JÖNKÖPING UNIVERSITY
*School of Engineering*

# From Text to Trust

## An LLM Multi-Agent System with Embedding Verification for ADAS Knowledge Graph Construction

**PAPER WITHIN** *Computer Science*
**AUTHORS:** *Miranda Rossana Martínez Rodríguez*
**TUTOR:** *Maria M. Hedblom*
**JÖNKÖPING** *June 2025*

This exam work has been carried out at the School of Engineering in Jönköping in the subject area Computer Science. The work is a part of the two-year Master of Science in AI Engineering programme. The authors take full responsibility for opinions, conclusions and findings presented.

| | |
|---|---|
| Examiner: | Vladimir Tarasov |
| Supervisor: | Maria M. Hedblom |
| Scope: | 30 credits |
| Date: | 2025-06-18 |


| Mailing address: | Visiting address: | Phone: |
|---|---|---|
| Box 1026<br>551 11 Jönköping | Gjuterigatan 5 | 036-10 10 00 (vx) |

# Contents

# Acknowledgments

What's the skin behind the concept?

The quiet breakdowns, the loud doubts, the strange peace that comes at 2am when you're questioning your life choices and suddenly a vision appears with the possible answer you were looking for, the stress, the hope, the idea that didn't work out, the idea that did work out but you are always trying to improve, the pressure, the relief.

There are moments in life that don't just change you, they redefine you. This thesis is one of them. Not because of what's written in its pages, but because of everything that supports the walls around it.

To my parents — mamá, papá — gracias por enseñarme que mis sueños nunca son demasiado grandes. Gracias por todo lo que hicieron, lo que dieron, lo que dejaron. For your sacrifices, your strength, your prayers when I didn't have the words. To my best friend, Ross, for always holding my hand when I'm scared of the dark and for teaching me how to find the way through it. Te admiro tanto.

Para mi familia, gracias por el apoyo que me brindaron. For showing up in different ways when I needed it the most. "Thank you" will never be enough. This is for you.

To my academic supervisor, Maria Hedblom, thank you for letting me fall forward. For giving me the space to think big and the structure to land on solid ground. Your patience and insight carried this work far beyond what I imagined. You changed my mind.

To Volvo Cars, for supporting this thesis and giving me the opportunity to work amongst great minds. From the ADAS Department, Ali Nouri and Zhennan Fei, for encouraging me to believe in my ideas. For asking the important questions that led to important solutions, sharing your knowledge with me was the greatest lesson. Thank you for trusting the vision.

To the quiet force inside me, to the version of myself that was scared but never stopped showing up, thank you. For enduring the loneliness, the uncertainty, the pain. For always keeping your curiosity alive when everything else felt heavy. You didn't give up. Not even to the blood.

For the faith, thank You for the silence that still held presence. For the moments of peace that made no logical sense. I didn't walk alone, and I never truly doubted that, even when it was hard to see.

And to Mary, thank you for untying my knots.

# 1 Introduction

The automotive industry is undergoing a major transformation driven by the rapid advancement of Autonomous Driving (AD) and Advanced Driver Assistance Systems (ADAS). These technologies promise to significantly improve road safety, mitigate human error, and improve transportation efficiency. However, the increasing sophistication of the ADAS/AD functionalities introduces unprecedented levels of system complexity. Modern vehicles are evolving into Software-Intensive Systems of Systems (SISoS) (Maier, 1998) comprising numerous interacting subsystems for perception, planning, decision-making, and control, all heavily reliant on intricate software logic. Managing this complexity and ensuring the safety, reliability, and regulatory compliance of these systems presents a formidable engineering challenge.

This progression towards more autonomous and intelligent systems is part of a broader technological evolution where artificial intelligence (AI) enables systems not only to process vast amounts of data but also to reason, learn, and adapt to dynamic environments. Central to this shift are knowledge-driven approaches, particularly the use of structured knowledge representations like ontologies and Knowledge Graphs (KGs). These formalisms allow complex concepts, relationships, and real-world behaviors to be explicitly modeled, supporting tasks from reasoning and decision-making to semantic search and validation. When combined with advancements in natural language processing (NLP) and machine learning, approaches that fall under the umbrella of neuro-symbolic AI (d'Avila Garcez & Lamb, 2020), such structured knowledge forms a cornerstone for systems capable of extracting, organizing, and verifying information from diverse, often unstructured, sources. Furthermore, the application of multi-agent architectures and powerful transformer-based language models has presented new avenues for distributing cognitive functions and emulating collaborative workflows for information processing and knowledge refinement.

Within this evolving technological landscape, addressing the specific information management challenges in ADAS/AD engineering is critical. The convergence of Large Language Models (LLMs) and Knowledge Graphs (KGs) offers a transformative opportunity LLMs possess remarkable capabilities in understanding and processing natural language text, while KGs provide a powerful framework for representing complex information and relationships in a structured, verifiable manner. However, deploying these technologies effectively in the safety-critical automotive domain requires careful consideration; direct application of LLMs, for instance, necessitates robust validation mechanisms to address limitations regarding domain-specific accuracy and factual reliability (Nouri et al., 2024). Recognizing these factors, and inspired by the potential of collaborative agent-based systems shown to be effective for complex KG curation tasks (Krishna et al., 2024), this thesis proposes a novel methodology: a Multi-Agentic Knowledge Graph (MAKG) system tailored for ADAS systems engineering.

This MAKG system utilizes a Multi-Agent System (MAS) architecture where specialized LLM-powered agents collaborate in a structured workflow. By assigning distinct roles, such as document parsing, information extraction, knowledge validation, entity linking, and KG integration, the system aims to automate the process of transforming unstructured technical documentation into a coherent, domain-specific KG. This approach leverages the strengths of LLMs for text processing while incorporating explicit validation steps and structured collaboration protocols, drawing inspiration from effective MAS paradigms (Krishna et al., 2024) to mitigate known LLM challenges in safety-critical applications (Nouri et al., 2024). The resulting KG is envisioned as a dynamic, verifiable repository of ADAS engineering knowledge.

## 1.1 Research Questions and Objectives

This thesis seeks to address the challenges of automated knowledge management in ADAS systems engineering by investigating the following key research questions. The proposed Multi-Agentic Knowledge Graph (MAKG) system, detailed in Chapter 4, serves as the primary artifact and methodology through

which these questions are explored:

- **RQ1:** How can collaborating LLM-powered agents within a Multi-Agent System automate the construction of a Knowledge Graph from unstructured ADAS technical documentation?
- **RQ2:** How can the accuracy and reliability of the knowledge extracted by LLM-driven components be improved within an automated Knowledge Graph Construction workflow?
- **RQ3:** How can semantic consistency and appropriate structure be maintained in an ADAS Knowledge Graph automatically generated from text?

To answer these research questions, the primary objectives of this research are:

- To design and implement a novel agent-inspired pipeline system, the Multi-Agentic Knowledge Graph (MAKG) system, employing specialized LLM-powered components and deterministic modules for processing ADAS technical documents and constructing a domain-specific KG.
- To develop, integrate, and evaluate specific validation and normalization mechanisms within the MAKG workflow, including embedding-based semantic checks and an inductive reasoning loop, aimed at improving the accuracy and semantic coherence of the generated KG.
- To assess the feasibility and effectiveness of the proposed MAKG system in constructing a structured KG from sample ADAS documentation, thereby demonstrating a viable approach to the research questions.

## 1.2 Scope and delimitations

This research focuses on the design, implementation, and methodological evaluation of the proposed Multi-Agentic Knowledge Graph (MAKG) system for structuring ADAS engineering knowledge primarily from unstructured English-language technical text. The scope includes defining the agent-inspired pipeline architecture, integrating pre-trained Large Language Models (LLMs like Llama 3 Instruct via Ollama) and semantic embedding models (e.g., SBERT, E5, Numberbatch) as tools within specialized components, and generating a Knowledge Graph (e.g., in Neo4j using RDF-aligned principles and Cypher) as the primary output. The evaluation will concentrate on the system's KGC workflow execution, the structural quality and semantic coherence of the generated KG, and the effectiveness of its validation and normalization mechanisms, using representative ADAS documentation as input. Key delimitations include the use of existing pre-trained AI models without undertaking new model training or extensive fine-tuning. The evaluation aims to demonstrate the proposed methodology's viability rather than conducting exhaustive benchmarking against all extant KGC techniques or optimizing for real-time processing speed. Furthermore, the constructed KG will reflect the information within the selected input documents and does not aim for complete domain coverage of all ADAS knowledge. The development of a user interface or specific downstream applications directly consuming the KG is outside the current scope, which centers on the KGC process and artifact itself. While the validation mechanisms aim to enhance factual accuracy and consistency, they do not constitute formal mathematical verification.

## 1.3 Project Outline

This thesis is organized into the following chapters, systematically addressing the research objectives and detailing the development and evaluation of the MAKG system:

- **Chapter 1 - Introduction:** Provides the research context, defines the problem motivating this work, introduces the proposed Multi-Agentic Knowledge Graph (MAKG) system, and outlines the research questions, objectives, scope, and the overall structure of the thesis.
- **Chapter 2 - Research Foundation:** Details the essential theoretical background on Advanced Driver Assistance Systems (ADAS), Knowledge Graphs (KGs) and their construction, semantic embedding models, Large Language Models (LLMs), and Multi-Agent Systems (MAS).
- **Chapter 3 - Related Work:** Reviews pertinent existing research in automated Knowledge Graph Construction from text, synergies between LLMs and KGs, applications of semantic embeddings, Multi-Agent Systems for knowledge engineering, and knowledge representation efforts within the automotive and ADAS domain.
- **Chapter 4 - Methodology:** Describes the Design Science Research (DSR) approach adopted. It then elaborates on the design principles, architecture, and core mechanisms of the MAKG system, and concludes by outlining the comprehensive Evaluation Strategy.
- **Chapter 5 - Results:** Presents the empirical findings obtained from executing the evaluation strategy, including quantitative metrics and qualitative analyses of the MAKG system's performance.
- **Chapter 6 - Discussion:** Interprets the results in the context of the research questions, discusses their implications, and highlights the strengths and limitations of the MAKG system.
- **Chapter 7 - Conclusion and Future Work:** Summarizes the key contributions and findings, revisits the research questions, acknowledges limitations, and proposes directions for future research..

# 2 Research Foundation

Large Language Models (LLMs) and Knowledge Graphs (KGs) offer complementary strengths for representing and extracting knowledge. LLMs, such as those based on the GPT architecture (Brown et al., 2020; Radford et al., 2019), capture vast world knowledge within their parameters but function largely as opaque systems, sometimes prone to factual errors. In contrast, KGs store facts in a structured, interpretable, and verifiable form. However, constructing and maintaining KGs, especially from unstructured data, is traditionally a challenging and labor-intensive endeavor. The synergistic unification of LLMs and KGs (Pan et al., 2024) presents a promising approach, leveraging the generative abilities of LLMs to populate KGs while using the inherent structure of KGs to ground and validate LLM outputs.

This chapter outlines the specific theoretical foundations underpinning the research presented in this thesis. It covers: Advanced Driver Assistance Systems (ADAS) as the primary application domain; the principles of Knowledge Graphs and established techniques for their construction; the role of embedding models in semantic representation; the core capabilities and limitations of Large Language Models; and the paradigm of Multi-Agent Systems (or agent-inspired approaches) relevant to collaborative knowledge engineering. A review of these key concepts provides the necessary context for understanding the design and contributions of the proposed MAKG system.

## 2.1 Advanced Driver Assistance Systems (ADAS)

Advanced Driver Assistance Systems (ADAS) represent a critical and rapidly evolving domain within the automotive industry, acting as a technological bridge between conventionally human-driven vehicles and the long-term vision of fully autonomous systems (Bengler et al., 2014). Fundamentally, the objective driving ADAS development is twofold: enhancing occupant safety and improving driving comfort and efficiency by selectively automating or supporting specific aspects of the dynamic driving task.

The primary motivation is the potential to mitigate the significant toll of traffic accidents, the vast majority of which are attributed, at least in part, to human error. By augmenting the driver's situational awareness through enhanced perception (e.g., blind-spot detection), providing timely warnings, or intervening directly in vehicle control, these systems aim to reduce driver cognitive load, compensate for momentary lapses in attention, and shorten reaction times in critical scenarios. However, this transition is not without its own complexities. While early ADAS implementations primarily offered passive warnings, contemporary systems increasingly assume active control over steering, braking, and acceleration, dynamically interacting with the vehicle's environment. This shift necessitates not only sophisticated technology but also careful consideration of human-machine interaction and the potential for mode confusion or over-reliance.

The technological foundation of modern ADAS is a complex interplay of diverse sensing modalities (e.g., cameras, radar, LiDAR), sophisticated algorithms, and high-performance computing. Integrating the data streams from these disparate sensors through sensor fusion is a critical but non-trivial task, requiring robust algorithms to construct a coherent and reliable real-time model of the vehicle's surroundings. This environmental model is the bedrock upon which perception functions, identifying and classifying objects and understanding road infrastructure, are built. Beyond perception lies the intricate logic of decision-making and control, often employing a hybrid of established rule-based systems and increasingly complex machine learning models (especially deep learning) to interpret the environment, predict future behaviors, and determine appropriate actions.

Designing and validating these systems presents formidable engineering challenges. The need for real-time performance under stringent safety requirements (often governed by standards like ISO 26262 (ISO, 2018)) must be met despite inherent uncertainties arising from sensor noise, complex environmental interactions, and unpredictable road users. Errors or failures in perception, prediction, or planning can have catastrophic consequences, demanding extreme levels of reliability and robustness, particularly against edge cases and unforeseen "corner case" scenarios (Koopman & Wagner, 2017). Furthermore, the increasing reliance on data-driven machine learning components introduces challenges related to explainability, validation sufficiency, and ensuring predictable behavior across the vast range of potential driving situations. This inherent system complexity, the intricate interplay between hardware and a massive software codebase, and the paramount importance of safety assurance underscore the critical need for advanced methodologies and tools, such as the knowledge management approach proposed in this thesis, to effectively engineer and validate modern ADAS/AD systems.

### 2.1.1 Levels of Driving Automation (SAE J3016)

To standardize discussions about vehicle automation, SAE International developed the J3016 standard, which specifies six levels of driving automation. The definitions provided below summarize the core distinctions of each level as described in the standard (SAE, 2021).

- **Level 0 (No Automation):** The human driver performs all dynamic driving tasks (DDT).
- **Level 1 (Driver Assistance):** The system provides sustained lateral or longitudinal vehicle motion control support. The driver performs all other aspects of the DDT and must constantly supervise.
- **Level 2 (Partial Automation):** The system provides sustained lateral and longitudinal vehicle motion control support simultaneously. The driver must constantly supervise the system and the driving environment and remains fully responsible for the DDT, including Object and Event Detection and Response (OEDR).
- **Level 3 (Conditional Automation):** The system performs the entire DDT within its specific Operational Design Domain (ODD). The human driver does not need to supervise while the feature is engaged within its ODD but must be receptive and ready to take back control when requested by

the system (fallback-ready user (Zhao et al., 2024)). The system performs OEDR when engaged within its ODD.

- **Level 4 (High Automation)**: The system performs the entire DDT and DDT fallback functions within its ODD without expecting human intervention, capable of achieving a minimal risk condition autonomously if needed.
- **Level 5 (Full Automation)**: The system performs the entire DDT under all conditions manageable by a human driver; no ODD limitations apply regarding DDT performance. No human driver needed.

Currently, most ADAS features commercially available fall into Levels 1 and 2. The transition to higher levels of automation (Level 3 and beyond) demands a significant increase in system complexity, requiring capabilities often described as human-like cognitive functions (Koopman & Wagner, 2017). These include robust environmental perception through multi-sensor fusion, reliable prediction of behavior, complex planning, and continuous self-monitoring. Achieving this level of performance relies heavily on advances in deep learning, sensor fusion, and planning algorithms. However, ensuring the safety and reliability of such systems, particularly regarding functional insufficiencies (like sensor limitations or algorithm performance issues) rather than just component failures, remains a major challenge addressed by standards like ISO 21448 (SOTIF - Safety of the Intended Functionality) (ISO, 2022).

## 2.2 Knowledge Graphs and their construction

A Knowledge Graph (KG) is a structured representation of knowledge typically modeled as a graph. In its common form, nodes represent entities (e.g., objects, concepts, people) and edges represent semantic relations between these entities. Factual information is often stored as triplets of the form (head entity, relation, tail entity), such as (AdaptiveCruiseControl, isFeatureOf, ADAS). This representation aligns well with the Resource Description Framework (RDF) model prominent in semantic web technologies (W3C, 2014). KGs provide a formal structure for representing real-world information, enabling applications like semantic search, question answering, and data integration. Well-known public KGs include Wikidata and DBpedia, often built from structured or semi-structured sources adhering to RDF principles.

To practically store, manage, and query these graph structures, specialized database systems have emerged. One prominent example is **Neo4j**[1], a native graph database management system widely used for implementing Knowledge Graphs. Unlike traditional relational databases, Neo4j is optimized for storing and traversing graph data directly. It employs the **Labeled Property Graph (LPG)** model (Robinson et al., 2015). In the LPG model:

- **Nodes:** Represent entities (similar to RDF resources/subjects/objects). Nodes can have zero or more **Labels** which act as type classifiers (e.g., :Sensor, :Requirement).
- **Relationships:** Represent connections between nodes (similar to RDF predicates). Relationships are directed, always have a start node and an end node, and crucially, they must have exactly one **Type** (e.g., hasPart, validatedBy).
- **Properties:** Both nodes and relationships can have key-value pairs called properties, allowing arbitrary attributes to be stored (e.g., a node labeled :Sensor might have a property name :Camera_Front).

Neo4j utilizes **Cypher**[2], a declarative graph query language designed for pattern matching and traversing relationships efficiently. Its focus on optimized graph traversals makes it well-suited for tasks common

[1] https://neo4j.com/
[2] https://neo4j.com/developer/cypher/

in KG applications, such as finding complex relationship paths, identifying connected components, or performing graph-based reasoning and querying, making it a practical choice for deploying KGs derived from sources like technical documentation.

Knowledge Graph Construction (KGC) is the process of creating or populating a KG, whether it conforms strictly to RDF or utilizes an LPG model like Neo4j's. When dealing with unstructured text, KGC typically involves several Information Extraction (IE) sub-tasks:

- **Named Entity Recognition (NER):** Identifying mentions of entities (e.g., "Adaptive Cruise Control", "camera") in the text.
- **Entity Linking (EL) / Disambiguation:** Mapping identified mentions to unique entities (nodes) within the KG (or creating new entities).
- **Relation Extraction (RE):** Identifying semantic relationships between entities mentioned in the text (which become relationship types between nodes).
- **(Optional) Event Extraction:** Identifying occurrences of events and their participants.

Traditionally, automated KGC from text often relied on pipeline approaches, where these sub-tasks are performed sequentially. While foundational, this approach can potentially propagate errors from earlier steps to later ones. More recently, generative approaches leveraging powerful sequence-to-sequence models, such as those based on the Transformer architecture like BART (M. Lewis et al., 2020) or T5 (Raffel et al., 2023), have emerged. These methods aim to directly generate structured outputs (like triples or graph elements) from unstructured text input, potentially performing multiple IE tasks jointly. This paradigm shift seeks to mitigate some limitations of sequential processing. A key challenge in KGC, regardless of the method, is relation normalization (or predicate canonicalization): mapping the diverse linguistic expressions of a relationship found in text to a standardized relation type defined in the KG's schema.

## 2.3 Embedding Models for Semantic Representation

Embedding models are fundamental to modern NLP and KG applications. They learn dense, low-dimensional vector representations (embeddings) of symbolic data like words, sentences, or KG entities. The key idea is that semantic similarity between the original data points should correspond to proximity in the vector space. In NLP, early influential models like Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) learned word embeddings from large text corpora, capturing semantic relationships like analogy (e.g., performing vector arithmetic such that the vector for Paris minus the vector for France plus the vector for Germany results in a vector close to the embedding for Berlin, capturing the `'capital of'` relationship). Subsequent developments led to contextualized embeddings from models like BERT (Devlin et al., 2019) and sentence-level embeddings using architectures like Sentence-BERT (SBERT) (Reimers & Gurevych, 2019), which capture the meaning of longer text spans. These text embeddings are highly effective for tasks requiring semantic understanding, such as information retrieval, clustering, and measuring textual similarity functions, crucial within a KGC pipeline for identifying related concepts or linking textual mentions to KG entities (L. Wu et al., 2020).

Within the KG domain, Knowledge Graph Embedding (KGE) techniques learn vector representations for entities and relations directly from the graph structure. Seminal methods like TransE (Bordes et al., 2013) model relations as translation vectors, such that $\textbf{embedding}(\textbf{head}) + \textbf{embedding}(\textbf{relation}) \approx \textbf{embedding}(\textbf{tail})$ for known triples. Other KGE models (e.g., DistMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016)) use different scoring functions but share the goal of encoding structural information for tasks like predicting missing links (KG completion) or identifying potentially erroneous triples.

Embeddings derived from text or knowledge bases can also be utilized to assess the semantic plausibility or coherence of information, potentially leveraging commonsense knowledge encoded in models like ConceptNet Numberbatch (Speer et al., 2017) or the contextual understanding of sentence encoders like SBERT or E5 (Wang et al., 2024).

## 2.4 Large Language Models

Large Language Models (LLMs) are deep neural networks, typically based on the Transformer architecture (Vaswani et al., 2023), trained on vast amounts of text data. Their primary training objective is usually language modeling, predicting the next word in a sequence. This process imbues them with extensive knowledge about language structure, facts, and reasoning patterns implicitly encoded within their billions of parameters.

Models like GPT-3 (Brown et al., 2020) demonstrated remarkable few-shot learning capabilities, performing various downstream tasks with minimal task-specific examples provided only in the input prompt. Subsequent models like ChatGPT/GPT-4 (OpenAI et al., 2024) further enhanced instruction-following and conversational abilities.These models exhibit several core capabilities relevant for knowledge-intensive tasks:

- **Natural Language Understanding (NLU):** This refers to the ability to comprehend the meaning, syntax, and context of human language. Effective NLU is essential for interpreting complex or specialized text accurately.
- **Information Extraction (IE):** This involves identifying and extracting specific pieces of structured information, such as named entities, relationships between them, or key attributes, from within unstructured or semi-structured text sources. This capability is fundamental for populating structured knowledge bases.
- **Natural Language Generation (NLG):** This is the capability to produce coherent, contextually appropriate human-like text. While often used for summarization or dialogue, NLG can also be employed to generate structured outputs (like JSON or code) based on internal representations or instructions.
- **Reasoning:** LLMs possess emergent reasoning capabilities, allowing them to perform limited forms of inference, pattern matching, analogy, and commonsense deduction based on the vast statistical patterns learned during pre-training. This enables tasks like contextual interpretation, disambiguation, and potentially hypothesizing implicit connections within data.

The accessibility and practical application of LLMs for research and development have been significantly advanced by the availability of powerful open-source models and frameworks for local execution. For instance, models like Meta's Llama 3 series (Grattafiori et al., 2024), particularly instruction-tuned variants such as Llama 3 Instruct, offer strong performance across a range of tasks.

Frameworks like **Ollama**[3] simplify the process of running these large models locally on standard hardware, enabling researchers and developers to leverage their capabilities without relying solely on proprietary cloud-based APIs. This local execution capability is particularly relevant for processing potentially sensitive engineering documents or for conducting reproducible research. While powerful, LLMs are not infallible knowledge bases. They can generate plausible-sounding but incorrect information ("hallucinations"), and tracing the roots of their generated statements is difficult (Yao et al., 2023). This motivates integrating them with external, verifiable knowledge sources or incorporating explicit validation mechanisms.

[3] https://ollama.com/

## 2.5 Multi-agent Systems for Knowledge Engineering

Building upon the capabilities of Large Language Models (Section 2.4), a promising architectural paradigm for orchestrating complex knowledge-intensive tasks like Knowledge Graph Construction (KGC) is the Multi-Agent System (MAS). A MAS is conceptualized as a system composed of multiple computational entities, known as agents, which operate within a defined context or environment, functioning with a degree of independence to achieve individual or collective goals (Wooldridge, 2009). Key characteristics commonly attributed to agents include (Wooldridge, 2009):

- **Autonomy:** Agents operate without direct, continuous human intervention and exercise control over their internal state and actions.
- **Reactivity:** Agents perceive aspects of their environment and respond in a timely fashion to changes that occur in it. While classic MAS research often focuses on reactivity to dynamic, unpredictable external worlds (e.g., robotics), agents within more constrained workflow systems (like those potentially used for KGC) exhibit reactivity primarily by responding to specific inputs or triggers within that defined workflow (e.g., receiving data from another component).
- **Pro-activeness:** Agents do not simply act in response to their environment; they are capable of exhibiting goal-directed behavior by taking the initiative.
- **Social Ability:** Agents interact with other agents (and possibly humans) to achieve their goals. In theoretical MAS, this can involve complex communication protocols, negotiation, and coordination. Within structured, workflow-oriented LLM systems, this 'social' interaction is often realized through more constrained mechanisms, such as passing structured data, results, or status updates according to predefined protocols rather than open-ended communication.

Historically, MAS architectures have proven valuable for tackling problems characterized by distribution, complexity, or the need for coordinated action among specialized entities, finding application in fields like distributed problem solving, supply chain management, automated negotiation, and complex simulations (Dorri et al., 2018). The architectural characteristics commonly employed in MAS design, such as the decomposition of a large problem into tasks handled by specialized, autonomous or semi-autonomous agents, naturally lead to potential benefits. For example, this decomposition directly supports modularity, facilitating easier development, testing, and maintenance of individual agent components. The distribution of tasks across multiple agents enables inherent parallelism, allowing concurrent operation on different parts of a problem. Furthermore, the modular nature can contribute to robustness, as the overall system may be able to degrade gracefully or adapt if individual agents encounter failures.

The recent integration of powerful LLMs (Section 2.4) as the cognitive engine within individual agents has spurred the creation of sophisticated LLM-powered agent systems (Wang et al., 2023). This paradigm seeks to combine the flexible reasoning and natural language capabilities of LLMs with the structuring principles derived from MAS research. The practical implementation of such systems is increasingly supported by development frameworks. **LangChain**[4] provides a popular open-source library offering modular components for building LLM-driven applications, including tools for prompt management, model interaction, data connection, memory, and chaining operations. Building on such foundations, frameworks like **LangGraph**[5] specifically target the creation of stateful, multi-actor applications by allowing developers to define workflows as graphs where nodes represent agents or tools and edges manage the flow of a shared state, naturally supporting cycles and complex agent interactions necessary for many MAS designs. These tools lower the barrier to entry for developing complex, agent-based LLM applications.

[4] https://www.langchain.com/
[5] https://python.langchain.com/docs/langgraph/

Conceptually applying MAS principles to the domain of Knowledge Engineering, and specifically to the challenges of KGC (discussed in Section 2.2), offers a potentially compelling strategy. The inherently multi-stage process of knowledge extraction, structuring, validation, and integration aligns well with the MAS philosophy of task decomposition and specialization. Theoretically, breaking down the KGC workflow could allow specialized agents, potentially leveraging LLMs, to focus on distinct subtasks (e.g., extraction, validation, linking). Furthermore, a structured interaction protocol between agents could enable collaborative refinement or validation loops, potentially mitigating known LLM weaknesses like hallucination through targeted checks. The modularity inherent in this approach could also facilitate scalability and the integration of human oversight at specific workflow points. Understanding these theoretical benefits and the enabling role of modern frameworks provides valuable context for considering novel approaches to automated KGC. .

# 3 Related work

While Section 2 established the theoretical underpinnings of Knowledge Graphs (KGs), Large Language Models (LLMs), embedding models, Multi-Agent Systems (MAS), and the ADAS domain, this section delves into specific research efforts that directly inform the objectives of this thesis. Constructing comprehensive KGs automatically, particularly from specialized and unstructured text, remains a significant challenge. The recent advances in LLMs and agent-based systems present new opportunities for tackling this, especially within complex engineering fields like ADAS. This review focuses on prior work most pertinent to our approach, examining existing research in: (1) automated KG construction techniques, with an emphasis on LLM-driven methods; (2) the application of multi-agent systems and agent-inspired architectures, particularly those powered by LLMs, to complex information processing and knowledge engineering tasks; (3) methods for validating extracted knowledge, ensuring commonsense plausibility, and maintaining coinsistency within KGs; and (4) existing efforts related to domain-specific KG construction, specifically within the automotive and ADAS context. By analyzing these related works, we aim to situate our proposed agent-inspired KG construction pipeline and highlight its contributions.

## 3.1 Knowledge Graph Construction from Text

The automated creation of KGs from unstructured text is a well-established research field. Early influential approaches often relied on pipeline architectures. NELL (Never-Ending Language Learner) (Carlson et al., 2010) exemplified this by using coupled semi-supervised learners for entity classification and relation extraction from web corpora, aiming for continuous knowledge acquisition. However, its reliance on bootstrapping and separate modules made it susceptible to semantic drift and error propagation (Ye et al., 2023). Knowledge Vault (Dong et al., 2014) also extracted triples from the web but integrated probabilistic scoring based on existing KGs to improve fact confidence, though still adhering to a multi-stage extraction process. The primary drawback of such pipelines remains the potential for errors in initial steps (like NER or entity typing) to severely impact the quality of downstream relation extraction.

To overcome these limitations, joint modeling and, more recently, end-to-end generative models based on pre-trained transformers have been explored. REBEL (Huguet Cabot & Navigli, 2021) demonstrated the viability of fine-tuning BART (M. Lewis et al., 2020) to directly generate linearized triples from input sentences, performing joint entity recognition and relation extraction. While effective for certain benchmarks, its performance can be sensitive to the target relation schema and may struggle with implicit relations requiring multi-sentence reasoning. That is, relationships not explicitly stated within a single sentence, but rather implied across multiple sentences or requiring inference from broader context, may be missed by models processing text primarily on a sentence-by-sentence basis. Other frameworks like TANL (Paolini et al., 2021) and UIE (Lu et al., 2022) leveraged T5-like models (Raffel et al., 2023) to unify various Information Extraction (IE) tasks under a text-to-text generation paradigm. These models

offer great flexibility but may lack the robustness or specialized mechanisms needed for high-precision extraction from dense, domain-specific technical documents like those found in ADAS engineering, which often feature complex terminology and sentence structures (cf. Nouri et al., 2024, discussed further in Section 3.5, who note LLM limitations with domain specificity). Furthermore, these generative models often lack integrated mechanisms for explicit validation against domain constraints or external knowledge sources. While some research focuses specifically on KGC from scientific or technical literature (e.g., Luan et al., 2018 using DyGIE++; Beltagy et al., 2019 developing SciBERT), these often require domain-specific fine-tuning or architectures. Work on extracting knowledge from requirements specifications exists (e.g., Hou et al., 2020 using NLP pipelines; Waseem and Idris, 2020 exploring ontologies from requirements), but often uses more traditional NLP methods or focuses narrowly on requirements traceability rather than broader system knowledge. More advanced methods are leveraging multi-agent architectures, such as the CoMaKG-RAG framework (Krishna et al., 2024), which employs collaborative agents for KG curation from multi-modal sources, aiming for richer semantic structures (discussed further in Section 3.4).

## 3.2 Synergies between LLMs and Knowledge Graphs

The potential for LLMs and KGs to mutually enhance each other is a rapidly growing research area (Pan et al., 2024). One direction involves using KGs to improve LLM performance. Techniques often fall under the umbrella of Retrieval-Augmented Generation (RAG), where relevant factual snippets from a KG or text corpus are retrieved and provided as context to the LLM during generation (P. Lewis et al., 2020). This aims to ground LLM outputs in verifiable facts, reducing hallucinations and enhancing faithfulness, particularly for knowledge-intensive tasks. Studies like LAMA (LAnguage Model Analysis) (Petroni et al., 2019) quantitatively showed that LLMs implicitly store factual knowledge but also highlighted inconsistencies and limitations in reliably retrieving it.

Conversely, LLMs are being leveraged for KG construction and maintenance. AutoKG (Chen & Bertozzi, 2023) explored using few-shot prompting with GPT-3.5 to perform KG completion tasks, demonstrating the potential of LLMs with minimal task-specific training. (Wadhwa et al., 2023) specifically investigated LLMs for relation normalization, showing their ability to map diverse textual expressions onto canonical KG relation types. Other works explore LLMs for entity linking (Kolitsas et al., 2018 pre-dating large LLMs but relevant for context; newer works often use LLMs for zero-shot linking) or even schema induction. However, relying solely on LLM prompts for KGC faces challenges: results can be sensitive to prompt formulation (Wei et al., 2023), ensuring consistency across large documents is difficult, scaling prompts for complex, multi-stage extraction workflows can be inefficient, and validating the veracity of extracted information remains critical, especially given the potential for hallucination (Yao et al., 2023). Frameworks integrating LLMs more deeply into the KGC process, such as multi-agent systems (e.g., Krishna et al., 2024), attempt to mitigate these issues through structured interaction and specialized task handling.

## 3.3 Semantic Representation and Linking

Semantic embeddings are crucial tools supporting KGC. Text embeddings, such as those generated by Sentence-BERT (SBERT) (Reimers & Gurevych, 2019) or newer models like E5 (Wang et al., 2024), map sentences or text passages to vectors where semantic similarity corresponds to spatial proximity. This capability is vital for entity linking, where mentions in text must be disambiguated and linked to canonical entities in the KG. Dense retrieval methods, which encode mentions and entity candidates in the same vector space and use nearest neighbor search, are common (L. Wu et al., 2020; Gillick et al., 2019). Knowledge Graph Embeddings (KGEs), such as TransE (Bordes et al., 2013), DistMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016), or RotatE (Sun et al., 2019), focus on learning embeddings

for entities and relations directly from the KG structure, primarily for internal tasks like link prediction or identifying inconsistent facts within the graph. Some approaches also leverage embeddings for plausibility checking of extracted facts, comparing them against general semantic spaces or commonsense knowledge bases like ConceptNet (Speer et al., 2017) using embeddings like Numberbatch.

## 3.4 Multi-Agent Systems for Complex Tasks

Multi-Agent Systems (MAS), composed of autonomous, goal-oriented, interacting agents (Wooldridge, 2009), offer a paradigm for modularizing complex problem-solving. The advent of powerful LLMs has recently catalyzed the development of LLM-powered agent frameworks (Wang et al., 2023). Systems like AutoGen (Q. Wu et al., 2023) provide frameworks for building conversational agents that collaborate to accomplish tasks, often through simulated conversations and role-playing. Other widely discussed conceptual explorations, such as BabyAGI and AutoGPT, popularized the notion of autonomous LLM agents capable of planning, tool use, and memory management for open-ended tasks. More formal research has explored LLM agents for tasks like software development (Park et al., 2023, simulating human interactions in a virtual "sandbox"), scientific experimentation (Boiko et al., 2023), and complex reasoning requiring decomposition. Applying MAS or agent-based thinking specifically to KGC or broader knowledge engineering is an emerging area. Illustrating the direct application of MAS to KGC, (Krishna et al., 2024) proposed CoMaKG-RAG, a collaborative multi-agent LLM framework explicitly designed for KG curation and querying from multi-modal sources. Their system utilizes distinct agents for query generation, domain model creation, population, KG curation (using the graph database Neo4j, introduced in Section 2.2), and querying, emphasizing inter-agent feedback loops and the creation of intermediate domain models to build more semantically rich and hierarchical KGs than traditional methods. This approach closely aligns with the principles underpinning our work, demonstrating the viability of using specialized, collaborative LLM agents for complex KGC tasks.

## 3.5 Knowledge Representation in Automotive Engineering

The application of explicit knowledge representation techniques, such as ontologies and Knowledge Graphs (KGs), have enabled significant progress in managing the inherent complexity of Advanced Driver Assistance Systems (ADAS) and Autonomous Driving (AD) (Halilaj et al., 2022). These structured approaches aim to formally model the driving environment, system behaviors, and safety considerations to support reasoning and decision-making. (Huang et al., 2019) exemplified this by developing an ontology for urban autonomous driving, encompassing scene elements, vehicle behaviors, and rules to drive situation assessment and decision logic using a Prolog reasoner. Building on the graph-based paradigm, (Halilaj et al., 2021) proposed the CoSI KG framework, which integrates diverse information about the driver, vehicle, and environment into a KG for situation comprehension, showing improved performance with Graph Neural Networks (GNNs) compared to traditional methods.

Further demonstrating the scale and utility of KGs in this domain, (Mlodzian et al., 2023) introduced the nuScenes Knowledge Graph (nuKG). Constructed from the extensive annotations of the popular nuScenes dataset, nuKG provides a large-scale, structured representation of complex driving scenes, capturing entities (vehicles, pedestrians, map elements), their attributes, and their spatio-temporal and semantic relationships. This work highlights the potential of KGs derived from sensor dataset annotations to serve as a rich resource for deeper scene understanding, enabling complex querying and reasoning beyond raw object detection.

While nuKG focuses on knowledge derived from annotated sensor data, other approaches leverage KGs for analyzing safety insights from textual sources. The methodology used by (Huo et al., 2024) in the domain of subway construction, for instance, shows how KGs built from accident reports can reveal coupling mechanisms between risk hazards through topological analysis, suggesting potential for applying

similar data-driven KG techniques to ADAS/AD incident data for safety learning.

However, populating these KGs or ontologies often relies on manual input or semi-automated methods tied to specific structured data sources. Automatically extracting and structuring comprehensive knowledge from the large volumes of unstructured technical documentation (e.g., design specifications, test reports, technical manuals) inherent in ADAS development remains a significant challenge.

Addressing this gap by leveraging recent AI advancements, (Nouri et al., 2024) specifically investigated the use of Large Language Models (LLMs) to support safety engineers. They developed an interactive LLM-based prompt pipeline to assist with Hazard Analysis and Risk Assessment (HARA) and the generation of safety requirements compliant with ISO 26262 (ISO, 2018) and ISO 21488 (SOTIF) (ISO, 2022). Their work, evaluated through expert reviews, emphasized both the capabilities of LLMs to process and generate relevant text and their significant limitations in the safety domain, such as difficulties with precise technical terminology, the need for highly detailed prompts to guide reasoning, and the absolute necessity of human expert validation due to potential inaccuracies or lack of domain grounding. This study underscores the challenges of applying general LLMs directly to safety-critical engineering knowledge tasks without robust structuring and validation mechanisms.

The need for more automated, reliable, and scalable methods for knowledge extraction and management is amplified by the increasing complexity of ADAS/AD functions moving towards higher automation levels (Level 3+ and beyond, see Section 2.1.1) and the associated requirements for rigorous ODD definition and safety assurance.

# 4 Methodology

## 4.1 Design Science Research

The investigation undertaken in this thesis is guided by the principles and practices of Design Science Research (DSR). DSR is a well-established research paradigm within information systems and related fields, distinguished by its focus on creating innovative artifacts to address real-world problems and extend the boundaries of existing capabilities (Hevner et al., 2004). Unlike explanatory research methodologies that primarily aim to understand or explain phenomena, DSR is inherently proactive and prescriptive; its core objective is to develop novel and purposeful artifacts, which may include constructs, models, methods, or instantiations and that solve identified problems and demonstrate utility within a specific context (Hevner et al., 2004; Peffers et al., 2007).

The adoption of a DSR methodology is particularly well-suited for the objectives of this research. The central goal is not simply to analyze the challenges associated with knowledge management in ADAS/AD systems engineering, but to actively design, implement, and evaluate a novel IT artifact, the Multi-Agentic Knowledge Graph (MAKG) system, conceived as a concrete solution to these identified challenges. This focus on artifact creation as a primary research output aligns directly with the fundamental tenets of DSR. Furthermore, DSR emphasizes the crucial interplay between practical relevance and academic rigor. This means that the research aims not only to produce an artifact that offers tangible utility in addressing real-world problems (problem relevance), such as the knowledge management bottleneck in ADAS development, but also to ensure that the design process, the artifact itself, and its evaluation are conducted systematically and contribute meaningfully to the existing academic knowledge base (research rigor) concerning automated knowledge graph construction, agent-inspired systems, and LLM applications in engineering domains (Hevner et al., 2004; vom Brocke et al., 2020).

An effective DSR requires adherence to principles that ensure both the quality of the artifact and the research process. (Hevner et al., 2004) outline key guidelines, including the explicit identification of the

artifact **(Guideline 1: Design as an Artifact)**, demonstration of its relevance to an important problem **(Guideline 2: Problem Relevance)**, rigorous evaluation of its utility **(Guideline 3: Design Evaluation)**, clear articulation of research contributions **(Guideline 4: Research Contributions)**, grounding in the existing knowledge base **(Guideline 5: Research Rigor)**, recognition of design as an iterative search process **(Guideline 6: Design as a Search Process)**, and clear communication of the findings **(Guideline 7: Communication of Research)**. This thesis aims to adhere to these guidelines throughout the research process. The DSR process itself is often characterized as iterative, involving cycles of problem understanding, solution design, artifact implementation, demonstration, and evaluation (Peffers et al., 2007; vom Brocke et al., 2020).

This iterative cycle allows for learning and refinement as the research progresses. The overall structure of this thesis aligns with the stages typical of a DSR project, as outlined in the project outline (Section 1.3 in Chapter 1). This current chapter (Chapter 4) focuses primarily on detailing the design and development methodology for the MAKG artifact (corresponding to Stage 3 of the DSR process model), outlining its architecture, component design principles, and implementation context. Subsequent chapters will then describe the demonstration and evaluation strategies and present the findings. This structured approach ensures that the development of the MAKG system is systematically conducted and rigorously evaluated according to established DSR practices.

## 4.2 Design Principles and Architecture

Following the Design Science Research methodology outlined in Section 4.1, the core of this research involves the design and development of the Multi-Agentic Knowledge Graph (MAKG) system artifact. This system is conceived as an interesting approach to automate the extraction, validation, and structuring of engineering knowledge from unstructured ADAS technical documents into a formal Knowledge Graph. This section details the underlying design principles, the overall system architecture, the specific roles assigned to the constituent components (including LLM-powered agents and deterministic modules), the workflow governing their collaboration, and the methodological approaches embedded within the core knowledge processing mechanisms.

### 4.2.1 An Agent-Inspired Pipeline Approach

The MAKG system architecture is fundamentally inspired by Multi-Agent System (MAS) principles, as introduced conceptually in Section 2.5, but implemented as a structured pipeline involving both LLM-powered agents and specialized code-based modules, an approach chosen for its suitability in addressing the complexities of automated Knowledge Graph Construction (KGC) from technical texts. This design is driven by several core principles. Firstly, the intricate KGC process naturally decomposes into distinct sub-tasks (extraction, normalization, validation, integration); this architecture embraces modularity and specialization by assigning each function to a dedicated component, facilitating focused development and maintenance, reflecting benefits often cited for MAS (Wooldridge, 2009). Secondly, a structured workflow is enforced, where these components interact via defined protocols in a largely sequential pipeline (with specific feedback loops), providing greater control over the knowledge construction process compared to more dynamic agent systems. Thirdly, this structure allows for strategically leveraging appropriate technologies: LLMs (Section 2.4) are employed for tasks demanding sophisticated language understanding (like extraction and normalization), while deterministic code utilizing semantic embeddings (Section 2.3) is used for operations requiring high precision and consistency (like validation and matching). Critically, this decomposition enables integrated validation for enhanced reliability; dedicated validation components (like the Commonsense Verifier module) are embedded within the pipeline to systematically mitigate known LLM weaknesses, such as hallucination (discussed in Section 2.4), thereby increasing the trustworthiness of the final KG for the safety-critical ADAS domain. Therefore, the MAKG system consists of this orchestrated pipeline of specialized components, intelligent agents and

deterministic modules, working collaboratively to transform input documents into a structured, validated Knowledge Graph repository.

### 4.2.2 System Components and Workflow

The agent-inspired pipeline architecture, justified in the previous section, is realized through a series of specialized components, including both LLM-powered agents and deterministic code-based modules. These components are orchestrated within a structured workflow designed to process unstructured ADAS technical documents and progressively build a domain-specific Knowledge Graph.

A cornerstone of the system's design is its configuration-based structure. Each core component operates based on parameters defined in dedicated JSON configuration files (see Appendix B). These files centralize settings such as the specific Large Language Model (LLM) to be utillized for agentic tasks, paths to prompt templates (see Appendix A), input/output file locations, name of embedding models, similarity thresholds for matching operations, API credentials (e.g., for Neo4j database connection), and operational parameters such as batch processing sizes. For LLM-based tasks within this research, the system primarly employs instruction-tuned models from the Llama 3 series (Grattafiori et al., 2024) (specifically, Llama 3 Instruct), accessed and run locally via the Ollama framework (as introduced in Section 2.4), ensuring control over the execution environment. This configuration-driven approach provides significant flexibility, allowing for systematic tuning of each individual component behavior and the overall pipeline dynamic, thereby facilitating experimentation, adaptation and reproducibility without requiring modifications to the core codebase.

The system's workflow and state management are implemented using **LangGraph**, a library that extends LangChain (discussed in Section 2.5) to build stateful, multi-actor applications. LangGraph allows the pipeline to be defined as a cyclical graph where nodes represent the configured processing components (agents or modules) and edges dictate the flow of data. A shared state object persists and transfers information between these nodes, enabling complex interactions including conditional branching and loops, which are integral to the proposed workflow. The target Knowledge Graph is managed using Neo4j graph database (Section 2.2).

The core components (graph nodes) and their roles within the MAKG pipeline are defined as follows:

- **Document Preprocessing (Implicit Start):** The process begins with ingesting the input PDF document and chunking it using the `read_pdf_chunked` tool, configured via parameters within the Triplet Extractor's JSON configuration file. The resulting list of text `chunks` is populated into the initial graph state for subsequent processing.
- **Triplet Extractor (LLM Agent Node):** This node receives the text `chunks` from the state. Its primary task is to extract candidate knowledge triples (Subject, Predicate, Object) from each chunk. The implementation includes error handling with retries and leverages the `repair_json_tool` to ensure valid JSON output. The aggregated list of extracted triples is stored in the `inferred_relations` key of the graph state.
- **Predicate Normalizer (LLM Agent Node):** This node takes the `inferred_relations` list. It uses a prompt that contains definitions of the canonical predicate vocabulary. For each input triplet, it aims to replace the raw, extracted predicate string with the semantically closest canonical predicate term. This node processes triplets in batches for efficiency and outputs the list of triplets with normalized predicates to the `normalized_triplets` state key.
- **Commonsense Verifier (Code-based Node):** This node operates in two potential passes, controlled by the `commonsense_pass` state variable ("first" or "second"). It processes triplets from

either `normalized_triplets` (Pass 1) or `inductive_triplets` (Pass 2). It applies a deterministic hybrid scoring logic using pre-loaded **Numberbatch** embeddings and **E5** sentence embeddings (Section 2.3) to assess the semantic plausibility of each triplet against the configured thresholds. Verified triplets are saved to output files and updated in the corresponding state keys (`first_verified_triplets` or `final_verified_triplets`).

- **Inductive Reasoner (LLM Agent Node - Conditional):** This optional node is triggered after the first pass of the Commonsense Verifier if verified triplets exist. It clusters the verified triplets using `cluster_triplets_util` and then uses a configured LLM to infer potentially novel triplets based on patterns within each cluster. The generated triples are output to the `inductive_triplets` state key, destined for verification in the second pass.
- **Set Second Pass (Utility Node):** A simple state-updating node that transitions the `commonsense_pass` variable to "second" after the Inductive Reasoner completes, ensuring the workflow correctly routes the induced triplets back to the Commonsense Verifier.
- **Triplet Matcher (Code-based Node):** This node executes after all verification steps. It takes the `final_verified_triplets` list. It first triggers Neo4j data export and then precomputes embeddings for existing KG triples, entities, and relations, saving them to configured paths. It then compares the embeddings of incoming `final_verified_triplets` against the precomputed KG triplet embeddings using cosine similarity via the `match_against_neo4j` function, applying the configured `matcher.threshold`. Triplets exceeding the threshold are deemed duplicates. The node outputs two lists to the state: `matched_triplets` and `unmatched_triplets`. This component operates in batches.
- **KG Curator (LLM Agent Node):** This node processes the `unmatched_triplets`. It performs: (a) Term Mapping: Using SBERT and precomputed KG embeddings, it maps incoming terms (subject, predicate, object) to existing KG terms if similarity exceeds the configured similarity threshold. (b) Entity Labeling & Query Generation: It employs the configured LLM and target labels from its config file (`target_kg_labels`) to assign appropiate Neo4j node labels. If unsure, it may assign a default label :Review. It then formats predicates and generates Neo4j Cypher `MERGE` queries for integrating the triplets to the database. Outputs are `curated_triplets` and `cypher_queries`. This node also runs in batches (size from config).
- **Graph Updater (Code-based Node):** The final node in the main pipeline. It takes the `cypher_queries` list. Using Neo4j connection details from the shared config, it connects to the database and executes each query within a transaction. Reports status to the `graph_update_status` state key.

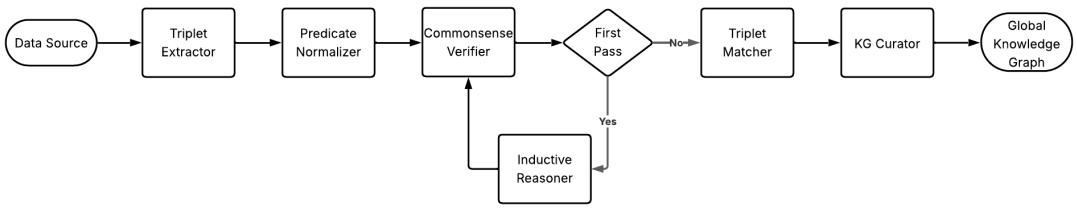


*Figure 1.* MAKG System Worklow

The MAKG system (as seen in Figure 1) operates through the sequential pipeline defined in the LangGraph application (`main.py`), incorporating a conditional loop for inductive reasoning. Input documents are chunked, followed by triplet extraction (Component 2) and predicate normalization (Component 3). The Commonsense Verifier (Component 4) performs its first pass. A conditional router (`route_after_verification`) then determines the path: if verified triples resulted from Pass 1, the

Inductive Reasoner (Component 5) generates new triples, the state is updated (Component 6), and the Commonsense Verifier (Component 4) runs its second pass on the induced triples. If no triples were verified in Pass 1, or after Pass 2 completes, the flow proceeds to the Triplet Matcher (Component 7), which filters out duplicates against the existing Neo4j KG. The remaining novel, verified triples are passed to the KG Curator (Component 8) for term mapping, entity labeling, predicate standardization, and Cypher query generation. Lastly, the Graph Updater (Component 9) executes these queries, persisting the new knowledge into the Neo4j database and concluding the graph execution. This intricate process, combining LLM agents with deterministic modules for verification and matching, aims to construct an accurate, consistent, and semantically rich Knowledge Graph from the source documents.

### 4.2.3 Core Mechanisms Design

The MAKG system's ability to transform unstructured ADAS technical documents into a structured Knowledge Graph hinges on the methodological design of its core processing mechanisms. These mechanisms are distributed across the system's components (agents and modules, as described in Section 4.2.2) and are engineered to synergistically leverage Large Language Models (LLMs), semantic embeddings, and rule-based logic within the defined pipeline.

**Knowledge Extraction using LLMs (Triplet Extractor Agent)** The foundational process of identifying candidate knowledge within raw text is entrusted to the Triplet Extractor Agent. The core methodology is strategic prompt engineering, designed to guide a pre-trained LLM (Section 2.4) to perform Information Extraction. This involves crafting prompts that precisely define the task (extracting subject-predicate-object triples relevant to ADAS concepts like system components, functions, or safety constraints), specify a structured output format (e.g., JSON, for subsequent parsing by `repair_json_tool`), and provide sufficient textual context from document chunks. The prompt design (`triplet_extractor.txt`) implicitly guides the LLM towards generating entities and relations amenable to later ontological alignment by suggesting preferred predicate styles (e.g., verb-based) and handling for definitional statements (e.g., using 'is_defined_as'). Methodologically, this stage aims to capture a comprehensive set of potential facts, with the understanding that subsequent pipeline stages will address precision through normalization and validation. The choice of LLMs for this task is driven by their inherent strength in natural language understanding and their ability to identify relationships not explicitly defined by rigid rules.

**Semantic Normalization of Predicates (Predicate Normalizer Agent)** To ensure semantic consistency within the generated KG, the Predicate Normalizer Agent employs an LLM-based methodology. Raw predicates extracted by the previous agent often exhibit considerable linguistic variability. This agent's task is to map these varied expressions to a predefined set of canonical relation types, as defined in its prompt file (`predicate_normalizer.txt`). The LLM is prompted to analyze the semantics of an input triplet's predicate in the context of its subject and object and to select or rephrase it into the most fitting canonical term. This leverages the LLM's understanding of paraphrase and semantic similarity, aiming to consolidate semantically equivalent relationships under a single, standardized representation, which is crucial for the KG's coherence and queryability.

**Embedding-Based Plausibility Validation (Commonsense Verifier Module)** A critical step for ensuring the quality of the KG is the validation of extracted triplets. The Commonsense Verifier module employs a deterministic, hybrid embedding-based methodology for this purpose, operating independently of LLM generative processes at this stage to provide an objective check against potentially erroneous or nonsensical extractions. The decision to use a hybrid approach, combining scores from two distinct types of semantic embeddings spaces, is driven by the complementary strengths they offer for assessing plausibility:

- **Commonsense Knowledge Embeddings (Numberbatch):** This components utilizes pre-trained word and concept embeddings from Numberbatch (derived from ConceptNet, as discussed in Section 2.3). Numberbatch embeddings are trained on a large commonse knowledge graph, and excel at capturing broad, decontextualized semantic relatedness and commonsense relationships between individual concepts. For a given triplet (Subject, Predicate, Object), the cosine similarity between the Numberbatch embeddings of the subject and object, and potentially the predicate, can indicate their general semantic coherence and whether they belong to types of entities that typically interact in the manner described by the predicate. This provides a baseline score for fundamental commonsense plausibility. For example, it can help filter out triplets where the subject and object are semantically very distant or mismatched for the given relation (e.g., “Car, isA, Banana").
- **Contextual Sentence Embeddings (E5):** To capture more nuanced, context-dependent plausibility, the module also employs powerful sentence embeddings generated by a model like E5 (Section 2.3). Instead of just looking at individual terms, E5 can enconde the entire triplet, or natural language statements derived from it (e.g., “Subject Predicate Object.", “The Subject is related to the Object via Predicate."), into a single contextual vector. The semantic similarity (cosine similarity) of this statement’s embedding to itself (as a measure of coherence) or to embeddings of known valid statements, or its position within the broader E5 embedding space relative to typical factual assertions, provides a measure of contextual plausibility. This helps capture whether the specific combination of entities and the relation makes sense as a complete factual statement, even if the individual terms are generally related. For instance, while “Vehicle" and “SteeringWheel" are commonsensically related, a sentence embedding approach can better assess the typicality of a statement like “Vehicle relatedTo SteeringWheel" compared to an atypical one.
- **Methodological Rationale for Hybrid Approach:** The methodology involves calculating scores from both Numberbatch and E5 embeddings and then combining them, often through a weighted average, to produce a final plausibility score. This hybrid strategy is adopted because neither embedding type alone perfectly captures all aspects of plausibility. Numberbatch provides robust general commonsense, while E5 offers richer contextual understanding for the specific assertion. By combining them, the aim is to create a more robust verifier that is less prone to the weaknesses of a single embedding type. For example, Numberbatch might struggle with highly domain-specific but valid relation if they are not common in general knowledge, whereas E5, by encoding the full statement, might better capture its coherence if similar phrasings have been seen in its vast training data. Triplets falling below a configurable combined threshold are deemed implausible and are filtered from the pipeline. This code-based, dual-embedding driven validation offers a systematic and reproducible check against semantically anomalous or nonsensical extractions prior to KG integration.

**LLM-Driven Inductive Reasoning for Knowledge Enrichment (Inductive Reasoner Agent)** To potentially expand the Knowledge Graph beyond explicitly stated facts, the Inductive Reasoner Agent employs the LLM’s emergent reasoning capabilities (Section 2.4). A key aspect of this agent’s methodology is the initial grouping of previously validated triples into semantically coherent clusters. This clustering step, performed by the `cluster_triplets_util` function, is designed to provide the LLM with more focused and contextually relevant sets of input facts for inference. The clustering methodology uses semantic embeddings generated by a SentenceTransformer model (e.g., `all-MiniLM-L6-v2`, as discussed in Section 2.3). Each triplet (Subject-Predicate-Object) is first converted into a single string representation. These string representations are then encoded into dense vector embeddings. Subsequently, K-Means clustering is applied to these embeddings to group semantically similar triplets. The number of clusters is a configurable parameter (`max_clusters`), allowing for control over the granularity of the input contexts provided to the LLM. Once these clusters are formed, each cluster of validated triples is provided as context to the configured LLM. The LLM is then prompted (via `prompts/inductive_reasoner.txt`) to infer novel, plausible relationships based on observed patterns, hierarchies, or implicit domain logic apparent within that specific semantic cluster. The intent

is not deductive logical proof but rather plausible hypothesis generation to uncover potentially valuable implicit knowledge. Critically, all new triples generated by this agent are routed back through the Commonsense Verifier module, subjecting them to the same rigorous embedding-based plausibility checks as initially extracted triples. This ensures that any inferred knowledge also meets a baseline quality standard before being considered for further processing.

**Consistency Checking and KG Integration (Triplet Matcher Module & KG Curator Agent)** The final stages ensure the novelty, consistency, and proper integration of knowledge into the Neo4j graph database.

- **Duplicate Prevention (Triplet Matcher Module):** This code-based module employs an embedding-based methodology to prevent the addition of semantically redundant triples. It compares the SBERT embedding of each validated candidate triple against a precomputed index of embeddings for all existing triples in the Neo4j KG. Triples with a cosine similarity above a set threshold are identified as duplicates. This ensures that only semantically novel information proceeds.
- **Term Standardization and Entity Labeling (KG Curator Agent):** Before final insertion, the KG Curator Agent performs two key operations. Firstly, using SBERT embeddings, it maps the subject, predicate, and object terms of novel triples to the most semantically similar terms already present in the Neo4j graph, enforcing vocabulary consistency. Secondly, it employs the configured LLM (guided by `kg_curator.txt` and a list of target Neo4j labels) to assign the most appropriate ontological type (node label) to each unique entity.
- **Cypher Query Generation and Execution (KG Curator Agent & Graph Updater Module):** The KG Curator Agent methodically translates the fully processed (validated, normalized, linked, de-duplicated, term-standardized, and entity-labeled) triples into Cypher `MERGE` queries. These queries are designed to correctly create nodes with appropriate labels and properties, and relationships with correct types and directions, adhering to the Labeled Property Graph model (Section 2.2). The Graph Updater module then executes these queries against the Neo4j database in a transactional manner.

This layered approach to extraction, normalization, validation, reasoning and integration, combining LLM flexibility with deterministic checks, defines the core methodology for constructing the MAKG.

## 4.3 Target Knowledge Graph Schema and Initial State

The Multi-Agentic Knowledge Graph (MAKG) system is designed to extract, structure, and integrate information into a Neo4j graph database that adheres to a predefined domain-specific schema. This schema provides the semantic framework, controlled vocabularies for node labels (entity types) and relationship types (predicates), and defines the foundational structure of the Knowledge Graph (KG) before any document processing begins.

The target schema is based on the Labeled Property Graph (LPG) model utilized by Neo4j (as introduced in Section 2.2). Key entity types defined for the ADAS domain include, for example, `:System`, `:Sensor`, `:Requirement`, `:Hazard`, and `:Feature`. These types are assigned as labels to nodes in the graph. Similarly, a set of canonical predicate types, such as `isA`, `hasPart`, `defines`, `triggers`, and `references`, are used to define the relationships between these nodes. The Predicate Normalizer Agent (Section 4.2.2) is specifically tasked with mapping extracted textual relations to these canonical types, while the KG Curator Agent ensures appropriate node labeling.

The full list of target Neo4j node labels used in this research is provided in Appendix D. The complete canonical predicate vocabulary, along with their intended definitions (as utilized in the Predicate Normalizer prompt), is detailed in Appendix E.

For the evaluation experiments, the Neo4j database is initialized with a base set of triples representing this core ontological structure. This initial KG state includes nodes for each defined entity type and key high-level relationships establishing the foundational taxonomy and core domain concepts (e.g., defining that a `:Sensor` `isA` a `:Component`, or that an `ISO 26262` node `isA` a `:Standard`). The complete list of triples constituting this pre-loaded base ontology is available in Appendix C. This initial state serves as the foundation upon which new, instance-level knowledge extracted from ADAS documents is integrated by the MAKG pipeline.

## 4.4 Evaluation Strategy

The evaluation of the Multi-Agentic Knowledge Graph (MAKG) system artifact is a cornerstone of the Design Science Research (DSR) methodology employed in this thesis, as outlined in Section 4.1. The primary goal of this evaluation is to systematically assess the system's effectiveness in automatically constructing a coherent, accurate, and domain-relevant Knowledge Graph from unstructured ADAS technical documentation. Furthermore, this evaluation seeks to systematically analyze the specific contributions and performance of the core components within the agent-inspired pipeline, with a particular focus on the integrated validation and normalization mechanisms. This section details the overarching evaluation objectives, the specific metrics that will be used for assessment, and the planned experimental procedures designed to gather empirical evidence regarding the MAKG system's capabilities.

### 4.4.1 Evaluation Objectives

The evaluation strategy is directly aligned with the research questions articulated in Section 1.1 and aims to demonstrate the utility and novelty of the MAKG artifact. The specific objectives are:

- **Objective 1:** To establish the feasibility and operational functionality of the end-to-end MAKG pipeline in automatically processing ADAS technical documents and generating a populated, structured Knowledge Graph. This involves assessing the system's ability to execute the workflow, addressing aspects of **RQ1**.
- **Objective 2:** To evaluate the effectiveness of the integrated validation mechanisms, particularly the Commonsense Verifier (including its two-pass operation with the Inductive Reasoner), in enhancing the accuracy and reliability of the knowledge by analyzing its filtering behavior on triples generated during the pipeline execution, addressing aspects of **RQ2**.
- **Objective 3:** To assess the system's ability to maintain semantic consistency. This includes evaluating the effectiveness of the Predicate Normalizer in standardizing relationship types and the Triplet Matcher in identifying and managing redundant information against the existing KG, addressing aspects of **RQ3**.

### 4.4.2 Evaluation Metrics and Procedure

A multi-faceted approach, combining quantitative metrics and qualitative analysis, will be employed to assess the MAKG system against the evaluation objectives outlined in Section 4.4.1. The core evaluation will involve processing a representative ADAS technical document (referred to as 'document1') through the pipeline.

**System Functionality and KG Generation (Objective 1, RQ1)** The MAKG system's fundamental ability to process 'document1' and generate a Knowledge Graph will be evaluated. **Metrics** will include the successful completion of the pipeline and throughput statistics (counts of triples) at each key stage (Extractor, Normalizer, Verifier, Reasoner, Matcher, Curator), collected via an automated script. **Qualitative assessment** will involve an inspection of the KG Curator's entity label distribution (particularly the use of default label) and visualizations of representative subgraphs from the populated Neo4j database to illustrate the nature of the constructed KG.

**Effectiveness of Commonsense Verification (Objective 2, RQ2)** The Commonsense Verifier's role in enhancing the perceived accuracy and reliability of the extracted knowledge will be assessed through pipeline data analysis. The rejection and acceptance rates of the Verifier during its first pass (on normalized extracted triples from `document1`) and its second pass (on inductively reasoned triples) will be recorded from the pipeline statistics. A qualitative review of sampled accepted and rejected triples from both passes will be performed to assess the appropriateness and impact of the verifier's decisions within the operational context of the pipeline.

**Semantic Consistency Mechanisms (Objective 3, RQ3)** The system's capabilities for maintaining semantic consistency will be evaluated by examining:

- **Predicate Normalization:** Statistics from the pipeline run on `document1` will show the reduction in distinct predicate types before and after processing by the Predicate Normalizer agent. The distribution of canonical predicates in the normalized output will be analyzed, alongside a qualitative review of a sample of normalization decisions for semantic appropriateness.
- **Triplet Matching for Redundancy Management:** The Triplet Matcher's effectiveness will be evaluated by first running the full pipeline on `document1` to populate the Neo4j KG (establishing a baseline against the pre-loaded ontology). A second run of the pipeline using the same `document1` as input will then be performed against the KG populated by the first run. The number of triples identified as "matched" (i.e., already existing from the first run) by the Triplet Matcher in this second run will be the primary metric. A qualitative review of sampled matched triples from the second run will be conducted.

The quantitative data from the automated statistics script (for Run 1 and Run 2 of `document1`), alongside qualitative observations derived from manual inspection of component outputs (e.g., normalized predicates, verifier rejections/acceptances, matcher decisions, curator labels) and the final Neo4j graph visualizations, will be used to interpret the system's performance. The detailed presentation of these findings will be documented in Chapter 5, followed by an in-depth discussion in Chapter 6.

# 5 Results

The primary focus of the experiments detailed in here is to assess the system's capabilities according to the evaluation strategy outlined in Section 4.4 and analyze the baseline performance of the MAKG system in constructing a Knowledge Graph from a representative ADAS glossary document (henceforth referred to as `document1`), with the Neo4j database initially containing only the base ontological structure defined in Appendix C.

## 5.1 Pipeline Functionality and Overall Knowledge Graph Generation

### 5.1.1 Overall Data Flow and Throughput

The MAKG pipeline successfully processed `document1` end-to-end. The quantitative statistics for the data flow through each key component are summarized in Table **??**. The Triplet Extractor initially identified 148 raw candidate triples from the input document. The Predicate Normalizer processed these, invalidating 7 and outputting 141 normalized triples; this stage also standardized predicate representations, reducing the count of distinct predicates from 71 to 61. The first pass of the Commonsense Verifier then processed these 141 triples, accepting 130 and rejecting 11. The Inductive Reasoner, operating on these 130 verified triples, generated 35 new candidate facts. A second pass of the Commonsense Verifier assessed these 35 induced triples, accepting 34 and rejecting only 1. Consequently, a total of 164 triples (130 from the initial extraction and 34 from induction) were deemed verified and passed to the Triplet Matcher. The Matcher, comparing these 164 triples against the base ontological KG, identified 0 existing matches, thus classifying all 164 as novel and passing them to the KG Curator. The Curator's internal pre-mapping and de-duplication (Pre-LLM) stage then removed 5 redundant triples from this set, yielding 159 unique triples. These were subsequently labeled, resulting in 159 final curated triples for which Cypher queries were generated. Ultimately, these 159 Cypher queries were successfully executed to populate the Neo4j graph.

Table 1

*Pipeline Throughput Statistics (Run 1)*

| **Pipeline Stage / Metric** | **Count** |
|---|---|
| Raw Triplets Extracted (Extractor Output) | 148 |
| Triplets after Normalization (Normalizer Output) | 141 |
| Triplets Invalidated by Normalizer | 7 |
| Distinct Predicates Before Normalization | 71 |
| Distinct Predicates After Normalization | 61 |
| Verifier Pass 1 Input (from Normalizer) | 141 |
| Verifier Pass 1 Accepted Triplets | 130 |
| Verifier Pass 1 Rejected Triplets | 11 |
| Triplets Generated by Inductive Reasoner | 35 |
| Verifier Pass 2 Input (from Inductive Reasoner) | 35 |
| Verifier Pass 2 Accepted Induced Triplets | 34 |
| Verifier Pass 2 Rejected Induced Triplets | 1 |
| Total Verified Triplets for Matching (P1 Accepted + P2 Accepted) | 164 |
| Matcher Input Triplets | 164 |
| Triplets Matched to Existing KG (Ontology) by Matcher | 0 |
| Novel Unmatched Triplets after Matching (Input to Curator) | 164 |
| Curator Input after its Internal Pre-mapping/De-duplication | 159 |
| Triplets De-duplicated by Curator (Pre-LLM) | 5 |
| Final Curated (Labeled) Triplets | 159 |
| Cypher Queries Generated for KG Insertion | 159 |

### 5.1.2 Characteristics of the Generated Knowledge Graph

The processing of `document1` resulted in the addition of 159 novel, structured facts to the initial ontological Knowledge Graph. Within these curated triples, the KG Curator identified and labeled a total of 204 unique entities using 41 distinct Neo4j node labels from the predefined target list (see Appendix D). The distribution of these assigned labels is noteworthy. While many entities received specific ADAS domain

labels such as ":Feature" (12 instances), ":System" (23 instances), or ":ScenarioElement" (4 instances), a significant portion, 74 out of 204 unique entities (approximately 36%), were assigned the fallback ":Review" label.

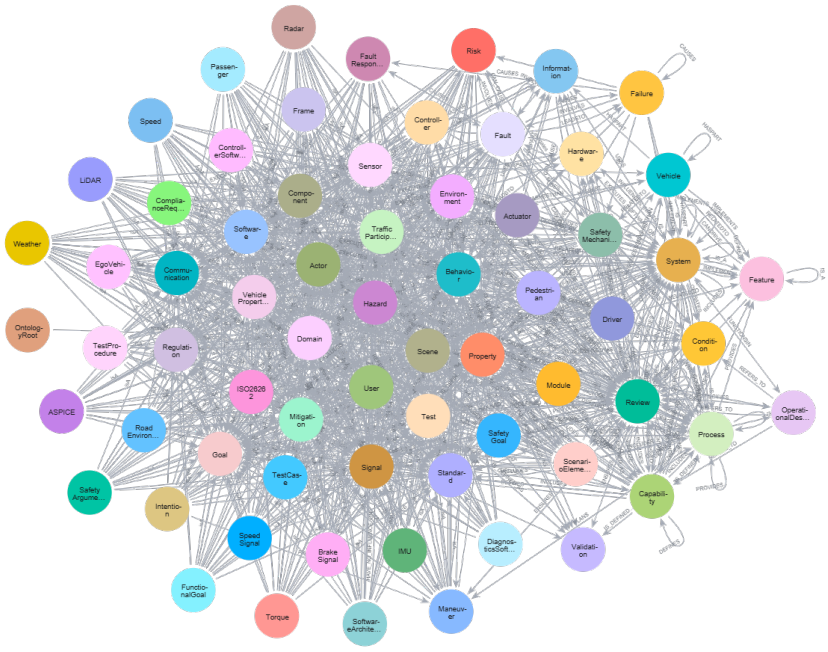


*Figure 2.* Visualization of the ADAS Knowledge Graph after processing Document1 (Run 1), integrating new facts with the base ontology

As seen in Figure 2, the Run 1 was completed succesfully populating the Neo4j knowledge graph. Figure 4 and Figure 3 illustrate representative subgraphs from the Neo4j database after processing `document1`. Figure 4 showcases the extracted definition and related components for the glossary term "[Example Feature]", demonstrating how the system structures definitional knowledge. Figure 3 depicts the relationships generated for "[V2X]". These visualizations confirm the system's capability to automatically generate a structured graph representation from the input text.

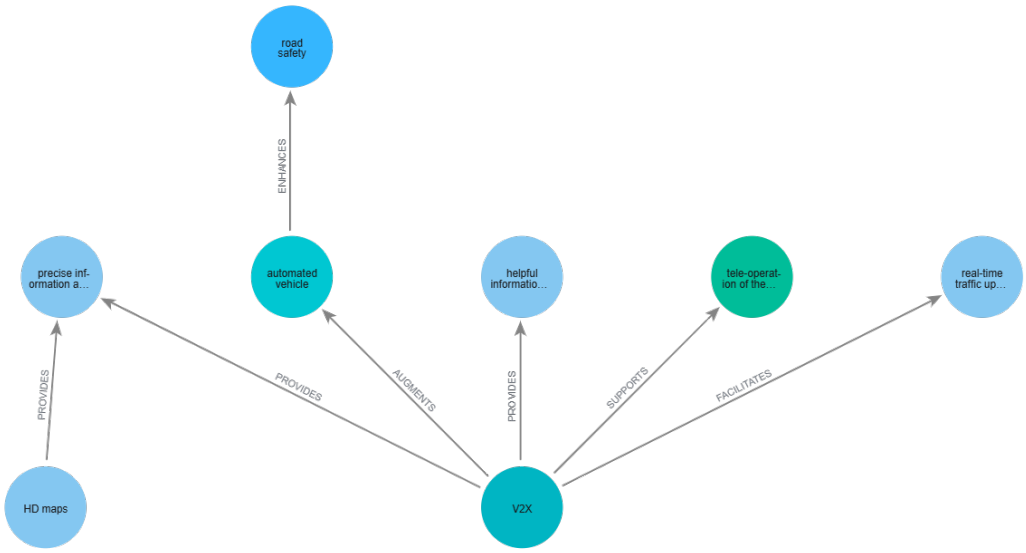


*Figure 3.* Example of sub-graph (V2X node)


*Figure 4.* Example of sub-graph (Example Feature node)

## 5.2 Effectiveness of Knowledge Validation and Refinement Mechanisms

A key objective of this research was to evaluate the system's internal mechanisms designed to improve the accuracy and reliability of the extracted knowledge, thereby addressing RQ2. This involved assessing the performance of the Commonsense Verifier across its two operational passes and the contribution of the Inductive Reasoner.

### 5.2.1 Commonsense Verifier Performance

The Commonsense Verifier module was applied to candidate triples after predicate normalization (Pass 1) and again to triples generated by the Inductive Reasoner (Pass 2), playing a crucial role in enhancing the reliability of the knowledge base.

**Pass 1 Verification (Normalized Extracted Triples):** The Commonsense Verifier (Pass 1) processed the 141 triples received from the Predicate Normalizer. Of these, 130 triples were accepted as plausible, representing an acceptance rate of approximately 92.2%, while 11 triples were rejected. A qualitative review of a sample of the 130 accepted triples from this pass, such as `(accident, involves, material damage)` (Final Score: 0.9175) and `(HD maps, uses, automated driving systems)` (Final Score: 0.9102) from the provided logs, indicated that the verifier successfully validated many factually coherent statements extracted from the glossary. Analysis of the 11 rejected triples would typically reveal instances where the combined Numberbatch and E5 scores fell below the 0.8 plausibility threshold. Such rejections might occur for triples representing more abstract or less common relationships not well represented in the general commonsense models, or for genuinely erroneous extractions from the LLM. For instance, the rejected triple `(features, can drive, vehicle)` was appropriately filtered due to semantic incongruence.

**Inductive Reasoner and Pass 2 Verification (Induced Triples):** Following the first verification pass, the 130 accepted triples were provided to the Inductive Reasoner. This component generated 35 new candidate triples aiming to enrich the knowledge base. These 35 induced triples were then processed by the Commonsense Verifier in its second pass. A significant portion, 34 triples, were accepted as plausible (an acceptance rate of approximately 97.1%), with only 1 triple being rejected. The detailed verification log for this second pass is available in

This high acceptance rate for induced triples suggests that the Inductive Reasoner, when primed with the contextually related and already verified facts from the glossary, tends to generate new statements that largely align with the Commonsense Verifier's plausibility assessment model. Examples of accepted induced triples that demonstrate plausible knowledge extension include:

- `(Scene, includes, artificial marker))` (Final Score: 0.9204)
- `(road user, includes, children)` (Final Score: 0.887)
- `(TAKEOVER, requires, operator attention)` (Final Score: 0.9215)
- `(dependability, includes, fault tolerance)` (Final Score: 0.8367)

These examples illustrate the generation of meaningful relationships that extend the explicitly stated information. The single induced triple rejected by the verifier was `(Simulation, represents, Scene)` with a final score of 0.7844, which fell below the 0.8 threshold, potentially due to the predicate "represents" being too generic in this context or the concepts being too broad for a strong specific link without

further context. The effectiveness of these induced additions in terms of true novelty and contribution to the KG's utility will be further considered in the Discussion chapter.

## 5.3 Semantic Consistency and Structural Integrity

Maintaining semantic consistency and an appropriate, non-redundant graph structure are vital for the usability and integrity of the generated Knowledge Graph. This section evaluates the MAKG system's components designed to address these aspects.

### 5.3.1 Triplet Matcher Performance

The Triplet Matcher aims to prevent the addition of redundant information to the Knowledge Graph by comparing incoming verified triples against the existing KG content. Its performance was assessed in two stages.

In the initial processing of `document1` (Run 1), the Neo4j database contained only the base ontological structure (Appendix C). The Triplet Matcher received 164 verified triples (130 from initial extraction and 34 from induction, as detailed in Table 1). As anticipated, when compared against this primarily schema-level KG, **0 triples were identified as direct semantic matches** by the matcher. Consequently, all 164 triples were passed as novel to the KG Curator for the initial population of factual content related to `document1`.

To specifically assess the matcher's de-duplication capability once facts are ingested, `document1` was processed a second time (Run 2) as seen in Table 2, targeting the Neo4j KG that had been populated by Run 1. In this second run, the pipeline generated and verified a comparable set of candidate triples. The Triplet Matcher received **162 verified triples** as input. Of these, **84 triples were identified as matches** to the triplets already present in the KG from Run 1. This represents a **duplicate detection rate of approximately 51.9%** (84 matched / 162 input). The remaining **78 triples were passed as novel (unmatched)** to the KG Curator.

Table 2

*Pipeline Throughput Statistics (Run 2 - Against Populated KG)*

| **Pipeline Stage / Metric (Run 2)** | **Count** |
|---|---|
| Raw Triplets Extracted (Extractor Output) | 151 |
| Triplets after Normalization (Normalizer Output) | 138 |
| Triplets Filtered/Invalidated by Normalizer | 13 |
| Distinct Predicates Before Normalization | 83 |
| Distinct Predicates After Normalization | 60 |
| Verifier Pass 1 Input (from Normalizer) | 138 |
| Verifier Pass 1 Accepted Triplets | 127 |
| Verifier Pass 1 Rejected Triplets | 11 |
| Triplets Generated by Inductive Reasoner | 35 |
| Verifier Pass 2 Input (from Inductive Reasoner) | 35 |
| Verifier Pass 2 Accepted Induced Triplets | 35 |
| Verifier Pass 2 Rejected Induced Triplets | 0 |
| Total Verified Triplets for Matching (P1 Accepted + P2 Accepted) | 162 |
| Matcher Input Triplets | 162 |
| Triplets Matched to Existing KG (from Run 1) by Matcher | 84 |
| Novel Unmatched Triplets after Matching (Input to Curator) | 78 |
| Curator Input after its Internal Pre-mapping/De-duplication | 78 |
| Triplets De-duplicated by Curator (Pre-LLM) | 0 |
| Final Curated (Labeled) Triplets | 78 |
| Cypher Queries Generated for KG Insertion | 78 |

A manual inspection of a sample of the 84 matched triples recorded from Run 2 confirmed that they were indeed semantically redundant with information that had been successfully extracted, curated, and added to the KG during Run 1. For instance, the incoming triplet from Run 2:


```
{
  "subject": "accident",
  "predicate": "involves",
  "object": "material damage",
  "match": true,
  "similarity": 1.000000238418579,
  "matched_kg_text": "accident INVOLVES material damage"
}
```


was correctly identified as a duplicate, with a similarity score of 1.0, against the identical triplet ingested in Run 1. Even with slight variations in phrasing or case due to the extraction process, the matcher demonstrated robustness. For example, the Run 2 triplet:


```
{
  "subject": "vulnerable road user",
  "predicate": "includes",
  "object": "pedestrians",
  "match": true,
  "similarity": 0.8306937217712402,
  "matched_kg_text": "road user INCLUDES Pedestrian"
}
```

was successfully matched (similarity 0.83) to a pre-existing triplet from Run 1 despite minor differences in the subject ("vulnerable road user" vs. "road user") and object casing ("pedestrians" vs. "Pedestrian"). This validates the Triplet Matcher's effectiveness in significantly reducing the re-insertion of identical or highly similar factual statements when processing overlapping content, thereby contributing to the overall consistency and conciseness of the evolving Knowledge Graph. The 78 unmatched triples from Run 2 represent variations due to LLM stochasticity during earlier pipeline stages in Run 2 that led to generated triples different enough from Run 1's output to fall below the matching threshold, or potentially minor new facts extracted due to subtle differences in LLM processing between runs.

### 5.3.2 Predicate Normalization

The Predicate Normalizer agent is tasked with standardizing the varied predicate expressions extracted by the Triplet Extractor to a predefined canonical vocabulary (see Appendix E), a crucial step for ensuring semantic consistency in the resulting Knowledge Graph.

For `document1`, the Triplet Extractor initially generated 148 candidate triples containing 71 distinct raw predicate strings. After processing by the Predicate Normalizer agent, 141 triples remained (indicating that 7 triples were likely filtered or became invalid during normalization). The number of distinct predicate strings in this processed set was successfully reduced to **61 unique predicate types**. This consolidation represents a 14.1% decrease in predicate diversity (from 71 to 61 unique types), quantitatively demonstrating the normalizer's primary function in standardizing the relational vocabulary.

A qualitative analysis was performed by comparing the original extracted predicates with their normalized counterparts for the first 68 triplets where both raw and normalized versions were available from the provided output snippets. Out of these 68 instances, 15 distinct raw predicates underwent a change during normalization (either semantic mapping or case/inflection adjustment), while the remaining predicates were either already in a canonical form or were passed through unchanged if no direct canonical mapping was found. All 15 of these observed changes resulted in a predicate that is either directly from the canonical list or a correctly cased/inflected version of a canonical predicate, suggesting a high local accuracy for the transformations it did perform.

Examples of successful normalizations observed in the sample include:

- Syntactic standardization, such as mapping the raw predicate `is_defined_as` (e.g., in '(road user, is_defined_as, anyone who uses a road)') to the consistent canonical form `isDefinedAs`. Similarly, `has_part` was normalized to `hasPart`.
- Semantic mapping of ambiguous verbs, for instance, the raw predicate `is` (e.g., in '(artificial marker, is, object or painting)') was correctly mapped to the more specific `isA`.
- Handling of verb inflections, where the raw predicate `carry` (in '(marker, carry, coded information)') was normalized to its canonical present tense form `carries`.
- Effective semantic interpretation of phrases, such as normalizing `is equipped with` (in '(automated vehicle, is equipped with, ...automated driving system)') to the structural predicate `hasPart`.
- Plausible interpretation of relational meaning, for example, mapping `comprises` (in '(automated driving system, comprises, set of elements)') to `includes`.

These examples illustrate the normalizer's capability to standardize syntactic variations and map different phrasings to a consistent set of predefined relations.

However, the normalization process is constrained by the provided canonical vocabulary. Several extracted predicates that did not have a direct or close semantic equivalent in the canonical list were passed through unchanged by the normalizer in the provided samples. For instance:

- The predicate `can be extracted and decoded by` (in '(marker, can be extracted and decoded by, automated driving system)') remained unmapped.
- Similarly, the predicate `offers` (in '(automated driving system, offers, ...use case)') was passed through without normalization.
- Predicates such as `targets` (from '(driver-in-the-loop (DIL), targets, prototypical_or_target_hardware)') and `modifies` (from '(driver-in-the-loop (DIL), modifies, environment_with_virtual_stimuli)') also did not map to a canonical predicate from the list.

These instances underscore that the effectiveness of the LLM-based normalization is significantly influenced by the comprehensiveness and design of the target canonical predicate list. Predicates not normalized at this stage would then depend on the KG Curator's subsequent term mapping capabilities for potential alignment or might be retained as custom relationship types in the Knowledge Graph if they represent valid and necessary domain-specific relations.

# 6 Discussion

## 6.1 Methodology Discussion

This section reflects on the design choices made in developing the Multi-Agentic Knowledge Graph (MAKG) system (detailed in Chapter 4), contextualizing them within the principles of Design Science Research and the related work explored in Chapter 3.

**The Agent-Inspired Pipeline Architecture (Relates to RQ1)** The decision to structure the MAKG system as an agent-inspired pipeline, rather than a monolithic system or a more classical, purely autonomous Multi-Agent System (MAS), was a deliberate methodological choice. While drawing inspiration from MAS principles such as modularity and specialization (Wooldridge, 2009) by decomposing the complex KGC task into distinct components (Extractor, Normalizer, Verifier, etc.), the pipeline enforces a more structured workflow. This controlled flow, orchestrated by LangGraph, was deemed more appropriate for the initial goal of achieving reliable KGC from technical documents compared to more dynamic or conversational agent frameworks like AutoGen (Q. Wu et al., 2023). The CoMaKG-RAG framework (Krishna et al., 2024), which also employs collaborative agents for KG curation, provides a relevant precedent for the utility of multi-component LLM systems in KGC, though our MAKG system integrates both LLM-powered agents and deterministic code-based modules for specific tasks like commonsense verification and triplet matching. This hybrid approach aimed to balance the flexibility of LLMs for NLU-intensive tasks with the precision of algorithmic methods for validation. The configuration-driven nature of each component, allowing for parameter tuning without code changes, was designed to facilitate iterative development and experimentation, aligning with the iterative nature of DSR (Peffers et al., 2007).

**Leveraging LLMs for Core KGC Tasks (Relates to RQ1, RQ2, RQ3)** The methodology heavily relies on LLMs (specifically Llama 3 Instruct via Ollama) for core KGC sub-tasks: initial triplet extraction, predicate normalization, inductive reasoning, and entity labeling within the KG Curator. This choice is grounded in the demonstrated capabilities of LLMs in NLU, IE, and NLG (Section 2.4).

- **Extraction and Normalization:** Using LLMs for extraction and normalization, guided by strategic prompt engineering (Prompts in Appendix A), aligns with recent trends in generative KGC (e.g., REBEL (Huguet Cabot & Navigli, 2021), TANL (Paolini et al., 2021)) but applies it within a decomposed pipeline. The methodology aimed to overcome limitations of purely rule-based or traditional NLP pipelines by leveraging the LLM's broader semantic understanding for interpreting varied linguistic expressions in ADAS documents.
- **Inductive Reasoning:** The use of an LLM for inductive reasoning sought to explore its potential for knowledge discovery beyond explicitly stated facts, a more advanced capability that some LLMs exhibit (Brown et al., 2020).

The inherent challenge with LLMs is their potential for hallucination and lack of guaranteed factual accuracy (Nouri et al., 2024; Yao et al., 2023). This was a primary motivator for the multi-stage pipeline and the inclusion of dedicated validation components, as discussed next.

**Validation and Refinement Mechanisms (Relates to RQ2)** A core methodological contribution is the integration of explicit validation and refinement stages.

- **Commonsense Verifier:** The design of a deterministic, code-based Commonsense Verifier using a hybrid of Numberbatch and E5 embeddings (Section 2.3) was a specific choice to provide an objective check on LLM-generated triples, independent of further LLM processing. This contrasts with approaches that might use another LLM call for verification. The rationale was that embedding spaces trained on broad commonsense (Numberbatch) and contextual semantics (E5) could offer a different, complementary mode of plausibility assessment. The two-pass verification (re-verifying induced triples) was designed to ensure that even inferred knowledge is scrutinized.
- **Triplet Matcher:** The use of SBERT embeddings for matching candidate triples against the existing Neo4j KG (including the base ontology and previously ingested facts) addresses the need for managing redundancy, a common issue in incrementally built KGs. This aligns with the broader goal of maintaining KG quality over time.

These mechanisms directly reflect the DSR principle of building artifacts that address known problems (in this case, LLM reliability and KG consistency).

**Semantic Consistency and Schema Adherence (Relates to RQ3)** The methodology for maintaining semantic consistency relies on several components:

- **Predicate Normalizer:** Using an LLM to map varied predicates to a predefined canonical list (Appendix E) is a direct attempt to enforce a controlled vocabulary for relationships, crucial for KG queryability and interoperability. This is a common challenge in KGC from diverse text sources.
- **KG Curator (Term Mapping  Labeling):** The Curator's use of SBERT for mapping extracted entity/predicate terms to existing terms in the Neo4j base ontology (Appendix C) before LLM labeling, and then using an LLM to assign node labels from a target list (Appendix D), are designed to align new knowledge with the predefined schema and existing vocabulary. The generation of Cypher queries ensures data is structured according to the Labeled Property Graph model (Section 2.2).

The definition of a target ontology (labels and predicates) and a base structural KG upfront is a methodological choice to guide the extraction and ensure the resulting KG is not an arbitrary collection of facts but a structured representation of the ADAS domain.

## 6.2 Discussion of Results

This section interprets the empirical findings presented in Chapter 5 for Experiment 1 (processing of `document1`, the ADAS glossary), relating them to the research questions and discussing their implications.

**Pipeline Functionality and Automated KG Generation (RQ1)** The successful end-to-end processing of `document1` (Table 1) demonstrates the operational feasibility of the MAKG pipeline for automating KGC. The system was able to ingest raw text, process it through multiple stages of extraction, normalization, validation, and curation, and ultimately generate 159 Cypher queries that successfully populated the Neo4j graph. The throughput statistics illustrate a significant filtering and refinement process: starting from 148 raw extracted triples, the pipeline yielded 159 curated triples after accounting for normalization effects, two verification passes, inductive reasoning, and internal de-duplication by the curator. The resulting KG contains 204 unique entities classified under 41 distinct labels, indicating the system's ability to not only extract facts but also to categorize the entities involved according to the predefined ADAS ontology. This provides an affirmative, practical answer to how such a pipeline can automate KG construction. However, the KG Curator's assignment of the ":Review" label to 74 out of 204 entities (36%) highlights a current limitation in achieving fully specific automated semantic typing. This suggests that while the pipeline automates the structural creation of the graph, further refinement of either the target label ontology or the curator's LLM prompting is needed for more granular entity classification, a common challenge in domain-specific KGC as also noted by studies focusing on LLM limitations in specialized contexts (Nouri et al., 2024).

**Effectiveness of Validation Mechanisms (RQ2)** The Commonsense Verifier was designed to improve the accuracy and reliability of the extracted knowledge. In Pass 1, it rejected 11 out of 141 (7.8%) normalized triples. Qualitative review of these rejections (Section 5.2.1) largely supported the verifier's decisions, filtering out semantically incongruent or less plausible statements. This indicates its utility in cleaning the initial LLM extractions. The Inductive Reasoner generated 35 new triples, and impressively, the Commonsense Verifier (Pass 2) accepted 34 of these (97.1%). While this high acceptance rate might suggest the induced triples are highly plausible, it also warrants careful consideration. As discussed in Section 5.2.1, many accepted induced triples like `(Scene, includes, artificial marker)` appeared to be reasonable extensions. However, without a ground truth for these inferred facts, their ultimate correctness and novelty remain subject to further domain expert validation. The verifier's role here is primarily as a plausibility filter for LLM-generated hypotheses, rather than a definitive truth assessor. The overall process, including this verification of induced triples, contributes to a more refined set of candidates (164 triples) being passed to the matcher compared to the initial 148 raw extractions, suggesting an improvement in the overall reliability of the knowledge before final KG integration.

**Maintaining Semantic Consistency and Structure (RQ3)** Addressing RQ3, which questions how semantic consistency and appropriate structure can be maintained in the automatically generated ADAS Knowledge Graph, the MAKG system employed several targeted mechanisms. The **Predicate Normalizer** demonstrated significant effectiveness in standardizing relational vocabulary, reducing the 71 distinct raw predicate types from `document1` to 61 canonical forms (Section 5.3.2). Qualitative analysis confirmed a high accuracy for most normalization decisions, such as mapping textual variations like "is equipped with" to the canonical 'hasPart', thereby promoting relational consistency. However, its reliance on a predefined canonical list (Appendix E) meant that novel or highly specific extracted predicates without clear mappings were passed through unchanged, highlighting a boundary for full automation with a fixed vocabulary.

The **Triplet Matcher** proved crucial for managing redundancy and ensuring structural integrity as the

KG evolves. Its ability to distinguish novel, instance-level facts from `document1` against the primarily schema-level base ontology in Run 1 (0 matches found) was a key finding. More significantly, in Run 2, when reprocessing the same document against the populated KG, the matcher identified approximately 51.9% (84 out of 162) of incoming verified triples as duplicates of information ingested in Run 1 (Section 5.3.1). This confirms its capacity to prevent the re-insertion of redundant facts. The remaining unmatched triples in Run 2 underscore the inherent stochasticity of the LLM-driven extraction process, even on identical input.

Finally, the KG Curator contributes by attempting to map terms to the existing ontology and by assigning entities to a controlled set of node labels (Appendix D). While the challenge of precise entity typing remains (evidenced by the ":Review" labels), these combined mechanisms for predicate standardization, duplicate triplet filtering, and structured labeling collectively ensure that the generated KG maintains a significantly higher degree of semantic consistency and organized structure compared to raw, unrefined LLM extractions.

# 7 Conclusion

This thesis addressed the critical challenge of managing complex engineering knowledge within the ADAS/AD domain by designing and evaluating a Multi-Agentic Knowledge Graph (MAKG) system. The research demonstrated the feasibility of an agent-inspired pipeline, leveraging LLM-driven components and deterministic modules, to automate the construction of a domain-specific Knowledge Graph from unstructured ADAS technical documentation. The system successfully processed a representative glossary document, transforming raw text into a populated graph, thereby affirming the potential of such automated approaches (RQ1). Key internal mechanisms, including a two-pass Commonsense Verifier and an Inductive Reasoner, proved effective in filtering implausible data and enriching the knowledge base with verified inferred facts, contributing positively to the accuracy and reliability of the extracted knowledge (RQ2). Furthermore, the MAKG system's components for Predicate Normalization and Triplet Matching demonstrated tangible success in enhancing semantic consistency and managing redundancy. The normalizer effectively standardized relational vocabulary by significantly reducing predicate diversity, while the matcher, particularly when re-processing the same document against an already populated KG, identified and filtered a substantial portion of duplicate information (approximately 51.9%). These mechanisms, alongside the KG Curator's structured labeling (though with a notable reliance on a fallback ":Review" label for a portion of entities), collectively work towards producing a more organized and semantically coherent Knowledge Graph (RQ3) than would be achievable through unrefined LLM extractions alone.

While the current evaluation highlights the MAKG system's promise, particularly in its structured approach to KGC and integrated validation, it also underscores areas for continued development, most notably in achieving more granular and accurate automated entity typing by the KG Curator. Nevertheless, this work contributes a validated methodology and a functional artifact that showcases how orchestrated LLM capabilities combined with deterministic checks can tackle complex knowledge engineering tasks in safety-critical domains, offering a pathway to more efficient and reliable management of ADAS engineering knowledge.

## 7.1 Future Work

Building on the insights gained, future research will focus on several key areas. Firstly, a comprehensive evaluation incorporating manually curated gold standards for diverse ADAS document types (including narrative specifications) is essential to rigorously quantify end-to-end extraction accuracy (Precision, Recall, F1-score). Secondly, enhancing the KG Curator's entity labeling capabilities, perhaps through more

sophisticated prompting, few-shot learning, or semi-supervised techniques to reduce reliance on fallback labels, is a primary goal. Further work will also explore the integration of more advanced validation mechanisms, potentially incorporating external domain ontologies or rule-based consistency checks, and refining the Inductive Reasoner for generating more diverse and high-utility inferences. Finally, investigating the practical application of the generated ADAS KGs for specific engineering use cases, such as requirements traceability, automated compliance checking, or engineering question-answering, will be crucial for demonstrating its real-world utility.

# References

Beltagy, I., Lo, K., & Cohan, A. (2019, November).
SciBERT: A pretrained language model for scientific text.
In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 3615–3620).
Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1371

Bengler, K., Dietmayer, K., Färber, B., Maurer, M., Stiller, C., & Winner, H. (2014).
Three decades of driver assistance systems: Review and future perspectives.
*IEEE Intelligent Transportation Systems Magazine*, *6*(4), 6–22.
https://doi.org/10.1109/MITS.2014.2336271

Boiko, D. A., MacKnight, R., & Gomes, G. (2023).
Emergent autonomous scientific research capabilities of large language models.
https://arxiv.org/abs/2304.05332

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., & Yakhnenko, O. (2013).
Translating embeddings for modeling multi-relational data.
In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Weinberger (Eds.),
*Advances in neural information processing systems* (Vol. 26). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020).
Language models are few-shot learners. https://arxiv.org/abs/2005.14165

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E., & Mitchell, T. (2010).
Toward an architecture for never-ending language learning. *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI) Conference*, *3*.

Chen, B., & Bertozzi, A. L. (2023).
Autokg: Efficient automated knowledge graph generation for language models.
https://arxiv.org/abs/2311.14740

d'Avila Garcez, A., & Lamb, L. C. (2020). Neurosymbolic ai: The 3rd wave.
https://arxiv.org/abs/2012.05876

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019).
Bert: Pre-training of deep bidirectional transformers for language understanding.
https://arxiv.org/abs/1810.04805

Dong, X. L., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., & Zhang, W. (2014). Knowledge vault: A web-scale approach to probabilistic knowledge fusion [Evgeniy GabrilovichWilko HornNi LaoKevin MurphyThomas StrohmannShaohua SunWei ZhangGeremy Heitz]. *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, 601–610.
http://www.cs.cmu.edu/~nlao/publication/2014.kdd.pdf

Dorri, A., Kanhere, S. S., & Jurdak, R. (2018). Multi-agent systems: A survey. *IEEE Access*, *6*, 28573–28593. https://doi.org/10.1109/ACCESS.2018.2831228

Gillick, D., Kulkarni, S., Lansing, L., Presta, A., Tomkins, A., Verma, B., & Wang, E. (2019).
Learning dense representations for entity retrieval.

*Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, 528–537.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., … Ma, Z. (2024). The llama 3 herd of models. https://arxiv.org/abs/2407.21783

Halilaj, L., Dindorkar, I., Lüttin, J., & Rothermel, S. (2021). A knowledge graph-based approach for situation comprehension in driving scenarios. In R. Verborgh, K. Hose, H. Paulheim, P.-A. Champin, M. Maleshkova, O. Corcho, P. Ristoski, & M. Alam (Eds.), *The semantic web* (pp. 699–716). Springer International Publishing.

Halilaj, L., Luettin, J., Henson, C., & Monka, S. (2022). Knowledge graphs for automated driving. *2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, 98–105. https://doi.org/10.1109/AIKE55402.2022.00023

Hevner, A., R, A., March, S., T, S., Park, Park, J., Ram, & Sudha. (2004). Design science in information systems research. *Management Information Systems Quarterly*, *28*, 75–.

Hou, Y., Yang, X., Jiang, S., Yang, Q., Zhao, Y., & Ji, Y. (2020). Knowledge graph based requirements understanding. *2020 IEEE 28th International Requirements Engineering Conference (RE)*, 397–402.

Huang, L., Liang, H., Yu, B., Li, B., & Zhu, H. (2019). Ontology-based driving scene modeling, situation assessment and decision making for autonomous vehicles. *2019 4th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, 57–62. https://doi.org/10.1109/ACIRS.2019.8935984

Huguet Cabot, P.-L., & Navigli, R. (2021, November). REBEL: Relation extraction by end-to-end language generation. In M.-F. Moens, X. Huang, L. Specia, & S. W.-t. Yih (Eds.), *Findings of the association for computational linguistics: Emnlp 2021* (pp. 2370–2381). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.findings-emnlp.204

Huo, X., Yin, Y., Jiao, L., & Zhang, Y. (2024). A data-driven and knowledge graph-based analysis of the risk hazard coupling mechanism in subway construction accidents. *Reliability Engineering and System Safety*, *250*, 110254. https://doi.org/https://doi.org/10.1016/j.ress.2024.110254

ISO. (2018). *ISO 26262:2018 Road vehicles – Functional safety* (Standard). International Organization for Standardization. Geneva, CH.

ISO. (2022). *ISO 21448:2022 Road vehicles – Safety of the intended functionality* (Standard). International Organization for Standardization. Geneva, CH.

Kolitsas, N., Ganea, O.-E., & Hofmann, T. (2018). End-to-end neural entity linking. *Proceedings of the 22nd Conference on Computational Natural Language Learning*, 519–529.

Koopman, P., & Wagner, M. (2017). Challenges in autonomous vehicle testing and validation. *SAE International Journal of Transportation Safety*, *4*(1), 15–24. https://doi.org/10.4271/2016-01-0128

Krishna, A., surya ardham, Malhotra, C., & Shinde, A. P. (2024). A collaborative multi-agent LLM approach for knowledge graph curation and query from multimodal data sources. https://openreview.net/forum?id=qXwVXj03nO

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2020). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. https://arxiv.org/abs/1910.13461

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Kiela, D., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, *33*, 9459–9474.

Lu, Y., Liu, Q., Dai, D., Xiao, X., Lin, H., Han, X., Sun, L., & Wu, H. (2022, May). Unified structure generation for universal information extraction. In S. Muresan, P. Nakov, & A. Villavicencio (Eds.), *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 5755–5772). Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.acl-long.395

Luan, Y., He, L., Ostendorf, M., & Hajishirzi, H. (2018). Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In E. Riloff, D. Chiang, J. Hockenmaier, & J. Tsujii (Eds.), *Proceedings of the 2018 conference on empirical methods in natural language processing* (pp. 3219–3232). Association for Computational Linguistics. https://doi.org/10.18653/v1/D18-1360

Maier, M. W. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, *1*(4), 267–284. https://doi.org/https://doi.org/10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. https://arxiv.org/abs/1301.3781

Mlodzian, L., Sun, Z., Berkemeyer, H., Monka, S., Wang, Z., Dietze, S., Halilaj, L., & Luettin, J. (2023). Nuscenes knowledge graph - a comprehensive semantic representation of traffic scenes for trajectory prediction. *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 42–52. https://doi.org/10.1109/iccvw60793.2023.00011

Nouri, A., Cabrero-Daniel, B., Törner, F., Sivencrona, H., & Berger, C. (2024). Engineering safety requirements for autonomous driving with large language models [Accepted to 32nd IEEE International Requirements Engineering Conference (RE'24)]. https://arxiv.org/abs/2403.16289

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., … Zoph, B. (2024). Gpt-4 technical report. https://arxiv.org/abs/2303.08774

Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., & Wu, X. (2024). Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, *36*(7), 3580–3599. https://doi.org/10.1109/tkde.2024.3352100

Paolini, G., Athiwaratkun, B., Krone, J., Ma, J., Achille, A., ANUBHAI, R., dos Santos, C. N., Xiang, B., & Soatto, S. (2021). Structured prediction as translation between augmented natural languages. *International Conference on Learning Representations*. https://openreview.net/forum?id=US-TP-xnXI

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. https://arxiv.org/abs/2304.03442

Peffers, K., Tuunanen, T., Rothenberger, M., & Chatterjee, S. (2007).
A design science research methodology for information systems research.
*Journal of Management Information Systems*, *24*, 45–77.

Pennington, J., Socher, R., & Manning, C. (2014, October).
GloVe: Global vectors for word representation.
In A. Moschitti, B. Pang, & W. Daelemans (Eds.), *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543).
Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1162

Petroni, F., Rocktäschel, T., Lewis, P., Bakhtin, A., Wu, Y., Miller, A. H., & Riedel, S. (2019).
Language models as knowledge bases? *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2463–2473.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019).
Language models are unsupervised multitask learners.
https://api.semanticscholar.org/CorpusID:160025533

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2023). Exploring the limits of transfer learning with a unified text-to-text transformer.
https://arxiv.org/abs/1910.10683

Reimers, N., & Gurevych, I. (2019, November).
Sentence-BERT: Sentence embeddings using Siamese BERT-networks.
In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 3982–3992).
Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1410

Robinson, I., Webber, J., & Eifrem, E. (2015).
*Graph databases: New opportunities for connected data* (Second). O'Reilly Media, Inc.

SAE. (2021, April). *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*.

Speer, R., Chin, J., & Havasi, C. (2017).
Conceptnet 5.5: An open multilingual graph of general knowledge.
https://arxiv.org/abs/1612.03975

Sun, Z., Deng, Z.-H., Nie, J.-Y., & Tang, J. (2019).
Rotate: Knowledge graph embedding by relational rotation in complex space.
https://arxiv.org/abs/1902.10197

Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., & Bouchard, G. (2016).
Complex embeddings for simple link prediction. https://arxiv.org/abs/1606.06357

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). Attention is all you need. https://arxiv.org/abs/1706.03762

vom Brocke, J., Hevner, A., & Maedche, A. (2020). *Design science research. cases*. Springer.
https://link.springer.com/book/10.1007/978-3-030-46781-4

W3C. (2014, February). RDF 1.1 Concepts and Abstract Syntax [Editors: Richard Cyganiak, David Wood, Markus Lanthaler.]. https://www.w3.org/TR/rdf11-concepts/

Wadhwa, S., Amir, S., & Wallace, B. (2023, July).
Revisiting relation extraction in the era of large language models.
In A. Rogers, J. Boyd-Graber, & N. Okazaki (Eds.), *Proceedings of the 61st annual meeting of*

*the association for computational linguistics (volume 1: Long papers)* (pp. 15566–15589). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.acl-long.868

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Wang, B., & Qiu, M. (2023). A survey on large language model based autonomous agents.

Wang, L., Yang, N., Huang, X., Jiao, B., Yang, L., Jiang, D., Majumder, R., & Wei, F. (2024). Text embeddings by weakly-supervised contrastive pre-training. https://arxiv.org/abs/2212.03533

Waseem, U., & Idris, N. (2020). Supporting requirements traceability by constructing ontologies from requirements specifications. *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, 951–956.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2023). Chain-of-thought prompting elicits reasoning in large language models. https://arxiv.org/abs/2201.11903

Wooldridge, M. (2009). *An introduction to multiagent systems* (2nd). John Wiley & Sons.

Wu, L., Petroni, F., Josifoski, M., Riedel, S., & Zettlemoyer, L. (2020, November). Scalable zero-shot entity linking with dense entity retrieval. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 conference on empirical methods in natural language processing (emnlp)* (pp. 6397–6407). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.519

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). Autogen: Enabling next-gen llm applications via multi-agent conversation. https://arxiv.org/abs/2308.08155

Yang, B., Yih, W.-t., He, X., Gao, J., & Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. https://arxiv.org/abs/1412.6575

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). React: Synergizing reasoning and acting in language models. https://arxiv.org/abs/2210.03629

Ye, H., Zhang, N., Chen, H., & Chen, H. (2023). Generative knowledge graph construction: A review. https://arxiv.org/abs/2210.12714

Zhao, M., Bellet, T., Richard, B., Giralt, A., Beurier, G., & Wang, X. (2024). Effects of non-driving related postures on takeover performance during conditionally automated driving. *Accident Analysis & Prevention*, *208*, 107793. https://doi.org/https://doi.org/10.1016/j.aap.2024.107793

# Appendices

This appendix contains supplementary materials referenced in the main body of the thesis, including detailed prompt templates and examples of key configuration files used by the MAKG system.

## Appendix A Prompt Templates

### A.1 Triplet Extractor Prompt (`prompts/triplet_extractor.txt`)

```
You are an advanced knowledge extraction agent. Your task is to extract
**meaningful subject-predicate-object relationships** from the following technical
document.

These relationships should be represented as **triplets** where:
- **subject** = the "head" or starting entity
- **predicate** = the "relation" that connects them
- **object** = the "tail" or target entity

Text:
{{text}}

Your output must be in **JSON** format with this structure:
‘‘‘json
{
  "triplets": [
    {
      "subject": "LaneKeepingAssist",
      "predicate": "uses",
      "object": "SteeringActuator"
    }
  ]
}

+ Use entity names that are reusable and consistent across triplets.

Rules:


- Use concise, lowercase, verb-based predicates, using underscores for multi-word
verbs (e.g., is_defined_as, requires_input, has_part). Avoid long phrases as
predicates.
- When extracting the primary definition of a TERM, try to use the predicate
'is_defined_as'. Extract other facts within the definition as separate triplets.
- Do not include explanations, markdown, or intro text.
- Extract all relevant triplets found in the provided text.
- Only return the final JSON. No comments or additional explanations.
```

## A.2 Predicate Normalizer Prompt (prompts/predicate_normalizer.txt)

```
You are an expert Knowledge Graph Ontology Specialist. Your task is to
**normalize the raw relationship phrase (predicate)** within a given
Subject-Predicate-Object triplet, selecting the most appropriate term
from a predefined canonical vocabulary.

### **Input Format**
You will receive a single JSON object representing the triplets extracted
previously:
‘‘‘json
{
  "triplets": [
    {
      "subject": "ExtractedSubject",
      "predicate": "RawPredicate",
      "object": "ExtractedObject"
    }
  ]
}

Input triplets:
{{ triplets }}

### **Your Task**
1. **Normalize Predicates**:
   - Replace the ‘predicate‘ field with the most appropriate term from the
   **Canonical Predicates** list below.
   - Do **not** modify the ‘subject‘ or ‘object‘ fields. They must remain
   exactly as provided.

2. **Output Requirements**:
   - Return the modified triplets in the same JSON format.
   - Ensure the number of triplets remains the same as the input.
   - Do not include explanations, comments, or markdown outside the JSON object.

### **Canonical Predicates & Definitions**
*   **‘isA‘**: Represents an instance-of relationship
(e.g., "Fido ‘isA‘ Dog").
*   **‘subClassOf‘**: Represents a type-of relationship
(e.g., "Dog ‘subClassOf‘ Mammal").
*   **‘hasPart‘**: Represents physical or logical composition
(e.g., "Car ‘hasPart‘ Wheel", "System ‘hasPart‘ Module").
*   **‘partOf‘**: The inverse of ‘hasPart‘ (e.g., "Wheel ‘partOf‘ Car").
*   **‘hasProperty‘**: Links an entity to its characteristic or attribute name
(e.g., "Sensor ‘hasProperty‘ FieldOfView").
*   **‘hasValue‘**: Links an entity or property to its specific value
(e.g., "FieldOfView ‘hasValue‘ ’90 degrees’").
*   **‘uses‘**: Indicates that the subject utilizes, employs, or consumes the object
(e.g., "System ‘uses‘ SensorData").
*   **‘produces‘**: Indicates that the subject generates or creates the object
(e.g., "Sensor ‘produces‘ Signal").
*   **‘controls‘**: Indicates that the subject directs or regulates the object
```

```
(e.g., "ECU ‘controls‘ Actuator").
*   **‘triggers‘**: Indicates the subject initiates the object
(e.g., "Fault ‘triggers‘ FailSafe").
*   **‘enables‘**: Indicates the subject makes the object possible or activates
it (e.g., "PowerSupply ‘enables‘ Sensor").
*   **‘requires‘**: Indicates the subject needs the object to function or exist
(e.g., "ADS ‘requires‘ SensorInput").
*   **‘implements‘**: Indicates the subject realizes or executes the object
(e.g., "Software ‘implements‘ Algorithm").
*   **‘defines‘**: Indicates the subject provides the definition or specification
for the object (e.g., "Standard ‘defines‘ Term").
*   **‘specifies‘**: Similar to defines, often used for requirements or parameters
(e.g., "UseCase ‘specifies‘ ODD").
*   **‘measures‘**: Indicates the subject quantifies the object
(e.g., "Sensor ‘measures‘ Temperature").
*   **‘represents‘**: Indicates the subject stands for or symbolizes the object
(e.g., "Model ‘represents‘ System").
*   **‘includes‘**: Represents membership in a set or category
(e.g., "RoadUser ‘includes‘ Pedestrian").
*   **‘locatedIn‘**: Specifies physical or logical location
(e.g., "Sensor ‘locatedIn‘ Bumper").
*   **‘causes‘**: Indicates the subject leads to the object
(e.g., "Fault ‘causes‘ Failure").
*   **‘mitigates‘**: Indicates the subject reduces the negative impact of the
object (e.g., "SafetyMeasure ‘mitigates‘ Hazard").
*   **‘references‘**: Indicates the subject refers to or cites the object
(e.g., "Document ‘references‘ Standard").
*   **‘equivalentTo‘**: Indicates two entities represent the same concept
(e.g., "ADS ‘equivalentTo‘ AutomatedDrivingSystem").
*   **‘relatedTo‘**: A generic relationship - **use only if no other predicate fits
well**.

### **Output Format**
Return **only** the modified JSON triplet structure. Do not include explanations,
comments, or markdown outside the JSON object.
```

### A.3 Inductive Reasoner Prompt (`prompts/inductive_reasoner.txt`)

```
You are an expert in knowledge graph design and inductive reasoning.
Given a group of semantically related knowledge graph triplets, your task is to
infer new, plausible triplets that follow the same domain logic and structure.

Use the following rules:
- Maintain subject-predicate-object formatting.
- Only include triplets that are logically consistent and meaningful based on
the cluster’s patterns.
- Avoid duplicating input triplets or making trivial rewordings (e.g., just
changing capitalization).
- - Think creatively but reasonably. Look for patterns, potential hierarchies,
cause-effect, part-whole, or other semantic relationships implied by the cluster.
- Do not infer relationships that contradict the given triplets or general
common sense for this domain.
```

```
Here is a cluster of triplets:

{{ triplets }}

Now, generate 5-7 new triplets that extend this knowledge graph in a useful
and intelligent way. Prioritize the quality and novelty of the inferred triplets
over quantity.

Format your output like this:
‘‘‘json
{
  "triplets": [
    {"subject": "...", "predicate": "...", "object": "..."},
    {"subject": "...", "predicate": "...", "object": "..."}
  ]
}

Return **only** the final JSON. Do not include explanations, comments, or
markdown outside the JSON object.
```

### A.4 KG Curator (Entity Labeling) Prompt (prompts/kg_curator.txt)

```
You are a precise Knowledge Graph Ontology Specialist. Your task is to accurately
classify entity names into predefined Neo4j labels. You MUST choose ONLY from the
provided list. Accuracy is paramount.

RULES:
1. Analyze EACH entity name provided in the input list.
2. Compare the entity name against the descriptions implicit in the VALID TARGET
LABELS list.
3. Assign the SINGLE MOST specific and appropriate label from the VALID TARGET
LABELS list.
4. Context: Assume entities relate to automated driving, vehicle systems,
safety, software, standards, sensors, actors, processes, or general technical
concepts.
5. **CRITICAL FALLBACK:** If, and ONLY if, you cannot confidently determine a
specific label from the list for an entity name (due to ambiguity, genericness
like 'item'), you MUST assign the specific label: 'Review'. Do NOT invent
labels or use labels not present in the VALID TARGET LABELS list.
6. Output ONLY a single, valid JSON dictionary mapping EACH input entity name to
its assigned label string (e.g., "Sensor", "Standard", "Review"). Do NOT include
the leading colon ':'.
7. The output JSON dictionary MUST contain exactly one entry for every entity name
provided in the input list.

VALID TARGET LABELS (Use ONLY these strings):
{{ target_labels_list_str }}

--- EXAMPLES ---

### Example 1:
INPUT ENTITY NAMES:
```

```
["LiDAR", "Automated Driving System", "Fault", "Driver", "ISO 26262", "Thingamajig"]

OUTPUT JSON DICTIONARY:
{% raw %}{"LiDAR": "LiDAR", "Automated Driving System": "System", "Fault": "Fault",
"Driver": "Driver", "ISO 26262": "Standard", "Thingamajig": "Review"}{% endraw %}

### Example 2:
INPUT ENTITY NAMES:
["SensorData", "ECU", "Vehicle", "Pedestrian", "item or object", "complex traffic
conditions"]

OUTPUT JSON DICTIONARY:
{% raw %}{"SensorData": "Information", "ECU": "Controller", "Vehicle": "Vehicle",
"Pedestrian": "Pedestrian", "item or object": "Review", "complex traffic conditions":
"Environment"}{% endraw %}

### End Examples

--- TASK ---

INPUT ENTITY NAMES:
{{ entity_list_json }}

### **Output Format**
Return **only** the final JSON. Do not include explanations, comments, or
markdown outside the JSON object.
```

# Appendix B Key Configuration File Examples

This section provides illustrative examples of the JSON configuration files used to define the parameters for each component of the MAKG system. For brevity, only key parameters or representative files are shown. Sensitive information such as API keys or full database URIs has been redacted or replaced with placeholders.

## B.1 Triplet Extractor Configuration (`configs/triplet_extractor.json`)

```
{
  "name": "triplet_extractor",
  "role": "Triplet Extractor",
  "llm": "llama3:instruct",
  "prompt_path": "prompts/triplet_extractor.txt",
  "tool": "read_pdf_chunked",
  "tool_params": {
    "path": "files/safety-first-for-automated-driving.pdf",
    "chunk_size": 1200,
    "chunk_overlap": 300
  },
  "output_path": "outputs/triplet_extractor/all_inferred_relations.json",
  "output_dir": "outputs/triplet_extractor",
```

```
  "log_dir": "outputs/triplet_extractor/logs"
}
```

### B.2 Predicate Normalizer Configuration (`configs/predicate_normalizer.json`)

```
{
    "name": "predicate_normalizer",
    "role": "Predicate Normalizer",
    "llm": "llama3:instruct",
    "prompt_path": "prompts/predicate_normalizer.txt",
    "tool": "repair_json",
    "tool_params": {
      "input_path": "outputs/triplet_extractor/all_triplets.json"
    },
    "output_path": "outputs/predicate_normalizer/normalized_triplets.json",
    "log_dir": "outputs/predicate_normalizer/logs",
    "output_dir": "outputs/predicate_normalizer",
    "batch_size": 10
  }
```

### B.3 Commonsense Verifier Configuration (`configs/commonsense_verifier.json`)

```
{
    "tool_params": {
        "input_path": "outputs/predicate_normalizer/predicate_normalizer.json",
        "emb_path": "data/numberbatch-en-19.08.txt",
        "output_dir": "outputs/commonsense_verifier",
        "threshold_nb": 0.2,
        "threshold_e5": 0.75
    }
}
```

### B.4 Inductive Reasoner Configuration (`configs/inductive_reasoner.json`)

```
{
    "name": "inductive_reasoner",
    "role": "Inductive Reasoning Agent",
    "llm": "llama3:instruct",
    "prompt_path": "prompts/inductive_reasoner.txt",
    "tool": "repair_json",
    "tool_params": {
      "input_path": "outputs/commonsense_verifier/verified_triplets.json"
    },
    "embedding_model": "all-MiniLM-L6-v2",
    "output_dir": "outputs/inductive_reasoner",
    "log_dir": "outputs/inductive_reasoner/logs",
    "max_clusters": 7
```

```
  }
```

### B.5 Triplet Matcher Configuration (configs/triplet_matcher.json)

```
{
    "neo4j": {
        "uri": [],
        "user": [],
        "password": [],
        "output_path": "outputs/neo4j/triplets_export.csv",
        "entities_output_path": "outputs/neo4j/entities_export.csv",
        "relations_output_path": "outputs/neo4j/relations_export.csv"
    },
    "embeddings": {
        "triplet_csv_path": "outputs/neo4j/triplets_export.csv",
        "triplet_emb_output_path":
        "outputs/triplet_matcher/embeddings/neo4j_triplet_embeddings.npy",
        "triplet_text_output_path":
        "outputs/triplet_matcher/embeddings/neo4j_triplet_text.json",

        "entities_csv_path": "outputs/neo4j/entities_export.csv",
        "entity_emb_output_path":
        "outputs/triplet_matcher/embeddings/neo4j_entity_embeddings.npy",
        "entity_text_output_path":
        "outputs/triplet_matcher/embeddings/neo4j_entity_text.json",

        "relations_csv_path": "outputs/neo4j/relations_export.csv",
        "relation_emb_output_path":
        "outputs/triplet_matcher/embeddings/neo4j_relation_embeddings.npy",
        "relation_text_output_path":
        "outputs/triplet_matcher/embeddings/neo4j_relation_text.json",

        "model_name": "all-MiniLM-L6-v2"
    },
    "matcher": {
        "triplet_file": "outputs/commonsense_verifier/final_verified_triplets.json",
        "embeddings_dir": "outputs/triplet_matcher/embeddings/",
        "threshold": 0.8,
        "batch_size": 10,
        "output_dir": "outputs/triplet_matcher"
    }
}
```

### B.6 KG Curator Configuration (configs/kg_curator.json)

```
{
    "name": "kg_curator",
    "role": "Knowledge Graph Curator",
    "llm": "llama3:instruct",
    "embedding_model": "all-MiniLM-L6-v2",
```

```
    "labeler_prompt_path": "prompts/kg_curator.txt",
    "target_kg_labels": [
        "System", "Sensor", "Actuator", "Controller", "Module",
        "Software", "SoftwareArchitecture", "Communication", "Signal", "Property",
        "VehicleProperty", "User", "Driver", "Passenger", "Actor",
        "TrafficParticipant", "Pedestrian", "Environment",
        "Weather", "Behavior", "Maneuver", "Intention", "Scene", "ScenarioElement",
        "Frame", "Hazard", "Risk", "Fault", "Failure", "Goal", "SafetyGoal",
        "FunctionalGoal", "Mitigation", "SafetyMechanism", "SafetyArgument",
        "FaultResponse", "Test", "TestProcedure", "TestCase", "Standard", "Regulation",
        "Requirement", "Information", "Data", "OperationalDesignDomain",
        "Condition", "Process", "Validation", "Verification", "Capability", "Vehicle", "Feature",
    ],
    "default_label": "Review",

    "neo4j_entity_embeddings_path":
    "outputs/triplet_matcher/embeddings/neo4j_entity_embeddings.npy",
    "neo4j_entity_text_path":
    "outputs/triplet_matcher/embeddings/neo4j_entity_text.json",
    "neo4j_relation_embeddings_path":
    "outputs/triplet_matcher/embeddings/neo4j_relation_embeddings.npy",
    "neo4j_relation_text_path":
    "outputs/triplet_matcher/embeddings/neo4j_relation_text.json",

    "output_dir": "outputs/kg_curator",
    "llm_log_dir": "outputs/kg_curator/llm_logs",
    "matcher_log_dir": "outputs/kg_curator/matcher_logs",
    "similarity_threshold": 0.75,
    "batch_size": 10
}
```

# Appendix C Base Ontological Knowledge Graph Triples

Prior to processing any ADAS documents for evaluation, the Neo4j database is initialized with the following set of triples. These represent the core ontological structure, defining types, hierarchies, and high-level domain concepts. The format shown is (Source Name `[:SourceLabel]`, RelationshipType, Target Name `[:TargetLabel]`).

```
(ASPICE [:Standard], isA, Standard [:Standard])
(Actor [:Actor], belongsToDomain, ScenarioModel [:Domain])
(Actuator [:Actuator], belongsToDomain, EgoVehicleModel [:Domain])
(Actuator [:Actuator], isA, Component [:Component])
(Behavior [:Behavior], belongsToDomain, ScenarioModel [:Domain])
(BrakeSignal [:BrakeSignal], belongsToDomain, EgoVehicleModel [:Domain])
(BrakeSignal [:BrakeSignal], isA, Signal [:Signal])
(Communication [:Communication], belongsToDomain, EgoVehicleModel [:Domain])
(ComplianceRequirement [:ComplianceRequirement], basedOn, Regulation [:Regulation])
(ComplianceRequirement [:ComplianceRequirement], belongsToDomain, ComplianceModel [:Domain])
(ComplianceRequirement [:ComplianceRequirement], isA, Regulation [:Regulation])
```

```
(Component [:Component], belongsToDomain, EgoVehicleModel [:Domain])
(Controller [:Controller], belongsToDomain, EgoVehicleModel [:Domain])
(Controller [:Controller], consumes, Signal [:Signal])
(Controller [:Controller], isA, Component [:Component])
(ControllerSoftware [:ControllerSoftware], belongsToDomain, EgoVehicleModel [:Domain])
(ControllerSoftware [:ControllerSoftware], hasInput, BrakeSignal [:BrakeSignal])
(ControllerSoftware [:ControllerSoftware], hasOutput, Actuator [:Actuator])
(ControllerSoftware [:ControllerSoftware], isA, Software [:Software])
(DiagnosticsSoftware [:DiagnosticsSoftware], belongsToDomain, EgoVehicleModel [:Domain])
(DiagnosticsSoftware [:DiagnosticsSoftware], isA, Software [:Software])
(Driver [:Driver], belongsToDomain, EgoVehicleModel [:Domain])
(Driver [:Driver], isA, User [:User])
(EgoVehicle [:EgoVehicle], belongsToDomain, ScenarioModel [:Domain])
(EgoVehicle [:EgoVehicle], isA, TrafficParticipant [:TrafficParticipant])
(Environment [:Environment], belongsToDomain, ScenarioModel [:Domain])
(Fault [:Fault], belongsToDomain, SafetyModel [:Domain])
(Fault [:Fault], isA, Hazard [:Hazard])
(FaultResponse [:FaultResponse], belongsToDomain, SafetyModel [:Domain])
(FaultResponse [:FaultResponse], isA, Mitigation [:Mitigation])
(Frame [:Frame], belongsToDomain, ScenarioModel [:Domain])
(Frame [:Frame], isA, Scene [:Scene])
(FunctionalGoal [:FunctionalGoal], belongsToDomain, SafetyModel [:Domain])
(FunctionalGoal [:FunctionalGoal], isA, Goal [:Goal])
(Goal [:Goal], belongsToDomain, SafetyModel [:Domain])
(Hazard [:Hazard], belongsToDomain, SafetyModel [:Domain])
(Hazard [:Hazard], causedBy, Fault [:Fault])
(IMU [:IMU], belongsToDomain, EgoVehicleModel [:Domain])
(IMU [:IMU], isA, Sensor [:Sensor])
(ISO26262 [:Standard], isA, Standard [:Standard])
(Intention [:Intention], belongsToDomain, ScenarioModel [:Domain])
(Intention [:Intention], isA, Behavior [:Behavior])
(Intention [:Intention], precedes, Maneuver [:Maneuver])
(LiDAR [:LiDAR], belongsToDomain, EgoVehicleModel [:Domain])
(LiDAR [:LiDAR], isA, Sensor [:Sensor])
(Maneuver [:Maneuver], belongsToDomain, ScenarioModel [:Domain])
(Maneuver [:Maneuver], isA, Behavior [:Behavior])
(Maneuver [:Maneuver], performedBy, TrafficParticipant [:TrafficParticipant])
(Mitigation [:Mitigation], belongsToDomain, SafetyModel [:Domain])
(Module [:Module], belongsToDomain, EgoVehicleModel [:Domain])
(Module [:Module], isA, Component [:Component])
(OntologyRoot [:OntologyRoot], ADASModel, hasSubModel, EgoVehicleModel [:Domain])
(OntologyRoot [:OntologyRoot], ADASModel, hasSubModel, ScenarioModel [:Domain])
(OntologyRoot [:OntologyRoot], ADASModel, hasSubModel, SafetyModel [:Domain])
(OntologyRoot [:OntologyRoot], ADASModel, hasSubModel, ComplianceModel [:Domain])
(Passenger [:Passenger], belongsToDomain, EgoVehicleModel [:Domain])
(Passenger [:Passenger], isA, User [:User])
(Pedestrian [:Pedestrian], belongsToDomain, ScenarioModel [:Domain])
(Pedestrian [:Pedestrian], isA, TrafficParticipant [:TrafficParticipant])
(Property [:Property], belongsToDomain, EgoVehicleModel [:Domain])
(Radar [:Radar], belongsToDomain, EgoVehicleModel [:Domain])
(Radar [:Radar], isA, Sensor [:Sensor])
(Regulation [:Regulation], belongsToDomain, ComplianceModel [:Domain])
(Risk [:Risk], belongsToDomain, SafetyModel [:Domain])
```

```
(Risk [:Risk], isA, Hazard [:Hazard])
(Risk [:Risk], quantifies, Hazard [:Hazard])
(RoadEnvironment [:RoadEnvironment], belongsToDomain, ScenarioModel [:Domain])
(RoadEnvironment [:RoadEnvironment], isA, Environment [:Environment])
(SafetyArgument [:SafetyArgument], belongsToDomain, SafetyModel [:Domain])
(SafetyArgument [:SafetyArgument], isA, Mitigation [:Mitigation])
(SafetyGoal [:SafetyGoal], addresses, Hazard [:Hazard])
(SafetyGoal [:SafetyGoal], belongsToDomain, SafetyModel [:Domain])
(SafetyGoal [:SafetyGoal], isA, Goal [:Goal])
(SafetyMechanism [:SafetyMechanism], belongsToDomain, SafetyModel [:Domain])
(SafetyMechanism [:SafetyMechanism], isA, Mitigation [:Mitigation])
(SafetyMechanism [:SafetyMechanism], mitigates, Risk [:Risk])
(ScenarioElement [:ScenarioElement], belongsToDomain, ScenarioModel [:Domain])
(ScenarioElement [:ScenarioElement], involves, Actor [:Actor])
(ScenarioElement [:ScenarioElement], isA, Scene [:Scene])
(Scene [:Scene], belongsToDomain, ScenarioModel [:Domain])
(Scene [:Scene], contains, ScenarioElement [:ScenarioElement])
(Sensor [:Sensor], belongsToDomain, EgoVehicleModel [:Domain])
(Sensor [:Sensor], generates, Signal [:Signal])
(Sensor [:Sensor], isA, Component [:Component])
(Signal [:Signal], affects, VehicleProperty [:VehicleProperty])
(Signal [:Signal], belongsToDomain, EgoVehicleModel [:Domain])
(Signal [:Signal], isA, Communication [:Communication])
(Software [:Software], belongsToDomain, EgoVehicleModel [:Domain])
(SoftwareArchitecture [:SoftwareArchitecture], belongsToDomain, EgoVehicleModel [:Domain])
(SoftwareArchitecture [:SoftwareArchitecture], isA, Software [:Software])
(Speed [:Speed], belongsToDomain, EgoVehicleModel [:Domain])
(Speed [:Speed], isA, VehicleProperty [:VehicleProperty])
(SpeedSignal [:SpeedSignal], belongsToDomain, EgoVehicleModel [:Domain])
(SpeedSignal [:SpeedSignal], isA, Signal [:Signal])
(Standard [:Standard], belongsToDomain, ComplianceModel [:Domain])
(Test [:Test], belongsToDomain, ComplianceModel [:Domain])
(TestCase [:TestCase], belongsToDomain, ComplianceModel [:Domain])
(TestCase [:TestCase], isA, Test [:Test])
(TestCase [:TestCase], verifies, SafetyGoal [:SafetyGoal])
(TestProcedure [:TestProcedure], belongsToDomain, ComplianceModel [:Domain])
(TestProcedure [:TestProcedure], executes, TestCase [:TestCase])
(TestProcedure [:TestProcedure], isA, Test [:Test])
(Torque [:Torque], belongsToDomain, EgoVehicleModel [:Domain])
(Torque [:Torque], isA, VehicleProperty [:VehicleProperty])
(TrafficParticipant [:TrafficParticipant], belongsToDomain, ScenarioModel [:Domain])
(TrafficParticipant [:TrafficParticipant], isA, Actor [:Actor])
(User [:User], belongsToDomain, EgoVehicleModel [:Domain])
(VehicleProperty [:VehicleProperty], belongsToDomain, EgoVehicleModel [:Domain])
(VehicleProperty [:VehicleProperty], isA, Property [:Property])
(Weather [:Weather], belongsToDomain, ScenarioModel [:Domain])
(Weather [:Weather], isA, Environment [:Environment])
```

# Appendix D Target Entity Labels (Neo4j Node Labels)

- System
- Sensor
- Actuator
- Controller
- Module
- Software
- SoftwareArchitecture
- Communication
- Signal
- Property
- VehicleProperty
- User
- Driver
- Passenger
- Actor
- TrafficParticipant
- Pedestrian
- Environment
- Weather
- Behavior
- Maneuver
- Intention
- Scene
- ScenarioElement
- Frame
- Hazard
- Risk
- Fault
- Failure
- Goal
- SafetyGoal

- FunctionalGoal
- Mitigation
- SafetyMechanism
- SafetyArgument
- FaultResponse
- Test
- TestProcedure
- TestCase
- Standard
- Regulation
- Requirement
- Information
- Data
- OperationalDesignDomain
- Condition
- Process
- Validation
- Verification
- Capability
- Vehicle
- Feature
- Hardware
- Review (fallback)

# Appendix E Canonical Predicate Vocabulary

- `isA`
- `subClassOf`
- `hasPart`
- `partOf`
- `hasProperty`
- `hasValue`
- `uses`

- produces
- controls
- triggers
- enables
- requires
- implements
- defines
- specifies
- measures
- represents
- includes
- locatedIn
- causes
- mitigates
- references
- equivalentTo
- relatedTo