

# KGG4SE: A Knowledge Graph Generation Framework for Systems Engineering

Frank Wawrzik <sup>a,\*</sup>, Khushnood Adil Rafique <sup>a</sup>, Theogene Urimubenshi <sup>a</sup> and Christoph Grimm <sup>a</sup>

<sup>a</sup> *Department of Computer Science, University of Kaiserslautern-Landau (RPTU), Rhineland-Palatinate, Germany*  
*E-mails: wawrzik@cs.uni-kl.de, khushnood.rafiq@cs.uni-kl.de, t\_urimuben16@cs.uni-kl.de, grim@cs.uni-kl.de*

**Abstract.** In this paper we introduce the Knowledge Graph Generation Framework for Systems Engineering (KGG4SE). Based on the GENIAL! Basic Ontology (GBO), a variety of large language models and prompt engineering, we generate numerous knowledge graphs from a diverse set of input sources. One of the key features of the framework is the reasoning-in-the-loop integration. The generated classes are structurally consistency checked and inconsistent classes are removed. Further features include the generation of research articles, technology videos and datasheets. Also quality control prompts are used and the framework is integrated into a system engineering tool (SysMD) with frontend and backend. This makes content of the knowledge graph accessible to users in MBSE (Model-Based Systems Engineering) ecosystems. Finally we outline results of the generation process and content of the graphs as well as the reasoning process with disjoint axioms. The results show an improved graph quality and structure in comparison to existing approaches. In terms of succinctness and conciseness, we remove an overall of 67.4% of classes that do not adhere to the ontological entailment or domain. Although graph quality is sometimes difficult to qualify and quantify and the implementation needs further work, we believe the methodology of this framework is a good way forward to produce better quality graphs, which are scalable.

**Keywords:** ontology web language, knowledge graph construction, quality assurance, reasoning

## 1. Introduction

In today's semantic web, we reached a point where building ontologies has developed from a black art into a skilled and comprehensive engineering process. We see more ontologies across many disciplines and domains and more widespread application. One of the next steps is to make the vast amount of data on the web understandable and processable. Large language models (LLMs) with their brain-like structure have shown great performance in understanding text, among other areas [31]. In this work we introduce a generation framework for systems engineering, which produces high-quality knowledge graphs by combining OWL 2 DL reasoning with LLMs in a hybrid approach. Well-defined axioms ensure a core quality of the knowledge graph (KG), while also being embedded into a wider system engineering tool context. We use a variety of knowledge sources and generate our graphs from the system engineering domain, including:

- General electronics hardware and devices
- Amplifiers, their characteristics and their properties
- LIDAR and other sensor technologies
- Current hardware trends

---

\*Corresponding author. E-mail: wawrzik@cs.uni-kl.de.

## – Automotive general components libraries

While the area of knowledge base construction (KBC) has gained momentum with the advent of transformers and large language models, it is a relatively new and arising research area. Currently there are no KBC approaches that combine consistency reasoning with graph generation. Further we are the first to generate and integrate other areas like research articles, data from datasheets and from expert technology videos. In the area of systems engineering there are few known larger knowledge graphs. In comparison to Wikidata [42], which contains an overview of related knowledge, for example automotive components, processors or electronic components, this framework targets the generation of expert knowledge and is scalable. We combine a variety of different knowledge graph generation approaches: we use several prompt approaches and combine them differently. Sometimes we use defined specifications, standards, descriptions or other modalities like roles to generate the prompt. Further we also combine Llama 3 with a RAG approach or generate definitions from existing classes to structure the knowledge base rather than just generating content triples. The variety of approaches are supplemented via github repositories in section 6. We complement the knowledge graph generation framework with a frontend and backend and give glimpses into its supplementation with SysML v2. The paper is structured as follows: the next section describes state-of-the-art (2) and section 3 explains the framework with all components (including the GENIAL! Basic Ontology (3.8)) in detail. Section 4 presents results and elaborates and discusses those results. Finally section 5 concludes.

## 2. State-of-the-Art

### 2.1. Related work

This section reviews how knowledge graphs improve large language models and gives some general remarks. The authors of [49] showcase a system to input knowledge graphs and causal graphs into large language models. They do this to improve safety and bias of LLMs as well as to improve personalization and explainability. Mostly they compare these three technologies. Rather than using the LLM for knowledge graph construction, the other way around, they improve the LLM's output with knowledge graphs. In [27] the authors use a prompt-based approach with mind maps to improve the LLMs reasoning capabilities. They use chain-of-thought and claim it mitigates the need for model training. In [52] the knowledge graph helps to mitigate and reduce hallucinations of LLMs. Here, the LLM is used to make knowledge graph queries and to optimize them. The work in [20] again exemplifies how knowledge graphs can improve LLMs. Mostly LLMs are trained on unstructured text data, but this work trains them with structured knowledge graphs. To do this, they transform the graph into text with a linearization process and preserve the knowledge graph structure. They report that their approach enhances the LLMs ability to reason and improve question answering tasks. [29] also generates knowledge graphs with LLMs to integrate them into knowledge management systems. With their method they report an improved precision and relevance of knowledge management systems. They improve the performance of LLMs for domain specific tasks using a RAG approach. The paper in [32] presents GUNDAM, an approach which leverages the graph structure understanding of KGs to perform better and more complex reasoning tasks. Further they include reasoning paths to improve LLMs. [48] further shows how KGs complement LLMs. A significant recent debate is how LLMs and knowledge graphs differ and where there is overlap. Ontologists and knowledge graph experts have to gain a good understanding of the technologies, limitations and opportunities of LLMs first [11, 17, 25]. Various other works compare, analyze and discuss differences and opportunities [19, 30, 33, 34, 39, 50]. Conclusions range from that LLMs will make KGs obsolete to them being a good complement. The authors of the work of this paper expect KGs and LLMs to further co-exist for some time. Even though LLMs can access the required information, reason with it and can even be trained on vocabularies, ontologies in OWL yield different capabilities and focus. For example one focus is on definitions, data and information quality. Well-crafted ontologies need considerable thought, structure and even philosophical considerations. Errors of any kind produced by either the LLM or false data are counter or disadvantages for reference schemas that ontologies are supposed to represent. Further ontologies in OWL have an explicit data format that is transparent to access, process, use and can be easily combined with other data-driven methodologies.

## 2.2. State-of-the-art for knowledge graph generation

This section reviews current knowledge graph construction methods. In [12] the authors address that knowledge graphs are often outdated and manually maintained like Wikidata. Here, they update the graph with RAG (Augmented Retrieval Generation) techniques and utilize LLM reasoning to update outdated facts. In [9] they utilize knowledge graph generation for the life sciences. They assess consistency, scalability and completeness of the generated graph or ontologies. The paper [51] makes a systematic review of current knowledge graph generation methods with large language models including reasoning. Here, the authors look at methods like link prediction, entity recognition, question answering, and relation extraction with eight datasets. They found that new models like GPT-4 have good performance in knowledge graph construction, even surpassing fine tuned models. Further they are particularly strong in reasoning. [18] is also a systematic review examining 28 papers of KG-powered LLMs and similar technologies. In [40] the authors compare REBEL with large language models in regards to knowledge graph construction. Their conclusion is that LLMs perform better in regards to relevancy and accuracy. The work in [1] concludes that LLMs can aid in knowledge graph construction. But they are in general not suitable enough for ontology learning that requires a high degree of domain knowledge and in order to do so need to improve in quality. [15] create their own dataset to evaluate LLMs. There are more works with a specific focus like multi-agent systems, large scale construction of knowledge graphs and more [13, 26, 38, 43]. We found around 15 more papers all dealing with knowledge base construction. However, their methods are similar. In comparison to the other works, our work is the only one using T-Box reasoning to check and control the semantic graph structure of the generated graph. Further only this work consists of loops of quality assurance.

The following works focus more on the particular generation of definitions in comparison to general knowledge graph construction. Chatrule [28] is a mining system for logical rules for knowledge graphs. Here the LLM also checks and corrects the logical correctness of the rules, ranks the rules and leverages chain-of-thought reasoning. The authors use rules rather than axioms, with an expressiveness of SWRL and first-order logic, but are limited to rather simple ones like generating a grandmother relationship with a father and mother class. Rather than generating the actual definition the logical correctness of definitions is generated. In [36] the authors generate definitions according to BFO (Basic Formal Ontology) that are aristotelian. This means that a class definition is based on its super class; however they handle only informal definitions with annotations with LLMs (GPT-4). In a similar work [3] GPT-4 is used to generate ontologies that conform to BFO and to support ontologists that use BFO. The authors develop their own GPT model and advocate to align LLM-generated ontologies with BFO. They also report that their method was LLM dependent and updating the LLM disrupted their progress. Rather than generating BFO-conformant definitions, in this work we generate informal definitions from formal definitions from recognized standards.

## 3. The Knowledge Graph Generation for Systems Engineering (KGG4SE) Framework

### 3.1. Overview

The Knowledge Graph Generation for Systems Engineering (KGG4SE) Framework is a versatile and multimodal framework created to address the following needs:

1. Automate the growing amount of data and information into a machine processable and understandable knowledge graphs.
2. Advance graph quality by quality assurance prompts and consistency reasoning.
3. Integrate it into larger system engineering contexts like modeling with SysML v2 and other MBSE tools.

Figure 1 gives an overview of all parts of the framework and how they work together. To generate our graph we use a variety of large language models like for example Gemini, Llama 3, GPT-4 and DeepSeek R1. Here we follow the strategy of updating the LLM every 6-12 months by considering current cost, performance and impact on current prompts. We use a variety of prompt approaches (see resources 6) to generate entities (classes, concepts), restrictions, triple knowledge graph structures and data properties with values into our graph model. Currently we

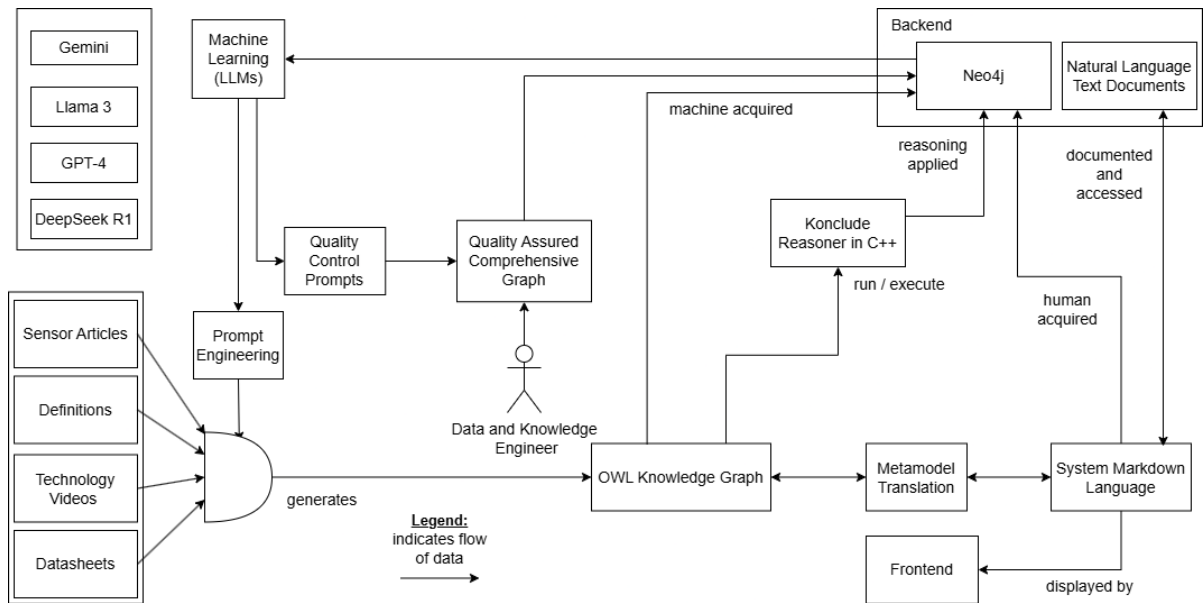


Fig. 1. Knowledge Graph Generation Framework for Systems Engineering (KGG4SE) overview

are feeding research articles of sensors and sensor technologies, definitions of various hardware parts, technology videos and datasheets into the graph. For quality assurance we use prompts and reasoning with Konclude. An example for a quality assurance prompt is for example asking the LLM if a certain sensor is a subclass of a certain category or supporting and controlling subclass hierarchies. An example for the reasoning quality assurance is for example the consistency checking of disjoint classes and has\_part hierarchies. These checks are explained in detail with examples in section 4. To better understand the reasoning, we also introduce the ontology with its axioms in section 3.8 in detail. This approach and methodology partitions the OWL knowledge graph into three major categories (complete graph, consistent graph and inconsistent graph). The quality assured comprehensive graph receives oversight and supervision from a knowledge engineer and ontologist. The ontologist reports back the current quality and engineers quality metrics and their applications. Table 1 shows all parts of our current partitioning.

Table 1  
Graph Partitioning [46]

Graph Partition	Partition Description
Basic Formal Ontology	top-level ontology to distinguish fundamental concepts like objects, processes and functions
GENIAL! Basic Ontology	electronic domain ontology to describe hardware, software and systems, properties, functions, but also dependencies and mechanical parts. Contains all axioms
complete graph	complete NLP generated graph of our current dataset, containing classes, instances and object properties
consistent graph	contains entities that pass the Konclude / Pellet consistency reasoning
inconsistent graph	contains entities that do not pass the consistency reasoning
unclassified graph	contains classes and individuals which are not allocated to a GENIAL! Basic Ontology (GBO) class

In the Figure, we can observe a loop: the consistency checking with Konclude is applied continuously to the new data coming in and the quality control prompts sort out new incorrect classes from the consistent graph as well. The OWL graph is then stored in the backend within a Neo4j graph database. Further we store text documents to be used with LLMs in parallel. In the frontend we now traverse into the world of MBSE and systems engineering. The

frontened captures documentation in the form of Markdown syntax including pictures, videos and links. Based on SysML v2 various models are captured (e.g. automotive components). In the frontened we calculate dependencies between various parameters via complex back and forward propagation algorithms. This helps to make estimation on system properties, exclude non-feasible configurations and estimate corner cases. To describe our models and constraints we use our self-developed language called SysMD (see section 3.6 and 3.9.1). The metamodel translates from SysMD (SysML v2) to OWL and back and is work in progress.

### 3.2. Definitions

Frequently knowledge graphs contain populated instance knowledge or connections with classes that represent content. For example what properties a certain processor might have or what objects are located withing a room. Ontologies however are often used as reference models to define concepts or data in well-formed precise ways. In this part of the framework we generated actual definitions with LLMs that are usually carefully designed by human craftsmanship. In this sense the TBox can contain definitions and their content. Existential restrictions need to represent that "content" to reason with it and be an ontological correct description logic representations. A processor might for example have the property frequency, which needs to be defined in an existential TBox restriction. Whereas the concrete frequency of a specific processor is an ABox triple. Figure 2 shows the general procedure. First an existing hardware knowledge base is searched for its classes without informal definition and missing restrictions. This is done with prompt engineering. Further classes are searched for missing object properties and unrelated classes in the ontology that are required for the definition. The next step consists of refinement steps that support helping the ontology definitions stay sufficiently distinguished. Then we prompt GPT-4 to get new fitting object properties. For example we ask with a prompt what other relationships are useful for sensors. This results in relationships of "used by" and "connects to." Further we create "more density" between the already existing classes by creating new object properties to fill gaps. The existing classes in the ontology are now connected as well as their alternatives generated. Finally a logical consistency analysis is made and the result is checked and controlled for ambiguity. In the following box, two example prompts within the pipeline are illustrated:

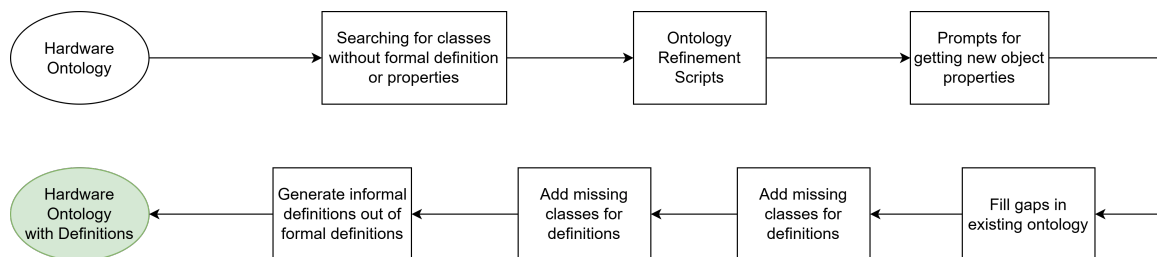


Fig. 2. Method to generate formal and informal definitions

#### GPT-4 Prompt

**Generating Relations Between Classes:** Analyze the following list of classes: [Class\_List] and identify all possible relationships between them based on functional, structural, and computational dependencies. Consider domain-specific standards (such as ISO/IEC 2382:2015) and provide meaningful object property names that best describe these relationships in an ontology-based knowledge representation system.

**Generating Formal Class Definitions Using ISO Standards and Relationships:** Generate formal definitions for the following classes [Class\_List], considering their functional role, structural properties, and relationships with other entities in the system. Ensure that each definition follows ISO/IEC 2382:2015 or a relevant domain standard and explicitly references its connections to other classes.

### 3.3. Datasheets

Another data source utilized by our framework are datasheets. Datasheets document various electronic components and their properties. We utilized amplifier datasheets from the website [datasheets.com](https://www.datasheets.com/de/categories?parametric=1) (<https://www.datasheets.com/de/categories?parametric=1>). We extracted different types of amplifiers and their electrical characteristics from PDFs represented in tabular form. Since each PDF is structured differently it was a challenge for extraction. The downloaded data was cleaned and converted to CSV so that the LLM could understand it well. We formulated precise, effective and contextually relevant prompts to elicit the desired response from the model. Different prompts were created for each amplifier type, since electrical characteristics differ for each item. The input prompt contains text description, entity, relation type and example data. The output prompt contains the output format in JSON. In below box a shortened sample input prompt is displayed. Owlready2 python package was used to load, modify and save the owl ontology.

#### Gemini Prompt

```
input_prompt = ""Based on the following example, extract entities and relations for each row from the provided specification. Use the following entity types: entity_types = "PowerDissipation" "SingleSupplyVoltage" ... Use the following relation types: relation_types = "hasType", "hasProperty", ... # specification Part, Type, PowerSupplyType, ... TDA7265, Class-AB, Single, Single, Differential, 1-Channel Mono ,8,315,V,2,V,5.5,V,-40,°C,85,°C,-,mW TDA7851A, Class-AB, Single, DifferentialSingle, Differential, 1-Channel Mono, 8,315,V,2.2,V,5.5,V,-40,°C,85,°C,-,mW ""
```

### 3.4. Technology Videos

In this approach we incorporated a video data source with domain technology trends expertise. We selected a Youtube channel based on strong domain expertise in the area of hardware, CPUs and AI. Then we crawled the videos using PyTube and then transcribed the videos using BuzzCaptions. Finally we did triple extraction with GTP-4, entity resolution and generated owl files. Prior triple extraction was performed with Llama 2 and Mistral, but yielded considerably weaker results. The results were integrated in the SysMD notebook front end (see Figure 5). The technology stack consisted of NodeJS, NPM, ReactJS, MaterialUI and Redux.

### 3.5. Sensor Knowledge Graph Generation

In this approach, we generate a knowledge graph from PDF's using retrieval augmented generation (RAG) from LIDAR sensor information. The approach is simplified and illustrated in Figure 3. To view the generated graphs look up the OWL files in the Github repository under section 6 point 5.

**Methodology:** We do this by extracting relevant LIDAR sensor data using a retrieval mechanism. We process the retrieved text using an LLM (Llama 3, DeepSeek R1) finding complex relationships between sensor data. We then convert the extracted information into a knowledge graph. We use FAISS and BM25 retrievals because they are fast and accurate and minimize unnecessary data processing. Further they capture semantically relevant text instead of relying solely on string matching. The retrieval works by indexing the PDF content into a vector store using embeddings. We use a small prompt to find relevant text snippets and rank retrieved passages based on relevance scores. Finally we pass the most relevant documents to the LLM for further processing and knowledge extraction.

**Validation:** The validation process for Lidar sensor extraction data verifies that the sensor names generated by the LLM matches the information in the original PDF document or not. The following approaches were used:

1. String Matching. This approach uses an exact text search matching technique to determine if the extracted sensor name exists in the PDF text or not. It is a check that returns a true or false result. However, it is sensitive to tiny errors such as typos and formatting inconsistencies.

2. Fuzzy Verification. A Python library called fuzzywuzzy was used to approximate string matching in order to overcome the drawbacks of exact string matching. The token set ratio between each extracted PDF text chunk and the extracted sensor name is calculated. For every comparison, a similarity score (ranging from 0 to 100) is

generated. To decide whether or not the sensor name adequately matches a section of the text, a threshold is applied. This approach can withstand some degree of variations in formatting and language.

3. Semantic Verification. A pre-trained sentence-transformer model is used in semantic verification to extract the sensor name’s contextual meaning. This method involved creating a query by fusing the name of the sensor with its category (for example, "LiDAR sensor model Velodyne VLP-32C used in automotive lidar"). The cosine similarity between each PDF chunk and the query is calculated after they have both been encoded into embeddings. To determine a semantic match measure, the average of the top few scores was used rather than depending only on the highest similarity value. When the specific phrase differs but the general idea remains the same, this approach can be helpful.

**Confidence Score:** For each sensor, a weighted confidence score was computed by combining the results of the following methods:

- String match (contributes 40%)
- Fuzzy matching (contributes 30%)
- Semantic matching (contributes 30%)

Scatter plots were used to display the results. Semantic vs. confidence scores show the relationship between overall confidence and semantic similarity. A better contextual fit is typically indicated by higher semantic scores. Confidence vs. fuzzy scores show how the confidence is affected by approximate string matching. Semantic score vs. fuzzy scores aid in understanding how context-sensitive and string-based matching techniques relate to one another.

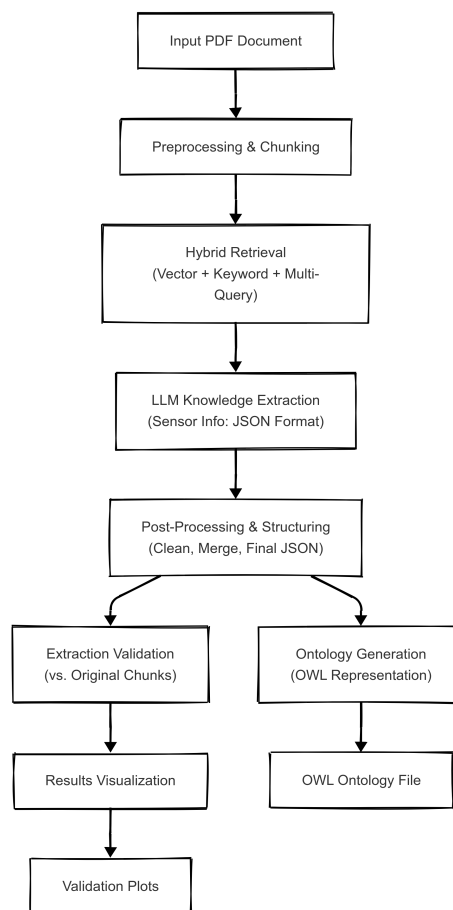


Fig. 3. RAG approach with Llama3 and DeepSeek R1 to generate knowledge graphs from LIDAR research articles

### 3.6. Frontend

Another part of the KGG4SE Framework is the front end. Here, we use a web-based application that serves the same goals as the SysMD multiplatform notebook discussed in this work [5, 24]. SysMD and its goals are outlined in section 3.9.1. Web-based applications are overtaking traditional desktop applications for several reasons. Authors in [22] discussed the cornerstone of web applications compared to native desktop applications, such as ease of use and minimal hardware resources, since all users need is just a browser. The emerging web technologies such as WebGPU and WebGL [23] are also contributing to the rapid growth of web applications. On top of the findings of different authors regarding web application preferences compared to desktop applications, we also experienced the drawbacks of desktop applications over web applications. The following are a few of them

1. Simplified DevOps: Development deployment and updates are done by the developing team.
2. Minimal hardware requirements for end-user side: Once an application is deployed, a user needs a browser and internet to use an application.
3. Single application for all platforms: Web applications are platform-independent.
4. Less Risks: Fear of installing new applications in industry companies is an opportunity for web application.

Back to React-based web frontend, a modelling tool that supports SysMD, SysML v2 and OWL languages. As illustrated in Figure 4 below, the interface is structured into three main components: the editing area (on the right), the tree view area (on the left), and the top bar menu (at the top). The tree view area displays projects, which are SysMD documents and contain class trees, relationship types ("HasA" next to projects) and related information.

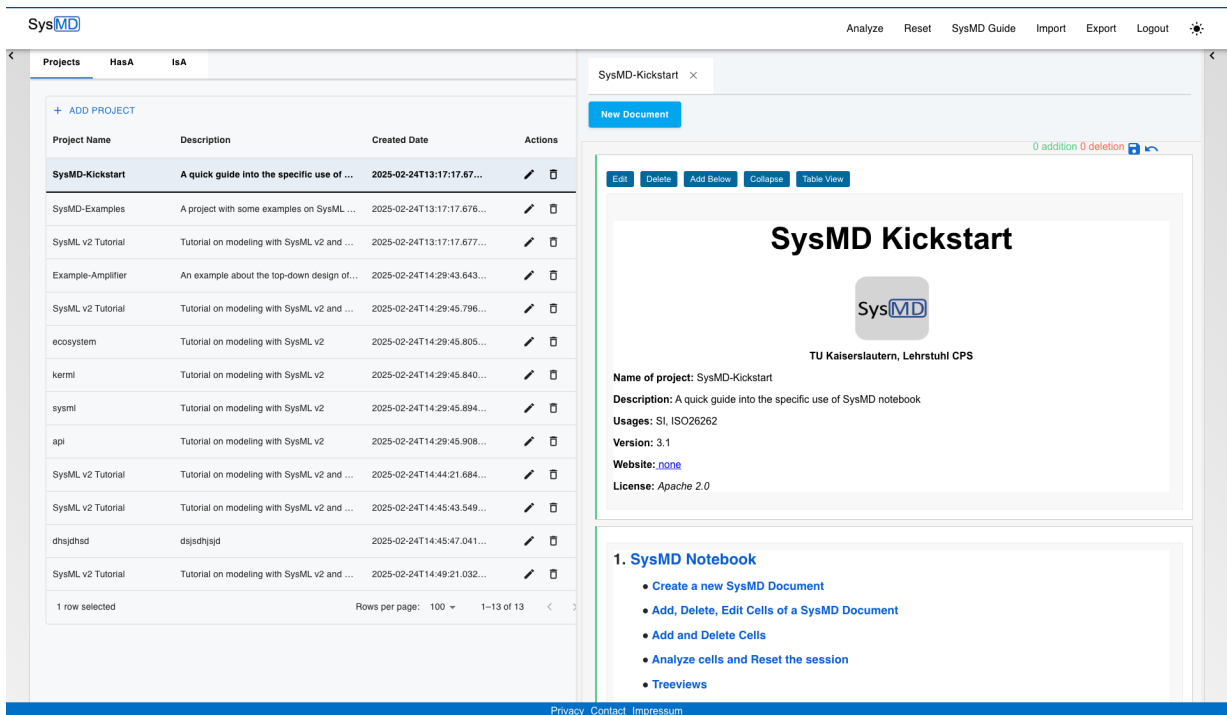


Fig. 4. SysMD Notebook: a web-based frontend for SysMD and SysML v2 languages

The WYSIWYG editor enables SysMD and SysML v2 model creation and renders markdown documents and OWL files for human readable format as shown in Figure 5.

The SysMD notebook is structured into cells, where each cell can contain SysMD or SysML v2 code or documentation (comments). This documentation may include images, videos, or audio, enhancing comprehensibility.

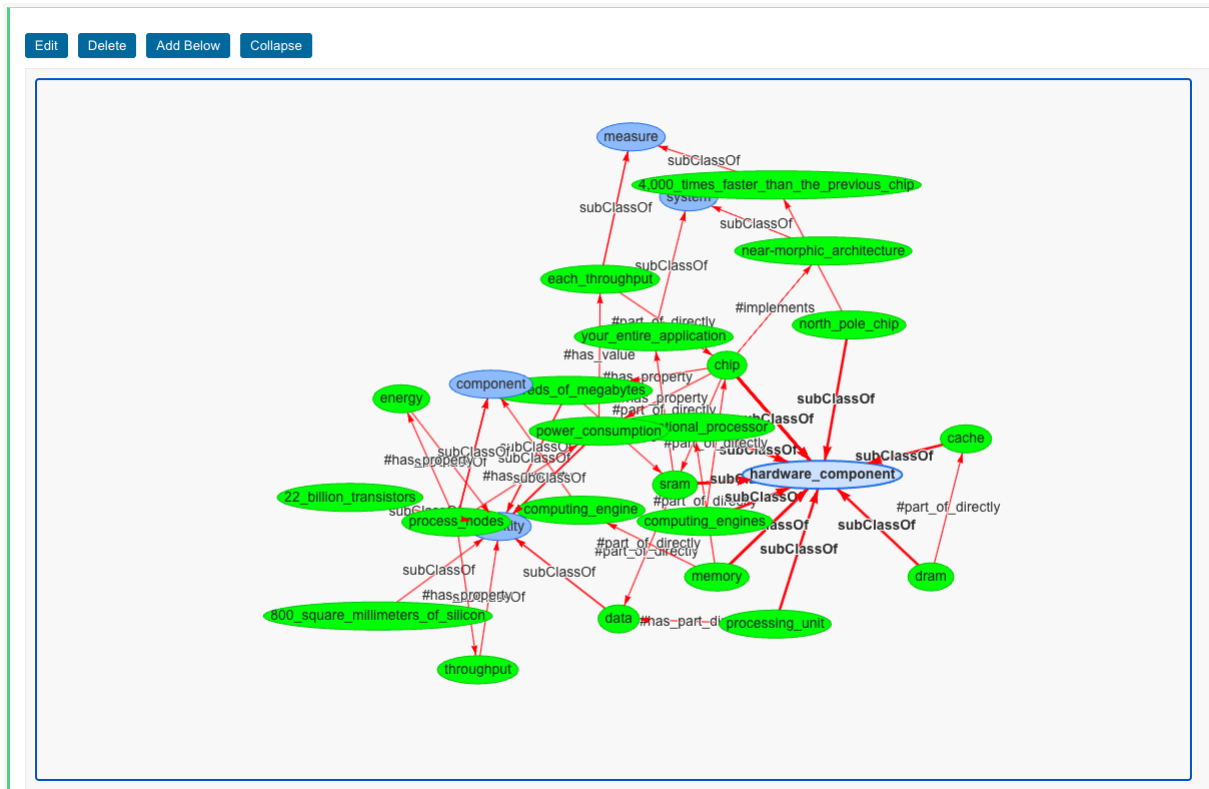


Fig. 5. SysMD Notebook: Rendered OWL knowledge graph

According to the Systems Engineering Vision 2035<sup>1</sup>, systems should be characterized by simplicity and ease of use. In line with this vision, the SysMD notebook adopts Markdown due to its simple syntax, portability, and platform independence. However, simplicity does not always equate to ease of use. To further enhance usability, we introduce an AI-based approach that generates Markdown syntax from natural language with a single click.

The authors discuss details on this Markdown syntax generation approach, which outperforms large language models (LLMs) such as ChatGPT, in [37]. Users can generate multiple versions until they are satisfied with the output. A key feature of this AI-driven syntax generation is its ability to suggest context-based titles, extract code from natural language, and predict text structures, such as tables, based on contextual understanding.

### 3.7. Backend

The backend is work in progress. Currently the backend provides a repository for SysML v2 projects and models. There are REST endpoints for the project, entities and branches. We are developing an interface for digital twins to upload and download data for development and operations purposes (DevOps). Current works are the metamodel for both SysML v2 and OWL and integration in Neo4j. Further user management features to access various levels and packages of the graph and documentation are created. Finally we consider security and safety issues for example regarding current DSVG (General Data Protection Regulation, GDPR) conformity.

<sup>1</sup>[https://www.incose.org/docs/default-source/se-vision/incose-se-vision-2035.pdf?sfvrsn=e32063c7\\_10](https://www.incose.org/docs/default-source/se-vision/incose-se-vision-2035.pdf?sfvrsn=e32063c7_10)

### 3.8. GENIAL! Basic Ontology (GBO)

An integral part of the KGG4SE Framework are its reasoning capabilities for improved quality assurance. To achieve quality assurance we built the GENIAL! Basic Ontology with a heavyweight approach. For a detailed explanation see [44, 45]. Within a control loop, reasoning is applied, inconsistent classes are sorted out and stored under a different namespace, whereas consistent classes and relationships are kept as the main quality-assured knowledge graph. In this subsection we explain considerations for how and why we constructed reasoning axioms for GBO. In general, there are two types of reasoning: 1. ontology-based reasoning and 2. rule-based reasoning. Whereas ontology-based reasoning is performed via axiom definitions in OWL 2 Description Logic, rules are often encoded in SWRL (Semantic Web Rule Language) or logic languages. Especially ontology-based reasoning can help keep the ontology consistent, draw implicit conclusions and deductions, and fill gaps. Large language models and transformers can mitigate some of their own drawbacks with assisted ontology-based reasoning [46]. In this paper, we extend the approach in [46] by integrating front-end and back-end and adding various generation approaches with large language models and information sources. Formalizing definitions and checking for contradictions is one application of reasoning. Commonly, most ontologies today are lightweight, often connected through existential restrictions to form a network of connected classes. This has drawbacks, for example that classes can be used more loosely and vaguely. This creates problems in classification tasks like over or underclassification, ontology or class merging issues, or simply a lack of control of classification correctness. Often, definitions are not strictly formalized and lack a consistent structured framework to be applied meaningfully. In GBO we strictly formalized each informal definition into a formal one to make precise use of each class.

Figure 6 shows some basic reasoning constructs. We see cardinality restrictions, which limit the number of Abox triples of a certain object property between two classes. Missing from the list is also a cardinality restriction which limits to exactly a particular number. In the Tbox the cardinality restrictions also help for more precise definitions and avoid inconsistencies. SomeValuesFrom or also called existential restrictions indicate that there should be a connection between two classes via a specific object property. This is however not enforced by a constraint and the relationship can be missing as we apply the open world assumption (OWA). SomeValuesFrom restrictions are an important part of forming definitions and building semantic networks. However sometimes their correct applications gets confused with domain and range axioms. AllValuesFrom restrictions are very strict and allow an object property connection only from one class to another by excluding other connections. This requires a thorough use of disjoint class axioms to be functionally useful for the reasoner. By default the use of someValuesFrom is recommended. UnionOf and intersectionOf connect expressions. These can be conjunctions or disjunctions of expressions of restrictions and even nested ones.

Constructor	DL Syntax	Example	FOL Syntax
intersectionOf	$C_1 \sqcap \dots \sqcap C_n$	Human $\sqcap$ Male	$C_1(x) \wedge \dots \wedge C_n(x)$
unionOf	$C_1 \sqcup \dots \sqcup C_n$	Doctor $\sqcup$ Lawyer	$C_1(x) \vee \dots \vee C_n(x)$
complementOf	$\neg C$	$\neg$ Male	$\neg C(x)$
oneOf	$\{x_1\} \sqcup \dots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}	$x = x_1 \vee \dots \vee x = x_n$
allValuesFrom	$\forall P.C$	$\forall$ hasChild.Doctor	$\forall y.P(x, y) \rightarrow C(y)$
someValuesFrom	$\exists P.C$	$\exists$ hasChild.Lawyer	$\exists y.P(x, y) \wedge C(y)$
maxCardinality	$\leq nP$	$\leq 1$ hasChild	$\exists^{\leq n} y.P(x, y)$
minCardinality	$\geq nP$	$\geq 2$ hasChild	$\exists^{\geq n} y.P(x, y)$

Fig. 6. Reasoning constructs with example, DL and FOL syntax 1

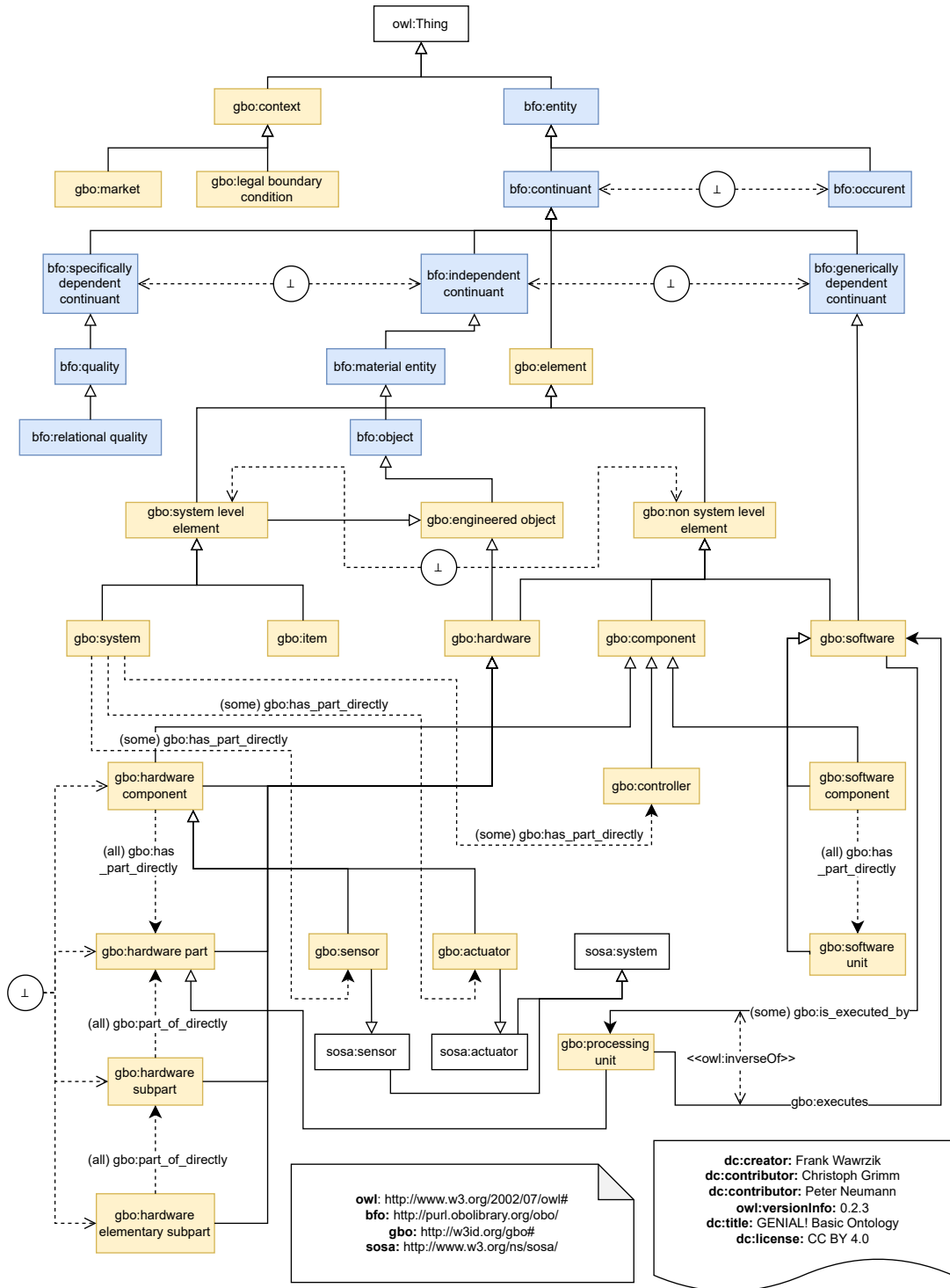


Fig. 7. GBO Core Part 1 Overview - Parts [44]

OWL Syntax	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
transitiveProperty	$P^+ \sqsubseteq P$	ancestor <sup>+</sup> $\sqsubseteq$ ancestor

Fig. 8. Reasoning constructs with example and DL syntax 2

Figure 8 shows more axioms. SubClassOf axioms are the means to create taxonomies and make instances be found on the correct hierarchical level when performing queries on instances. EquivalentClasses automatically categorize Tbox and Abox classes and instances. The equivalent restrictions supports the deductions of classifying those instances and classes under the equivalent class if the restriction expression is met. Not mentioned in Figure 6 are for example covering axioms. Last but not least, there are object property axioms. For example subPropertyOf is for subordinating object properties in hierarchies similar to subclassing. Equivalent properties set two properties as equal. This is useful for example in merging ontologies. Transitive properties help propagate certain characteristics or individuals through hierarchies. Not mentioned in Figure 8 are for example data property restrictions.

Figure 7 shows an overview of the core structure of GBO. It imports Basic Formal Ontology (BFO), the ontology of units of measure and related concepts (OM-2) and is build upon the ISO 26262 standard. The following classes are part of its core definitions:

- entity, continuant, occurrent, specifically dependent continuant, independent continuant, generally dependent continuant, quality, relational quality, material entity, object (from Basic Formal Ontology)
- context, market, legal boundary condition, element, system level element, non system level element, engineered object, system, item, hardware, hardware component, hardware part, hardware subpart, hardware elementary subpart, component, controller, software, software component, software unit, sensor, actuator, processing unit (most classes are from the ISO 26262 standard)

There are currently just three cardinality restrictions in GBO and they are all part of the informal ISO26262 standard. We use many someValuesFrom restrictions often in combination with allValuesFrom restrictions to provide the desired capability for deductions. AllValuesFrom is used in hierarchies of software and hardware in combination with disjoint axioms to achieve consistency reasoning. We made a frequent use of disjoint axioms and equivalent classes. And also combine various restrictions. Overall the number of object and data properties have been kept to a minimum. This is a modeling best practice derived from BFO-compliant ontologies.

### 3.9. Natural Language Processing for SysMD and SysML V.2

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and human language. It encompasses tasks such as text classification, named entity recognition, machine translation, and text generation. Traditional NLP techniques relied on rule-based systems and statistical methods, but recent advancements have shifted towards deep learning models, particularly transformers and large language models (LLMs). The introduction of the transformer model [41] revolutionized NLP by introducing the self-attention mechanism, which allowed models to capture long-range dependencies in text. This architecture led to the development of powerful LLMs such as BERT [7], GPT-4 [31], and CodeBERT [10]. These models have significantly enhanced the capabilities of NLP applications, including code generation and system modeling. Using the advanced NLP and LLM techniques, the technology can facilitate the automated generation of high-quality knowledge graphs. These graphs can be used for a variety of applications, such as information retrieval, semantic search, decision support systems, and even for model-based analysis in system engineering and design. In domains such as AI-powered systems engineering, the ability to automatically generate and update knowledge graphs from evolving technical documentation and system models can lead to more efficient and up-to-date knowledge management.

### 3.9.1. SysMD: An Accessible Systems Modeling Language

SysMD [6] is a modeling tool and language based on SysML v2 [2], designed to make systems modeling accessible to domain experts with minimal modeling experience. It leverages statements in near-natural language and integrates seamlessly with markdown files, enabling the combination of code and documentation within a single document. SysMD facilitates knowledge base creation by incorporating markdown documents with model comments, ensuring maintainability. With its simplified syntax and continuous consistency checking, SysMD enhances the modeling process, supporting a collaborative approach to system development. SysML v2, on the other hand, extends the capabilities of its predecessor by improving expressiveness, precision, and interoperability. Introduces a more formal syntax, improved traceability, and better integration with modern software engineering practices. SysML v2 supports various modeling paradigms, allowing for more comprehensive and structured representations of the system.

### 3.9.2. Generating SysMD Code Using Transformers

SysMD code generation can be achieved using transformer-based models like CodeBERT, following the methodologies outlined in [47]. This involves utilizing PyTorch Lightning [8] for efficient model development, SpaCy [14] for tokenization, and Torchtext [35] for vocabulary management. The training data are structured in JSON format and validated using the CodeSearchNet dataset [16]. Data augmentation, including replacing function and variable names with numerical representations, enhances the robustness of the model. Hyperparameter tuning, such as optimizing learning rate, embedding dimensions, dropout, and optimizer selection, is crucial, with the Adam optimizer [21] preferred over SGD [4] for superior performance. These methodologies ensure that the generated SysMD code aligns with the SysML v2 standard while maintaining clarity and accessibility.

### 3.9.3. Generating SysML v2 Code Using LLMs

Large Language Models (LLMs), such as GPT-4 and CodeT5, have shown potential to generate structured code for complex modeling languages such as SysML v2. These models leverage advanced techniques such as prompt engineering or zero-shot training and fine-tuning to achieve this. Prompt engineering involves carefully crafting inputs that guide the model to produce outputs that align with the desired format and content. Fine-tuning, on the other hand, entails training an already pretrained model on a specialized dataset that includes both natural language descriptions and their corresponding SysML v2 representations. To facilitate the translation of domain-specific language into structured system models, LLMs can be trained on annotated datasets. These datasets may consist of natural language statements describing system components, processes, and interactions, alongside their corresponding SysML v2 model code. Through this training, LLMs learn the patterns and mappings between the unstructured descriptions and the formalized language of SysML v2, enabling them to generate accurate system models from textual inputs. Moreover, advanced techniques like reinforcement learning and retrieval-augmented generation (RAG) can further enhance the model's performance. Reinforcement learning involves training the model to optimize for specific outcomes, such as the accuracy of the generated SysML v2 code or the quality of the model's logical consistency. By using rewards and penalties, the model can be guided to generate more relevant and correct outputs over time. Retrieval-augmented generation, on the other hand, enriches the model's responses by incorporating external knowledge sources during the generation process. This can significantly improve the accuracy of generated models by allowing the model to access a wider range of relevant information beyond what was captured during its initial training.

## 4. Results

In this section we give an overview of results from the various approaches in the knowledge graph generation framework: from the generated definitions, datasheets and technology videos. Finally, we outline results of the reasoning-in-the-loop process with the hybrid approach with transformers. Here we give qualitative and quantitative results regarding disjoint axiom definitions in GBO. To view the results in detail with the corresponding generated OWL files, we refer to the Github repositories in the resource section.

#### 4.1. Ontology Definition Generation with GPT-4

In this section we look at the definitions generated from our framework and discuss them. Our hardware ontology contained 166 classes, 6 object properties and 246 relationships (triples plus restrictions). After the generation we obtained 180 classes. Meaning 14 were supplemented to complete definitions. 11 new object properties were created, yielding 17 in total. The most substantial increase was in relationship creation of almost 50% bringing the total relationships to 354.

##### 4.1.1. Discussion

###### Manual Construction vs. Automatic Construction of Definitions

The framework and approach did not only construct the missing definitions but we also let it try to define our manually constructed definitions. Figure 9 shows the definitions of hardware component. The "comprised\_of" object property is set as equivalent with the "has\_part\_directly" object property, which is non-transitive. This means that in (a) a hardware component has as parts only hardware parts and at least two of them. This is in contrast to (b) where we can see that the GPT-Model does not yet have an awareness of the kind of restriction. And also in other examples only constructs existential "some" restrictions. It also constructed the hardware part with a different relation, which can potentially be set to equivalent with "has\_parts", but usually also "connects" horizontal connections. Here it could be mapped dependent on context. It took the literal definition of interface with hardware for its definition, knowing that sensor is a hardware. This decision of GPT-4 and the framework is debatable as it "makes sense" and is additional information, but does not yield an exact definition. In this case the manual definition is superior as it can be handled by the reasoner in a clear way and is also more helpful in constructing the knowledge base in a more structured way. Secondly, in (a) no informal definition was provided, because it was implicitly derived from the component definition and no ISO 26262 definition of hardware component existed. With GPT-4 the informal definition was more narrow and contextualized towards automotive. Similar in some regards and different in others. It is important to have a common baseline for definitions and avoid inconsistencies for the integration of knowledge.

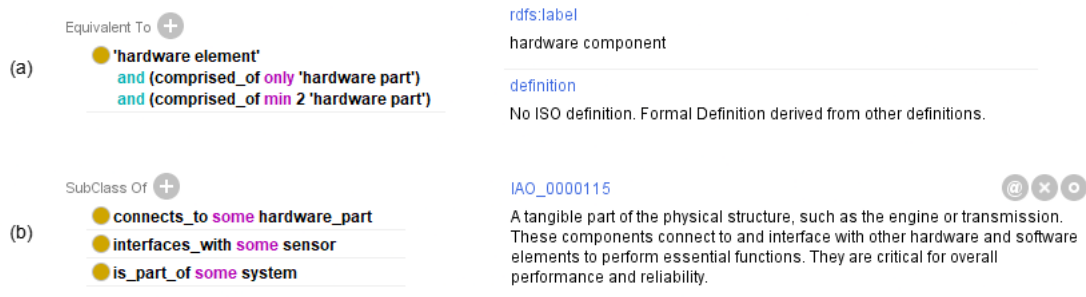


Fig. 9. Definition of hardware component in gbo and in the automatically constructed hardware ontology

###### Automatic Constructions of Definitions

Further many formal definitions and informal definitions that were lacking in the knowledge base were generated. Here are a few representative examples given, that outline and illustrate the results that GPT-4 and the approach yield. Figure 10 shows both definition types. In (c) the memory interface is shown. We can see that it created a new relationship to the memory map based on the word "memory components" it found in the definition. This definition is correct, but only well-educated domain experts are able to evaluate nuances on ontological grounds. Based on the word "data transfer" another correct relationship was created that helps definitions. In (d) computer architecture on the other hand we see an ontological problem. Here the relationship "has\_part" was falsely exchanged with the inverse "is\_part\_of." Further the existential restrictions memory and input output systems could have been created and were missed by GPT-4. In (e) we again see a good performance with the memory access unit. We see a knowledge graph completion rather than a definition. Further an additional quantity was added to complete the formal definition which is fitting.

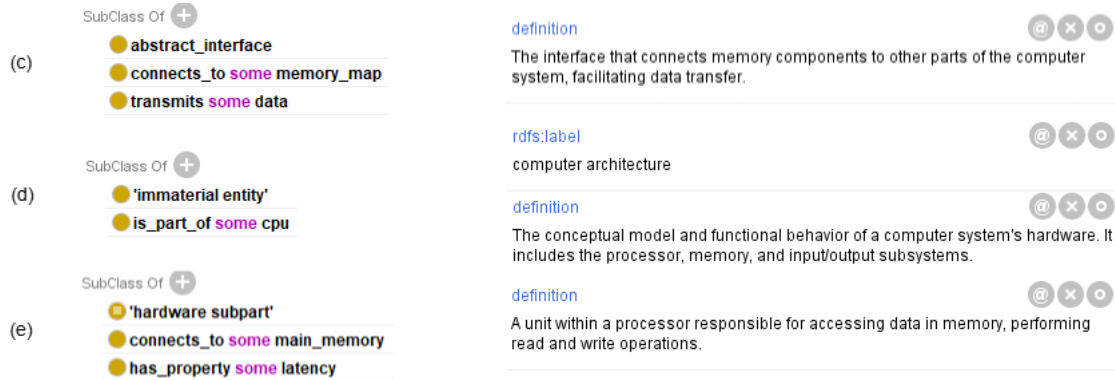


Fig. 10. Automatic definitions pros and cons

#### 4.2. Generation of Amplifiers and their Properties from Datasheets with Gemini

With our approach outlined in the overall framework from Figure 1 and section 3.3 we generated about 40 amplifiers in 12 categories with over 300 properties. Figure 11 shows an outline of an video amplifier and its properties. Below are the generated categories and properties of the amplifiers listed:

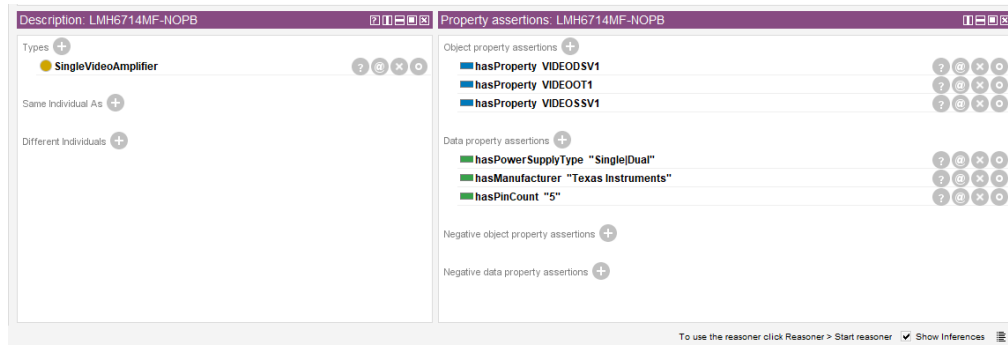


Fig. 11. Instantiation of single video amplifier LMH6714MF-NOPB and its properties

- **Amplifier Taxonomy:** audio amplifier, (classAB, classD, classG), GPS amplifier (low-noise GPS amplifier), instrumental amplifier (dual instrumental amplifier, single instrumental amplifier), operational amplifier (auto-zero operational amplifier, chopper operational amplifier, general purpose operational amplifier, high-speed operational amplifier, low-noise operational amplifier low-offset voltage operational amplifier, micro-power operational amplifier, precision operational amplifier, wide-band operational amplifier), video amplifier (dual video amplifier, hex video amplifier, quad video amplifier, single video amplifier, triple video amplifier)
- **Amplifier Properties:** gain, gain bandwidth product, input bias current, noise figure, operating frequency, operating supply voltage, operating temperature, output current, power dissipation, CMRR, PSRR, single supply voltage, supply current, supply voltage

#### 4.3. Generation of Technology Videos with GPT-4

The style of the content and amount of data posed several challenges. We transcribed about 70 videos out of which the transcription quality was good. However the amount of generated classes and quality of relationships was less than in other data sources and approaches. Further the cost involved with a paid GPT-4 version was no longer neglectable, especially in regards to further scaling. Figure 5 shows the visualization of the Youtube video "New

IBM AI Chip Explained Faster than Nvidia GPUs and the Rest!" An excerpt from this knowledge graph is shown in Figure 12. Overall the generation of properties was good, however the `has_parts` relationships could be improved in this sub-approach. For example "chip" is here related to the "SRAM" class with the "has\_part\_directly" object property. However, both chip and SRAM are on the same hierarchical level. Better would have been a "connected\_to" object property. But this also highlights the strength of the overall approach. This inconsistency next to others can be sorted out with the reasoning. The following section gives examples with disjoint reasoning.



```

ExtractedTriples.json
[{"head": "chip", "relation": "implements", "tail": "near-morphic architecture"},
{"head": "chip", "relation": "has part directly", "tail": "SRAM"},
{"head": "chip", "relation": "has Property", "tail": "hundreds of megabytes"},
{"head": "hundreds of megabytes", "relation": "has value", "tail": "SRAM"},
{"head": "SRAM", "relation": "part of directly", "tail": "your entire application"},
{"head": "SRAM", "relation": "part of directly", "tail": "data"},
{"head": "processing unit", "relation": "has part directly", "tail": "data"},
{"head": "memory", "relation": "part of directly", "tail": "conventional processor"},
{"head": "DRAM", "relation": "part of directly", "tail": "cache"},
{"head": "North Pole chip", "relation": "has Property", "tail": "4,000 times faster than the previous chip"},
{"head": "computing engines", "relation": "part of directly", "tail": "chip"},
{"head": "memory", "relation": "part of directly", "tail": "computing engine"},
{"head": "process nodes", "relation": "has Property", "tail": "power consumption"},
{"head": "process nodes", "relation": "has Property", "tail": "throughput"},
{"head": "process nodes", "relation": "has Property", "tail": "energy"},
{"head": "power consumption", "relation": "has value", "tail": "each throughput"},
{"head": "each throughput", "relation": "part of directly", "tail": "chip"},
{"head": "chip", "relation": "has Property", "tail": "power consumption"}

```

Fig. 12. Triple excerpt from video generation in JSON

#### 4.4. Reasoning Evaluation

The reasoning evaluation in this section was applied using a previous method using transformers (see resources point 1). This prior work with transformers [46] and its results were comparable to knowledge graphs generated with large language models. We explore first the results of creating the ontology. The ontology has 3623 classes, 93 object properties, 239 individuals, 17399 axioms, and 8373 "subclass of" relations. Which is considerably larger than the manually developed models for specific use cases, that we were able to create in the past with considerable amounts of invested time and manual effort.

Figure 13 shows the "hardware element" or hardware class from GBO. It has "hardware component", "hardware elementary subpart", "hardware part" and "hardware subpart" as subclasses. Here we show just the hardware part class of the inconsistent graph. We see that this class has many subclasses like `adc`, `clock generator`, `cpu`, `crystal resonator`, `disks flash memory`, `serial ata` and others. Which are often indeed hardware parts, but are inconsistent. Again those classes are machine-created and also machine classified as hardware part.

For example the class "clock generator" is inconsistent, because we have the rule that "hardware component", "hardware elementary subpart", "hardware part" and "hardware subpart" are disjoint classes. And the class clock generator is directly a subclass of "hardware component" and "hardware part." For this reason it is deleted from the main ontology and added to the inconsistent graph file.

We evaluated the results according to several metrics of conciseness and consistency<sup>2</sup>. Conciseness refers to "avoiding schema and data elements irrelevant for the domain." Consistency measures the degree to which "(i.e., inconsistencies) with respect to the particular logical entailment considered." In order to handle complexity, we used only the transitive "has\_part" relationship for classification rather than the "has\_part\_directly" object property. This is equivalent with a less strict sorting out. First, we quantitatively measured the percentage of classes that the reasoner categorized. We counted the inconsistent and consistent classes with proteges metrics and subtracted

<sup>2</sup><https://www.emse.fr/~zimmermann/KGBook/Multifile/quality-assessment/>

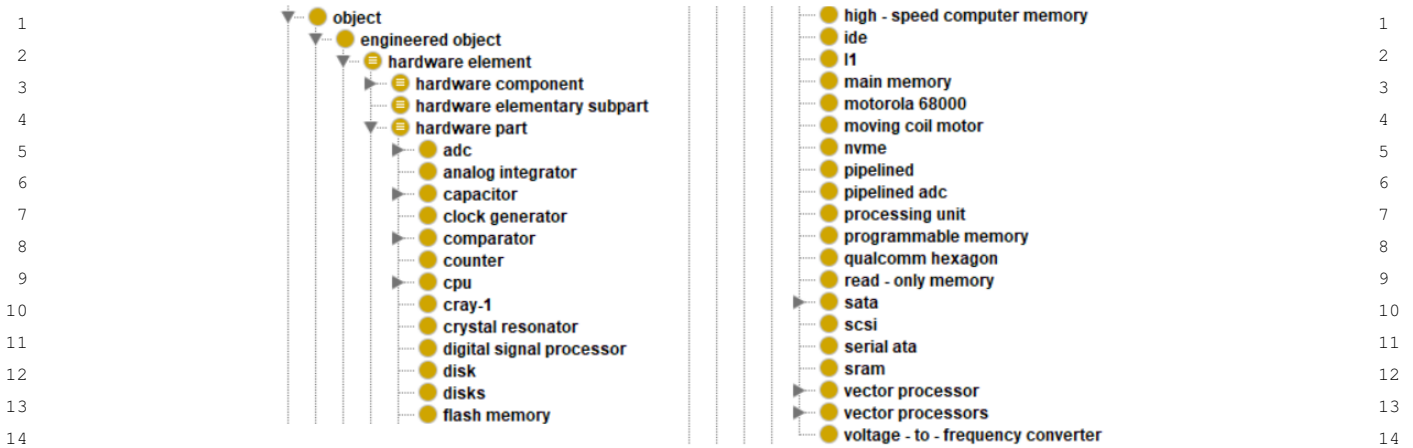


Fig. 13. Inconsistent ontology OWL file overview with class "clock generator" [46]

Table 2  
Graph Partitioning after Reasoning [46]

Graph Partition	Amount of Occurrence
unclassified classes	2218
unclassified individuals	239
GBO classes	83
inconsistent classes	165 (248-83)
inconsistent individuals	9
consistent classes	3399 (3482-83)
consistent individuals	194
complete graph classes	3623
complete graph individuals	239

the classes of both BFO and GBO. Whereas in the whole graph we did not subtract, as we see both ontologies as an integral element; unclassified classes were also counted directly. Table 2 gives an overview. Second, the amount of spared time by the knowledge engineer is significant. As the amount of data and its connections are hard to comprehend. This methods makes this process supervisable. Apart from the number of connections, some evaluations and considerations might simply be missed. The next paragraph gives examples. Third, we decided to qualitatively analyze and evaluate the results and discuss their significance. Here, we exemplify the results based on examining the correctness through disjointness of the classes as well as the categorization through the "has\_parts" relationship. Figure 14 gives a sample of a few inconsistencies detected by the reasoner. In the following we will give explanations to each (1-5) case:

1. "moving coil motor" appears to be an unexpected classification by the neural net transformer, which puts it as a subclass of actuator (not hardware component) and contradicts our vocabulary.
2. "rop" appears at first glance to be an abbreviation of a hardware part. Examining the training dataset clears up the reason for the inconsistency. Rop is once classified as an abbreviation of "render output pipelines", a hardware part. And on another occurrence as an abbreviation of "return oriented programming", a software. Both terms are merged in our graph and thus create the inconsistency, because the abbreviation is referring to two different entities.
3. "calculation pipeline" was identifiably falsely assigned as a function in the tagging dataset by our trainee. The transformer in this case was correct by classifying it as a type of alu (arithmetic logic unit). In combination with our ontology the error was recognized.

- 1 4. here we see the "converter" class and its relationships. This class has as a part ("has\_part") a clock. For  
 2 example, in digital-to-analog converters (DACs) and analog-to-digital converters (ADCs), a clock is often used  
 3 to synchronize the conversion process. Also this class has as a part a comparator. In ADCs, the comparator is  
 4 used to compare the input analog signal to a reference voltage and generate a digital output based on the result  
 5 of the comparison. And this class uses some signal processing. In the context of converters, signal processing  
 6 is used to transform the input signal into a format that can be processed and converted into the desired output  
 7 signal. The color green indicates that the converter in itself is a consistent class. However, its relationships  
 8 where removed, since both the comparator and the clock where found to be inconsistent classes.  
 9  
 10 5. "snr" (signal to noise ratio) was correctly tagged in the dataset as a quantity and thus correctly placed in the  
 11 graph. The transformer neural network however, decided due to context to place the snr as a subclass of adc  
 12 (as well as create the "facet\_of" relationship). An adc is a hardware part though, creating the inconsistency.

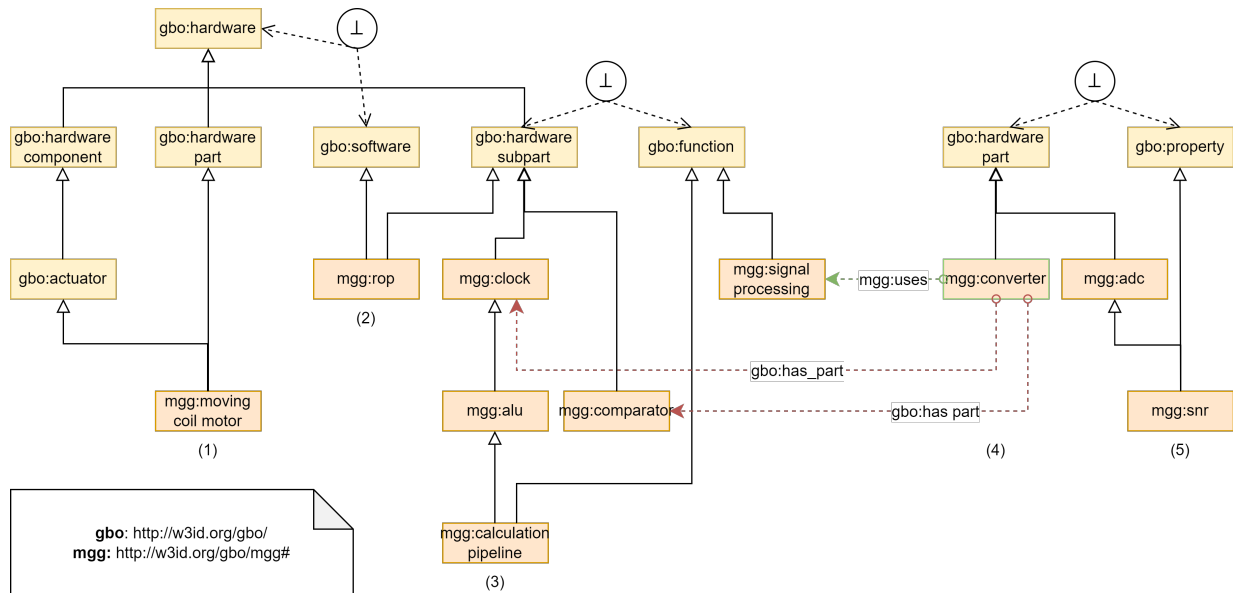


Fig. 14. Examples for inconsistencies with disjointness axiom and "has\_part" object property [46]

Overall the dataset was quantitatively improved by 4.7 % in coherency with inconsistency analysis alone. The question of relevancy is of similar importance to improve graph quality. The removing of unclassified classes achieved an improvement of 62,7 % in conciseness. Altogether our method improved our knowledge graph about 67,4 %. Table 3 summarizes the results.

Table 3  
Quantification of Evaluation [46]

Metric	Percentage of Improvement
unclassified removed classes	62,7%
inconsistent removed classes	4,7%
overall removed classes	67,4%

## 5. Conclusion

In this paper we introduced the Knowledge Graph Generation Framework for Systems Engineering (KGG4SE). We explained the overall framework including its parts with large language model knowledge graph generation, consistency reasoning and frontend and backend integration. We introduced the GENIAL! Basic Ontology with its reasoning elements. Further we generated knowledge graphs from research articles, incomplete definitions, datasheets and technology videos. The application of reasoning helped removing inconsistent classes and improve the overall quality of the knowledge graph. In future works we further want to focus on quality assurance aspects. We want to apply the complete reasoning not only to disjoint classes, but to all restrictions, including AllValuesFrom and SomeValuesFrom with real-time processing. This is expected to significantly improve the number of sorted out inconsistent classes and further underline the benefits of the approach. Further work also needs the application of all approaches to the reasoning loop and the backend integration. We want to integrate reinforcement learning to reclassify inconsistent classes. And add refined prompt-loops to assess and control the quality of the graph. In addition, we also want to further scale the approach and work on more rigorous quality metrics to further reduce human intervention in the quality assurance process. Finally, we want to finish the metamodel translation between SysML v2 and OWL.

## 6. Resources

1. Hybrid AI and reasoning with konclude: <https://github.com/OntologyResearcher/Supplementarymaterial>
2. Generating formal definitions from informal ones: [https://github.com/khanmujeeb687/ontology\\_completion](https://github.com/khanmujeeb687/ontology_completion)
3. Generation of amplifiers and their properties from datasheets: <https://github.com/oovk/masters-thesis/tree/main>
4. Knowledge graph generation from video: <https://github.com/wawrzik/Knowledge-Graph-Construction-from-Video>
5. Sensor knowledge graph generation with RAG approach: [https://github.com/rocknegi/Generating\\_KG\\_from\\_LLMs/tree/main](https://github.com/rocknegi/Generating_KG_from_LLMs/tree/main)
6. Knowledge graph construction with GPT-4: <https://github.com/savanvekariya/Knowledge-Graph-Construction-for-Technology-Intelligence-and-Planning-of-Cyber-Physical-Systems/tree/main>

## References

- [1] H. Babaei Giglou, J. D'Souza and S. Auer, *LLMs4OL: Large Language Models for Ontology Learning*, 2023, pp. 408–427. ISBN 978-3-031-47239-8. doi:10.1007/978-3-031-47240-4\_22.
- [2] M. Bajaj, S. Friedenthal, C. Delp, R.D. Schultz, M. Hause and J.A. Estefan, *The Systems Modeling Language (SysML) v2: An Overview of the Proposed Specification*, Vol. 32, John Wiley & Sons, 2022, pp. 1567–1584. doi:10.1002/iis2.12387.
- [3] C.-B. Benson, A. Sculley, A. Liebers and J. Beverley, My Ontologist: Evaluating BFO-Based AI for Definition Support, 2024. doi:10.48550/arXiv.2407.17657.
- [4] L. Bottou, Large-Scale Machine Learning with Stochastic Gradient Descent, in: *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT 2010)*, 2010, pp. 177–186. <https://hal.inria.fr/inria-00405995>.
- [5] S. Dalecke and C. Grimm, SysMD: An Inclusive Modelling Tool, in: *2024 19th Annual System of Systems Engineering Conference (SoSE)*, 2024, pp. 178–183. doi:10.1109/SoSE62659.2024.10620956.
- [6] S. Dalecke, K.A. Rafique, A. Ratzke, C. Grimm and J. Koch, SysMD: Towards “Inclusive” Systems Engineering, in: *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2022, pp. 1–6. doi:10.1109/ICPS51978.2022.9816856.
- [7] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, *arXiv preprint arXiv:1810.04805* (2018). <https://arxiv.org/abs/1810.04805>.
- [8] G.F. Falcon, M.G. Schmitt, A. Baran et al., PyTorch Lightning, *GitHub repository* (2019). <https://github.com/PyTorchLightning/pytorch-lightning>.
- [9] N. Fathallah, S. Staab and A. Algergawy, LLMs4Life: Large Language Models for Ontology Learning in Life Sciences, 2024. doi:10.48550/arXiv.2412.02035.
- [10] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang and M. Zhou, CodeBERT: A Pre-Trained Model for Programming and Natural Languages, *arXiv preprint arXiv:2002.08155* (2020). <https://arxiv.org/abs/2002.08155>.

- [11] W. fred, Comprehensive Overview of Large Language Models (LLMs): Grasping Their Essence, 2023. doi:10.31219/osf.io/xbtzw.
- [12] S. Hatem, G. Khoriba, M.H. Gad-Elrab and M. ElHelw, Up To Date: Automatic Updating Knowledge Graphs Using LLMs, *Procedia Computer Science* **244** (2024), 327–334, 6th International Conference on AI in Computational Linguistics. doi:https://doi.org/10.1016/j.procs.2024.10.206. https://www.sciencedirect.com/science/article/pii/S1877050924030072.
- [13] Q. He, X. Wang, J. Liang and Y. Xiao, MAPS-KB: A Million-Scale Probabilistic Simile Knowledge Base, *Proceedings of the AAAI Conference on Artificial Intelligence* **37** (2023), 6398–6406. doi:10.1609/aaai.v37i5.25787.
- [14] M. Honnibal, I. Montani, S.V. Landeghem et al., spaCy: Industrial-Strength Natural Language Processing, 2020. https://spacy.io/.
- [15] J. Huang, M. Li, Z. Yao, Z. Yang, Y. Xiao, F. Ouyang, X. Li, S. Han and H. Yu, RiTeK: A Dataset for Large Language Models Complex Reasoning over Textual Knowledge Graphs, 2024. doi:10.48550/arXiv.2410.13987.
- [16] S. Husain, S. Ding, M.H.L. Kuo, J. Tang and R. Singh, CodeSearchNet: Evaluating Large Language Models on Code Understanding Tasks, in: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019, pp. 3319–3329. https://arxiv.org/abs/1909.09436.
- [17] S. Johnson and D. Hyland-Wood, A Primer on Large Language Models and their Limitations, 2024. doi:10.48550/arXiv.2412.04503.
- [18] A. Kau, X. He, A. Nambissan, A. Astudillo, H. Yin and A. Aryani, Combining Knowledge Graphs and Large Language Models, 2024. doi:10.48550/arXiv.2407.06564.
- [19] H. Khorashadizadeh, F.Z. Amara, M. Ezzabady, F. Ieng, S. Mishra Tiwari, N. Mihindukulasooriya, J. Groppe, S. Soror, F. Benamara and S. Groppe, Research Trends for the Interplay between Large Language Models and Knowledge Graphs, 2024. doi:10.48550/arXiv.2406.08223.
- [20] W. Kim, H. Jung and W. Kim, Knowledge Graph as Pre-Training Corpus for Structural Reasoning via Multi-Hop Linearization, *IEEE Access* **PP** (2024), 1–1. doi:10.1109/ACCESS.2024.3523579.
- [21] D.P. Kingma and J.B. Ba, Adam: A Method for Stochastic Optimization, *arXiv preprint arXiv:1412.6980* (2014). https://arxiv.org/abs/1412.6980.
- [22] J. Kiruthika, S. Khaddaj, D. Greenhill and J. Francik, User experience design in web applications, in: *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, IEEE, 2016, pp. 642–646.
- [23] M. Kligge, Comparison of WebGL and WebGPU as alternatives for implementing GPGPU computing in the browser, *Abschlussbericht FEP 2023/2024* (2024), 13.
- [24] J. Koch, A. Wansch and C. Grimm, Knowledge modeling of power grids with SysMD, in: *2022 10th Workshop on Modelling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, 2022, pp. 1–6. doi:10.1109/MSCPES55116.2022.9770147.
- [25] E. Koutsiana, J. Walker, M. Nwachukwu, A. Meroño-Peñuela and E. Simperl, Knowledge Prompting: How Knowledge Engineers Use Large Language Models, 2024. doi:10.48550/arXiv.2408.08878.
- [26] A. Krishna and C. Malhotra, A COLLABORATIVE MULTI-AGENT LLM APPROACH FOR KNOWLEDGE GRAPH CURATION AND QUERY FROM MULTI-MODAL DATA SOURCES, 2024. doi:10.13140/RG.2.2.36306.77761.
- [27] X. Lai, L. Tang, X. Zhu, L. Xiao, Z. Chen and J. Yang, KG-CQAM: knowledge graph and mind-mapping-based complex question answering for large language models, in: *Fifth International Conference on Control, Robotics, and Intelligent System (CCRIS 2024)*, Vol. 13404, C. Yang, ed., SPIE, 2024, p. 134040G, International Society for Optics and Photonics. doi:10.1117/12.3050113.
- [28] L. Luo, J. Ju, B. Xiong, Y.-F. Li, G. Haffari and S. Pan, ChatRule: Mining Logical Rules with Large Language Models for Knowledge Graph Reasoning, 2023. doi:10.48550/arXiv.2309.01538.
- [29] A. Makin, Ontology-Driven Knowledge Management Systems Enhanced by Large Language Models, 2024. doi:10.13140/RG.2.2.24648.23043.
- [30] F. Neuhaus, Ontologies in the era of large language models – a perspective, *Applied Ontology* **18** (2023), 399–407. doi:10.3233/AO-230072.
- [31] OpenAI, GPT-4 Technical Report, *arXiv preprint arXiv:2303.08774* (2023). https://arxiv.org/abs/2303.08774.
- [32] S. Ouyang, Y. Hu, G. Chen and Y. Liu, GUNDAM: Aligning Large Language Models with Graph Understanding, 2024. doi:10.48550/arXiv.2409.20053.
- [33] J. Pan, S. Razniewski, J.-C. Kalo, S. Singhanian, J. Chen, S. Dietze, H. Jabeen, J. Omeliyanenko, W. Zhang, M. Lissandrini, R. Biswas, G. Melo, A. Bonifati, E. Vakaj, M. Dragoni and D. Graux, Large Language Models and Knowledge Graphs: Opportunities and Challenges, 2023. doi:10.48550/arXiv.2308.06374.
- [34] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang and X. Wu, Unifying Large Language Models and Knowledge Graphs: A Roadmap, *IEEE Transactions on Knowledge and Data Engineering* **36**(7) (2024), 3580–3599. doi:10.1109/TKDE.2024.3352100.
- [35] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark and L. Zettlemoyer, Deep Contextualized Word Representations, *arXiv preprint arXiv:1802.05365* (2018). https://arxiv.org/abs/1802.05365.
- [36] T. Procko, O. Ochoa and T. Elvira, Automatic Generation of BFO-Compliant Aristotelian Definitions in OWL Ontologies with GPT, 2023, pp. 141–146. doi:10.1109/TransAI60598.2023.00042.
- [37] K.A. Rafique, N. Khaliq and C. Grimm, HyMark: Application of Hybrid AI for Markdown Syntax Generation, in: *2024 IEEE Conference on Artificial Intelligence (CAI)*, IEEE, 2024, pp. 687–694.
- [38] D. Reales, R. Manrique and C. Grévisse, Core Concept Identification in Educational Resources via Knowledge Graphs and Large Language Models, *SN Computer Science* **5** (2024). doi:10.1007/s42979-024-03341-y.
- [39] K. Suri, A. Singh, P. Mishra, S. Rout and R. Sabapathy, Language Models sound the Death Knell of Knowledge Graphs, 2023. doi:10.48550/arXiv.2301.03980.
- [40] M. Trajanoska, R. Stojanov and D. Trajanov, Enhancing Knowledge Graph Construction Using Large Language Models, 2023. doi:10.48550/arXiv.2305.04676.

- [41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser and I. Polosukhin, Attention Is All You Need, in: *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 30, 2017. <https://arxiv.org/abs/1706.03762>.
- [42] D. Vrandečić and M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Commun. ACM* **57**(10) (2014), 78–85. doi:10.1145/2629489.
- [43] J. Walker, E. Koutsiana, M. Nwachukwu, A. Peñuela and E. Simperl, The Promise and Challenge of Large Language Models for Knowledge Engineering: Insights from a Hackathon, 2024, pp. 1–9. doi:10.1145/3613905.3650844.
- [44] F. Wawrzik, Knowledge Representation in Engineering 4.0, doctoralthesis, Technische Universität Kaiserslautern, 2022. doi:10.26204/KLUEDO/6888. <http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-68887>.
- [45] F. Wawrzik and A. Lober, A Reasoner-Challenging Ontology from the Microelectronics Domain, in: *Proceedings of the Semantic Reasoning Evaluation Challenge (SemREC 2021) co-located with the 20th International Semantic Web Conference (ISWC 2021), Virtual Event, October 27th, 2021*, G. Singh, R. Mutharaju and P. Kapanipathi, eds, CEUR Workshop Proceedings, Vol. 3123, CEUR-WS.org, 2021, pp. 1–12. <http://ceur-ws.org/Vol-3123/paper1.pdf>.
- [46] F. Wawrzik, R. Dhouib, K.A. Rafique and C. Grimm, Hybrid AI Approach for Knowledge Graph Construction, in: *Proceedings of the Workshop, Poster and Demonstration Sessions at IJCKG 2023 co-located with 12th International Joint Conference on Knowledge Graphs (IJCKG 2023), Tokyo, Japan, December 8-9, 2023*, A. Yamaguchi, S. Egami, K. Kozaki, T. Kawamura, B. Villazón-Terrazas and M. Buranarach, eds, CEUR Workshop Proceedings, Vol. 3659, CEUR-WS.org, 2023, pp. 48–63. [https://ceur-ws.org/Vol-3659/IJCKG\\_2023\\_WS2.pdf](https://ceur-ws.org/Vol-3659/IJCKG_2023_WS2.pdf).
- [47] F. Wawrzik, J. Koch, K.A. Rafique, S. Kwasigroch, M. Herzog and C. Grimm, Linking System and Circuit Design by AI Techniques, in: *AI-Enabled Electronic Circuit and System Design*, A. Iranmanesh and H. Sayadi, eds, Springer, Cham, 2025, pp. 1–24. doi:10.1007/978-3-031-71436-8\_2. [https://doi.org/10.1007/978-3-031-71436-8\\_2](https://doi.org/10.1007/978-3-031-71436-8_2).
- [48] X. Wu and K. Tsioutsoulis, Thinking with Knowledge Graphs: Enhancing LLM Reasoning Through Structured Data, 2024. doi:10.48550/arXiv.2412.10654.
- [49] Z. Yang, I. Azimi, M. Zaki, M. Gaur, O. Seneviratne, D. McGuinness and S. Rashid, Transforming Personal Health AI: Integrating Knowledge and Causal Graphs with Large Language Models, 2024. doi:10.13140/RG.2.2.35236.90242.
- [50] Y. Zhang, L. Chen, S. Li, N. Cao, Y. Shi, J. Ding, P. Zhou and Y. Bai, Way to Specialist: Closing Loop Between Specialized LLM and Evolving Domain Knowledge Graph, 2024. doi:10.48550/arXiv.2411.19064.
- [51] Y. Zhu, X. Wang, J. Chen, S. Qiao, Y. Ou, Y. Yao, S. Deng, H. Chen and N. Zhang, LLMs for knowledge graph construction and reasoning: recent capabilities and future opportunities, *World Wide Web* **27** (2024). doi:10.1007/s11280-024-01297-w.
- [52] Z. Zhu, G. Qi, G. Shang, Q. He, W. Zhang, N. Li, Y. Chen, L. Hu, W. Zhang and F. Dang, Enhancing Large Language Models with Knowledge Graphs for Robust Question Answering, in: *2024 IEEE 30th International Conference on Parallel and Distributed Systems (ICPADS)*, 2024, pp. 262–269. doi:10.1109/ICPADS63350.2024.00042.