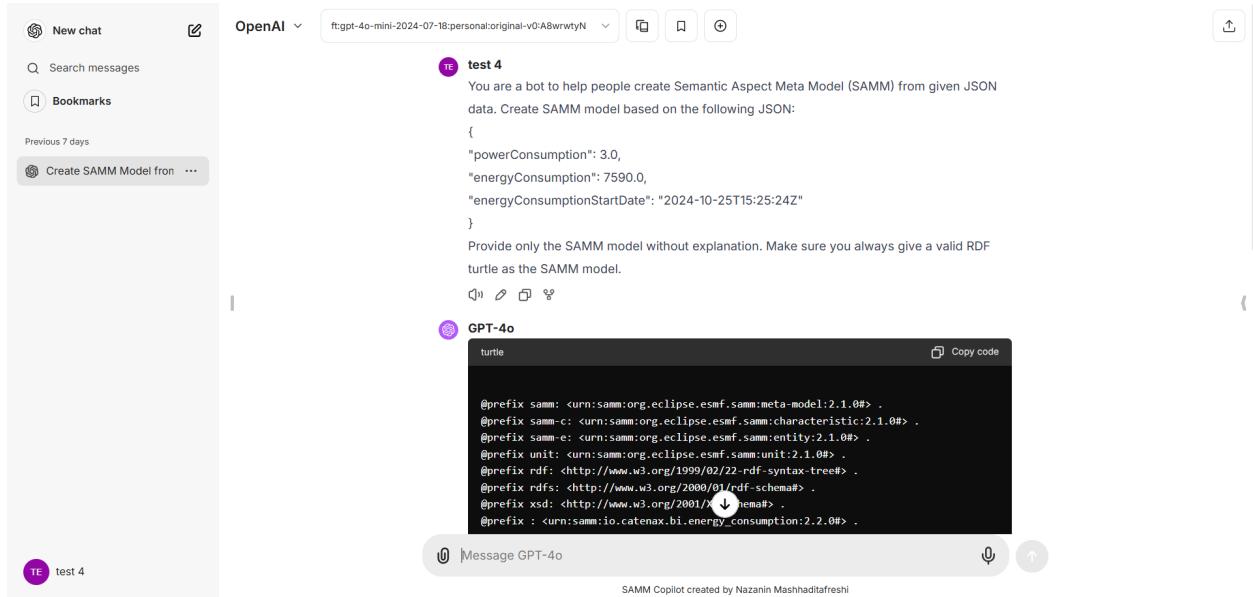


Figure 5.8: SAMM Copilot platform user interface.

the process is depicted in Figure 5.11. As shown in Figure 5.11a, 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 5.11b. For example, they could ask the AI to add a regular expression constraint to a property. In 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.

5.13 Summary

This chapter has detailed the practical aspects of the research. Input data formats, cloud and local fine-tuning approaches, LLM evaluation methodology, prompting techniques, and human evaluations have been covered. Additionally, software tools, as well as the end-solution architecture and prototype, have been explained in detail.

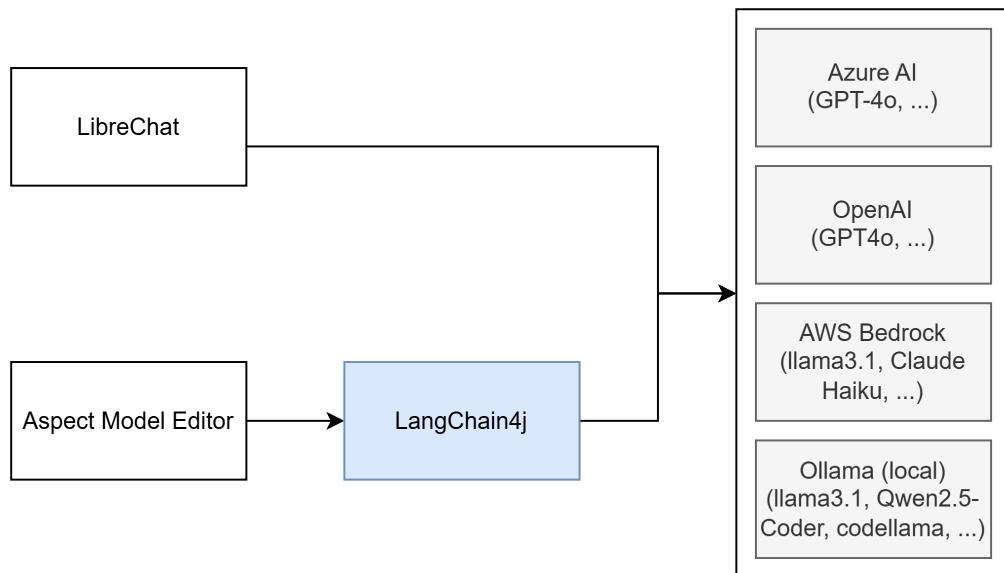


Figure 5.9: Overall architecture and components of the end-solution.

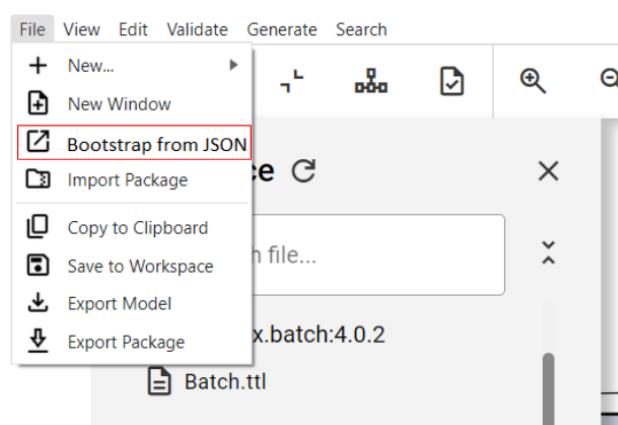
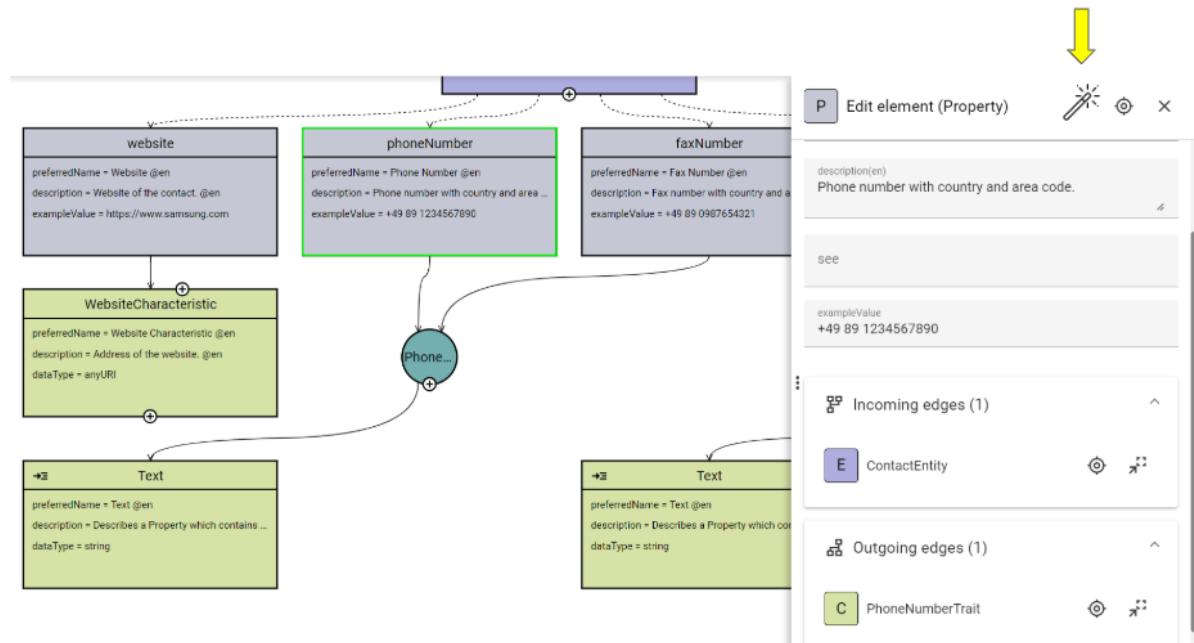
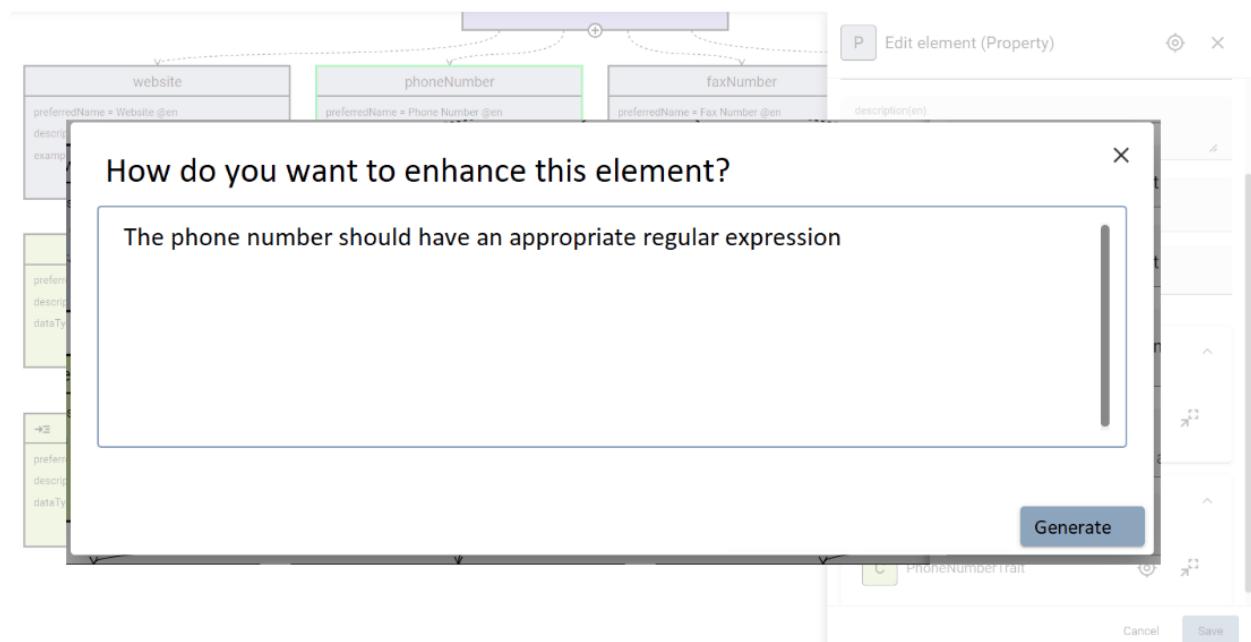


Figure 5.10: Bootstrap Aspect Model from JSON example in the Aspect Model Editor.



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



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

Figure 5.11: Enhance an element of the Aspect Model by AI using natural language specification.

6 Results

6.1 Fine-Tuning Statistics

Part of the fine-tuning process was conducted on the OpenAI platform. Figure 6.1 shows the fine-tuning job details for GPT-4o-mini on the original dataset, including information such as batch size, number of epochs, learning rate multiplier, and seed. With this information, it would be possible to more easily reproduce the results. The validation loss metrics for the fine-tuning process are depicted in Figure 6.2. The job details of fine-tuning with augmented data are shown in Figure 6.3, and the corresponding validation loss chart is presented in Figure 6.4.

To fine-tune GPT-4o, the suggested parameters from the Azure OpenAI service were used, as depicted in Figure 6.5. These parameters included a batch size of 1, the default learning rate multiplier, and 3 epochs. The training process took a total of 2 hours. The training and validation loss metrics for this fine-tuning process are shown in Figure 6.6.

Figure 6.7 and Figure 6.9 depict the job details for fine-tuning with different batch size settings. The corresponding loss charts are shown in Figure 6.8 and Figure 6.10. The validation and training loss of fine-tuning the Qwen2.5-Coder model, on the original dataset, are shown in Figure 6.11. Chapter 7 explains all the results in more detail.

MODEL	
ft:gpt-4o-mini-2024-07-18:personal:original-v0:A8wrwtyN	
○ Status	Succeeded
① Job ID	ft:job-2cdBSBZKM8oFEcqpLuohysh5
✉ Suffix	original-v0
↳ Base model	gpt-4o-mini-2024-07-18
↳ Output model	ft:gpt-4o-mini-2024-07-18:personal:original-v0:A8wrwtyN
⌚ Created at	Sep 18, 2024, 11:31 PM
⋙ Trained tokens	1,314,291
⌚ Epochs	3
⌘ Batch size	1
ߡ LR multiplier	1.8
ߜ Seed	34
ߡ Metrics	
Training loss	0.1819
Validation loss	0.3143
Full validation loss	0.2744

Figure 6.1: Job details for fine-tuning GPT-4o-mini on the original dataset.

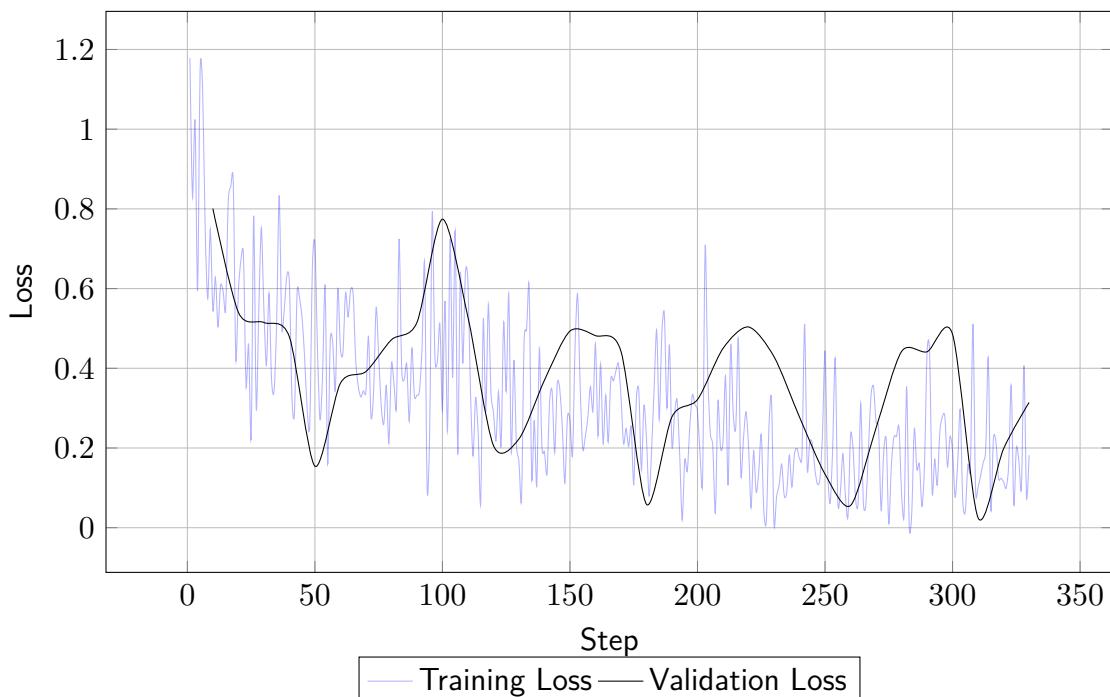


Figure 6.2: Training and validation losses for the fine-tuned GPT-4o-mini on the original dataset.

MODEL	
ft:gpt-4o-mini-2024-07-18:personal:augmented-rename-drop-02-05-v1:A8x9uqo0	
○ Status	Succeeded
○ Job ID	ftjob-2RrZ3risQpRVZQSwGvI8sk4j
☒ Suffix	augmented-rename-drop-02-05-v1
☒ Base model	gpt-4o-mini-2024-07-18
☒ Output model	ft:gpt-4o-mini-2024-07-18:personal:augmented-rename-drop-02-05-v1:A8x9uqo0
○ Created at	Sep 18, 2024, 11:46 PM
☒ Trained tokens	1,932,537
☒ Epochs	3
☒ Batch size	1
☒ LR multiplier	1.8
☒ Seed	33
☒ Metrics	
Training loss	0.0098
Full validation loss	0.3019

Figure 6.3: Job details for fine-tuning GPT-4o-mini on the augmented dataset.

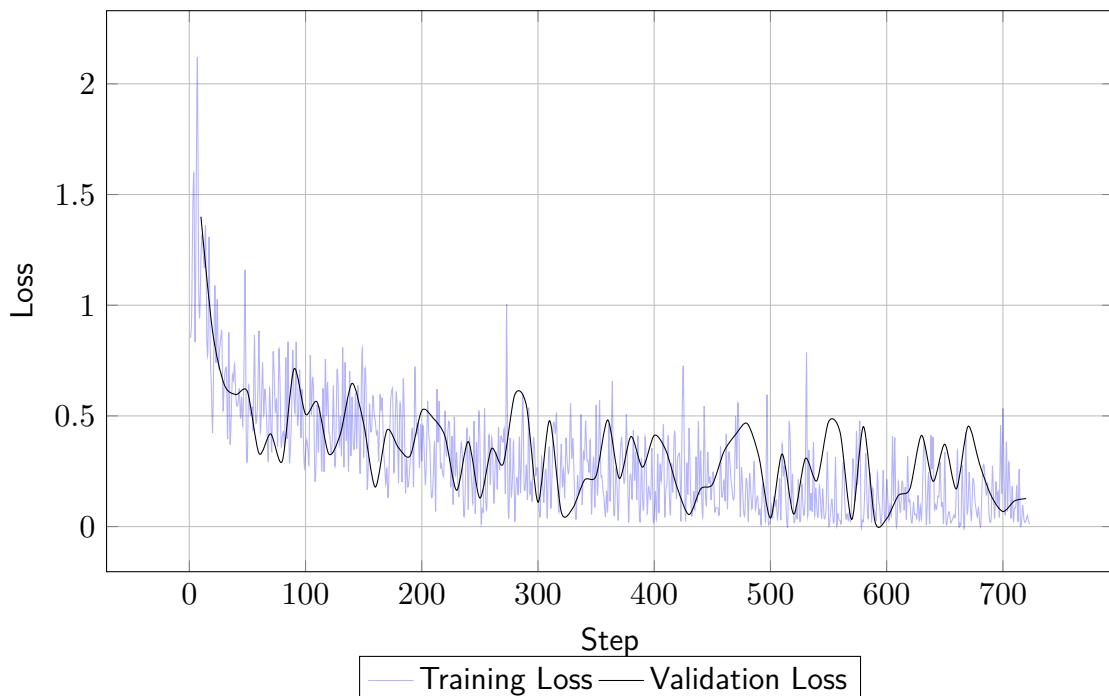


Figure 6.4: Training and validation losses for the fine-tuned GPT-4o-mini model trained on the augmented dataset.

Model attributes		Task parameters	
ID	ftjob-d363fc64e90440a791a116544f8e1231	Batch size	1
Status	Completed	Learning rate multiplier	Default
Created on	Nov 11, 2024 10:12 AM	Number of epochs	3
Validation file	output_dataset_validation_2024-09-18.jsonl	Seed	411962825
Weights & Biases integration enabled?	No		
Training completed on	Nov 11, 2024 12:05 PM	Duration	1h 53m 1s
Training tokens billed	1,315,000		
Download results		Download training file	Download validation file

Figure 6.5: Job details for fine-tuning GPT-4o on the original dataset.

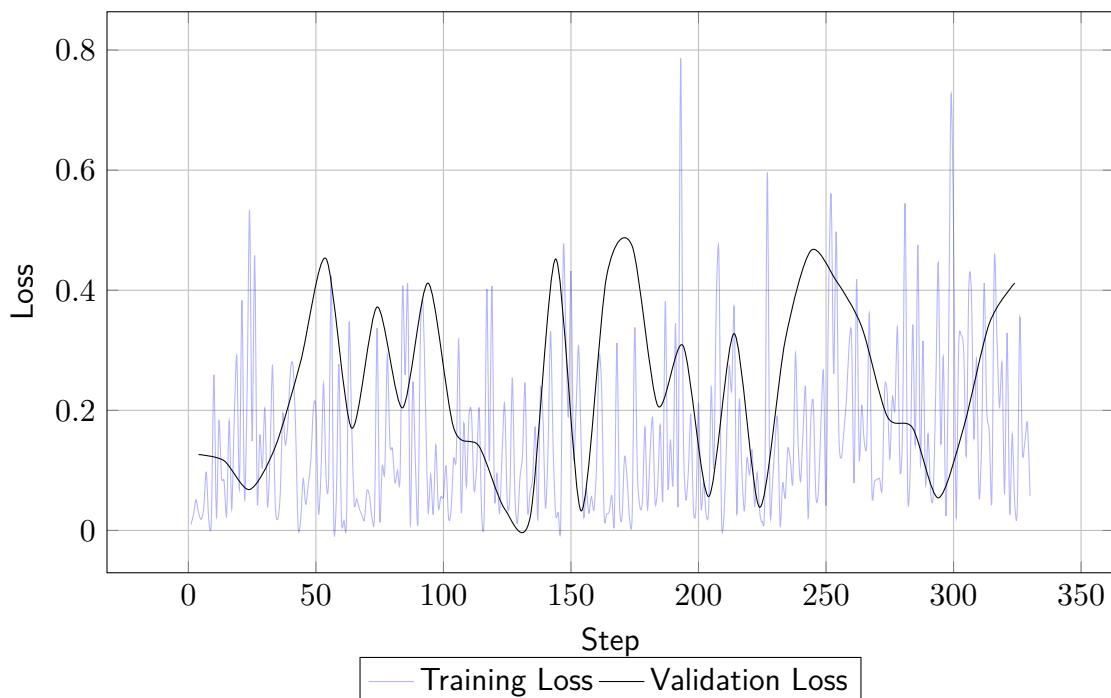


Figure 6.6: Training and validation losses for the fine-tuned GPT-4o trained on the original dataset.

Model attributes		Task parameters	
ID	ftjob-aabbbfbdec70404bb1bdb067f8e74b72	Batch size	8
Status	Completed	Learning rate multiplier	0.2
Created on	Nov 22, 2024 12:43 PM	Number of epochs	3
Validation file	output_dataset_validation_2024-09-18.jsonl	Seed	34
Weights & Biases integration enabled?	No		
Training completed on	Nov 22, 2024 1:23 PM	Duration	40m 16s
Training tokens billed	1,315,000		
Download results		Download training file	Download validation file

Figure 6.7: Details of fine-tuning GPT-4o-mini on the original dataset with batch size of 2.

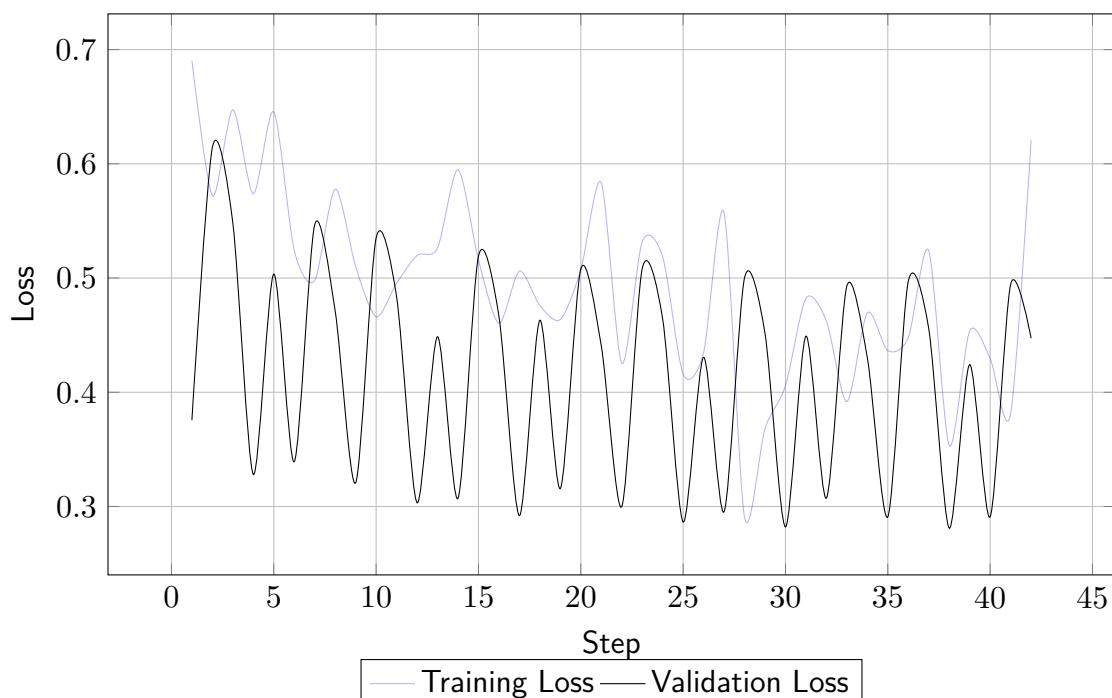


Figure 6.8: Training and validation losses for fine-tuning GPT-4o-mini using original dataset with batch size of 2.

Model attributes		Task parameters	
ID	ftjob-aabbbfbdec70404bb1bdb067f8e74b72	Batch size	8
Status	Completed	Learning rate multiplier	0.2
Created on	Nov 22, 2024 12:43 PM	Number of epochs	3
Validation file	output_dataset_validation_2024-09-18.jsonl	Seed	34
Weights & Biases integration enabled?	No		
Training completed on	Nov 22, 2024 1:23 PM	Duration	40m 16s
Training tokens billed	1,315,000		
Download results		Download training file	Download validation file

Figure 6.9: Job details of fine-tuning GPT-4o-mini on the original dataset with batch size of 8.

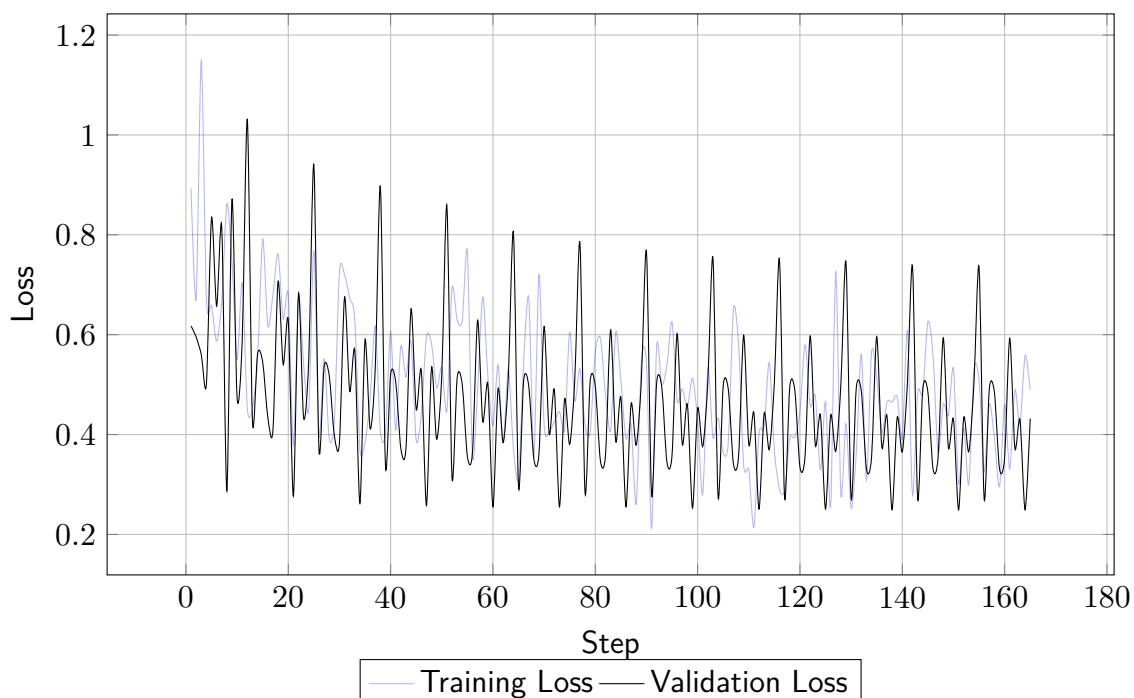


Figure 6.10: Training and validation losses for the fine-tuned GPT-4o-mini using original dataset with batch size of 8.

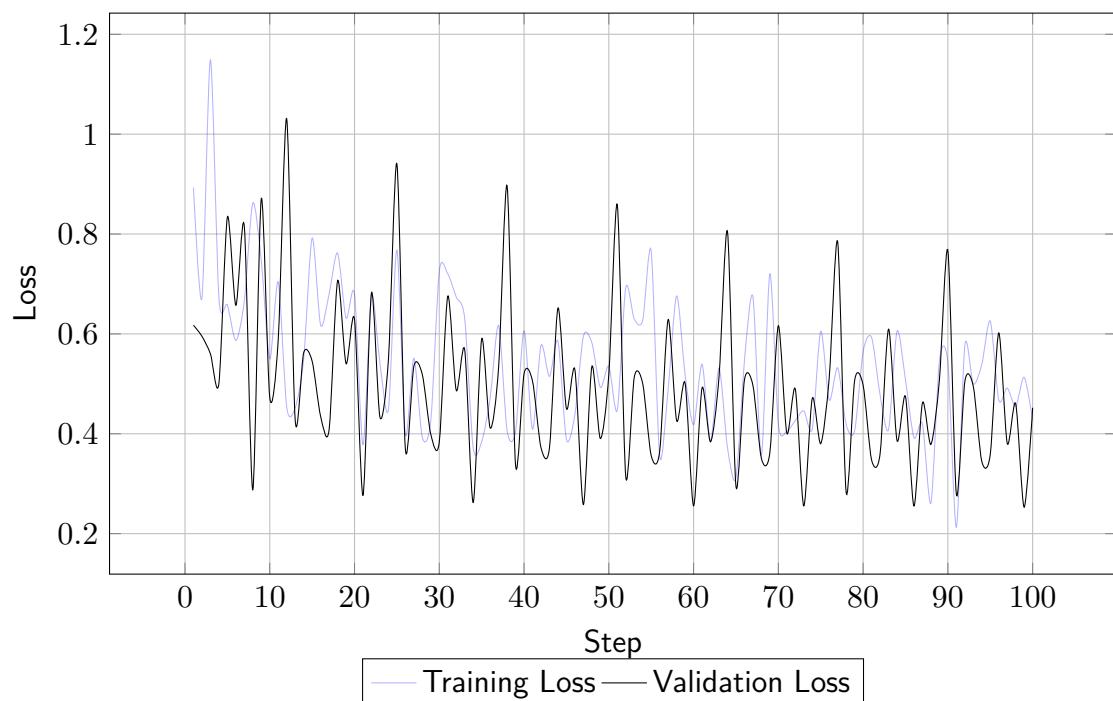


Figure 6.11: Training and validation losses for the fine-tuned Qwen2.5-Coder trained on the original dataset.

6.2 Experiments Result

Experiment 1: Effect of Temperature

In Experiment 1, the effect of temperature on the model's performance was evaluated. Table 6.1 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 same results, with 41 "Valid SAMM" outputs, and 32 "Correct" outputs. In Snippet 6.1, the actual output generated at different temperature settings is shown.

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%)

Table 6.1: Effect of temperature on the performance of fine-tuned GPT-4o-mini model (one-shot prompting) trained on the original data.

```

1 ######
2 # Temperature 0.0 #
3 #####
4 :classificationDescription a samm:Property ;
5   samm:preferredName "Classification Description"@en ;
6   samm:description "Description of the classification."@en ;
7   samm:characteristic samm-c:Text ;
8   samm:exampleValue "Standard data element types with associated classification
9     ↪ scheme for electric components." .
10
11 :ClassificationStandardCharacteristic a samm-c:Enumeration ;
12   samm:preferredName "Classification Standard Characteristic"@en ;
13   samm:description "The standard according which the part is classified."@en ;
14   samm:dataType xsd:string ;
15   samm-c:values ( "AATCC" "ANSI" "AS" "ASME" "ASTM" "DIN" "DIN EN" "EN" "GB"
16     ↪ "IEC" "IEEE" "ISO" "IS" "JASO" "JIS" "MIL" "SAE" "UL" ) .
17 #####
18 # Temperature 0.7 #
19 #####
20 :classificationDescription a samm:Property ;
21   samm:preferredName "Classification Description"@en ;
22   samm:description "Description of the classification standard entry."@en ;
23   samm:characteristic samm-c:Text ;
24   samm:exampleValue "Standard data element types with associated classification
25     ↪ scheme for electric components." .

```

```

26   samm:description "The standard according to which the part is classified."@en ;
27   samm:see <https://www.iec.ch/> ;
28   samm:dataType xsd:string ;
29   samm-c:values ( "AIST" "AS" "ASME" "DIN" "DIN EN" "EN" "GB" "IEC" "IEEE" "ISO"
    ↵ "IS" "JASO" "JIS" ) .

```

Snippet 6.1: Comparison of the two generated elements using different temperatures.

Experiment 2: Effect of Examples and Number of Attempts

Experiment 2 explored the effect of example complexity and the number of attempts on the model's performance. Table 6.2 shows the results of this experiment. Using 'Waste' 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' 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.

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

Table 6.2: Effect of example complexity on the performance of fine-tuned GPT-4o-mini in one-shot prompting after three attempts (pass@3).

The effect of multiple attempts is shown in Table 6.3, 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.

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

Table 6.3: Effect of multiple attempts on the performance of the fine-tuned GPT-4o-mini (pass@3 metric) in one-shot prompting.

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

Experiment 3 compared the performance of different open-source models without fine-tuning. As seen in Table 6.4, 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. In Table 6.11, results of open-source and closed-source LLMs are presented together.

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

Table 6.4: Performance of open-source models without fine-tuning (one-shot prompting).

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

The impact of using more shots on Qwen2.5-Coder-7b and Llama 3.1 8B is presented in Table 6.5. 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.

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 (%)	0
Llama 3.1 8B (one-shot)	29 (30%)	0	0

Table 6.5: Comparison of one-shot and two-shot prompting on the Qwen2.5-Coder and Llama 3.1 base models.

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

The impact of batch size on fine-tuning was explored with GPT-4o-mini. As shown in Table 6.6, 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.

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

Table 6.7 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

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%)

Table 6.6: Effect of batch size on model performance with one-shot prompting trained on original data.

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.

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%)

Table 6.7: Performance of GPT-4o-mini under various prompting and fine-tuning scenarios.

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

The comparison of iterative prompting on fine-tuned GPT-4o-mini is shown in Table 6.8. 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.

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

The results for one-shot and two-shot for fine-tuned Qwen2.5-Coder are described in Table 6.9.

Table 6.10 presents GPU memory consumption of Qwen2.5-Coder with varying context window sizes. The memory usage was 5.3 GB at 2,500 and 5,000 tokens, 6.0 GB at 10,000 tokens, 7.1 GB at 20,000 tokens, 9.3 GB at 40,000 tokens, and increased to 13.9 GB at 80,000 tokens.

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)

Table 6.8: Performance of fine-tuned GPT-4o-mini using simple retries versus iterative feedback.

Summary of Experiments

Table 6.11 provides a comprehensive performance overview of various LLMs across different fine-tuning and prompt configurations. The results clearly highlight the superior capabilities of GPT-4o-mini, especially when fine-tuned, while Qwen2.5-Coder remains the most competitive open-source option.

6.3 Human Evaluation

The evaluation was conducted with six experienced modelers who provided feedback on the SAMM Copilot tool. Table 6.12 summarizes the quantitative and qualitative responses collected. Figure 6.12 visualizes the time taken to model manually versus using SAMM Copilot.

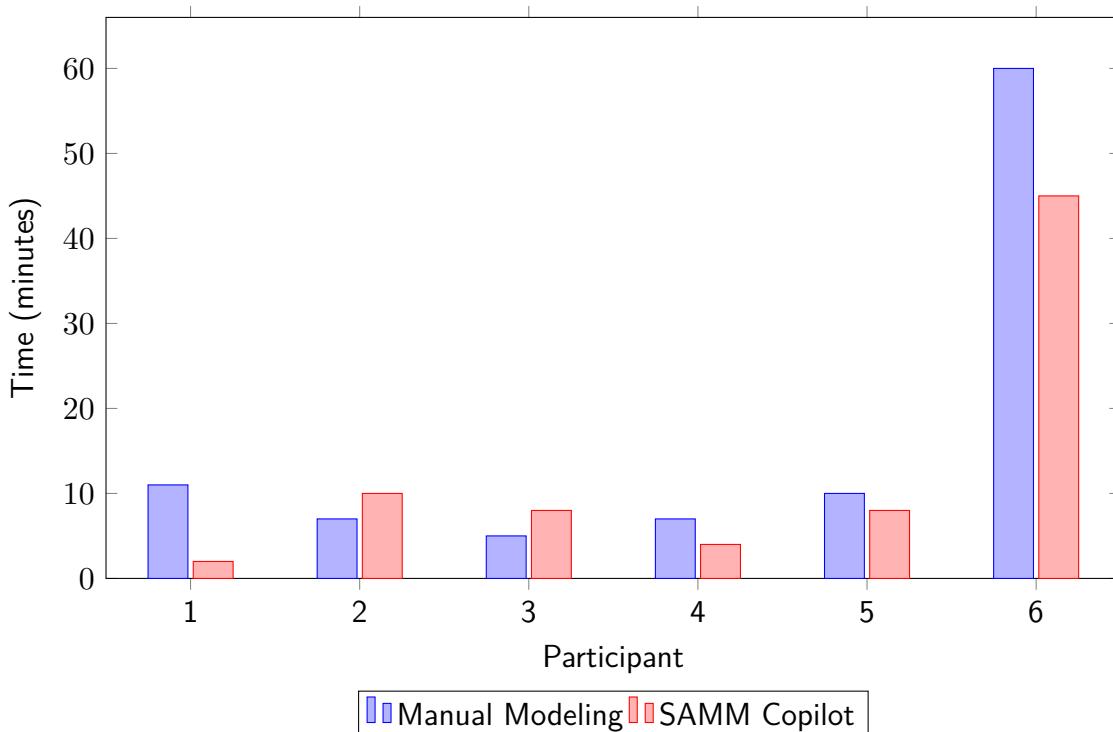


Figure 6.12: Comparison of time taken for manual modeling and using SAMM Copilot.

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%)

Table 6.9: Performance comparison of Qwen2.5-Coder 7B base and fine-tuned models.

Context Window Size	GPU Memory Usage (GB)
2,500	5.3
5,000	5.3
10,000	6.0
20,000	7.1
40,000	9.3
80,000	13.9

Table 6.10: GPU memory consumption of Qwen2.5-Coder 7B for varying context window sizes.

Model	Valid Turtle	Valid SAMM	Correct
Fine-tuned GPT-4o-mini (3 iterations)	86 (89%)	63 (65%)	42 (43%)
Fine-tuned GPT-4o-mini (one-shot)	85 (88%)	41 (42%)	32 (33%)
Fine-tuned GPT-4o-mini (two-shot)	80 (83%)	39 (40%)	31 (32%)
Fine-tuned GPT-4o-mini (zero-shot)	79 (82%)	20 (20%)	21 (21%)
Base GPT-4o-mini (two-shot)	83 (86%)	34 (35%)	9 (9%)
Fine-tuned Qwen2.5-Coder 7B (one-shot)	43 (44%)	10 (10%)	6 (6%)
Base GPT-4o-mini (one-shot)	88 (91%)	24 (25%)	4 (4%)
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%)
Fine-tuned Qwen2.5-Coder 7B (zero-shot)	53 (55%)	5 (5%)	1 (1%)
Base Llama 3.1 8B (two-shot)	24 (25%)	4 (%)	0
Base GPT-4o-mini (zero-shot)	75 (78%)	0	0
Base Qwen2.5-Coder 7B (zero-shot)	40 (41%)	0	0
Base Qwen2.5-Coder 7B (one-shot)	39 (40%)	0	0
Base CodeLlama 7B (one-shot)	31 (32%)	0	0
Base Llama 3.1 8B (one-shot)	29 (30%)	0	0
Base Llama 3.2 3B (one-shot)	5 (5%)	0	0

Table 6.11: Summary of the performance of different LLMs.

Participant	Manual Modeling Time	External Sources Used	SAMM Copilot Time	Satisfaction with SAMM Copilot
1	11 minutes	No	2 minutes	Yes, valid and complete model
2	7 minutes	No	>10 minutes	No, invalid model
3	5 minutes	None	8 minutes	Yes, valid but incomplete model
4	7 minutes	SAMM specification, Smart Plug Datasheet	4 minutes	Yes, valid but incomplete model
5	60 minutes	ESMF documentation	45 minutes	No, valid but incorrect model
6	10 minutes	No	Less than 3 minutes	No, invalid model

Table 6.12: Summary of human evaluation results.

7 Discussion

7.1 Fine-Tuning Insights

An ideal loss curve for LLM fine-tuning should initially start with a higher value and gradually decrease as the model adapts to the new data. Ideally, the curve should plateau once the model reaches optimal performance. Sharp fluctuations in the loss curve or long plateaus may indicate issues such as insufficient data, improper learning rates, or overfitting.

However, token loss (like cross-entropy loss) may not always be the best indicator for fine-tuning LLMs, especially for complex code generation tasks. While token loss helps optimize the model's ability to predict the next token, it does not necessarily correlate with task-specific performance, such as generating correct semantic models.

However, looking at the loss charts for GPT-4o-mini, depicted in Figure 6.2 and Figure 6.4, an overall decline trend can be observed. However, this behavior is not visible in the loss charts for fine-tuning GPT-4o, shown in Figure 6.6. This may be due to the fact that more hyperparameter tuning, especially adjusting the learning rate multiplier or additional epochs, may be required to properly fine-tune the larger GPT-4o model. As shown in Figure 6.11, such a decline pattern is also visible for the fine-tuning of Qwen2.5-Coder.

7.2 Experiments

Experiment 1: Effect of Temperature

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 such as `samm:description` might become more apparent with varied temperature settings, as they are more sensitive to next-token probability distributions. As shown in Snippet 6.1, higher temperature settings sometimes resulted in the model adding more elements like `samm:see`.

Experiment 2: Effect of Examples and Number of Attempts

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

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 semantic modeling tasks, and the need for fine-tuning these models.

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

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

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

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

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

The results in Table 6.9 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.

GPU Consumption of Qwen2.5-Coder Relative to Context Window Size

The GPU memory usage of Qwen2.5-Coder scales with the size of the context window. This highlights the trade-off between extending the context window for improved model performance and the associated GPU resource allocation.

7.3 Human Evaluation

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. However, occasional issues such as the "Insufficient Funds" error and unclear error messages extended the effective usage time for some users, which reduced the tool's overall time-saving benefit.

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.

Error handling and usability issues emerged as notable challenges. Several participants encountered vague error messages while trying to integrate the generated Aspect Model into the Aspect Model Editor. These messages lacked clear guidance on how to resolve the issues. For example, one user struggled with missing prefixes in the generated models, which prevented successful integration. Such challenges emphasize the need for better error communication to address integration issues effectively. Additionally, this can be seen as a limitation of the Aspect Model Editor itself, as some error messages are very generic and do not help to resolve and locate the problem. A direct integration of the generation approach into the Aspect Model Editor, which is capable of automatically handling errors in the background, can significantly ease the process.

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.

7.4 Ethical Considerations

When SAMM Copilot or similar LLMs are trained on datasets under the Creative Commons Attribution 4.0 International License (CC-BY-4.0), the LLM learns patterns for constructing SAMM Aspect Models rather than directly copying content. As the LLM generalizes from training ex-

amples, the outputs generated are intended to be novel, although they may sometimes look like existing models.

In scenarios where the LLM does not create entirely new models but rather links to existing models or draws upon learned patterns to suggest related content, attribution may not be necessary. Here, the model is augmenting the knowledge base, connecting users to relevant content, rather than reproducing specific work. However, when the model generates a new SAMM Aspect Model that is entirely similar to an existing one, especially if the original work is the intellectual property of others, attribution becomes essential to avoid unfair usage.

Commercializing a tool like SAMM Copilot introduces additional complexities. The use of CC-BY-licensed data in commercial applications must be carefully managed to ensure that the attribution requirements of the license are respected. Failing to properly attribute the original creators when the tool generates outputs that resemble their work could result in legal or ethical complications. Additionally, developers of commercial tools must consider the reputational risks of using someone else's work without proper credit, as well as the potential impact on user trust.

To mitigate these concerns, developers should ensure that the model does not unfairly replicate existing work. Instead, the tool should facilitate linking to existing models or suggesting related models, rather than generating exact replicas. Clear and transparent attribution, as well as proper licensing for commercial use, must be implemented to ensure that the tool complies with the legal requirements of the CC-BY-4.0 license.

8 Conclusion and Future Work

This thesis 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 the effectiveness of them 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 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 can be used for future studies. This work also outlined how LLMs can be used by end-users and integrated with existing tools for generating semantic models.

This thesis 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. These experiments highlighted the effectiveness of LLMs in generating semantic models. 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. These implementations highlighted the importance of usability and interaction design in promoting effective use of LLMs in semantic modeling tasks.
- **RQ6: Leveraging RDF-based knowledge graphs for SAMM and AAS.** Representing AAS and SAMM as RDF-based knowledge graphs provides a powerful framework for digital twins, enhancing data interoperability and enabling advanced capabilities. LLMs were shown to utilize these knowledge graphs effectively, performing tasks such as question-answering and knowledge discovery.

These findings collectively demonstrate the potential of LLMs to address key challenges in semantic modeling, showcasing their capability to automate, validate, and enhance the development of semantic models.

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. However, proprietary licensing often restricts the commercial application of model distillation across different providers, limiting its practical implementation outside a specific provider's ecosystem, such as OpenAI. Despite these constraints, 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 Unslot library. In the future, as more advanced models become compatible with libraries like Unslot, they can be included in similar studies to enhance outcomes and broaden the scope of

comparisons. Furthermore, 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.

For commercial LLMs, the scope was limited to OpenAI's GPT models. Due to budget and time constraints, the effect of all hyperparameters, such as the learning rate multiplier, could not be explored. Moreover, other providers, such as Claude (Anthropic), Google Gemini, and additional platforms offering fine-tuning capabilities, could be considered in future research. Budget constraints were the primary reason for the exclusion of these alternatives from the current study. Future work with greater resources could investigate these models to provide a more comprehensive analysis.

A1 Prompt Guidelines

The following prompts were used during the iterative prompting as feedback to the model. When the model is not capable of generating a valid RDF, some hints about the structure of RDF Turtle are provided as context, which is depicted in Snippet A1.1. When the issue is due to the structural differences between input JSON and generated JSON from LLM output Aspect Model, some instructions are provided to guide the LLM, outlined in Snippet A1.2. In other cases, Snippet A1.3 will be used in the prompt as a guideline.

```

1 You can follow these instructions for Generating a Valid Turtle Document
2 Start with Prefixes (Optional but Recommended):
3
4     Declare namespaces using the @prefix or PREFIX keyword followed by a prefix,
5         ↪ a colon, and the IRI in angle brackets.
6     End the declaration with a period (.).
7     Example:
8         ```turtle
9             @prefix ex: <http://example.org/> .
10
11 Define a Subject:
12
13     A subject is a resource (IRI or blank node) about which you want to make
14         ↪ statements.
15 Represent it using:
16     An IRI enclosed in angle brackets, e.g., <http://example.org/resource1>, or
17     A prefixed name like ex:Resource1.
18 Connect the Subject to Properties (Predicates):
19
20     Use predicates (verbs) to describe relationships or attributes of the subject.
21     Represent predicates using:
22     An IRI in angle brackets, e.g., <http://example.org/property>, or
23     A prefixed name like ex:Property.
24 Provide Object Values:
25
26     Objects can be:
         IRIs (as subjects or resources),
         Literals (data values like strings, numbers, dates),

```

27 Blank nodes (anonymous resources).
28 Literals should be enclosed in quotes, e.g., "Alice", or formatted as
 ↳ datatype literals, e.g.,
 ↳ "42"^^<<http://www.w3.org/2001/XMLSchema#integer>>.
29 End Statements with a Period:
30
31 Each triple (subject-predicate-object) must end with a period (.).
32 Group Properties for the Same Subject:
33
34 Use a semicolon (;) to separate multiple predicates for the same subject.
35 Example:
36 ```turtle
37 ex:Resource1 ex:hasName "Alice";
38 ex:hasAge 30 .
39 ```
40 Use Commas to List Multiple Objects for the Same Predicate:
41
42 Use commas (,) to separate multiple objects for a single predicate.
43 Example:
44 ```turtle
45 ex:Resource1 ex:hasFriend ex:Resource2, ex:Resource3 .
46 ```
47 Include Comments for Clarity:
48
49 Use the # symbol for comments, which are ignored by parsers.
50 Example:
51 ```turtle
52 # This is a comment
53 ```
54 Follow RDF Data Model Constraints:
55
56 Ensure that:
57 Subjects and predicates are always IRIs or prefixed names.
58 Objects can be IRIs, prefixed names, literals, or blank nodes.
59
60 Example valid RDF:
61 ```turtle
62 @prefix : <<urn:smm:com.mycompany.myapplication:1.0.0#>> .
63 @prefix samm: <<urn:smm:org.eclipse.esmf.samm:meta-model:2.1.0#>> .
64 @prefix samm-c: <<urn:smm:org.eclipse.esmf.samm:characteristic:2.1.0#>> .
65 @prefix samm-e: <<urn:smm:org.eclipse.esmf.samm:entity:2.1.0#>> .
66 @prefix unit: <<urn:smm:org.eclipse.esmf.samm:unit:2.1.0#>> .
67 @prefix xsd: <<http://www.w3.org/2001/XMLSchema#>> .

```

68 :MyAspect a samm:Aspect ;
69   samm:preferredName "My Aspect"@en ;
70   samm:preferredName "Mein Aspekt"@de ;
71   samm:description "This Aspect is an example."@en ;
72   samm:description "Dieser Aspekt ist ein Beispiel."@de ;
73   samm:properties ( :materialNumber ) ;
74   samm:operations ( ) ;
75   samm:events ( ) .

76
77
78 :materialNumber a samm:Property ;
79   samm:preferredName "Material number"@en ;
80   samm:description "A material number"@en ;
81   samm:exampleValue "ABC123456-000" ;
82   samm:characteristic samm-c:Text .

83
84 ```

```

Snippet A1.1: Hints inserted in the prompt when occurred exception in previous attempt was related to RDF.

```

1 * Relation between JSON and SAMM Aspect Model:
2 For understanding the construction rules, we define the following terms:
3
4 A Constraint of the 'Propertys Characteristic defined with a Trait is applied to
   ↳ the characteristic referred in the samm-c:baseCharacteristic .
5
6 A 'Propertys effective data type means the 'Propertys 'Characteristics
   ↳ samm:dataType.
7
8 A data type is scalar, if it is one of the allowed data types, but not a
   ↳ samm:Entity.
9
10 In order to create JSON payloads that correspond to an Aspect Model, the
    ↳ following rules are applied. The other way round they can also be used to
    ↳ describe a validation algorithm.
11
12 An Aspect Model is always serialized as an unnamed JSON object.
13
14 For each Property:
15
16 If it is marked as optional, it may or may not be included in the payload. If,
   ↳ and only if, the Property is marked as optional and is included in the
   ↳ payload, then its value may be null, which is equivalent to it not being

```

```

17      ↳ included in the payload.

18 If the 'Property's effective data type is scalar with any date type other than
    ↳ rdf:langString, the Property is serialized as ${propertyName}: ${value}
    ↳ where ${value} is the JSON serialization of the respective 'Property's
    ↳ value, details on mapping of the data types are given in Data type
    ↳ mappings. The value must adhere to the value range defined by the
    ↳ 'Property's effective data type and possible Constraints on the 'Property's
    ↳ Characteristic.

19
20 If the 'Property's effective data type is scalar with the data type
    ↳ rdf:langString, the Property is serialized as a named JSON object (with
    ↳ ${propertyName} being the name of the JSON property), with keys for each
    ↳ available language tag of the Property and the corresponding localized
    ↳ string as the value.

21
22 If the 'Property's effective data type is not scalar, it is serialized as a named
    ↳ JSON object (with ${propertyName} being the name of the JSON property),
    ↳ recursively using the same rules.

23
24 If the 'Property's effective data type is an Entity which extends another Entity,
    ↳ it is serialized as a named JSON object (with ${propertyName} being the
    ↳ name of the JSON property). The Properties included for the Entity, are
    ↳ the Properties from the Entity itself as well as all Properties from the
    ↳ extended Entities, i.e. all Properties from ?thisEntity samm:extends* [].

25
26 If the 'Property's Characteristic is a Collection, List, Set or Sorted Set, it is
    ↳ serialized as a named JSON array (with ${propertyName} being the name of
    ↳ the JSON array property).

27
28 Characteristics defined in the Aspect Model other than the ones mentioned above
    ↳ are not subject to serialization.

29
30 Operations defined in the Aspect Model are not subject to serialization.

31
32 Events defined in the Aspect Model are not subject to serialization.

33 1- Example JSON:
34 {
35   "isMoving": true,
36   "speed": 0.5
37 }
38 1- Expected SAMM Aspect Model:
39 @prefix : <urn:samm:com.mycompany.myapplication:1.0.0#> .

```

```

40 @prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#> .
41 @prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#> .
42 @prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#> .
43 @prefix unit: <urn:samm:org.eclipse.esmf.samm:unit:2.1.0#> .
44 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

45
46 :Movement a samm:Aspect ;
47   samm:preferredName "movement"@en ;
48   samm:properties ( :isMoving :speed ) ;
49   samm:operations ( ) ;
50   samm:events ( ) .

51
52 :isMoving a samm:Property ;
53   samm:preferredName "is moving"@en ;
54   samm:characteristic samm-c:Boolean .

55
56 :speed a samm:Property ;
57   samm:preferredName "speed"@en ;
58   samm:characteristic :Speed ;
59   samm:exampleValue "0.5"^^xsd:float .

60
61 :Speed a samm-c:Measurement ;
62   samm:preferredName "speed"@en ;
63   samm:dataType xsd:float ;
64   samm-c:unit unit:kilometrePerHour .

65
66
67 3- Example JSON:
68 {
69   "categoricalData": "enum1",
70   "property1": 0,
71   "multiLang": {
72     "en": "multilanguage"
73   },
74   "simpleKey": "simpleValue",
75   "complexList": [
76     {
77       "anotherKey": "value"
78     }
79   ],
80   "complex": {
81     "property2": "2024-11-29T02:02:49.228+01:00",
82     "property3": true

```

```

83     }
84 }
85
86 3-Expected SAMM Aspect Model
87 @prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#> .
88 @prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#> .
89 @prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#> .
90 @prefix unit: <urn:samm:org.eclipse.esmf.samm:unit:2.1.0#> .
91 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
92 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
93 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
94 @prefix : <urn:samm:example.example.contact_property:7.0.0#> .
95
96 :ContactProperty a samm:Aspect ;
97   samm:preferredName "Contact Property"@en ;
98   samm:description "The contact property aspect contains essential contact
99     ↳ information such as fax number, website, phone number, and email."@en ;
100    samm:properties ( :simpleKey :complexList :complex :categoricalData :multiLang
101      ↳ :property1 ) ;
102    samm:operations ( ) ;
103    samm:events ( ) .
104
105 :simpleKey a samm:Property ;
106   samm:characteristic samm-c:Text ;
107   samm:exampleValue "simpleValue" .
108
109 :complexList a samm:Property ;
110   samm:characteristic :ComplexValueCharacteristic .
111
112 :complex a samm:Property ;
113   samm:characteristic :Characteristic1 .
114
115 :categoricalData a samm:Property ;
116   samm:characteristic :Characteristic2 .
117
118 :multiLang a samm:Property ;
119   samm:characteristic samm-c:MultiLanguageText ;
120   samm:exampleValue "multilanguage"@en .
121
122 :property1 a samm:Property ;
123   samm:characteristic :Characteristic3 .
124
125 :ComplexValueCharacteristic a samm-c>List ;

```

```

124 :sammm:dataType :DataTypeOfComplexValue .
125
126 :Characteristic1 a samm:Characteristic ;
127   samm:datatype :Entity1 .
128
129 :Characteristic2 a samm-c:Enumeration ;
130   samm:datatype xsd:string ;
131   samm-c:values ( "enum1" "enum2" ) .
132
133 :Characteristic3 a samm-c:Measurement ;
134   samm:datatype xsd:float ;
135   samm-c:unit unit:angstrom .
136
137 :DataTypeOfComplexValue a samm:Entity ;
138   samm:properties ( :anotherKey ) .
139
140 :Entity1 a samm:Entity ;
141   samm:properties ( :property2 :property3 ) .
142
143 :anotherKey a samm:Property ;
144   samm:characteristic :AnotherValueCharacteristic ;
145   samm:exampleValue "value" .
146
147 :property2 a samm:Property ;
148   samm:characteristic samm-c:Timestamp .
149
150 :property3 a samm:Property ;
151   samm:characteristic samm-c:Boolean .
152
153 :AnotherValueCharacteristic a samm:Characteristic ;
154   samm:datatype xsd:string .

```

Snippet A1.2: Hints inserted in the prompt when occurred exception in previous attempt was related to JSON.

```

1 Use the following hints for correction:
2 * General RDF and SAMM Model Rules:
3
4   Output Format: Always ensure the output is a valid RDF Turtle file. Don't
      ↢ write any explanation.
5   Model Validity: Adhere to SAMM standards to create valid Aspect Models.
6
7 * Properties and Characteristics:
8

```

9 Each Entity or Aspect must have exactly one samm:properties field containing
10 ↳ a list of properties.
11 Example (Invalid):
12 ```turtle
13 :Person a samm:Entity ;
14 samm:properties (:name) ;
15 samm:properties ([samm:property :surname ; samm:payloadName "familyName"
16 ↳]) .
17 ```\n
18 Example (Correct):
19 ```turtle
20 :Person a samm:Entity ;
21 samm:properties (:name [samm:property :surname ; samm:payloadName
22 ↳ "familyName"]) .
23 ```\n
24 Every samm:Property must have a samm:characteristic.
25 A Property and its Characteristic cannot have the same name.
26 Ensure the samm:dataType of a property matches the data type of its
27 ↳ characteristic.
28
29 Example (Invalid):
30 ```turtle
31 :ToolTemperature a samm-c:Measurement ;
32 samm:datatype xsd:float ;
33 samm-c:unit unit:degreeCelsius .
34
35 ```\n
36 Example (Correct):
37 ```turtle
38 :ToolTemperature a samm-c:Measurement ;
39 samm:datatype xsd:float ;
40 samm-c:unit unit:degreeCelsius .
41
42 ```\n
43 :drillHeadTemperature a samm:Property ;
44 samm:characteristic :ToolTemperature ;
45 samm:exampleValue "12"^^xsd:int .
46
47 ```\n
48 * Error Handling:

```
48 If you encounter errors like "No model file", remove dependencies or
  ↪ references to external namespaces.
49 If the example value type is incorrect, either correct it or remove the
  ↪ samm:exampleValue.
50
51 * Constraints on Properties:
52
53 A Property must not have both samm:notInPayload true and samm:payloadName.
54 If Property name cannot have special characters like "@id" or "@type", you
  ↪ need to use alphanumeric name for Property and use samm:payloadName
55 Correct Example:
56 ````turtle
57 :Person a samm:Entity ;
58   samm:properties ( :name [ samm:property :surname ; samm:payloadName
  ↪ "@SName" ] ) .
59 ````

60 * Mandatory Fields:
61 All model elements must have:
62   samm:preferredName: Human-readable, uses normal orthography, no camelCase.
63   samm:description: Short, comprehensible, and consistent.
64   preferredName and description must not be identical.
65 * Traits and Enumerations:
66 A Trait adds constraints to a "base characteristic." Traits inherit the data
  ↪ type from their samm-c:baseCharacteristic and do not define their own
  ↪ samm:datatype.
67 For enumerations, list valid literal values with samm-c:values.
68 Example:
69 ````turtle
70 :Status a samm-c:Enumeration ;
71   samm:datatype xsd:string ;
72   samm-c:values ( "Complete" "In Progress" "Created" ) .
73 ````

74 * Measurements:
75
76 A samm-c:Measurement must include a samm-c:unit.
77 Example:
78 ````turtle
79 :TemperatureMeasurement a samm-c:Measurement ;
80   samm:datatype xsd:float ;
81   samm-c:unit unit:degreeCelsius .
82 ````

83
84 * Aspect Model Naming:
```

85 Names must be singular unless the aspect contains a Collection, List, or Set as
86 ↳ its only property, in which case plural naming is allowed.

87 * External Standards:
88 Use `samm:see` to reference external standards.

89 * Units:
90 Always use units from the SAMM unit catalog when applicable.

91 * Constraints:
92
93 Explicitly define known constraints for a use case within the Aspect Model.

94
95 * Example Values:
96 All properties with simple types should have an `samm:exampleValue` that matches
97 ↳ the data type of the characteristic.

98 Characteristics Classes: Guidelines for Valid Instances

99 1- Allowed Instances

100 Only specific instances or subclasses within the `samm-c` namespace
101 ↳ (`urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#`) are valid.
102 If invalid instances (e.g.,
103 ↳ `urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#Map` or
104 ↳ `samm-c:Map`) are found, they must be replaced with one of the valid
105 ↳ characteristics listed below.

106 2- Valid Characteristics and Their Required Attributes

107 Ensure each characteristic includes its mandatory attributes for the model
108 ↳ to be valid:

109 `samm-c:Trait`
110 `samm-c:Quantifiable`
111 `samm-c:Measurement`
112 Required attributes: `samm-c:unit`
113 `samm-c:Enumeration`
114 Required attributes: `samm-c:values`
115 `samm-c:State`
116 Required attributes: `samm-c:defaultValue`
117 `samm-c:Duration`
118 Required attributes: `samm-c:unit`
119 Example values: `unit:week`, `unit:year`, `unit:month`, `unit:millisecond`
120 `samm-c:Collection`
121 `samm-c>List`
122 `samm-c:Set`
123 `samm-c:SortedSet`

```
121     samm-c:TimeSeries
122         Required attributes: samm:dataType
123     samm-c:Code
124 3- Discouraged Characteristics
125      While the following characteristics are valid, avoid using them unless
126      ↪ necessary:
127      samm-c:Either
128          Required attributes: samm-c:left, samm-c:right
129      samm-c:SingleEntity
130      samm-c:StructuredValue
131          Required attributes: samm-c:deconstructionRule, samm-c:elements
132 4- Important Notes
133
134      Always include the required attributes for any characteristic. Models
135      ↪ missing these attributes are invalid.
136
137 * Characteristics Instances
138 These are already instances of Characteristics. If you see
139     ↪ `:ExampleCharacteristic a samm-c:Timestamp` then it is wrong because
140     ↪ samm-c:Timestamp is already an instance and you need to correct it.
141 - samm-c:Timestamp (Data Type: xsd:dateTime)
142 - samm-c:Text (Data Type: xsd:string)
143 - samm-c:MultiLanguageText (Data Type: rdf:langString)
144
145 - samm-c:Language
146     Description: Represents a property containing a language as per ISO 639-1
147     ↪ (e.g., "en" for English).
148     Data Type: xsd:string
149
150 - samm-c:Locale
151     Description: Represents a property containing a locale as per IETF BCP 47
152     ↪ (e.g., "en-US").
153     Data Type: xsd:string
154
155 - samm-c:Boolean
156     Description: Represents a property containing a boolean value (true/false).
157     Data Type: xsd:boolean
158
159 - samm-c:ResourcePath
160     Description: Represents a property containing the path to a relative or
161     ↪ absolute resource.
162     Data Type: xsd:anyURI
163
```

```
157 - samm-c:MimeType
158     Description: Represents a property containing a MIME type as defined by RFC
159         ↪ 2046 (e.g., "application/json").
160     Data Type: xsd:string
161
162 - samm-c:UnitReference
163     Description: Represents a property containing a reference to a unit in the
164         ↪ Unit catalog.
165     Data Type: samm:curie
166
167 - samm-c:Reference
168     Description: Represents a property containing a reference to a concept such
169         ↪ as a resource or model element (e.g., URI or SAMM identifier).
170     Data Type: Based on samm:see reference conventions.
```

Snippet A1.3: Generic hints inserted in the prompt to help the model correct the output.

A2 Example Outputs

This appendix contains the output of 3 models using one-shot and zero-shot prompt. For zero-shot prompt, we use Snippet A2.1, and for one-shot, we use Snippet A2.2.

```

1 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:
2 {
3   "powerConsumption": 3.0,
4   "energyConsumption": 7590.0,
5   "energyConsumptionStartDate": "2024-10-25T15:25:24Z"
6 }
7 Provide only the SAMM model without explanation. Make sure you always give a
   ↵ valid RDF turtle as the SAMM model.

```

Snippet A2.1: Zero-shot prompt template for one example

```

1 This is an example SAMM model:
2 @prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#> .
3 @prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#> .
4 @prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#> .
5 @prefix unit: <urn:samm:org.eclipse.esmf.samm:unit:2.1.0#> .
6 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
7 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
8 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
9 @prefix : <urn:samm:io.catenax.shared.secondary_material_content:1.0.0#> .
10
11 :SecondaryMaterialContent a samm:Aspect ;
12 ##### OMITTED TO SAVE SPACE #####
13 This is its corresponding JSON example:
14 {
15   "secondaryMaterialContent" : [ {
16     ##### OMITTED TO SAVE SPACE #####
17 Your task is to create a SAMM model from a JSON Example.
18 Json Example:
19 {
20   "powerConsumption": 3.0,
21   "energyConsumption": 7590.0,

```

```

22 "energyConsumptionStartDate": "2024-10-25T15:25:24Z"
23 }
24 Provide only the SAMM model without explanation, make sure that the output is a
   ↳ valid RDF Turtle format.

```

Snippet A2.2: One-shot prompt template for one example

A2.1 GPT-4o-mini

Zero-Shot

```

1 @prefix ex: <http://example.org/> .
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5
6 ex:EnergyData a ex:EnergyConsumptionRecord ;
7   ex:powerConsumption 3.0^^xsd:float ;
8   ex:energyConsumption 7590.0^^xsd:float ;
9   ex:energyConsumptionStartDate "2024-10-25T15:25:24Z"^^xsd:dateTime .

```

Snippet A2.3: Output of GPT-4o-mini after applying zero-shot prompt template for one example

One-Shot

```

1 @prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#> .
2 @prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#> .
3 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
5 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
6 @prefix : <urn:samm:io.catenax.energy_consumption:1.0.0#> .
7
8 :EnergyConsumption a samm:Aspect ;
9   samm:preferredName "Energy Consumption"@en ;
10  samm:description "Model describing the energy consumption of a system."@en ;
11  samm:properties ( :powerConsumption :energyConsumption
12    ↳ :energyConsumptionStartDate ) ;
13  samm:operations ( ) ;
14  samm:events ( ) .
15
16 :powerConsumption a samm:Property ;
   samm:preferredName "Power Consumption"@en ;

```

```

17  samm:description "Amount of power consumed, in kilowatts."@en ;
18  samm:characteristic :PowerConsumptionCharacteristic ;
19  samm:exampleValue "3.0"^^xsd:float .

20
21 :energyConsumption a samm:Property ;
22   samm:preferredName "Energy Consumption"@en ;
23   samm:description "Total energy consumption in kilowatt-hours."@en ;
24   samm:characteristic :EnergyConsumptionCharacteristic ;
25   samm:exampleValue "7590.0"^^xsd:float .

26
27 :energyConsumptionStartDate a samm:Property ;
28   samm:preferredName "Energy Consumption Start Date"@en ;
29   samm:description "Start date for the energy consumption measurement."@en ;
30   samm:characteristic :EnergyConsumptionStartDateCharacteristic ;
31   samm:exampleValue "2024-10-25T15:25:24Z"^^xsd:dateTime .

32
33 :PowerConsumptionCharacteristic a samm:Characteristic ;
34   samm:preferredName "Power Consumption Characteristic"@en ;
35   samm:description "Characteristic of power consumption measured in
36     ↪ kilowatts."@en ;
37   samm:dataType xsd:float .

38 :EnergyConsumptionCharacteristic a samm:Characteristic ;
39   samm:preferredName "Energy Consumption Characteristic"@en ;
40   samm:description "Characteristic of total energy consumption measured in
41     ↪ kilowatt-hours."@en ;
42   samm:dataType xsd:float .

43 :EnergyConsumptionStartDateCharacteristic a samm:Characteristic ;
44   samm:preferredName "Energy Consumption Start Date Characteristic"@en ;
45   samm:description "Characteristic defining the start date of energy consumption
46     ↪ measurement."@en ;
        samm:dataType xsd:dateTime .

```

Snippet A2.4: Output of GPT-4o-mini after applying one-shot prompt template for one example

A2.2 Fine-tuned Qwen2.5-Coder

Zero-Shot

```

1 @prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#> .
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

```

```
4 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
5 @prefix : <urn:samm:io.catenax.energy_consumption:4.0.0#> .  
6  
7 :EnergyConsumption a samm:Aspect ;  
8   samm:preferredName "Energy Consumption"@en ;  
9   samm:description "Data structure representing the energy consumption of an  
10    ↳ asset."@en ;  
11   samm:properties ( :energyConsumption :powerConsumption  
12    ↳ :energyConsumptionStartDate ) ;  
13   samm:operations () ;  
14   samm:events () .  
15  
16 :energyConsumption a samm:Property ;  
17   samm:preferredName "Energy Consumption"@en ;  
18   samm:description "The total energy consumption of an asset during the  
19    ↳ reporting period."@en ;  
20   samm:characteristic :EnergyConsumptionCharacteristic ;  
21  
22 :powerConsumption a samm:Property ;  
23   samm:preferredName "Power Consumption"@en ;  
24   samm:description "The power consumption of an asset during the reporting  
25    ↳ period."@en ;  
26   samm:characteristic :PowerConsumptionCharacteristic ;  
27  
28 :energyConsumptionStartDate a samm:Property ;  
29   samm:preferredName "Energy Consumption Start Date"@en ;  
30   samm:description "The starting date of the reporting period."@en ;  
31   samm:characteristic :DateTimeCharacteristic ;  
32  
33   samm:exampleValue "2024-10-25T15:25:24Z"^^xsd:dateTime .  
34  
35 :EnergyConsumptionCharacteristic a samm:Characteristic ;  
36   samm:preferredName "Energy Consumption Characteristic"@en ;  
37   samm:description "Characterization of an energy consumption value."@en ;  
38   samm:dataType xsd:float .  
39  
40 :PowerConsumptionCharacteristic a samm:Characteristic ;  
41   samm:preferredName "Power Consumption Characteristic"@en ;
```

```

42    samm:description "The power consumption is the amount of electric energy that
43        ↳ an asset draws from the electrical grid to perform its functions."@en ;
44    samm:dataType xsd:float .
45
46 :DateTimeCharacteristic a samm:Characteristic ;
47     samm:preferredName "Date Time Characteristic"@en ;
48     samm:description "Characterization for date time values."@en ;
49     samm:dataType xsd:dateTime .

```

Snippet A2.5: Output of fine-tuned Qwen2.5-Coder after applying zero-shot prompt template for one example

One-Shot

```

1 @prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#> .
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5 @prefix : <urn:samm:io.catenax.energy_consumption:4.0.0#> .
6
7 :EnergyConsumption a samm:Aspect ;
8     samm:preferredName "Energy Consumption"@en ;
9     samm:description "This model describes the energy consumption of an industrial
10        ↳ site."@en ;
11     samm:properties ( :powerConsumption :energyConsumption
12         ↳ :energyConsumptionStartDate ) ;
13     samm:operations ( ) ;
14     samm:events ( ) .
15
16 :powerConsumption a samm:Property ;
17     samm:preferredName "Power Consumption"@en ;
18     samm:description "Amount of electrical power consumed, expressed in
19         ↳ kilowatt."@en ;
20     samm:characteristic :PowerConsumptionCharacteristic ;
21     samm:exampleValue "3.0"^^xsd:float .
22
23 :energyConsumption a samm:Property ;
24     samm:preferredName "Energy Consumption"@en ;
25     samm:description "Total energy consumption in kilowatt-hours."@en ;
26     samm:characteristic :EnergyConsumptionCharacteristic ;
27     samm:exampleValue "7590.0"^^xsd:float .
28
29 :energyConsumptionStartDate a samm:Property ;

```

```

27  samm:preferredName "Energy Consumption Start Date"@en ;
28  samm:description "Start date for the energy consumption measurement
   ↪ period."@en ;
29  samm:characteristic :EnergyConsumptionDateCharacteristic ;
30  samm:exampleValue "2024-10-25T15:25:24Z"^^xsd:dateTime .

31
32 :PowerConsumptionCharacteristic a samm:Characteristic ;
33  samm:preferredName "Power Consumption Characteristic"@en ;
34  samm:description "The power consumption expressed in kilowatt."@en ;
35  samm:dataType xsd:float .

36
37 :EnergyConsumptionCharacteristic a samm:Characteristic ;
38  samm:preferredName "Energy Consumption Characteristic"@en ;
39  samm:description "Energy consumption in kilowatt-hours."@en ;
40  samm:dataType xsd:float .

41
42 :EnergyConsumptionDateCharacteristic a samm:Characteristic ;
43  samm:preferredName "Energy Consumption Date Characteristic"@en ;
44  samm:description "Start date for the energy consumption measurement
   ↪ period."@en ;
45  samm:dataType xsd:dateTime .

```

Snippet A2.6: Output of fine-tuned Qwen2.5-Coder after applying one-shot prompt template for one example

A2.3 Fine-tuned GPT-4o-mini

Zero-Shot

```

1 @prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#>.
2 @prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#>.
3 @prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#>.
4 @prefix unit: <urn:samm:org.eclipse.esmf.samm:unit:2.1.0#>.
5 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
6 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
7 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
8 @prefix : <urn:samm:io.catenax.shared.electric_energy_meter:2.0.0#>.

9
10 :ElectricEnergyMeter a samm:Aspect;
11   samm:preferredName "Electric Energy Meter"@en;
12   samm:description "Aspect for devices measuring electric energy"@en;
13   samm:properties (:energyConsumption :energyConsumptionStartDate [
   ↪ samm:property :powerConsumption; samm:optional true ]);
```

```

14    samm:operations ();
15    samm:events ().

16 :energyConsumption a samm:Property;
17   samm:preferredName "Accumulated Energy Consumption"@en;
18   samm:description "Energy meter property representing the accumulated energy
19     ↪ consumption of the device."@en;
20   samm:characteristic :EnergyMeasure;
21   samm:exampleValue "7590.0"^^xsd:decimal.

22 :energyConsumptionStartDate a samm:Property;
23   samm:preferredName "Energy Consumption Start Date"@en;
24   samm:description "Start date for the recorded energy consumption. The value
25     ↪ will be reset to 0 after this date."@en;
26   samm:characteristic :TimestampTrait;
27   samm:exampleValue "2024-10-25T15:25:24Z"^^xsd:dateTime.

28 :powerConsumption a samm:Property;
29   samm:preferredName "Power Consumption"@en;
30   samm:description "Energy meter property representing the current power
31     ↪ consumption of the device."@en;
32   samm:characteristic :PowerCharacteristic;
33   samm:exampleValue "3.0"^^xsd:double.

34 :EnergyMeasure a samm:Characteristic;
35   samm:preferredName "Energy Measure"@en;
36   samm:description "Energy meter Characteristic measuring energy in Wh(Watt
37     ↪ hour)."@en;
38   samm:dataType xsd:decimal;
39   samm:see <https://www.samm.rocks/io.catenax.shared.energycategories>.

40 :TimestampTrait a samm-c:Trait;
41   samm-c:baseCharacteristic samm-c:Timestamp;
42   samm-c:constraint :DateAndTimeRegularExpression.

43 :PowerCharacteristic a samm:Characteristic;
44   samm:preferredName "Power Characteristic"@en;
45   samm:description "Energy meter Characteristic measuring electric power in W
46     ↪ (Watt)."@en;
47   samm:dataType xsd:double;
48   samm:see <https://www.samm.rocks/io.catenax.shared.powercategories>.

49 :DateAndTimeRegularExpression a samm-c:RegularExpressionConstraint;
50   samm:preferredName "Date and Time with Time Zone Regular Expression"@en;
51   samm:description "Regular Expression Constraint for Date and Time with Time
52     ↪ Zone."@en;
53   samm:value "(?=.*[Zz])(?!.*[a-zA-Z]{2,4}$)[0-9]{4}-(0[1-9]|1[0-2])
54     -(0[1-9]|1[12])[0-9]|3[01])T(0[0-9]|1[0-9]|2[0-3])
55     :([0-5][0-9]):([0-5][0-9])(.[0-9]+)?[Zz]".

```

Snippet A2.7: Output of fine-tuned GPT-4o-mini after applying zero-shot prompt template for one example

One-Shot

```

1 @prefix samm: <urn:samm:org.eclipse.esmf.samm:meta-model:2.1.0#> .
2 @prefix samm-c: <urn:samm:org.eclipse.esmf.samm:characteristic:2.1.0#> .
3 @prefix samm-e: <urn:samm:org.eclipse.esmf.samm:entity:2.1.0#> .
4 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
6 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
7 @prefix : <urn:samm:io.catenax.energy_agent:3.0.0#> .

8
9 :EnergyAgent a samm:Aspect ;
10    samm:preferredName "Energy Agent"@en ;
11    samm:description "This model refers to one specific digital twin. In contrast
12      ↳ to the model EnergyConsumptionProperties, which describes the
13      ↳ properties, characteristics and correlation to other models, all
14      ↳ properties are directly embedded in this model. Only use this model when
15      ↳ referring to one specific Agent."@en ;
16    samm:properties ( :powerConsumption :energyConsumption
17      ↳ :energyConsumptionStartDate ) ;
18    samm:operations ( ) ;
19    samm:events ( ) .

20
21 :powerConsumption a samm:Property ;
22    samm:preferredName "Power Consumption"@en ;
23    samm:description "The current power consumption in watts."@en ;
24    samm:characteristic :PowerConsumptionCharacteristic ;
25    samm:exampleValue "3.0"^^xsd:float .

26
27 :energyConsumption a samm:Property ;
28    samm:preferredName "Energy Consumption"@en ;
29    samm:description "The energy consumption since the energyConsumptionStartDate
      ↳ in watt hours (Wh)."@en ;
30    samm:characteristic :EnergyConsumptionCharacteristic ;
31    samm:exampleValue "7590.0"^^xsd:float .

32
33 :energyConsumptionStartDate a samm:Property ;
34    samm:preferredName "Energy Consumption Start Date"@en ;

```

```

30    samm:description "The date and time when the energy measurement period
31      ↳ started. The date and time will be set to zero to create an absolute
32      ↳ measurement regardless of the time zone."@en ;
33
34 :PowerConsumptionCharacteristic a samm:Characteristic ;
35   samm:preferredName "Power Consumption Characteristic"@en ;
36   samm:description "The current power consumption in watts."@en ;
37   samm:dataType xsd:float .
38
39 :EnergyConsumptionCharacteristic a samm-c:Quantifiable ;
40   samm:preferredName "Energy Consumption Characteristic"@en ;
41   samm:description "The energy consumption since the energyConsumptionStartDate
42     ↳ in watt hours (Wh)."@en ;
43   samm:dataType xsd:float .
44
45 :EnergyConsumptionStartDateTrait a samm-c:Trait ;
46   samm:preferredName "Energy Consumption Start Date Trait"@en ;
47   samm:description "Trait for the EnergyConsumptionStartDate with Regex and
48     ↳ Compatibility to dateTime."@en ;
49   samm:see <https://www.odata.org/static/odatav4/abpl/updates/4.01/4.01.pdf> ;
50   samm-c:baseCharacteristic :DateTimeStamp ;
51   samm-c:constraint :DateTimeStampConstraint .
52
53 :DateTimeStamp a samm:Characteristic ;
54   samm:preferredName "Date Time Stamp"@en ;
55   samm:description "Date and Time with an optional timezone."@en ;
56   samm:dataType xsd:dateTime .
57
58 :DateTimeStampConstraint a samm-c:RegularExpressionConstraint ;
59   samm:preferredName "Date Time Stamp Constraint"@en ;
60   samm:description "Regular Expression which allows UTC and Timezone."@en ;
61   samm:value
62     ↳ "([-+]?(([1-9][0-9]{3}|0[0-9]{3})|[1-9][0-9]{3}))(-((0[1-9]|1-9)[0-9])([0-3][0-9])|[-].
63     ↳ .

```

Snippet A2.8: Output of fine-tuned GPT-4o-mini after applying one-shot prompt template for one example

Bibliography

- [ACHI23] ACHIAM, Josh; ADLER, Steven; AGARWAL, Sandhini; AHMAD, Lama; AKKAYA, Ilge; ALEMAN, Florencia L.; ALMEIDA, Diogo; ALTENSCHMIDT, Janko; ALTMAN, Sam; ANADKAT, Shyamal [u. a.]: Gpt-4 technical report. In: *arXiv preprint arXiv:2303.08774* (2023).
- [ADMInd] ADMIN-SHELL.IO: *Specification of the Asset Administration Shell: Asset Administration Shell Documentation*. <https://admin-shell-io.github.io/aas-specs-antora/index/home/index.html>. n.d.. – Accessed on 2025-01-12.
- [ALLE] ALLEGROGRAPH BLOG: *Graphs without Semantics Are Not Enough*. <https://allegrograph.com/property-graphs-are-not-enough/>. – Accessed on 2025-01-12.
- [BAAZ17] BAAZIZI, Mohamed-Amine; LAHMAR, Houssem B.; COLAZZO, Dario; GHELLI, Giorgio ; SARTIANI, Carlo: Schema inference for massive JSON datasets. In: *Extending Database Technology (EDBT)*, 2017.
- [BABA23] BABAEI GIGLOU, Hamed; D’SOUZA, Jennifer ; AUER, Sören: LLMs4OL: Large language models for ontology learning. In: *International Semantic Web Conference* Springer, 2023, S. 408–427.
- [BAUE16] BAUERNHANSL, Thomas; KRÜGER, Jörg; REINHART, Gunther ; SCHUH, Günther: Wgp-standpunkt industrie 4.0. (2016).
- [ECLI] ECLIPSE SEMANTIC MODELING FRAMEWORK (ESMF): *Meta Model Elements — Eclipse Semantic Modeling Framework (ESMF)*. <https://eclipse-esmf.github.io/samm-specification/snapshot/meta-model-elements.html>. – Accessed on 2025-01-12.
- [FATH24] FATHALLAH, Nadeen; DAS, Arunav; DE GIORGIS, Stefano; POLTRONIERI, Andrea; HAASE, Peter ; KOVRIGUINA, Liubov: NeOn-GPT: A Large Language Model-Powered Pipeline for Ontology Learning. In: *The Semantic Web: ESWC Satellite Events 2024* (2024).
- [GÄNG11] GÄNGLER, Dipl Med-Inf T.: Semantic Federation of Musical and Music-Related Information for Establishing a Personal Music Knowledge Base. (2011).

- [GRIE14] GRIEVES, Michael: Digital twin: manufacturing excellence through virtual factory replication. In: *White paper 1* (2014), Nr. 2014, S. 1–7.
- [HELU15] HELU, Moneer; HEDBERG JR, Thomas: Enabling smart manufacturing research and development using a product lifecycle test bed. In: *Procedia manufacturing* 1 (2015), S. 86–97.
- [HOČE24] HOČEVAR, Domen; KENDA, Klemen: *Integrating Knowledge Graphs and Large Language Models for Querying in an Industrial Environment*, Bachelor's Thesis. University of Ljubljana, Faculty of Computer, Information ..., Diss., 2024.
- [HU21] HU, Edward J.; SHEN, Yelong; WALLIS, Phillip; ALLEN-ZHU, Zeyuan; LI, Yuanzhi; WANG, Shean; WANG, Lu ; CHEN, Weizhu. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021.
- [HUI24] HUI, Binyuan; YANG, Jian; CUI, Zeyu; YANG, Jiaxi; LIU, Dayiheng; ZHANG, Lei; LIU, Tianyu; ZHANG, Jiajun; YU, Bowen; DANG, Kai [u. a.]: Qwen2. 5-Coder Technical Report. In: *arXiv preprint arXiv:2409.12186* (2024).
- [INTE23] INTERNATIONAL ELECTROTECHNICAL COMMISSION (IEC): *IEC 63278-1:2023 — Asset Administration Shell for Industrial Applications - Part 1: Asset Administration Shell Structure*. <https://webstore.iec.ch/en/publication/65628>. 2023. – Available online: <https://webstore.iec.ch/en/publication/65628>, accessed on 2025-01-12.
- [INTE24] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO); INTERNATIONAL ELECTROTECHNICAL COMMISSION (IEC): *ISO/IEC 39075:2024*. <https://www.iso.org/standard/76120.html>. 2024. – Accessed on 2025-01-12.
- [KÖCH20] KÖCHER, Aljosha; HILDEBRANDT, Constantin; DA SILVA, Luis Miguel V. ; FAY, Alexander: A formal capability and skill model for use in plug and produce scenarios. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* Bd. 1 IEEE, 2020, S. 1663–1670.
- [KÜMP21] KÜMPEL, Michaela; MUELLER, Christian A. ; BEETZ, Michael: 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, S. 129–153.
- [KUNZ24] KUNZE, Franz C.; FAY, Alexander: Automated Generation of AML Models for Industrial Plants Using LLM Chat Applications. In: *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)* IEEE, 2024, S. 1–4.
- [LASS01] LASSILA, Ora; HENDLER, J ; BERNERS-LEE, T: The semantic web. In: *Scientific*

- American 284 (2001), Nr. 5, S. 34–43.
- [MIOR24] MIOR, Michael J.: Large Language Models for JSON Schema Discovery. In: *arXiv preprint arXiv:2407.03286* (2024).
- [MUCT24] MUCTADIR, Hossain M.; MANRIQUE NEGRIN, David A.; GUNASEKARAN, Raghavendran; CLEOPHAS, Loek; VAN DEN BRAND, Mark ; HAVERKORT, Boudewijn R.: Current trends in digital twin development, maintenance, and operation: an interview study. In: *Software and Systems Modeling* (2024), S. 1–31.
- [PHUA24] PHUA, Shin Z.; HOFMEISTER, Markus; TSAI, Yi-Kai; PEPPARD, Oisín; LEE, Kok F.; COURTNEY, Seán; MOSBACH, Sebastian; AKROYD, Jethro ; KRAFT, Markus: Fostering urban resilience and accessibility in cities: A dynamic knowledge graph approach. In: *Sustainable Cities and Society* 113 (2024), S. 105708.
- [PIRE19] PIRES, Flávia; CACHADA, Ana; BARBOSA, José; MOREIRA, António P. ; LEITÃO, Paulo: Digital twin in industry 4.0: Technologies, applications and challenges. In: *2019 IEEE 17th international conference on industrial informatics (INDIN)* Bd. 1 IEEE, 2019, S. 721–726.
- [PIRO22] PIROMALIS, Dimitrios; KANTAROS, Antreas: Digital twins in the automotive industry: The road toward physical-digital convergence. In: *Applied System Innovation* 5 (2022), Nr. 4, S. 65.
- [PLATnd] PLATTFORM INDUSTRIE 4.0: *Fortschrittsbericht 2023*. <https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/2023-fortschrittsbericht.html>. n.d.. – Accessed on 2025-01-12.
- [RAFF20] RAFFEL, Colin; SHAZER, Noam; ROBERTS, Adam; LEE, Katherine; NARANG, Sharan; MATENA, Michael; ZHOU, Yanqi; LI, Wei ; LIU, Peter J.: Exploring the limits of transfer learning with a unified text-to-text transformer. In: *Journal of machine learning research* 21 (2020), Nr. 140, S. 1–67.
- [RIMA24] RIMAZ, Mohammad H.; PLOCIENNIK, Christiane ; RUSKOWSKI, Martin: Semantic Asset Administration Shell for Circular Economy. In: *Proceedings of the 2nd International Workshop on Knowledge Graphs for Sustainability (KG4S 2024), co-located with the 21st Extended Semantic Web Conference (ESWC 2024)* Bd. 3753, CEUR Workshop Proceedings, 2024, S. 1–14.
- [RONG23] RONGEN, Sjoerd; NIKOLOVA, Nikoletta ; VAN DER PAS, Mark: Modelling with AAS and RDF in industry 4.0. In: *Computers in Industry* 148 (2023), S. 103910.
- [ROZI23] ROZIERE, Baptiste; GEHRING, Jonas; GLOECKLE, Fabian; SOOTLA, Sten; GAT, Itai; TAN, Xiaoqing E.; ADI, Yossi; LIU, Jingyu; SAUVESTRE, Romain; REMEZ, Tal [u. a.]: Code llama: Open foundation models for code. In: *arXiv preprint*

- arXiv:2308.12950* (2023).
- [RÜBE23] RÜBEL, Pascal; MOAREVAND, Nastaran; MOTSCH, William; WAGNER, Achim ; RUSKOWSKI, Martin: Enabling Fault Diagnosis in Skill-Based Production Environments. In: *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)* IEEE, 2023, S. 1–8.
- [SANT21] SANTOS, Kécia Souza S.; PINHEIRO, Larissa Barbosa L. ; MACIEL, Rita Suzana P.: Interoperability types classifications: A tertiary study. In: *Proceedings of the XVII Brazilian Symposium on Information Systems*, 2021, S. 1–8.
- [SARA24] SARANI RAD, Fatemeh; HENDAWI, Rasha; YANG, Xinyi ; LI, Juan: Personalized Diabetes Management with Digital Twins: A Patient-Centric Knowledge Graph Approach. In: *Journal of Personalized Medicine* 14 (2024), Nr. 4, S. 359.
- [SILV24] DA SILVA, Luis Miguel V.; KOCHER, Aljosha; GEHLHOFF, Felix ; FAY, Alexander: 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, S. 1–8.
- [SOOR23] SOORI, Mohsen; AREZOO, B ; DASTRES, Roza: Sustainable Manufacturing and Service Economics. In: *Sustainable Manufacturing and Service Economics* 19 (2023), Nr. 1, S. 1–12.
- [SUÁR15] SUÁREZ-FIGUEROA, Mari C.; GÓMEZ-PÉREZ, Asunción ; FERNANDEZ-LOPEZ, Mariano: The NeOn Methodology framework: A scenario-based methodology for ontology development. In: *Applied ontology* 10 (2015), Nr. 2, S. 107–145.
- [TEAM24] TEAM, BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. (2024).
- [TEXT21] TEXTOR, Andreas; STADTMÜLLER, Steffen; BOSS, Birgit ; KRISTAN, Johannes: BAMM Aspect Meta Model. In: *ISWC (Posters/Demos/Industry)*, 2021.
- [TINN24] TINNES, Christof; RISTIN, Marko; HOHENSTEIN, Uwe; FATHI, Kiavash ; VAN DE VENN, Hans W.: From Unstructured Product Descriptions to Structured Data for Industry 4.0 with ChatGPT. In: *2024 IEEE 7th International Conference on Industrial Cyber-Physical Systems (ICPS)* IEEE, 2024, S. 1–8.
- [TOUV23] TOUVRON, Hugo; LAVRIL, Thibaut; IZACARD, Gautier; MARTINET, Xavier; LACHAUX, Marie-Anne; LACROIX, Timothée; ROZIÈRE, Baptiste; GOYAL, Naman; HAMBRO, Eric; AZHAR, Faisal [u. a.]: Llama: Open and efficient foundation language models. In: *arXiv preprint arXiv:2302.13971* (2023).
- [VASW23] VASWANI, Ashish; SHAZEE, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz ; POLOSUKHIN, Illia. *Attention Is All*

You Need. 2023.

- [XIA24] XIA, Yuchen; XIAO, Zhewen; JAZDI, Nasser ; WEYRICH, Michael: Generation of Asset Administration Shell with Large Language Model Agents: Towards Semantic Interoperability in Digital Twins in the Context of Industry 4.0. In: *IEEE Access* (2024).