Bakkalaureatsarbeit

Automatisierte Wissensgraph-Erstellung per Multi-Objekterkennung für einen KI-Wartungsassistenten

bearbeitet von

Robin Morgenstern

geboren am 27.01.2004 in Chemnitz

Technische Universität Dresden

Fakultät Informatik
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie

Betreuer: Dr.-Ing. Karsten Wendt
Hochschullehrer: Prof. Dr. rer. nat. habil. Uwe Aßmann

Eingereicht am 23. August 2025

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung

Die zunehmende Digitalisierung in der Industrie führt zu einem wachsenden Bedarf an Systemen, die Informationen verschiedenster Quellen effizient verarbeiten. Besonders im Bereich der Wartung von Industriemaschinen müssen große Mengen an textuellen und visuellen Daten für Wartungsaufgaben nutzbar gemacht werden [1]. Moderne KI-Systeme wie Large Language Models (LLMs) bieten hierbei großes Potential, da sie es ermöglichen, umfangreiches Wissen aus verschiedenen Quellen und für konkrete Anforderungen praktisch nutzbar bereitzustellen [2, 1]. In den letzten Jahren haben sich Wissensgraphen erfolgreich als Wissensspeicher etabliert, um Information strukturiert zu speichern und um LLMs effizient mit diesem Wissen anzureichern [3, 4, 5, 6].

Die Erstellung dieser Wissensgraphen erfordert bislang überwiegend manuelle Abläufe, was diese Erstellung sowohl zeitintensiv, als auch fehleranfällig gestaltet [7, 8]. Während es bereits zahlreiche Verfahren zur automatischen Wissensgraph-Erstellung aus Textdaten gibt, ist die automatisierte Erstellung aus Bilddaten nur kaum untersucht [3, 8]. Insbesondere fehlt ein Ansatz, um Objekterkennungsdaten in formale, von LLM effizient verarbeitbare Wissensstrukturen zu überführen. Eine Automatisierung dieses Prozesses würde die leistungsfähige, fehlerfreie Erstellung von (multimodalen) Wissensgraphen in Echtzeitsystemen ermöglichen. Diese Arbeit greift damit eine zentrale Lücke zwischen Objekterkennung aus Bilddateien und der Strukturierung dieser Informationen für die effiziente Weiterverarbeitung durch LLMs auf.

1.2 Zielsetzung der Arbeit und Forschungsfragen

Aus den beschriebenen Problemen ergibt sich die Notwendigkeit, Verfahren zur automatischen Wissensgraph-Erstellung zu erforschen. Diese Arbeit verfolgt das Ziel, einen solchen Prozess zur automatischen Erstellung von OWL-konformen Wissensgraphen aus annotierten Bilddateien, welche aus der Ausgabe eines Multi-Objekterkennungsmodells stammen, zu entwickeln und zu evaluieren. Dabei soll der Prozess anhand einer CSV-Datei, welche die erkannten Objekte und deren Koordinaten enthält, automatisiert einen Wissensgraph erstellen. Dieser Graph soll alle Relationen ausschließlich von den Informationen der CSV-Datei herleiten und keine externen Wissensquellen einbeziehen, um eine universell einsetzbare methodische Basis zu schaffen. Da es kaum Forschung zur geeigneten Modellierung von Wissensgraphen mit visuellen Informationen gibt, sollen mehrere verschiedene Modellierungsansätze implementiert werden. Um den für den praktischen Einsatz am besten geeigneten Modellierungsansatz zu bestimmen, soll das Zusammenspiel dieser Graphen mit LLMs evaluiert werden. Da die Objekterkennung einzelner Kameras Fehler enthalten können, soll auch die Wissensgraph-Erstellung anhand mehrerer Kameras in dieser Arbeit implementiert und evaluiert werden, um die Robustheit zu erhöhen und Fehlzuordnungen zu reduzieren. Um für den praktischen Einsatz als KI-Wartungsassistent eine klare Tendenz ableiten zu können, muss darüber hinaus auch der Einfluss von unterschiedli-

cher Datenformate der Wissensgraphen und unterschiedlichen LLMs auf die Antwortqualität des LLMs untersucht werden. Aus dieser Zielsetzung wurden folgende drei Forschungsfragen für diese Arbeit abgeleitet:

1. Kann die Erstellung eines Wissensgraphen aus annotierten Bilddateien automatisiert werden?
2. Welche Wissensgraphmodellierung eignet sich am besten für den Einsatz in einem LLM-basierten KI-Wartungsassistenten?
3. Welche Modelle sind für den Einsatz mit Wissensgraphen geeignet?

Die Beantwortung dieser Fragen bildet die Grundlage für eine spätere Integration in den KI-Wartungsassistenten und eröffnen Perspektiven für den Einsatz multimodaler Wissensgraphen in anderen Domänen.

1.3 Aufbau der Arbeit

Um die Herangehensweise dieser Arbeit nachvollziehbar zu machen, wird in diesem Abschnitt der Aufbau der Arbeit erklärt. Dabei ist der Aufbau so gestaltet, dass zunächst die theoretischen Grundlagen erläutert werden, bevor die entwickelte Methodik, deren Evaluation sowie die daraus abgeleiteten Schlussfolgerungen thematisiert werden. Im theoretischen werden zunächst die Grundlegenden Konzepte dieser Arbeit erläutert, also Wissensgraphen, Ontologien, Objekterkennung und LLMs. Doch auch der aktuelle Stand der Technik zu Verfahren der Wissensgraph-Erstellung wird thematisiert, um eine Einordnung dieser Arbeit in den Kontext bisheriger Forschung setzen zu können.

2 Theoretischer Hintergrund

2.1 Wissensgraphen in der semantischen Datenverarbeitung

Wissensgraphen (Knowledge Graphs, KGs) sind eine Form der strukturierten Wissensrepräsentation, die sich im Bereich der Künstlichen Intelligenz etabliert haben. Sie bilden durch Entitäten, deren Beziehungen sowie weiteren semantisch beschriebenen Merkmalen, komplexe Sachverhalte ab. Während die Knoten eines KGs reale oder abstrakte Objekte repräsentieren, modellieren die Kanten Relationen zwischen diesen Entitäten [9].

Ein weit verbreiteter Standard für Wissensgraphen ist das Resource Description Framework (RDF). Nach diesem Standard ist das Tripel die grundlegende Struktureinheit eines Wissensgraphen und besteht aus Subjekt, Prädikat und Objekt [9, 3, 4]. Diese Subjekte und Prädikate werden dabei durch IRIs (Internationalized Resource Identifiers) identifiziert, welche einzigartige Kennzeichner-URLs des Semantic Webs sind. Mit ihnen kann man Entitäten und Relationen global identifizieren[5, 4]. Objekte hingegen können auch Literalwerte (z. B. Zahlen oder Zeichenketten) sein [5, 4]. Die Tripelstruktur kann man sich folgendermaßen vorstellen:

Subjekt	**Prädikat**	**Objekt**
Lamp	above	Motor
Lamp	x_minimum	222

Wissensgraphen bieten verschiedene Vorteile im Vergleich zu anderen Arten der Wissensrepräsentation. So sind sie durch ihr graphbasiertes Datenmodell leicht verständlich und erlauben eine prägnante und intuitive Abstraktion. RDF-Wissensgralhen besitzen zudem im Unterschied zu relationalen Datenbanken kein festes Schema, was eine flexible Entwicklung und eine einfache Erweiterbarkeit ermöglicht [5]. Ihr Anwendungsgebiet ist sehr breit und umfasst unter anderem die Informationsbeschaffung und Entscheidungsunterstützungssysteme in der Medizin, der Bildung, der wissenschaftlichen Forschung sowie sozialen Netzwerke (LinkedIn, Facebook) [3, 4]. Zudem haben sie sich auch in Abfrageverarbeitung, in Suchmaschinen (z. B. Google Knowledge Graph) und in E-Commerce (z. B. eBay, IBM) etabliert [4, 5]. Wissensgraphen verbessern außerdem die Qualität von KI-Systemen, insbesondere für Frage-Antwort-Systeme wie KI-Assistenten, die eine besondere Rolle in dieser Arbeit spielt [3].

2.2 Ontologien als formale Grundlage für Wissensgraphen

Ontologien legen die Basis für eine formale Darstellung von Wissen. Sie fungieren dabei als eine Art Konvention oder Richtlinie [5]. Sie sind in der Lage, die Vereinheitlichung terminologischer Konzepte zu fördern und ein konsistentes Verständnis zwischen unterschiedlichen Domänen zu ermöglichen [7]. Eine Ontologie enthält Entitäten, Relationen, Eigenschaften sowie je nach Ontologiesprache auch Axiome [5, 7]. Mithilfe taxonomischer Relationen können Konzept-Hierarchien

zwischen Entitäten definiert werden. Die Beziehungen zwischen einzelnen Entitäten können mithilfe sogenannter nicht-taxonomischer Relationen (in OWL auch Object-Properties genannt) beschrieben werden [7].

Ein weit verbreiteter und vom World Wide Web Consortium (W3C) empfohlener Standard ist RDFS (Resource Description Framework Schema)[10, 11]. Dieser erlaubt die Definition von Klassen und Eigenschaften (Properties) sowie Hierarchisierung dieser durch taxonomische Relationen (mit rdfs:subClassOf) [11]. In dieser Arbeit spielt die darauf aufbauende die Ontologiesprache OWL (Web Ontology Language) eine besondere Rolle. Sie ist eine Erweiterung von RDFS, die speziell für die Erstellung von allgemeiner und wiederverwendbarer Wissensbasen ausgelegt ist [1]. OWL ermöglicht auch Reasoning und Inferenz, da es auf Description Logics (DLs) basiert [5]. Dadurch können neue Fakten aus bestehendem Wissen abgeleitet und die Konsistenz der Wissensbasis überprüft werden [5, 3, 4]. Die Grundbausteine des OWL-Standards umfassen Klassen, Individuen, Object Properties, Data Properties sowie Annotationen. Klassen dienen der Gruppierung von Konzepten und der Organisation von Entitäten in Hierarchien [1, 5, 11]. Individuen sind konkrete Instanzen der in der Ontologie definierten Klassen [1, 5]. Während Object Properties die Beziehungen zwischen zwei Individuen beschreiben (z. B. Lamp above Motor), verknüpfen Data Properties Individuen mit Datentypen (z. B. Lamp detection_score 0.89) [1, 5]. Annotationen ermöglichen die Ergänzung benutzerfreundlicher Metadaten für Elemente der Ontologie, etwa in Form von mehrsprachigen Namen oder Kommentaren zur Bereitstellung zusätzlicher Informationen [1, 11].

Ontologien dienen als formales Schema für die Erstellung von Wissensgraphen und können genauso wie Wissensgraphen als Graphstruktur modelliert werden [5, 3]. Somit enthalten Wissensgraphen Daten und instanziieren die Definition einer Ontologie [9, 3].

2.3 Objekterkennung und Aufbau der annotierten Bilddateien

Objekterkennung (Object Detection) ist ein zentrales Aufgabenfeld im Bereich der Computer Vision. Das Hauptziel der Objekterkennung besteht darin, in einem Bild Instanzen von definierten Klassen zu identifizieren und deren genaue Position im Bild zu bestimmen [12].

Ursprünglich bestand die Objekterkennung aus einem Verfahren von drei Phasen. Die Erste war die Regionenauswahl, deren Ziel es ist, die Bildregionen zu erkennen, in denen ein Objekt vorhanden sein könnte [12]. In der zweiten Phase erfolgte die Merkmalsextraktion, um eine robuste Repräsentation zu erhalten. Dafür wurden Methoden wie HOG, Haar-like Features und SIFT verwendet [12, 13]. Als Letztes fand die Klassifikation statt, bei welcher mithilfe eines Klassifikators, Zielobjekte identifiziert wurden [12]. Mit dem Aufkommen von Deep Learning im Jahr 2014 wurden viele verschiedene Architekturen von Convolutional Neural Networks (CNNs) entwickelt, welche die traditionellen Techniken verdrängten. Die CNN wiesen eine höhere Genauigkeit und Geschwindigkeit auf, waren skalierbarer und boten automatisches Lernen von Merkmalen an [12, 14, 13]. Dabei lassen sich die CNN in zwei Kategorien einteilen. Zwei-Stufen-Detektoren trennen die Objekterkennung und die Klassifizierung. Beispiele hiervon sind Varianten wie R-CNN, Fast R-CNN und Faster R-CNN. Ein-Stufen-Detektoren hingegen führen die Objektlokalisierung und -klassifizierung gleichzeitig im selben Durchlauf aus. In der Regel sind diese Detektoren schneller und damit besonders geeignet für Echtzeitanwendungen (wie etwa in einem KI-Wartungsassistenten). Beispiele hiervon sind YOLO (You Only Look

Once) und SSD (Single-Shot Multibox Detector) Architekturen. Die Objekterkennung des KI-Wartungsassistenten beruht auf der SSD-Architektur, da diese versucht, die Geschwindigkeit von YOLO mit der Genauigkeit von Zwei-Stufen-Methoden zu vereinen, indem sie Techniken der beiden Ansätze kombiniert [12].

Bei der Objekterkennung gibt es spezielle Herausforderungen, welche für das industrielle Umfeld bewältigt werden müssen. Zum einen können schwierige Lichtverhältnisse und anderweitige Umgebungsbedingungen die Richtigkeit der Erkennung negativ beeinflussen [13]. In beispielsweise Wäschereien können unterschiedliche Beleuchtungen die Klassifizierung und Segmentierung stark beeinträchtigen [15]. Auch verdeckte Bauteile (Occlusion) stellen eine große Herausforderung dar. Dabei werden bestimmte Teile einer Szene für eine Kamera oder andere Sensoren unsichtbar, da diese von anderen Objekten verdeckt werden [13]. Die Erkennung kleiner Objekte ist zusätzlich eine der größten Herausforderungen in der Objekterkennung. Vor allem DCNNs (Deep Convolutional Neural Networks) haben damit häufig Probleme, da kleine Objekte aufgrund ihrer geringen Größe nur wenig Kontextinformationen liefern [12]. Auch ähnliche Objekte sind problematisch, da Objekte durch Rotation und Skalierung flexibel erscheinen und so Fehlklassifizierungen ermöglichen [13, 12].

Die Positionen der erkannten Objekte können auf unterschiedliche Weise bestimmt werden, jedoch liegt der Fokus dieser Arbeit auf der Verwendung von Bounding Boxes. Dabei ermittelt die Objekterkennung den Begrenzungsrahmen, die Objektklasse sowie einen Konfidenzwert (der die geschätzte Wahrscheinlichkeit für die Klassenzuordnung angibt) [12]. Diese Informationen werden in einer CSV-Datei gespeichert. Bei den annotierten Bilddateien, die im Laufe der Arbeit verwendet werden, handelt es sich um den Output eines trainierten Objekterkennungsmodells. Die CSV-Datei soll nun in einen Wissensgraphen überführt werden, damit LLMs mithilfe dieser Daten Fragen zu den Maschinen beantworten können.

2.4 Large Language Models und Verbindung zu Wissensgraphen

Large Language Models (LLMs) sind Künstliche Intelligenz (KI) Sprachmodelle, die oft viele Milliarden Parameter umfassen und typischerweise auf der Transformer-Architektur basieren [16, 17]. Diese beruht traditionell auf einer Encoder-Decoder-Struktur [18]. Der Encoder nimmt die kodierte, tokenisierte Eingabesequenz entgegen und erfasst so die bidirektionalen Informationen der Eingabe [18]. Er ist in der Lage, ein grundlegendes Sprachverständnis und kontextuelle Beziehungen während des Vortrainings zu erlernen [17]. Der Decoder hingegen generiert dabei die Ausgabesequenz, indem er die Wahrscheinlichkeitsverteilung über das Zielvokabular berechnet [18]. Die genaue Architektur variiert dabei je nach Modell. So nutzt die GPT-Serie beispielsweise nur Decoder. Diese decoder-only-Architektur hat sich in letzter Zeit zunehmend für im textbasierten Gen-AI-Bereich durchgesetzt. Weitere Beispiele sind Llama oder OPT [18]. Ein weiteres Kernelement der Transformer-Architektur ist der Aufmerksamkeitsmechanismus. Dieser ermöglicht es dem Modell Sequenzen mit komplexen Beziehungen zu verarbeiten, indem er die gewichtete Repräsentation jedes Tokens der Eingabesequenz mithilfe seiner Relevanz zu den anderen Tokens berechnet [18].

Auch wenn mit der LLM-Begriff sowohl generative als auch nicht-generative Modelle umfasst, wird der Begriff in der Arbeit ausschließlich im Kontext generativer Sprachmodelle genutzt [17, 18]. Dabei fungieren LLMs als generative mathematische Modelle, die auf Textdaten im Umfang

von mehreren hundert Terabyte trainiert wurden [16]. Diese Modelle berechnen die statistischen Wahrscheinlichkeiten für die möglichen Fortsetzungen einer Wortsequenz und sagen so neue Token voraus – diese können einzelne Zeichen, Wortteile oder ganze Wörter sein [16]. Somit basieren die Antworten von LLMs auf rein statistischen Korrelationen. Dies kann dazu führen, dass Halluzinationen (irrelevante oder falsche Aussagen, die allerdings sprachlich plausibel erscheinen) auftreten [16]. Dabei sind kleinere LLMs (gemessen an der Anzahl der Parameter) anfälliger für Halluzinationen aber schneller und weniger ressourcenintensiv [1, 17, 2].

Die genaue Formulierung von Eingaben (Prompts) ist für die effektive Nutzung von LLMs entscheidend [19, 16]. Das Forschungsfeld rund um diese Prompts wird Prompt Engineering genannt und zielt darauf ab, LLMs für verschiedenste Aufgaben ohne zusätzliches Training anzupassen. Dabei werden die Eingaben durch Prompt-Präfixe erweitert, was es ermöglicht die Funktionsweise eines LLMs über die Mustervervollständigung des Kontextes zu steuern [16]. Bekannte Ansätze des Prompt Engineering sind beispielsweise Zero-Shot- und Few-Shot-Prompting sowie Chain-of-Thought-Techniken (CoT), die komplexes Schlussfolgern ermöglichen [19, 17, 16]. Da der KI-Wartungsassistent ebenfalls präzise und kontextrelevante Antworten geben soll, ist das Gebiet des Prompt Engineering auch in dieser Arbeit von Bedeutung.

Ein etablierter Benchmark für LLMs ist der MMLU-Pro-Score (Massive Multitask Language Understanding Pro), der den herkömmlichen MMLU-Score (welcher aus 16.000 Multiple-Choice-Fragen aus 57 akademischen Fächer bestand [2]) um größere Herausforderungen ergänzt, die insbesondere stärker auf Argumentation ausgelegt sind. Er umfasst 12.000 Fragen aus 14 Hauptkategorien, darunter beispielsweise Mathematik, Gesundheit und Recht [20, 2]. Diese fördern und belohnen Chain-of-Thought-Reasoning und bieten, dank ihrer Komplexität, einen Raum für zukünftige Verbesserungen [20, 2]. Der Benchmark ist von besonderer Bedeutung, da in dieser Arbeit die eingesetzten LLMs anhand ihres MMLU-Pro-Scores ausgewählt wurden.

Wissensgraphen können dem LLM "Wissen injizieren", um somit wissensbasierte Anwendungen zu ermöglichen, welche logische Schlussfolgerungen erlauben. OWL-basierte Wissensgraphen bieten weitere Vorteile, wie beispielsweise die Unterstützung mehrsprachigem Wissens sowie die Kategorisierung von Entitäten und Beziehungen, was mehrsprachige Assistenten erlaubt und die Identifikation relevanter Elemente erleichtert [1]. Aus diesen Gründen werden Wissensgraphen bereits in Frage-Antwort-Systemen eingesetzt [3, 9, 4]. Da der KI-Wartungsassistent ebenfalls ein Frage-Antwort-System darstellt, sind LLMs und ihr Umgang mit Wissensgraphen von großer Bedeutung für diese Arbeit. Die Wissensgraphen werden aus visuellen Daten generiert und ermöglichen die Beantwortung einfacher (einzelner Tripel) sowie komplexerer Abfragen. Diese können über mehrstufiges Schlussfolgern (multi-hop) anhand mehrerer Tripel des Wissensgraphen beantwortet werden [3]. Zudem tragen die implementierten Wissensgraphen zur Lösung des Black-Box-Problems bei, da man mithilfe dieser die Herkunft einer Antwort erklären und nachvollziehen kann, was das Vertrauen in die Vorhersagen stärken kann [9]. Die Kombination vereint somit die Fähigkeit von Wissensgraphen zur Speicherung und Abfrage von Wissen mit der Ausdrucksstärke von LLMs [9, 4, 16, 17].

2.5 Automatisierte Wissensgraphgenerierung - Stand der Technik

Die automatisierte Wissensgraphgenerierung – auch als Ontology Learning bezeichnet – bezeichnet den Prozess der (semi-)automatischen Erstellung und Erweiterung von Wissensgraphen [7].

Kernmodule der Generierung liegen in der Wissensextraktion sowie der Wissensverlinkung [4]. Die manuelle Erstellung von Wissensgraphen ist ein arbeitsintensiver und zugleich komplexer Prozess [8]. Bei großen Domänen kann dies sogar die menschlichen Fähigkeiten übersteigen [7]. Diesen manuellen Prozess versucht die automatische Wissensgraphgenerierung zu minimieren bzw. zu umgehen.

Der Stand der Technik umfasst dabei eine Vielzahl an Ansätzen und Technologien. Natural Language Processing (NLP) wird zur Analyse und Extraktion von Informationen aus Texten verwendet [7]. Dazu zählen Techniken wie Named Entity Recognition (NER), Relation Extraction oder die Analyse lexico-syntaktischer Muster [7, 4, 8]. Auch Methoden des maschinellen Lernens (ML) werden umfassend eingesetzt [4]. Aktuelle Forschung untersucht zudem den Einsatz von LLMs zur Extraktion, Schemaerstellung und Abfragegenerierung im Bereich des Knowledge Graph Engineering (KGE) [15].

Multimodale Wissensgraphen sind besondere Wissensgraphen, die Informationen verschiedener Modalitäten verknüpfen, beispielsweise Informationen aus Texten und Bildern [3]. Ihre Konstruktion ist allerdings weiterhin ein herausforderndes Problem, da die Herausforderungen jeder Modalität zusammenkommen [3, 9]. So weist die Wissensextraktion aus Textquellen ein Problem mit der Polysemie der natürlichen Sprache sowie mit Konflikten zwischen mehreren Wissensquellen auf [3, 4]. Die Extraktion von Wissen aus Bilddateien ist allerdings sehr rechenaufwendig und kann aufgrund der in 2.3 beschriebenen Herausforderungen der Objekterkennung fehlerbehaftet oder unvollständig sein [12, 13]. Bislang sind nur wenige Arbeiten bekannt, die sich mit einem Wissensgraphgenerator aus visuellen Quellen befassen. Genau hier setzt diese Arbeit an und untersucht, ob ein solcher Generator umsetzbar und effizient ist.

3 Methodik

3.1 Struktur der Datenbasis

Format und Aufbau der Annotationsdateien

Die Wissensgraph-Erstellung soll aus der Output-Datei eines Multi-Objekterkennungsmodell erfolgen. Dabei erhält jedes Bild eine eigene .csv Datei mit allen erkannten Objekten, die im Rahmen dieser Arbeit die erkannten Bauteile (Components) der Industriemaschinen darstellen. CSV-Dateien (Comma-Separated Values) sind Klartextdateien, welche jeden Datensatz, dessen Felder durch Kommas getrennt sind, in einer Zeile repräsentieren. Die Nutzung dieser Dateien bietet sich im Zusammenhang dieser Arbeit an, da diese eine einfache Struktur und eine breite Unterstützung bieten [21]. Die Spalten der Dateien enthalten die Detection Scores (Erkennungswerte, welche angeben wie sicher sich das Objekterkennungsmodell ist), Classes (Klassen der erkannten Bauteile), sowie die Werte x min, y min, x max und y max. Anhand dieser 4 Koordinaten lassen sich die Bounding Boxes der Bauteile herleiten, die es später ermöglichen, räumliche Relationen zwischen den erkannten Objekten abzuleiten. Diese Koordinaten sind pixelbasiert und beziehen sich direkt auf das Bildformat (in unserem Fall 512x512 Pixel). Die Objektklassen sind konsistent in englischer Sprache benannt (z. B. Motor, Switch, Lamp).

Detection Score	**Class**	**x min**	**y min**	**x max**	**y max**
0.927	Lamp	222	74	316	132
0.859	Grinding wheel	39	141	147	253
0.932	Grinding wheel	319	130	409	246
0.969	Motor	117	170	349	218
0.849	Switch	220	242	262	264
0.727	Fuse	143	299	196	350

Tabelle 3.1: Beispielhafter Ausschnitt einer CSV-Datei

Aufbau der Datenbasis

Die Datenbasis besteht aus insgesamt neun annotierten Bilddateien, die jeweils verschiedenen Perspektiven auf drei Industriemaschinen zeigen. Bei den Maschinen handelt es sich um eine Schleifmaschine, eine Bohrmaschine und um einen Schalter, die zwischen vier und sieben Bauteile aufweisen. Der Aufbau der CSV-Dateien orientiert sich hierbei am typischen Output eines Objekterkennungsmodells der SSD-Architektur. Da diese Modelle nicht immer fehlerfrei arbeiten, wurde der Ablauf innerhalb der Datenbasis simuliert und die Daten manuell erstellt. Dadurch kann die Richtigkeit und Vollständigkeit der Annotationen sichergestellt werden, was eine Grundlage für die nachfolgenden Schritte der Wissensgraphgenerierung bietet.

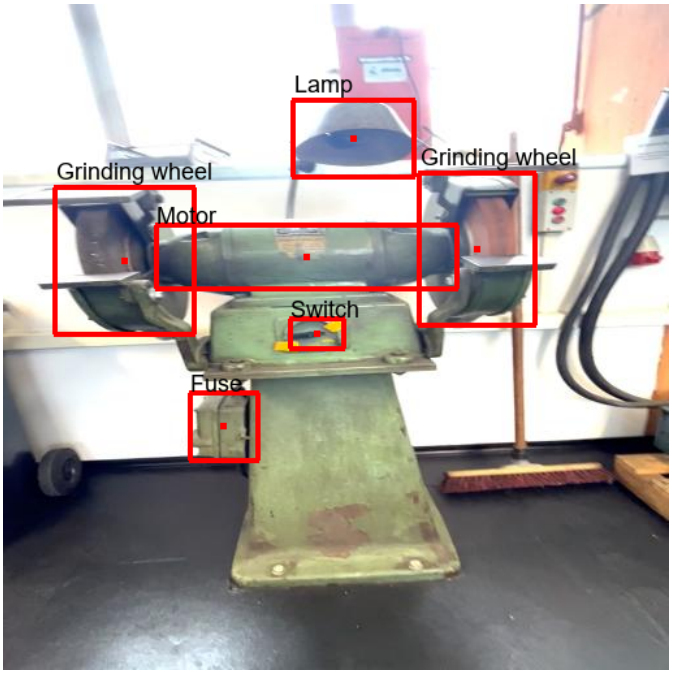

Abbildung 3.1: Visualisierte Boundingboxes des Beispiels:

3.2 Anforderung an die Graphstruktur

3.2.1 Semantische und strukturelle Anforderungen

Die Zielstruktur der generierten Wissensgraphen ist ein gerichteter RDF-Graph nach dem OWL-Standard. Dieser Repräsentiert Wissen durch Tripel, welche aus Subjekt - Prädikat - Objekt bestehen (z. B. Lamp above Motor) [3, 4, 1]. Jedes erkannte Bauteil soll als Individuum der OWL-Klasse Components instanziiert werden. Da alle Objekte eines Wissensgraphen eindeutig benannt werden müssen, müssen gleiche Bauteile anhand ihrer Position im Bild, von links nach rechts, nummeriert werden. Jeder Wissensgraph erhält außerdem ein Individuum Machine der OWL-Klasse Machines, durch welches man später die relative Position der Bauteile zur gesamten Maschine definieren kann. Die Relationen des Wissensgraphen werden auf OWL-Properties (Objekt-Properties und Data-Properties) abgebildet. Anhand der Relationen left_to, right_to, above und below sollen sich die Richtungsabhängigkeit zwischen den einzelnen Individuen beschreiben lassen. Die Relationen inside_of und outside_of sollen topologische (raumumschließende) Relationen zwischen den einzelnen Individuen modellieren. Außerdem soll mithilfe der in_the_middle_of Relation die relative Mitte der Maschine als Orientierungspunkt auf das mittig platzierte Bauteil der Maschine abgebildet werden. Dabei sollen inverse Relationen implementiert werden (also left_to - right_to, above - below, inside_of - outside_of), da diese für die Vollständigkeit des Wissensgraphen essentiell sind. Die RDF-Struktur soll dabei nicht auf eine Mindest- oder Maximalanzahl an Objekten begrenzt sein und theoretisch später anschlussfähig an domänspezifische Wissensgraphen mit faktenbasiertem Hintergrundwissen sein, die Informationen zu den Eigenschaften und Funktionen der jeweiligen Bauteile enthält.

3.2.2 Anforderungen an die technische und funktionale Nutzbarkeit

Der generierte Wissensgraph soll die etablierten Standards des semantischen Webs entsprechen, vor allem den Spezifikationen von RDF und OWL. Um eine maschinelle Weiterverarbeitung zu ermöglichen, soll der Wissensgraph sowohl als OWL-Datei, als auch in einem Tripeltextformat gespeichert werden können. Für die Einbindung in LLM-basierte Frage-Antwort-Systeme ist zudem erforderlich, dass die Relationen aussagekräftige und semantisch eindeutige Prädikate verwenden. Dies ermöglicht insbesondere mehrstufiges Schlussfolgern (multi-hop reasoning) auf Basis von logischen Ketten zwischen Entitäten. Darüber hinaus muss die Struktur des Graphen regelkonform und widerspruchsfrei sein. Das betrifft vor allem:

- die Verwendung von eindeutigen Namenskonventionen für Klassen, Individuen und Properties,
- die Vermeidung widersprüchlicher Relationen (z. B. A above B und B above A),
- die Einhaltung syntaktischer Regeln der OWL/RDF-Spezifikation,
- sowie die Unterstützung von inversen Relationen (da diese bei Richtungsangaben von großer Bedeutung sind)

Diese Anforderungen stellen sicher, dass der erzeugte Wissensgraph sowohl interoperabel und flexibel, als auch funktional für die Integration in wissensbasierte KI-Systeme geeignet ist.

3.3 Konzeption des automatischen Generierungsprozesses

3.3.1 Ablaufmodell des Konvertierungsprozesses

Ziel dieses Prozesses ist die automatische Übersetzung strukturierte Annotationsdaten (CSV) in einen OWL-konformen RDF-Graphstruktur. Dafür beginnt der Prozess im ersten Schritt mit dem Einlesen der Annotationsdaten. Dabei werden auch die weitere Werte x_center und y_center berechnet, welche die Mittelpunkte der Bounding Boxes darstellen. Anhand dieser Daten werden nun die einzelnen Bauteile als OWL-Individuen der Klasse Components instanziiert. Um die OWL und RDF Standards einzuhalten und Mehrfachbenennung zu umgehen, werden gleiche Bauteile nummeriert. Diese Nummerierung geschieht von links nach rechts im Bild anhand der x_center Koordinaten. Zusätzlich wird ein Individuum Machine der Klasse Machines erzeugt. Anschließend werden die Data Properties x_minimum, y_minimum, x_maximum, y_maximum, x_center, y_center sowie detection_score den jeweiligen Instanzen zugewiesen. Im nächsten Schritt werden nun mithilfe regelbasierter Algorithmen die räumlichen Relationen abgeleitet (siehe 3.3.2. Anhand dieser abgeleiteten Relationen werden nun die RDF-Tripel erstellt (z. B. Lamp above Motor). Am Ende wird die erzeugte Graphstruktur in einer OWL-Datei gespeichert, sowie durch einen extra Script in eine Textdatei aus Klartext-Tripeln konvertiert. Für eine visualisierte Version, siehe 3.2.

3.3.2 Logik zur Relationsermittlung

Um die Ableitung semantisch interpretierbarer Relationen aus rein geometrischen Daten zu ermöglichen, wurden regelbasierte Algorithmen definiert. Diese verwenden die Koordinaten der Bounding Boxes, sowie die daraus berechneten Mittelpunktkoordinaten (x_center und y_center).

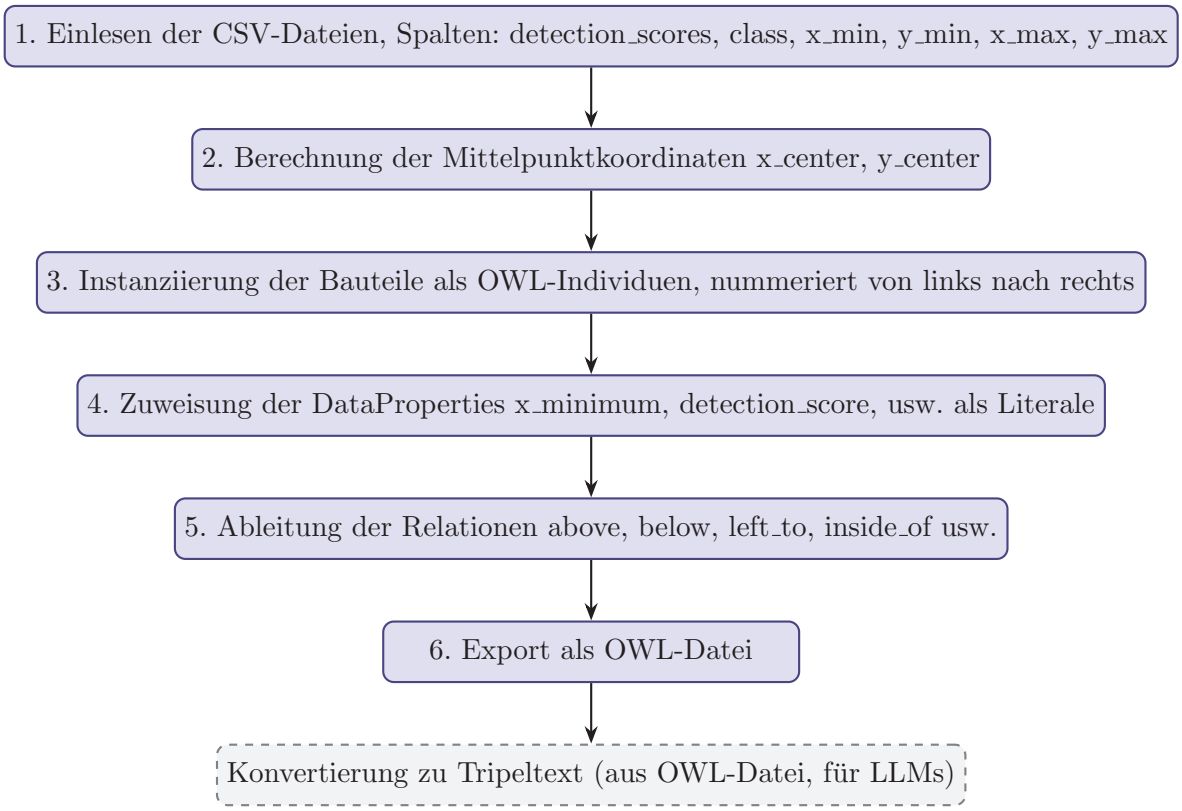

Abbildung 3.2: Flussdiagramm des Ablaufs

Die Bestimmung der horizontalen Relationen left_to und right_to basiert auf den Differenzen der x_center und y_center Werte zweier Individuen. Dabei dient das Vorzeichen der x-Differenz zur Richtungszuordnung. Um zu vermeiden, dass Bauteile horizontale Relationen erhalten, obwohl sie vertikal stark versetzt sind, wird zusätzlich das Verhältnis der y- zur x-Differenz betrachtet. Die Relation wird nur gesetzt, wenn das Verhältnis einen bestimmten Schwellenwert nicht übersteigt. Dafür prüft der Algorithmus folgende Schrägheitsbedingung:

$$|y_i - y_j| < \frac{1}{\texttt{threshold}} \cdot |x_i - x_j|$$

Die besten Ergebnisse wurden mit einem Wert von threshold = 0,7 empirisch ermittelt, was bedeutet, dass das Nachbar-Bauteil maximal eine Abweichung von ca. 55° von der horizontalen Achse des betrachtenden Bauteils abweichen darf (0 Grad heißt hierbei perfekt horizontal). Dies verhindert, dass Bauteile als links oder rechts erkannt werden, obwohl sie fast vertikal unter dem Bauteil angeordnet sind. Hierbei ist es also möglich, dass ein Individuum, z. B. sowohl links, als auch unter einem anderen Bauteil liegen kann. Durch die Verwendung von gerichteten Differenzen ist allerdings sichergestellt, dass keine widersprüchlichen Relationen (z. B. A left_to B und B left_to A) entstehen können, da das Vorzeichen der Differenz eindeutig bestimmt, in welcher Richtung die Bauteile voneinander liegen. Optional kann der Algorithmus sowohl zwischen impliziter und expliziter Erkennung unterscheiden: Im impliziten Modus wird nur die Relation des nächstgelegenen Bauteils in jede Richtung berücksichtigt, während im expliziten Modus alle zulässigen Richtungsrelationen gesetzt werden. Die vertikalen Relationen above und below werden analog zu den horizontalen Relationen bestimmt, wobei die x- und y-Differenzen

entsprechend vertauscht werden.

Die topologischen Relationen inside_of und outside_of werden anhand der minimalen und maximalen Koordinaten der Eckpunkte der Bounding Boxes hergeleitet (x_min, y_min, x_max, y_max). Dabei berechnet der Algorithmus die Überschneidung zweier Bounding Boxes und setzt diese ins Verhältnis zur Fläche der größeren Box:

$$\frac{\text{Area}(\text{Box}_i \cap \text{Box}_j)}{\max(\text{Area}(\text{Box}_i), \text{Area}(\text{Box}_j))} > \texttt{threshold}$$

Überschreitet dieser Wert einen definierten Schwellenwert (ebenfalls empirisch mit threshold = 0,7 bestimmt), wird das kleinere Objekt als inside_of dem größeren Individuum als Relation zugeordnet.

Für die Relation in_the_middle_of wird das Bauteil bestimmt, dessen Mittelpunktkoordinaten dem Durchschnitt der Mittelpunktkoordinaten aller Bauteile am nächsten liegt. Das eingesetzte Framework owlready2 ermöglicht außerdem die Definition von inverser Operationen des OWL-Standards, wodurch jede Relation automatisch eine semantisch äquivalente Gegenrelation erzeugt. Die beschriebenen Algorithmen gewährleisten nicht nur Konsistenz und Widerspruchsfreiheit, sondern erzeugen auch verständliche und reproduzierbare Ergebnisse.

3.4 Technische Umsetzung und Implementierungsdetails

3.4.1 Strukturelle Varianten der Generierung

Um die Umsetzung des Generierungsprozesses zu ermöglichen wurde ein modulares Programmsystem entwickelt. In diesem Kapitel werden Da der Forschungsstand zu Wissensgraphen mit visuellen Informationen sehr begrenzt ist, ist die Implementierung und Analyse unterschiedlicher Modellierungsansätzen von besonderer Bedeutung. Denn je nach Anwendungsfall und Datenlage könnten unterschiedliche Ansätze optimal sein.

Richtungsrelationen - implizit oder explizit: Der Wissensgraphgenerator unterstützt zwei Varianten zur Erzeugung von Richtungsrelationen: eine implizite und eine explizite Struktur. Bei der impliziten Relationserzeugung wird immer nur die Richtungsrelation des nächsten Nachbarbauteils gesetzt. Dies gilt je Richtung, sodass jedes Individuum maximal vier Richtungsrelation besitzen kann (von jeder Relation eine). Dadurch wird der Graph kompakter und deutlich weniger komplex, was insbesondere im Einsatz mit kleineren LLMs in einem Frage-Antwort-Modell dafür sorgen könnte, dass es den Wissensgraph besser verarbeiten kann. Allerdings müsste diese multi-hop-reasoning einsetzen, um z. B. alle Bauteile unter der Lampe zu nennen. Bei der expliziten Relationserzeugung hingegen werden alle zulässigen Richtungsrelation gesetzt, sodass jedes Individuum theoretisch maximal n-1 Relationen erhalten könnte (n = Anzahl der Individuuen). Hierbei würde der Graph deutlich komplexer, aber auch semantisch vollständiger. Auch das multi-hop-reasoning wird dadurch vermieden, allerdings könnte ein LLM es schwieriger haben, herauszufinden welches Bauteil beispielsweise direkt unter der Lampe ist.

Numerische Daten - mit oder ohne Koordinaten: Auch im Bezug auf numerische Informationen besitzt der Wissensgraphgenerator zwei Möglichkeiten: ohne Koordinaten und mit Koordinaten. Ohne die Speicherung der Koordinaten wird der erzeugte Wissensgraph deutlich

kompakter, was wie bereits erwähnt bei kleineren LLMs mit begrenzten Kontextfenster von großer Bedeutung sein könnte. Im Modus mit Koordinatenspeicherung werden diese in den Data-Properties x_minimum, y_minimum, x_maximum, y_maximum, x_center und y_center hinterlegt. Dadurch wird der Graph nachvollziehbarer und LLMs können beispielsweise räumliche Abstände abschätzen, ihre eigene Schlussfolgerungen ableiten oder Ergebnisse überprüfen.

Exportformate - OWL und Tripeltext: Auch der Exportprozess unterstützt zwei verschiedene Formate: OWL-Dateien und Textdateien im Tripel-Format. Wobei die OWL-Datei für den standardkonformen Export semantischer Graphen dient und Kompatibilität mit Ontologie-Editoren wie z. B. Protege ermöglicht. Allerdings kann es für LLMs unvorteilhafter sein mit diesen Dateien umgehen, als mit einfachem Textinput. Für diese mögliche Verbesserung der Ergebnisse für die Integration in LLM Frage-Antwort-Systeme ist außerdem der Export in eine Textdatei der RDF-Tripel des erzeugten Graphen.

All diese Varianten erlauben eine flexible Generierung der RDF-Struktur je nach gewünschtem Zielsystem. Alle Varianten können dabei beliebig per Konfigurationsdatei kombiniert werden, sodass sich der jeweils geeignetste Modellierungsansatz des Wissensgraphen je nach Zielsystem bestimmen lässt (siehe 4 zur LLM-Integration).

3.4.2 Codebasierte Umsetzung der Varianten

Das Python Skript OntologyGenerator.py wurde unter der Verwendung des owlready2 Frameworks entwickelt und stellt den zentralen Baustein der automatischen Generierung RDF-Wissensgraphen dar. Zunächst wird die Konfigurationsdatei eingelesen, deren Dateipfad beim Starten des Programms mit dem Argument -c übergeben wurde (z. B. -c config.ini). Aus dieser Datei werden die Parameter output_path, csv_path, add_coordinates, explicit_mode, remove_false und summarize_graph geladen.

Je nachdem, ob der csv_path eine CSV-Datei oder ein Ordner ist, wird der Single- oder der Multi-Cam-Modus aktiviert. Beim Single-Cam-Modus wird die CSV-Datei ausgelesen und das Skript berechnet bereits, welches Bauteil sich in der Mitte der Maschine befindet. Im Anschluss wird die Ontologie, die festen Klassen, die Object- und Data Properties sowie die inversen Properties mithilfe von owlready2 erstellt. Danach wird für jeden Eintrag der CSV-Datei ein neues Individuum der Klasse Components erstellt und ihm der zugehörige Detection Score zugeordnet. Falls der Parameter add_coordinates in der Configdatei auf true gesetzt ist, werden zusätzlich ihre Koordinaten ergänzt. Anschließend werden den Individuen ihre räumlichen Relationen zugeordnet. Dies geschieht, indem für jedes Individuum, der in 3.3.2 beschriebene Algorithmus durchgeführt wird. Dieser unterscheidet allerdings anhand der explicit_mode Flag, ob die Relationen implizit und explizit modelliert werden. Beim impliziten Modus der check_vertical Funktion werden die Werte above_min und below_min mit dem Wert 1024 initialisiert (da das der maximale Abstand ist, der bei einem 512x512 Bild möglich ist) und die leeren Listen above_class und below_class erstellt. Dabei wird für jedes Bauteil, das links oder rechts von dem Individuum liegt, der Abstand berechnet und verglichen, ob dieser unter dem Minimalabstand liegt. Wenn nein, ignoriert der Algorithmus das Bauteil, da das nähere Bauteil bereits in der Liste stehen muss. Wenn ja, werden die Einträge der Liste gelöscht, und das neue Bauteil wird der Liste hinzugefügt. Dabei wird der neue Minimalabstand in die left_min oder right_min Variable geschrieben. So kann der Algorithmus zuverlässig das nächstgelegene Bauteil für jede Richtung bestimmen. Beim expliziten Modus läuft das Programm analog ab, außer dass die Listenelemente nicht entfernt

werden und der Mindestabstand auf 1024 festgelegt bleibt. Dadurch können in die Liste mehrere Bauteile aufgenommen werden und ermöglichen so die explizite Modellierung. Der Algorithmus für die vertikalen Relationen läuft genauso ab und muss deswegen nicht nochmal erklärt werden.

Anschließend wird die Relation in_the_middle_of zwischen dem bereits am Anfang bestimmten mittleren Bauteil und dem Objekt Machine gesetzt. Danach wird die Funktion check_inside aufgerufen, welche die topologischen Relationen modelliert. Die Funktionsweise dieser wurde allerdings bereits in 3.3.2 ausführlich erleutert und weist keine wesentlichen codebasierten Besonderheiten auf. Das owlready2 Framework erkennt erst beim Speichern der OWL-Datei Dopplungen und inverse Relationen, was zu Problemen führen kann, da der Code mit den aktuellen Werten der owlready2-internen Relationslisten weiterarbeitet (vor allem im Multi-Cam-Modus, siehe 3.4.3). Aus diesem Grund werden anschließend die zwei Hilfsfunktionen reverse_properties und remove_redundant_properties aufgerufen. Dabei ergänzt die reverse_properties Funktion für jedes Tripel aus Subjekt - Prädikat - Objekt, das inverse Tripel aus Objekt - inverses Prädikat - Subjekt (z. B. Lamp above Motor - Motor below Lamp). Da bei dieser Funktion auch Dopplungen auftreten können, werden diese im Anschluss mit der remove_redundant_properties Funktion entfernt, indem die owlready2-Relationenlisten in ein Set und danach wieder in eine Liste umgewandelt werden. Da ein Set per Definition keine Duplikate enthalten kann, werden dadurch alle doppelten Einträge gelöscht.

Am Ende des Programms wird die RDF-Graphstruktur als OWL-Datei exportiert und kann, wenn notwendig, mit dem StatementGenerator.py Skript in eine Textdatei mit Tripeln umgewandelt werden. Dafür muss das Programm mit dem -o Parameter gestartet werden, welcher den Pfad der OWL-Datei angibt. Durch die Konfigurationsdatei ist es möglich, die verschiedenen Varianten miteinander beliebig zu kombinieren und diese jederzeit einfach zu ändern, was kombinierte Tests und Modellierungsvergleiche ermöglicht. Durch die modulare Struktur des Programms lässt es sich leicht erweitern und einzelne Komponenten können unabhängig voneinander getestet werden.

3.4.3 Erweiterung des Generierungsprozesses für mehrere Kameras

Die Wissensgraph-Erstellung aus einem Kamerawinkel ist für diesen optimiert, aber kann wichtige Informationen möglicherweise nicht erfassen. Denn je nach Blickwinkel können bestimmte Komponenten versteckt, gar nicht sichtbar in toten Winkeln oder von der Objekterkennung nicht erkannt worden sein. Um diese Probleme zu beheben, wurde der Generierungsprozess um einen Modus für mehrere Kameras (Multi-Cam-Modus) ergänzt. Er ermöglicht es dem Prozess, verschiedene Perspektiven zu modellieren und so dem LLM-basierten Frage-Antwort-System zusätzlichen Kontext bereitzustellen.

Der Multi-Cam-Modus wird gestartet indem der csv_path Parameter der Konfigurationsdatei auf einen Ordner mit mehreren CSV-Dateien zeigt und läuft analog zum Single-Cam-Modus ab, aber besitzt einige große Unterschiede. Dafür werden zuerst alle CSV-Dateien aus dem Ordner gefiltert. Danach führt der Prozess die bereits in 3.4.2 beschriebenen Schritte durch, allerdings unterscheiden sich in ein paar Implementierungsentscheidungen. Dabei wird zu Beginn des Einlesens jeder CSV-Datei, eine neue Klasse Camera_[Name der CSV-Datei] erstellt. Die Components und Machines Klassen werden dann innerhalb dieser Camera Klasse erstellt. Diese Klassen werden in einem Python Dictionary gespeichert und können so jederzeit aufgerufen werden. Nachdem das Erstellen der Klassen durchgeführt wurde, läuft der Prozess wie

im Single-Cam-Modus (siehe 3.4.2) ab und wiederholt sich für jede CSV-Datei. Dabei wird das Dictionary aller Individuen einer Iteration, im Dictionary all_individuals gespeichert, das am Ende des letzten Durchlaufs alle Individuen enthält.

Da nun nach erfolgreichem Durchlaufen des Prozesses ein verschachteltes Dictionary anstatt einem Einfachen vorliegt, müssen die check_inside, reverse_properties und remove_redundant_properties Funktionen dafür angepasst werden. Dafür wurden die check_inside_all, reverse_properties_all und remove_redundant_properties_all Funktionen implementiert, welche sich um die Organisation dieses verschachtelten Dictionarys kümmern. Dabei lesen die Funktionen nacheinander jedes Einfache Dictionary aus dem Verschachtelten, führen die Ursprungsfunktion aus und ersetzen das jeweilige Einzeldictionary durch die aktualisierte Version. Alle Änderungen an der Usprungsfunktion führen somit auch zu einer Änderung der zugehörigen _all Funktionen, da diese wie eine Art Vermittler fungieren.

Nachdem sowohl die check_inside_all Funktion zur Bestimmung der topologischen Relationen und die reverse_properties_all Funktion zum aktualiseren der owlready2-internen Relationslisten ausgeführt wurden, wird anschließend die same_individuals Funktion ausgeführt. Das Ziel dieser ist es, anhand eines regelbasierten Algorithmus zu erkennen, welche Bauteile der unterschiedlichen Kameras die identischen Bauteile sein könnten. Dabei soll er die OWL-native sameAs Relation zu den identischen Individuen hinzufügen. Dabei berechnet diese Funktion den Ähnlichkeitsgrad anhand der übereinstimmenden Relationen. Dafür werden die Individuen paarweise mit allen denen der anderen Kameras verglichen. Die Berechnung erfolgt mithilfe einer Ähnlichkeitsformel, deren Bestandteile im Folgenden definiert werden:

- A, B: Zu vergleichende Individuen
- R: Menge aller zu betrachteten Relationen (z. B. *above*, *below*, *left_to*, ...)
- $r(A)$: Menge aller Ziel-Individuen, mit denen A über die Relation r verknüpft ist
- $|M|$: Anzahl der Elemente in der Menge M

$$\text{similarity}(A, B) = \frac{\sum_{r \in R} |r(A) \cap r(B)|}{\sum_{r \in R} |r(A)|}$$

$$\text{threshold} = \begin{cases} \text{threshold}_{\text{equal}}, & \text{wenn name}(A) = \text{name}(B) \\ \text{threshold}_{\text{diff}}, & \text{wenn name}(A) \neq \text{name}(B) \end{cases}$$

$$\text{mit} \quad \begin{cases} \text{threshold}_{\text{equal}} = \text{threshold} - \text{tolerance} \\ \text{threshold}_{\text{diff}} = \text{threshold} \end{cases}$$

$$\text{Dann gilt:} \quad \text{similarity}(A, B) > \text{threshold} \quad \Rightarrow \quad A \equiv B$$

Dabei wird die Summe aller gleichen Relationen mit dem gleichen Ziel-Individuum von 2 Individuen gebildet und anschließend durch die Anzahl aller Relationen des zu betrachten Individuums geteilt. Wenn dieser berechnete Ähnlichkeitswert über einen bestimmten threshold liegt, erhalten die beiden Individuen die OWL-native sameAs-Relation. Wenn die beiden Individuen den

gleichen Namen mit derselben Nummerierung besitzen (z.B. Schleifscheibe1_cam1 und Schleifmaschine1_cam2), wird dem threshold ein tolerance Wert abgezogen, was die Ähnlichkeitsprüfung weniger strikt gestaltet. Dies ist gerechtfertigt, da der gleiche Name und gleiche Nummer garantieren, dass sie derselben Klasse und an derselben Position innerhalb der Reihenfolge von links nach rechts sind. Durch diese bereits hohe Wahrscheinlichkeit kann der Schwellenwert kleiner werden. Die besten Ergebnisse wurden hierbei empirisch bei threshold = 0.75 und tolerance = 0,45 festgestellt.

Anschließend wird, falls der remove_false Parameter in der Konfigurationsdatei auf true gesetzt wurde, die remove_false_detections Funktion aufgerufen. Diese ermöglicht es, mögliche Fehlerkennungen des Objekterkennungsmodell aus dem Wissensgraphen zu entfernen. Dafür wird zuerst die durchschnittliche Anzahl der sameAs-Relationen aller Individuen berechnet. Danach prüft die Funktion für alle Individuen, ob diese weniger sameAs-Relationen als der Durchschnitt besitzt, und entfernt diese falls sie einen Detection Score von unter einem gewissen threshold besitzen. Durch die Berücksichtigung des Detection Scores, werden nur Individuen gelöscht, die sowohl auf den anderen Kameras anders oder gar nicht erkannt wurden und bei der zu betrachtenden Kamera sogar ein niedriger Detection Score auf eine erhöhte Unsicherheit des Objekterkennungsmodells hinweist. Auch dieser Schwellenwert wurde empirisch auf 0.8 festgelegt.

Vor dem Speichern des RDF-Graphstruktur wird die ontologySummarizer Python Klasse aufgerufen, wenn der summarize_graph Parameter der Konfigurationsdatei auf true gesetzt wurde. Diese Klasse erzeugt zunächst für jedes Individuum ein Python Set, das sowohl sich selbst als auch alle über die sameAs-Relation verknüpften Individuen enthält. Diese generierten Gruppierungssets aller Individuen werden in einer Python Liste gespeichert, wenn dieses Set nicht bereits in der Liste vorkommt. Um dies zu bestimmen, iteriert der Algorithmus über alle Listenelemente und prüft, ob eine beidseitige Teilmengenbeziehung besteht. Wenn beide Subsetbedingungen erfüllt sind, befindet sich das Gruppierungsset bereits in der Liste und wird nicht erneut hinzugefügt. Nachdem alle Individuen in der Liste gruppiert wurden, erstellt der Algorithmus eine neue Ontologie mithilfe des owlready2 Frameworks, welche den exakt gleichen Aufbau der Ontologien des Single-Cam-Modus besitzt. Anschließend wird ein neues Individuum für jedes Gruppierungsset erstellt, welches den Namen enthält der am häufigsten in besagten Set vorkommt. Danach werden sämtliche Relationen aller Indiviuen eines Gruppierungssets dem neuen Individuum zugeordnet. So können alle Relationen der Kameras zuverlässig zusammengefasst werden. Am Ende wird die RDF-Graphstruktur als OWL-Datei exportiert und kann danach auch, wie bereits in 3.4.2 erwähnt, in einen Tripeltext umgewandelt werden.

Die Berücksichtigung mehrerer Kameraperspektiven erhöht die Robustheit des Systems gegenüber Unsicherheiten und Fehlern des Objekterkennungsmodells sowie verdeckten Bildbereichen. Das Zusammenführen mehrerer Perspektiven ermöglicht einen kompakten und semantisch reichhaltigen Wissensgraphen, der besonders für LLM-Systeme geeignet ist. Eine Einschränkung dieser Methode besteht jedoch darin, dass die Fusion der Perspektiven nur rein heuristisch erfolgt und keine Kalibrierung der Kameras oder zusätzliche externe Informationen verwendet. Eine Erweiterung um solche externen Faktoren und Informationen könnte perspektivisch eine 3D-Rekonstruktion ermöglichen , die eine noch präzisere Zusammenführung der Perspektiven erlaubt.

3.4.4 Allgemeine Nutzbarkeit und Integration

Die automatisierte Generierung eines OWL-konformen Wissensgraphen aus annotierten Bilddateien wurde mit dem bereits beschriebenen Skript umgesetzt, doch ein praktischer Mehrwert entsteht erst, wenn das System in reale Anwendungen eingebunden wird.

Der primäre Anwendungskontext, für welchen der Wissensgraphgenerator nativ entwickelt wurde, ist der KI-Wartungsassistent. Seine Vision ist es, die Wartung von Industriemaschinen zu erleichtern, damit nicht jedes Mal ein Wartungstechniker vor Ort sein muss. Ungeschultes Personal soll durch diese Erleichterung in der Lage sein, einige Maschinen selbst warten zu können. Der Assistent sollte hierbei bei z. B. der Wartung, Fehlerdiagnose und Bauteilidentifikation unterstützen. Dabei kann das Personal mit dem KI-Wartungsassistenten über eine Spracheingabe interagieren. Die aufgenommene Sprache wird dann mit einem Spracherkennungsmodell in einen Fließtext umgewandelt, welcher dann als Eingabe in ein LLM eingefügt wird. Damit das LLM in der Lage ist, Aussagen über räumliche Verhältnisse zu treffen, enthält diese den Wissensgraph in Form von Tripeltext zusätzlich als Eingabe. Das ermöglicht Fragen wie beispielsweise: "Wo befindet sich der Motor?öder auch "Welche Bauteile befinden sich unter der Lampe?ßu beantworten. Durch dieses Einfügen von Wissen über räumliche Verhältnisse, kann die Systemgenauigkeit erhöht und die Anzahl von Halluzinationen verringert werden. Der generierte Wissensgraph dient also in diesem Anwendungsfall als eine Art strukturierte Wissensquelle, welche erweiterbar und flexibel ist. So kann entweder der Wissensgraph anschlussfähig an einen domänspezifischen Wissensgraphen mit faktenbasiertem Hintergrundwissen sein, oder man fügt diesen domänspezifischen Graphen einzeln der Eingabe des LLMs hinzu, um auch zuverlässiges Faktenwissen dem KI-Wartungsassistenten zu ermöglichen.

Auch jenseits des Wartungsassistenten lässt sich die automatische Wissensgraph-Erstellung integrieren. Da der Wissensgraph-Generator nur mit den Informationen der CSV-Datei arbeitet und nur wenige fest implementierte Klassen besitzt, kann dieser auf beliebigen Maschinen oder sogar in anderen Szenarien eingesetzt werden. Hierfür müsste man die Konfigurationsdatei um die Namen von Klassen und Individuen ergänzen und diese im Code implementieren, da zur Zeit die Klassen Components, und Machines sowie das Individuum Machine und die Semanticweb-Adresse noch fest im Code vorgeschrieben sind. Alles andere basiert bereits nur auf den erkannten Objekten und wird somit auch automatisch auf völlig unterschiedliche Szenarien übertragen. Mit diesen Anpassungen wäre es möglich, den Wissensgraphgenerator in viele verschiedene LLM-basierte KI-Systeme zu integrieren, die eine optische oder räumliche Komponente besitzen sollen. Durch die bereits angesprochene Möglichkeit, den generierten Wissensgraphen mit faktenbasiertem Hintergrundwissen anzureichern, können so multimodale Wissensgraphen erstellt werden, die sowohl räumliche als auch logische Verbindungen und Hintergrundinformationen repräsentieren. Damit zeigt sich, dass das System nicht nur auf den Wartungsassistenten beschränkt ist, sondern auch Potenzial für viele unterschiedliche KI-Anwendungen bietet.

4 Evaluation

4.1 Validierungskonzept und Evaluationskriterien

Um zu ermitteln, wie die LLMs mit den jeweiligen Wissensgraph-Modellierungen und -Formaten interagieren, werden die generierten Wissensgraphen an vier verschiedenen LLMs evaluiert. Darunter befinden sich drei LLMs unterschiedlicher Größe (gemessen an der Anzahl der Parameter) sowie einem Reasoning-LLM. Da die Richtigkeit und Vollständigkeit der generierten Wissensgraphen bereits während der Implementierung überprüft wurde um potenzielle Fehler zu finden, steht nicht der Wissensgraph selbst im Fokus, sondern dessen Interaktion mit den LLMs. Dadurch können wichtige Erkenntnisse über den praktischen Einsatz des Generators für einen KI-Wartungsassistenten gewonnen werden.

Um die Antworten der verschiedenen LLMs vergleichen zu können, wurden folgende Evaluationsmetriken definiert:

- die Richtigkeit der Antwort,
- die Vollständigkeit der Antwort,
- sowie weitere Anmerkungen und Auffälligkeiten

Die Richtigkeit überprüft hier, ob alle Inhalte der Antwort sachlich korrekt sind. Die Vollständigkeit hingegen überprüft, ob alle relevanten Informationen berücksichtigt wurden. Somit ist es möglich, dass eine Antwort eines LLMs zwar nicht der Richtigkeit entspricht, allerdings trotzdem vollständig ist, wenn beispielsweise ein Bauteil zu viel genannt wurde. Auch umgekehrt kann eine Antwort richtig und unvollständig sein, wenn beispielsweise ein Bauteil zu wenig genannt wurde. Die Richtigkeit und Vollständigkeit sind folglich sehr objektive Metriken, die laut ihrer Definition wenig subjektiven Spielraum lassen. Die weiteren Anmerkungen und Auffälligkeiten hingegen dienen dazu, besonders auffällige, subjektive Beobachtungen wie Halluzinationen, unerwartete Muster oder Bemerkungen zur Antwortzeit zu dokumentieren. Um einheitliche Schlüsse und eine gleichmäßige Übersicht über die Ergebnisse zu ermöglichen, wurde außerdem ein aggregierter Score-Wert definiert, welcher aus allen Antworten eines LLMs und einer Wissensgraphmodellierung gebildet wird. Dabei gibt sowohl die Richtigkeit als auch die Vollständigkeit der Antwort jeweils 2 Punkte. Auch die besonderen Anmerkungen und Auffälligkeiten fließen in den Score-Wert ein. Dabei können sie entweder -1, 0 oder +1 Punkte geben, je nachdem ob die Anmerkungen sich negativ, neutral oder positiv auf die Antwort auswirken. Um im begrenzten Rahmen dieser Arbeit sowohl eine qualitative als auch quantitative Analyse zu ermöglichen, wurden die Metriken pragmatisch orientiert gewählt.

Um eine einheitliche und faire Evaluation der Antworten zu ermöglichen, erhalten alle LLMs identische Prompts. Auch die Nachbearbeitung der Antworten ist nicht erlaubt. Um eine einheitliche Anwendung der Evaluationskriterien zu gewährleisten, wurde die Bewertung konsistent durch den Autor vorgenommen.

4.2 Aufbau der Evaluation

4.2.1 Verglichene Wissensgraph-Varianten und LLMs

Um die Analyse des Einflusses der Modellierung und des Formates der Wissensgraphen auf die LLM-Antworten zu ermöglichen, müssen die unterschiedlichen Varianten getestet werden. Damit diese Evaluation vollständig ist, werden alle möglichen Kombinationen der Wissensgraphen generiert. Dies betrifft zehn verschiedene Modellierungen mit zwei Datenformaten. Die einzelnen Wissensgraphmodellierungen wurden einheitlich mit Abkürzungen benannt, die durch ihre Kombination ermöglichen, alle zehn Modellierungen im Verlauf der Arbeit eindeutig zu bezeichnen. Die Abkürzungen sind folgendermaßen definiert:

Abkürzung	**Bedeutung**
imp	implizite Relationsmodellierung
exp	explizite Relationsmodellierung
nc	ohne Koordinaten
c	mit Koordinaten
sim	Einzelperspektive (single image)
mim	Mehrfachperspektive (multi image)
sum	zusammengefasste Mehrfachperspektive (summary)

Tabelle 4.1: Verwendete Abkürzungen der Wissensgraphmodellierungen und deren Bedeutungen

Neben den unterschiedlichen Wissensgraphen wurden auch unterschiedliche LLMs ausgewählt, um eine möglichst umfassende Evaluation zu ermöglichen. Diese Modelle müssen öffentlich verfügbar, kostenlos nutzbar und vorzugsweise open-weight sein. Ausgewählt wurden die einzelnen Modelle anhand ihres MMLU-Pro-Werts (Juni 2025), der bereits in 2.4 thematisiert wurde.

LLM	**MMLU-Pro**	**Parameter (in Milliarden)**	**Kontextgröße**
DeepSeek-R1	0.84	671	128k
DeepSeek-V3-0324	0.813	671	128k
Llama-3.1-8b-instruct	0.4425	8	128k
Qwen-2.5-3B	0.4373	3	32,768

Tabelle 4.2: LLMs für die Evaluation

Im praktischen Einsatz des KI-Wartungsassistenten ist es nicht möglich, die beiden DeepSeek-Modelle lokal auf einem Server zu betreiben, da sie aufgrund ihrer hohen Parameteranzahl zu ressourcenintensiv sind. Diese müsssten über eine API angebunden werden. Das Llama- und Qwen-Modell hingegen würden es erlauben, die Textverarbeitung lokal auf dem Server auszuführen. Dies ist besonders relevant für den KI-Wartungsassistenten, der in ressourcenbeschränkten Umgebungen, wie beispielsweise internen Servern, betrieben werden könnte. Auch die Kontextgröße, die angibt wie viele Token ein Modell gleichzeitig verarbeiten kann, ist insbesondere bei umfangreichen Wissensgraphen von Bedeutung. Anhand der Evaluationsergebnisse lässt sich somit neben der Leistung auch die Praxistauglichkeit bewerten, das heißt, wie sich die Parameteranzahl auf die Informationsverarbeitung eines Wissensgraphen auswirkt und welche Modelle sich für welche Einsatzszenarien wissensgraphbasierter Anwendungen eignen.

4.2.2 Erstellung und Struktur der Testfragen

Um eine umfassende Evaluation zu ermöglichen, werden praxisnahe Fragen mit unterschiedlichen Komplexitätsstufen gestellt. Dabei werden pro Maschine zwei einfache Wissensabfragen, eine Multi-Hop-Wissensabfrage und eine Multi-Hop-Wissensabfrage mit gefordertem Allgemeinwissen erstellt. Während die beiden einfachen Wissensabfragen mit nur einem Tripel beantwortet werden können, muss das LLM bei den Multi-Hop-Wissensabfragen den gesamten Wissensgraph berücksichtigen, um eine finale Antwort geben zu können. Die Multi-Hop-Fragen mit gefordertem Allgemeinwissen sind die komplexesten Testfragen, da sie neben dem Verständnis des Wissensgraphen auch logisches Denken und Allgemeinwissen erfordern. Dabei wurden die Fragen in Englisch formuliert, da kleinere LLMs in Verbindung mit Wissensgraphen in der englischen Sprache besser abschneiden [1]. Um möglichst praxisnahe Wartungsszenarien abzudecken, wurden für die drei Maschinen (Schleifmaschine, Bohrmaschine und Schalter), zwölf Fragen wie folgt definiert:

Grindingmachine

1. Which component is located above the motor?
2. Which component is located left to the motor?
3. Where is the left grinding wheel located?
4. If the motor overheats, which parts would be affected?

Drillingmachine

1. Which component is located above the drilling table?
2. Which component is located left to the bottom lever?
3. Where is the wrench hanging?
4. Which components are involved in operating the machine?

Switch

1. Which component is located below the top button?
2. Which component is located above the power plug?
3. How many buttons are on the housing?
4. The supply of power is not working, what could be the issue?

Um die gewünschten und praxisgerechten Antworten der LLMs zu erhalten, werden den Modellen mittels Prompt Engineering präzise Anweisungen für ihre Antworten gegeben. So sollen die Antworten im Einsatzszenario eines KI-Wartungsassistenten möglichst prägnant alle relevanten Informationen wiedergeben. Zudem soll das LLM nicht erklären, aus welchen Tripeln des Wissensgraphen die Informationen stammen, da diese für ungeschultes Personal nicht praktisch nutzbar sind. Auch die Koordinaten sollen von den LLMs nicht in ihrer Antwort genannt werden, sondern lediglich als Hintergrundinformationen in die Antwort einfließen. Da die Koordinaten

relativ von der Kameraperspektive gemessen wurden, würden sie dem Personal im praktischen Betrieb nicht weiterhelfen. Die Anweisung enthält außerdem Beispiele und verdeutlicht dem LLM, wie es mit mehreren Kameraperspektiven umgehen soll, wenn diese im Wissensgraph vorhanden sind. Die vollständige Anweisung (Baseprompt) lautet:

You are a maintenance assistant who helps workers maintain machines. I will provide information about a machine and ask you questions. Answer the questions briefly and precisely, while still using all the information and deciding for yourself which of the information is important and helpful to the worker. Don't tell me where you got the answer from, just the answer itself. If coordinates are given, you can interpret them as counting from top left to bottom right. The coordinates are relative, so don't dwell on them too much since they can't help the workers, but they can help you see relationships between the components. If you have multiple camera perspectives, form your own complete picture from all the cameras. Don't go into each component of each camera individually, but generalize them. The components of the different cameras are marked with a "_".

4.2.3 Testumgebung und Ablauf

Die Evaluation wurde, je nach verwendetem LLM, in zwei verschiedenen Umgebungen durchgeführt. Die Modelle DeepSeek-R1, DeepSeek-V3-0324, Llama-3.1-8b-instruct wurden per API auf der Plattform openrouter.ai getestet. Dafür wurde der Baseprompt sowie die Frage zu einem Prompt zusammengefügt und der generierte Wissensgraph als Dateianhang hinzugefügt. Je nachdem, ob die Interaktion der LLMs mithilfe der OWL-Datei oder des Tripeltextes evaluiert wird, wurde die OWL-Wissensgraphdatei oder die TXT-Tripeltextdatei angehangen. Das Qwen-2.5-3B Modell hingegen wurde lokal auf einem Windows-Laptop mittels Ollama (Version 0.9.1) ausgeführt, da das Modell mit 3B Parametern sehr klein sowie open-weight ist und über openrouter.ai keine API-Ausführung angeboten wird. Der Laptop besitzt einen AMD Ryzen 7 5700U Prozessor, eine integrierte AMD Radeon(TM) Graphics GPU, einen 512 GB SSD Speicher sowie 16 GB Arbeitsspeicher. Da im Terminal des Windows-Laptops keine Dateianhänge möglich sind, wurde der Inhalt der Dateien vor jedem fertig zusammengefügten Prompt (bestehend aus Baseprompt und Frage) direkt am Beginn des Prompts ergänzt. Dafür wurde entweder der komplette OWL-Code der Wissensgraph-Datei oder der gesamte Text der TXT-Tripeltextdatei eingefügt.

Nach dem Ausführen der LLMs werden die Antworten auf Richtigkeit, Vollständigkeit und besondere Auffälligkeiten geprüft. Die Ergebnisse werden in einer tabellarischen Übersicht (Excel-Tabelle) protokolliert. Durch identische Prompts, den Verzicht auf Nachbearbeitung und gleichbleibende Evaluationsmatriken, wird die Fairness der Evaluation sichergestellt. Im Laufe der Evaluation wurden somit 960 Antworten der LLMs überprüft, da es 4 Fragen für jede Wissensgraphmodellierung (10) für jede Maschine (3) gab und diese sowohl für die OWL-Dateien als auch für die TXT-Dateien durchgeführt wurden (2). Im praktischen Verlauf kam es außerdem vereinzelt zu Verzögerungen durch API-Limits und längeren Antwortzeiten bei komplexen Anfragen, insbesondere bei dem Reasoning-Modell DeepSeek-R1 (bis zu 200 Sekunden). Diese Antwortzeiten wurden im Rahmen der besonderen Auffälligkeiten dokumentiert und flossen gezielt am Ende der Evaluation bei der Berechnung des Score-Werts negativ ein.

4.3 Analyse und zentrale Erkenntnisse

In diesem Kapitel werden die Ergebnisse der Evaluation analysiert, um zentrale Erkenntnisse über die Wechselwirkung von Wissensgraph-Modellierungen, Datenformaten und LLM-Leistung abzuleiten. Zur Verbildlichung der Analyse werden die Ergebnisse mittels Diagramme dargestellt.

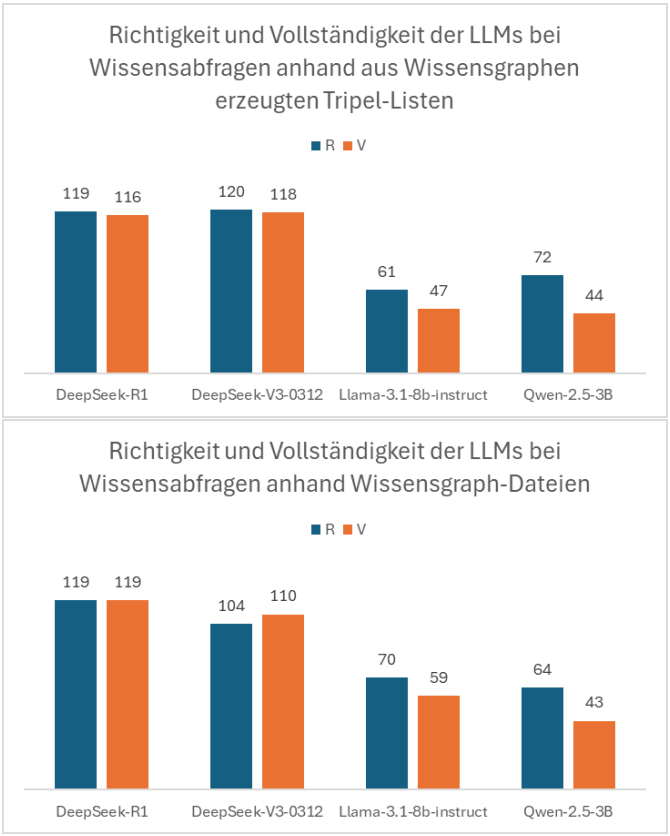

Abbildung 4.1: Evaluationsergebnisse der LLMs

Bei beiden Balkendiagrammen sind sowohl die Richtigkeit (R) als auch die Vollständigkeit (V) für jedes evaluierte LLM eingezeichnet. Die Beobachtung zeigt, dass sowohl DeepSeek-R1 als auch DeepSeek-V3-0312 durchgehend sehr gute Ergebnisse liefern. Die Ergebnisse zeigen, dass Llama-3.1-8b-instruct und Qwen-2.5-3B deutlich hinter den anderen beiden zurückbleiben, was auf die Limitierungen kleinerer Modelle hindeutet. Beim Wechsel von Tripeltext zu OWL-Dateien zeigt DeepSeek-V3-0312 einen leichten Leistungsabfall, während das Reasoning-Modell DeepSeek-R1 stabil bleibt. Qwen zeigt generell Schwächen, vor allem in Hinsicht auf Vollständigkeit.

DeepSeek-R1 weist in allen Tests eine hohe Stabilität auf. Es kann sowohl mit impliziten als auch mit expliziten Relationen umgehen und auch bei komplexen Multi-Image-Graphen gute Ergebnisse liefern. Die besten Ergebnisse zeigen sich aber bei dem Summary-Graphen (98 Punkte), dicht gefolgt von dem Single-Cam-Graphen (97 Punkte), da hier die einfache Struktur eine effektive Verarbeitung ermöglicht. Ein kleiner Leistungsabfall zeigt sich bei den Multi-Image-Graphen mit impliziten Relationen und ohne Koordinaten (89 Punkte), was darauf hindeutet, dass dem Modell möglicherweise wichtige Informationen zum Zusammenführen der Perspektiven

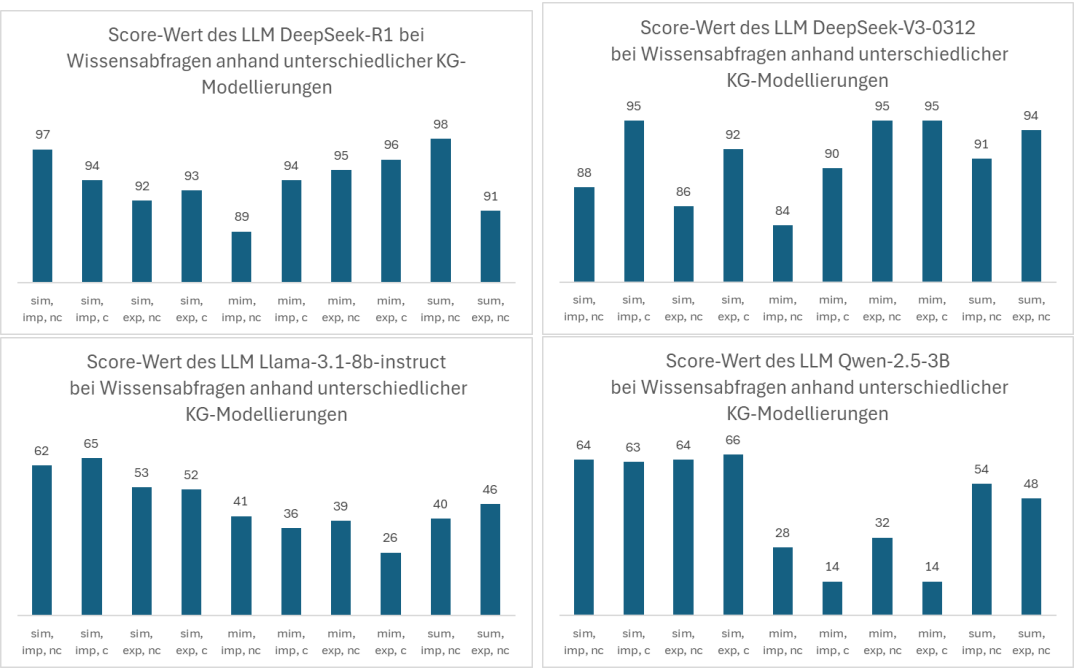

Abbildung 4.2: Evaluationsergebnisse der Wissensgraph-Modellierungen je LLM

fehlen. Auch bei OWL-Dateien bleibt die Leistung des LLMs stabil.

DeepSeek-V3-0312 erreicht fast gleichstarke Ergebnisse wie DeepSeek-R1, aber liegt bei den meisten Modellierungen leicht darunter. Besonders bemerkenswert ist, dass die Koordinaten bei diesem Modell die Leistung immer verbessert oder gleich hält. Auch mit mehreren Kameraperspektiven kommt es gut zurecht, da es sowohl bei Single-Image, Multi-Image und Summary-Graphen relativ ähnliche Punktzahlen von 95, 95 und 94 erzielt. Allerdings ist ein starker Performanceeinbruch bei OWL-Dateien erkennbar.

Llama-3.1-8b-instruct schneidet insgesamt deutlich schwächer als die beiden DeepSeek-Modelle ab. Bei einfachen, impliziten Single-Cam-Graphen erreicht es noch durchschnittliche Werte (62, 65), doch bei komplexeren Multi-Image-Graphen sinkt die Antwortqualität deutlich ab. Die schlechtesten Werte wurden bei expliziten Multi-Image-Graphen mit Koordinaten erzielt (26 Punkte). Bei Summary-Graphen erreicht das Modell wieder bessere Ergebnisse. Dies deutet darauf hin, dass es bei mehrschichtigen Graphstrukturen mit hoher Informationsdichte erhebliche Verarbeitungsprobleme hat.

Qwen-2.5-3B liefert ein deutliches, aber gemischtes Bild. Während es bei Single-Image-Graphen solide Werte erreicht (63 - 66 Punkte), bricht die Leistung bei Multi-Image-Graphen stark ein. Vor allem bei Multi-Image-Graphen mit Koordinaten, fallen die Werte enorm ab (bis auf 14 Punkte). Das deutet darauf hin, dass das Modell angesichts seiner kleinen Parameterzahl und geringen Kontextgröße mit komplexen, informationsreiche Graphen große Schwierigkeiten hat. Die Summary-Graphen reduzieren die Komplexität, wodurch das Modell Werte bis zu 54 Punkte erreichen kann.

5 Diskussion

5.1 Interpretation der Evaluationsergebnisse

In diesem Kapitel werden die vorher analysierten Ergebnisse hinsichtlich ihrer inhaltlichen Bedeutung gedeutet, um Zusammenhänge zwischen Modelltyp, Graphstruktur und Datenformat zu identifizieren.

Einfluss des Modelltyps Während Modelle wie DeepSeek-R1 und DeepSeek-V3-0312 mit ihrer Parameterzahl von 671 Milliarden eine hohe Antwortqualität erzielen, so ist ein deutlicher Abfall der Leistung bei den Modellen mit weniger Parametern erkennbar (vgl. Abb. 4.3). Im Gegensatz dazu lassen sich nur wenige Unterschiede zwischen Reasoning-Modellen und den generischen LLMs feststellen. Bei Tripeltexten schneiden die Modelle DeepSeek-R1 und DeepSeek-V3-0312 sehr ähnlich ab, bei den OWL-Dateien zeigte das Reasoning-Modell dagegen eine klar bessere Performance als DeepSeek-V3-0312 (vgl. Abb. 4.3). Die Hostingform des Modells beeinflusst die Qualität der Antworten hingegen nicht bemerkbar. Für den praktischen Einsatz der Modelle in einem KI-Wartungsassistenten, eignen sich, um die beste Qualität der Antworten zu erreichen, Modelle mit möglichst großer Parameterzahl (bei API-Hosting deutlich größer als bei lokalem Hosting). Reasoning-Modelle hingegen würden nicht für den praktischen Einsatz geeignet sein, da der minimale Leistungsbonus nicht die längeren Antwortzeiten rechtfertigt (bei manchen Antworten bis zu 200 Sekunden). Denn für ein Echtzeitsystem wie dem KI-Wartungsassistenten ist die Antwortzeit ein wichtiger Faktor. Diese Unterschiede zwischen den Modellen legen nahe, dass die Auswahl der LLMs eine zentrale Rolle für die Antwortqualität spielt. Um jedoch einen fundierten Schluss ziehen zu können, sollte auch der Einfluss des verwendeten Datenformats auf die Leistungsfähigkeit betrachtet werden.

Einfluss des Datenformats Bei den Datenformaten zeigt sich, dass die meisten LLMs mit Tripeltext besser zurechtkommen, als mit den Wissensgraphdateien (vgl. Abb. 4.3). Dies könnte daran liegen, dass die meisten LLMs mehr auf Text und weniger auf XML-Formaten (beispielsweise OWL-Dateien) trainiert wurden. Auch die Kontextlänge der Modelle könnte bei sehr großen Wissensgraphen zu Problemen führen, da durch die XML-Struktur der OWL-Wissensgraphdateien ein großer Overhead entsteht, welcher keine relevanten Daten enthält. Es ist anzumerken, dass dieser Schluss auch auf die Struktur der in dieser Arbeit verwendeten Dateien beeinflusst sein könnte und nicht zwingend in allen OWL-Implementierungen in gleicher Form und Stärke auftritt. Allerdings lässt sich auch erkennen, das Reasoning-Modelle wie DeepSeek-R1 ziemlich robust sind, und somit auch gute Ergebnisse mit Wissensgraphdateien erzielen können (vgl. Abb. 4.3). Für den KI-Wartungsassistenten sollten somit die Tripeltexte als Datenformate gegenüber den Wissensgraphdateien bevorzugt werden, um die höchste Antwortqualität mit den meisten LLMs zu erreichen. Die beobachteten Unterschiede zwischen Tripeltext und OWL-Dateien verdeutlichen, dass das Datenformat die Verarbeitungsqualität des Wissensgraphen durch LLMs wesentlich beeinflusst. Neben der Struktur der Daten, spielt allerdings auch die Struktur der Modellierung des Graphen eine erhebliche Rolle.

Einfluss der Graphmodellierung Bei dem Umgang der LLMs mittels der verschiedenen Wissensgraph-Modellierungen zeigen sich nicht so deutliche Muster, sondern eher LLM-spezifische-Auffälligkeiten. So fällt auf, dass gerade Modelle mit kleiner Parameteranzahl mit zunehmender Komplexität der Wissensgraphen (beispielsweise Multi-Image-Graphen) deutlich an Leistung verlieren (vgl. Abb. 4.3). Das wird deutlich erkennbar an beispielsweise dem Qwen-2.5-3B Modell (3 Milliarden Parameter), da dieses einen erheblichen Leistungsverlust verzeichnet, wenn zu den Multi-Image-Graph auch noch die Koordinaten der einzelnen Individuen hinzugefügt werden (vgl. Abb. 4.3). Insbesondere bei Llama-3.1-8b-instruct (8 Milliarden Parameter) zeigt sich, dass die implizite Modellierung signifikant bessere Ergebnisse erzielt als die Explizite. Besonders bemerkenswert ist, dass beide dieser Modelle bei der Single-Image-Modellierung durch das Hinzufügen der Koordinaten an Leistung gewinnen, allerdings bei der Multi-Image-Modellierung durch das Hinzufügen dieser an Leistung verlieren (vgl. Abb. 4.3). Die Ergebnisse legen also nahe, dass diese zusätzlichen Informationen nur bis zu einem Schwellenwert an Komplexität des Wissensgraphs hilfreich für die Verbesserung der Qualität der Antworten sind. Koordinaten sollten bei diesen Modellen nur dann hinzugefügt werden, wenn die Anzahl an Individuen des Wissensgraphen nicht einen gewissen Schwellenpunkt (bei Graphen mit unter 12 Individuen) übertritt. Der Schwellenwert von 12 Individuen basiert auf den kleinsten Wissensgraphen in der Evaluation, bei denen ein Leistungsabfall beim Hinzufügen von Koordinaten beobachtet wurden. In einem industriellen Wartungsszenario mit mehreren Kameras, könnte auch die Verwendung des Summary-Graphen die Fehlerrate kleinerer Modelle deutlich senken, da diese die Komplexität erheblich reduzieren.
Bei den Modellen mit großer Parameteranzahl hingegen lässt sich eine konstantere Qualität der Antworten erkennen. Vor allem bei DeepSeek-V3-0312 lässt sich allerdings eine deutliche Verbesserung der Antworten erkennen, wenn der Wissensgraph Koordinaten enthält (vgl. Abb. 4.3). Die Ergebnisse der impliziten und expliziten Relationsmodellierung sind bei diesen beiden Modellen kaum von Relevanz, da diese gute Ergebnisse mit beiden Modellierungsarten erzeugen. Auch der Summary-Graph verbessert die Leistung der Multi-Image-Graph nur sehr geringfügig oder gar nicht, was die Robustheit dieser Modelle zeigt (vgl. Abb. 4.3). Für den praktischen Einsatz empfehlen sich somit alle Modellierungen, die Koordinaten enthalten. Da viele hohe Werte über die implizite Modellierung erreicht wurden, ist diese im Einsatz für einen KI-Wartungsassistenten voraussichtlich von zentraler Bedeutung, wobei auch die explizite Modellierung praktikabel sein könnte.

Zusammenfassend lässt sich somit sagen, dass die Parameteranzahl der Modelle, das Datenformat sowie die Komplexität des Wissensgraphen und dessen Modellierungsart, die Qualität der Antworten erheblich beeinflussen. Diese Erkenntnisse bilden die Grundlage für die Beantwortung der Forschungsfragen in 5.4.

5.2 Methodische Herausforderungen und Limitationen

Um die Ergebnisse in einem reflektierten Kontext zu verstehen, ist jedoch die Kenntnisnahme der methodischen Herausforderungen und der Limitationen wichtig, welche die Vorbereitung, Entwicklung und Evaluation der automatisierten Wissensgraph-Erstellung beeinflusst haben.

Methodische Herausforderungen Der Datensatz, an welchem die Automatische Wissensgraph-Erstellung getestet und evaluiert wurde, bietet nur eine begrenzte Anzahl an annotierten Bildern (neun Bilder von 3 Maschinen, alle 3 Industriemaschinen). Somit bietet sich die Gefahr für zu

wenig Varianz der Datensätze. Auch die manuelle Erstellung der Bounding-Box-CSV-Dateien ist prinzipiell fehleranfällig, aber wurde durch genaue Kontrolle weitgehend minimiert. Die automatische Ermittlung von Relationen zwischen Individuen auf Basis zweidimensionaler geometrischer Daten in einem dreidimensionalen Raum stellt ebenfalls eine erhebliche Herausforderung dar. Beispielsweise ist die Entscheidung, ob ein Individuum unter einem anderen ist, je nach Interpretation dieser "below"Relation nicht immer eindeutig (beispiesweise Individuum A liegt sehr weit links, aber auch unter einem Individuum B). Die genauen Grenzen dieser Relationen sind oft von subjektiven Einschätzungen beeinflusst. Auch die Verschiebung der Individuen durch die Perspektivenänderung kann zu zusätzlichen Herausforderungen führen. So kann sich die Reihenfolge der Individuen ändern sowie die X- oder Y-Dimension gestaucht werden, wodurch sich Relationen verändern können. Das kann dazu führen, dass die Erkennung der gleichen erkannten Objekte im Multi-Image-Graphen zu Fehlern führt. Viele der getesteten Algorithmen zeigen geringe Robustheit beispielsweise bei dieser Erkennung, wenn viele Hebel sich in der Nähe befinden. Dabei kann es dazu kommen, dass manche Individuen die OWL-native same_as Relation mit zwei unterschiedlichen Individuen einer anderen Kamera erhalten, wenn diese sehr ähnliche Positionen und Nachbarrelationen aufweisen. Auch in der Evaluation gab es methodische Herausforderungen, so wurden lediglich vier verschiedene Modelle verglichen. Diese wurden zwar möglichst verschieden gewählt, um eine breite Evaluation zu ermöglichen, allerdings bietet dies keine vollständige Repräsentation der Vielfalt aktueller LLM-Architekturen. Auch die einzelnen Antwortstile der Modelle erschwerten eine einheitliche, objektive Bewertung. Die langen Antwortzeiten des Reasoning-Modells DeepSeek-R1 erschwerten ebenfalls die Testläufe der Evaluation.

Limitationen Da der Datensatz der klein und homogen ist, sind die Ergebnisse nicht ohne Weiteres auf andere Maschinen, Objektklassen oder andere Einsatzszenarien übertragbar. Auch die aus den Ergebnissen der Evaluation gezogenen Schlüsse in 5.1 können bei anderen gewählten Modellen und Trainingsdaten abweichen und sollten nicht als allgemeingültig aufgefasst werden. Die vollständig objektive Bewertung der Antworten ist ebenfalls nicht möglich und kann subjektive Einflüsse enthalten, die die Antwortbewertung verfälschen können. Ebenso können Kontextlängenbeschränkungen der LLMs im praktischen Einsatz die maximal nutzbare Graphgröße beschränken. Die Hostingform dieser LLMs wurde zwar ebenfalls getestet, allerdings nicht unter Aspekten wie Netzwerklatenz oder Kosten pro generiertem Token optimiert. Dadurch könnten im praktischen Einsatz via API weitere, nicht berücksichtigte Herausforderungen auftreten.

Der Fokus der Datenbasis und der Evaluation lag somit auf dem Szenario des Einsatzes des LLMs als KI-Wartungsassistent. Die gezogenen Schlüsse können nicht auf andere Domänen übertragen werden und sollten in diesen überprüft werden. Die Erkenntnisse dieser Arbeit können allerdings trotzdem als richtungsweisend betrachtet werden, und können so als Basis für zukünftige Forschung in anderen Domänen oder anderen Arten, Wissensgraphen aus Bildern abzuleiten dienen.

5.3 Einordnung in den aktuellen Forschungsstand

Um die Relevanz der Ergebnisse dieser Arbeit einschätzen zu können, werden sie in den aktuellen Forschungsstand eingeordnet. Während die Wissensgraph-Erstellung aus Texten und strukturierten Daten bereits länger ein Thema der Forschung ist, sind Ansätze aus der Bildverarbeitung deutlich seltener vertreten [3, 8]. Der Forschungsstand zur automatisierten Wissensgraph-

Erstellung aus annotierten Bilddateien ist bislang sehr begrenzt. Nur wenige Arbeiten befassen sich mit möglichen methodischen Ansätzen zur Umsetzung solcher Systeme. Eine Arbeit aus dem Jahre 2024 beschäftigte sich bereits mit diesem Thema, allerdings wurden die Wissensgraphen nicht direkt aus der Ausgabe eines Objekterkennungsmodells modelliert, sondern erforderten eine nachträgliche Datenaufbereitung. In diesem Paper wurden mittels Perspektivtransformation aus der Ausgabe und den Grundrissen der Gebäude, die Koordinaten absolut in Abhängigkeit des Grundrisses gesetzt [22]. Dadurch war es nicht nötig, positionsabhängige Relationen wie above, below, usw. herzuleiten. In diesem Ansatz war es somit nicht notwendig, mehrere Modellierungsansätze zu untersuchen, da absolute Koordinaten ausreichen, um die räumlichen Positionen eindeutig zu bestimmen. Doch im Gegensatz zu diesem Ansatz, sind nicht in jedem Einsatzgebiet ein Grundriss oder anderweitige Scans der Betriebsumgebung vorhanden. Zudem können Betriebsumgebung variabel und dynamisch sein, was die Verwendung eines solchen Ansatz wesentlich erschwert. Durch diese Limitierungen, sind die Ergebnisse dieser Arbeit von Relevanz, da sie die automatisierte Wissensgrapherstellung auf Basis eines universell einsetzbaren Szenarios betrachtet. Diese Arbeit leistet damit nicht nur einen Beitrag zur Automatisierung, sondern auch einen Vergleich der verschiedenen Modellierungsansätze im Kontext von LLMs - ein Aspekt, bislang kaum untersucht wurde.

5.4 Beantwortung der Forschungsfragen

In diesem Kapitel werden die, in 1.2 definierten Forschungsfragen, beantwortet. Dafür werden die Ergebnisse der Evaluation sowie die Reflexion der Methodik betrachtet.

Kann die Erstellung eines Wissensgraphen aus annotierten Bilddateien automatisiert werden? Wie bereits in 3 beschrieben, ist es möglich aus annotierten Bilddateien OWL-konforme Wissensgraphen zu generieren. Dafür können notwendige Relationen algorithmisch aus den Bounding-Box-Koordinaten abgeleitet werden. Allerdings sind alle generierten Wissensgraphen abhängig von der Perspektive der Kamera und somit nicht allgemeingültig. Aus diesen Ergebnissen lässt sich folgern, dass die Erstellung eines Wissensgraphen aus annotierten Bilddateien grundsätzlich automatisiert werden kann, allerdings bieten sich noch einige methodische Herausforderungen in Bezug auf Robustheit und Skalierbarkeit, die weiteren Forschungsbedarf eröffnen.

Welche Wissensgraphmodellierung eignet sich am besten für den Einsatz in einem LLM-basierten KI-Wartungsassistenten? Die Antwort dieser Frage ist, wie bereits in 4 erläutert, nicht eindeutig sondern hängt von mehreren Faktoren ab. Je nach verwendetem LLM und der Anzahl der zur Verfügung stehenden Kameras, bieten sich unterschiedliche Ansätze zur Modellierung der Wissensgraphen an. Bei der Generierung der Graphen mit lediglich einer Kamera, bieten sich implizite Modellierungen mit enthaltenen Koordinaten an. Kleinere Modelle konnten die Koordinaten zwar nur begrenzt in ihre Entscheidungen einbeziehen (hatten einen einen geringen Einfluss auf die Antwortqualität), doch insbesondere große Modelle konnten mit Koordinaten eine höhere Antwortqualität erzielen als ohne. Bei mehreren Kameras bietet sich eher der Einsatz der Summary-Graph-Modellierung an, was die Komplexität des Graphens stark reduzieren kann. Gerade kleinere LLMs wiesen bei komplexen Graphen Probleme bei der Verarbeitung auf. Zusammenfassend sollte somit bei einer Kamera die implizite Modellierung mit Koordinaten und bei mehreren Kameras der implizite Summary-Graph gewählt werden, da diese die konstanteste Performance erzielte. Dem LLM sollten die generierten Wissensgraphen als Tripeltext anstatt OWL-Dateien als Input mitgegeben werden, da diese Entscheidung die Leistung

in den meisten Fällen verbesserte. Diese Schlüsse müssten allerdings breiter getestet werden und deuten lediglich auf eine Tendenz für weitere Forschung oder den praktischen Betrieb hin.

Welche Modelle sind für den Einsatz mit Wissensgraphen geeignet? Diese Forschungsfrage kann leider nicht eindeutig beantwortet werden. Während dieser Arbeit wurden mehrere LLMs unterschiedlicher Architektur getestet und dabei ein positiver Zusammenhang zwischen der Anzahl der Parameter und dessen Leistung nachgewiesen. Allerdings gibt es in der Praxis eine Vielzahl von Modellen verschiedenster Architektur, welche die präzise Beantwortung dieser Frage erschweren. Reasoning-Modelle erwiesen sich allerdings aufgrund der möglichen langen Antwortzeiten als ungeeignet für den praktischen Einsatz in einer Echtzeitanwendung . In der Praxis sollte somit, je nach Implementierung, ein möglichst leistungsstarkes Modell mit hoher Parametrisierung gewählt werden, welches dennoch praktisch nutzbare Antwortzeiten bietet.

5.5 Ausblick: Weiterentwicklung und industrielle Anwendung

Die in dieser Arbeit erarbeiteten Methoden und Ergebnisse bieten eine Grundlage für potentielle Weiterentwicklung und Einsatz in industriellen Anwendungen.

Methodische Weiterentwicklung Die vorgestellte Methodik kann auf unterschiedlichen Ebenen verbessert werden. So könnten durch den Einsatz verbesserter und komplexerer Algorithmen zur Relationserkennung die Robustheit der Wissensgraphgenerierung verbessert werden. Ebenso wäre ein überarbeiteter Prozess denkbar, der die gleichen Bauteile in unterschiedlichen Perspektiven erkennt, die Skalierbarkeit und Robustheit erhöhen. Ein möglicher Ansatz dafür wäre beispielsweise, anhand der Bilddatei markante Eigenschaften jedes Bauteils zu erkennen und mittels auf Feature-Matching basierten Verfahren diese auszuwerten. Ein weiterer möglicher Ansatz bestünde in einer 3D-Beschreibung der Maschinen im Graphen beispielsweise mittels vier Kameras aus jeder Richtung.

Verbesserte Integration mit LLMs Auch die Integration mit LLM könnte auf unterschiedlichen Ebenen verbessert und angepasst werden. Dabei würde Prompt-Engineering ohne viel Änderung des Verfahrens erlauben, die Antworten der LLMs deutlich verbessern. Ein verbesserter, leicht abgeänderter Beispielprompt findet sich in Anhang A.1. Auch eine einheitliche und verbesserte OWL zu Tripeltext Transformation würde es ermöglichen, die Robustheit der Antworten zu erhöhen. So wäre es möglich anstatt jedes Tripel des Wissensgraphen in eine Textdatei zu schreiben, mittels Vorverarbeitung der Tripel die Leistung von vor allem kleineren LLMs deutlich verbessert werden. Beispielsweise könnte diese Vorverarbeitung die für die Beantwortung einer Frage irrelevanten Tripel, aus der Tripelliste entfernen um die Komplexität des Graphen zu reduzieren.

Mögliche industrielle Anwendung Der native Einsatz des in dieser Arbeit entwickelten Wissensgraphgenerators liegt in dem System eines KI-Wartungsassistenten. Dabei wird der entwickelte Prozess verwendet, um Wissensgraphen aus den Ausgaben einer Multi-Objekterkennung zu erstellen. Dies ermöglicht es dem Wartungsassistenten, Fragen zu visuellen Informationen zu beantworten und so dem Benutzer mehr praktische Hilfe anzubieten. Bei weiterer Entwicklung wäre es denkbar, diese generierten Wissensgraphen sogar mit Faktenwissen zu den einzelnen Bauteilen angereichert werden könnten. Darüber hinaus könnte der Prozess auch in anderen

Domänen eingesetzt werden, wie beispielsweise in der Medizin oder dem Bauwesen, solange es sich um ein LLM-basiertes System mit visuellen Informationen handelt. Für einen industriellen Einsatz müssten neben der technischen Umsetzung auch praktische Faktoren wie Datenverfügbarkeit und die Antwortzeit der LLMs abgewogen werden.

Wie diese Arbeit gezeigt hat, ist die automatische Erstellung von Wissensgraphen prinzipiell umsetzbar. Allerdings gibt es dafür noch einige methodische und technische Forschungsgebiete, die für eine möglichst effektive Nutzung noch tiefer untersucht werden sollten. Trotz dieser verbleibenden Hürden zeigt sich ein erhebliches Potenzial für die industrielle Anwendung in den verschiedensten Domänen, das durch weitere Forschung schrittweise erschlossen werden kann.

6 Literaturverzeichnis

[1] Heiner Ludwig, Thorsten Schmidt und Mathias Kühn. "An ontology-based retrieval augmented generation procedure for a voice-controlled maintenance assistant". In: *Computers in Industry* 169 (Aug. 2025), S. 104289. ISSN: 0166-3615. DOI: 10.1016/j.compind.2025.104289. URL: http://dx.doi.org/10.1016/j.compind.2025.104289.

[2] R. E. Hoyt u. a. "Evaluating Large Reasoning Model Performance on Complex Medical Scenarios In The MMLU-Pro Benchmark". In: (Apr. 2025). DOI: 10.1101/2025.04.07.25325385. URL: http://dx.doi.org/10.1101/2025.04.07.25325385.

[3] Ciyuan Peng u. a. "Knowledge Graphs: Opportunities and Challenges". In: *Artificial Intelligence Review* 56.11 (Apr. 2023), S. 13071–13102. ISSN: 1573-7462. DOI: 10.1007/s10462-023-10465-9.

[4] Sanju Tiwari, Fatima N. Al-Aswadi und Devottam Gaurav. "Recent trends in knowledge graphs: theory and practice". In: *Soft Computing* 25.13 (Apr. 2021), S. 8337–8355. ISSN: 1433-7479. DOI: 10.1007/s00500-021-05756-8. URL: http://dx.doi.org/10.1007/s00500-021-05756-8.

[5] Aidan Hogan u. a. "Knowledge Graphs". In: *ACM Computing Surveys* 54.4 (Juli 2021), S. 1–37. ISSN: 1557-7341. DOI: 10.1145/3447772.

[6] Hasan Abu-Rasheed, Christian Weber und Madjid Fathi. "Knowledge Graphs as Context Sources for LLM-Based Explanations of Learning Recommendations". In: Kos Island, Greece. Kos Island, Greece: IEEE, 2024, S. 1–5. ISBN: 979-8-3503-9403-0. DOI: 10.1109/EDUCON60312.2024.10578654.

[7] Lan Yang, Kathryn Cormican und Ming Yu. "Ontology Learning for Systems Engineering Body of Knowledge". In: *IEEE Transactions on Industrial Informatics* 17 (2 Feb. 2021), S. 1039–1047. ISSN: 1941-0050. DOI: 10.1109/TII.2020.2990953.

[8] Christopher Brewster u. a. "Issues in learning an ontology from text". In: *BMC Bioinformatics* 10.S5 (Mai 2009). ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-s5-s1.

[9] Shaoxiong Ji u. a. "A Survey on Knowledge Graphs: Representation, Acquisition, and Applications". In: *IEEE Transactions on Neural Networks and Learning Systems* 33 (2 Feb. 2022), S. 494–514. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2021.3070843.

[10] Dan Brickley und R.V. Guha. *RDF Schema 1.1 - W3C Recommendation 25 February 2014*. World Wide Web Consortium (W3C), Feb. 2014. URL: http://www.w3.org/TR/rdf-schema/.

[11] Brian McBride. "The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS". In: *Handbook on Ontologies*. Springer Berlin Heidelberg, 2004, S. 51–65. ISBN: 9783540247500. DOI: 10.1007/978-3-540-24750-0_3.

[12] Ravpreet Kaur und Sarbjeet Singh. "A comprehensive review of object detection with deep learning". In: *Digital Signal Processing* 132 (Jan. 2023), S. 103812. ISSN: 1051-2004. DOI: 10.1016/j.dsp.2022.103812. URL: http://dx.doi.org/10.1016/j.dsp.2022.103812.

[13] *Computer Vision. A Reference Guide*. 2nd ed. 2021. Cham: Imprint: Springer, 2021. 1568405 S. ISBN: 9783030634162.

[14] Ali Borji u. a. "Salient object detection: A survey". In: *Computational Visual Media* 5.2 (Juni 2019), S. 117–150. ISSN: 2096-0662. DOI: 10.1007/s41095-019-0149-9.

[15] Christian Zinke-Wehlmann und Julia Friedrich, Hrsg. *First Working Conference on Artificial Intelligence Development for a Resilient and Sustainable Tomorrow. AI Tomorrow 2023*. Informatik Aktuell Series. 4 Soft and Missing Spots of Human-Centered AI Implementation. Wiesbaden, Germany: Springer Vieweg, 2024. 1158 S. ISBN: 9783658437053.

[16] Murray Shanahan. "Talking about Large Language Models". In: *Communications of the ACM* 67.2 (Jan. 2024), S. 68–79. ISSN: 1557-7317. DOI: 10.1145/3624724. URL: http://dx.doi.org/10.1145/3624724.

[17] Haiyan Zhao u. a. "Explainability for Large Language Models: A Survey". In: *ACM Transactions on Intelligent Systems and Technology* 15.2 (Feb. 2024), S. 1–38. ISSN: 2157-6912. DOI: 10.1145/3639372. URL: http://dx.doi.org/10.1145/3639372.

[18] Yunpeng Huang u. a. *Advancing Transformer Architecture in Long-Context Large Language Models: A Comprehensive Survey*. 2023. DOI: 10.48550/ARXIV.2311.12351. URL: https://arxiv.org/abs/2311.12351.

[19] Thomas Heston und Charya Khun. "Prompt Engineering in Medical Education". In: *International Medical Education* 2.3 (Aug. 2023), S. 198–205. ISSN: 2813-141X. DOI: 10.3390/ime2030019. URL: http://dx.doi.org/10.3390/ime2030019.

[20] Yubo Wang u. a. "MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark". In: *Advances in Neural Information Processing Systems*. Hrsg. von A. Globerson u. a. Bd. 37. Curran Associates, Inc., 2024, S. 95266–95290. URL: https://proceedings.neurips.cc/paper_files/paper/2024/file/ad236edc564f3e3156e1b2feafb99a24-Paper-Datasets_and_Benchmarks_Track.pdf.

[21] Stephan Mäs u. a. "Generic schema descriptions for comma-separated values files of environmental data". In: *The 21th AGILE International Conference on Geographic Information Science*. 2018.

[22] Fabian Pfitzner, Alexander Braun und André Borrmann. "From data to knowledge: Construction process analysis through continuous image capturing, object detection, and knowledge graph creation". In: *Automation in Construction* 164 (Aug. 2024), S. 105451. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2024.105451. URL: http://dx.doi.org/10.1016/j.autcon.2024.105451.

A Appendix

A.1 Verbesserter Prompt

You are a maintenance assistant who helps workers maintain machines. I will provide you with detailed machine information and then ask you specific maintenance questions.

When answering:
- Give short, precise, and practical answers.
- Use all relevant information, but decide yourself which parts are important or helpful to the worker.
- Do NOT mention where the answer comes from.
- If coordinates are included, treat them as relative (top-left to bottom-right); use them only to understand spatial relationships but don't explain them to the worker.
- If multiple camera perspectives are given, merge them mentally into a complete, unified picture; generalize findings instead of listing per camera.
- Focus only on what helps the worker maintain or troubleshoot the machine.

Ready to assist.

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Dresden, den 23. August 2025