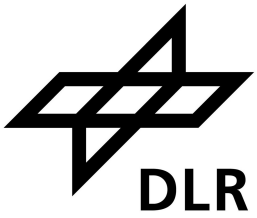

Bachelorstudiengang Wirtschaftsinformatik

Bachelorarbeit

Konzeption und Implementierung eines interaktiven Webtools zur Exploration und Kuration von sicherheitsrelevantem Wissen für hoch-automatisierte Systeme

vorgelegt von

Lars Streekmann

Betreuender Gutachter: Prof. Dr. Martin Fränzle
Zweiter Gutachter: Dr. rer. nat. Christian Neurohr

Oldenburg, den 04. November 2025

Zusammenfassung

Die HazardDB enthält 299 vernetzte Kritikalitätsphänomene für die Absicherung hochautomatisierter Fahrsysteme. Ihre Nutzung erfordert jedoch zeitaufwendige manuelle Durchsicht oder generische Tools ohne domänenspezifische Unterstützung. Die Kuration neuer Phänomene ist fehleranfällig und fehlende Anreize gefährden die langfristige Pflege der Wissensbasis.

Diese Arbeit entwickelt ein interaktives Webtool zur intuitiven Exploration und effizienten Pflege der HazardDB. Die Lösung basiert auf einer Drei-Schichten-Architektur mit RDF Triple Store, LLM-gestützter Kurations-Assistenz und automatisierten Qualitätsprüfungen. Das System kombiniert hierarchische Navigation, interaktive Graph-Visualisierung und intelligente Vorschläge für Beschreibungen, Tags und Relationen.

Die szenariobasierte Evaluation zeigt: Die Exploration ist intuitiv, die Kuration wird durch LLM-Assistenz erheblich beschleunigt und automatisierte Health Checks sichern die Datenqualität.

Abstract

HazardDB contains 299 interconnected criticality phenomena for safety assurance of highly automated driving systems. However, its use requires time-consuming manual review or generic tools without domain-specific support. Curating new phenomena is error-prone, and lacking incentives threaten long-term maintenance.

This thesis develops an interactive web tool for intuitive exploration and efficient maintenance of HazardDB. The solution is based on a three-tier architecture with RDF triple store, LLM-assisted curation, and automated quality checks. The system combines hierarchical navigation, interactive graph visualization, and intelligent suggestions for descriptions, tags, and relations.

Scenario-based evaluation shows: exploration is intuitive, curation is significantly accelerated by LLM assistance, and automated health checks ensure data quality.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

ADS	Automated Driving System
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LLM	Large Language Model
OWL	Web Ontology Language
RAG	Retrieval-Augmented Generation
RDF	Resource Description Framework
RDFS	RDF Schema
REST	Representational State Transfer
RLHF	Reinforcement Learning from Human Feedback
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VVM	Verification & Validation Methods
W3C	World Wide Web Consortium

1. Einleitung

1.1. Motivation und Problemstellung

Bei der Erstellung einer Sicherheitsargumentation für hochautomatisierte Fahrsysteme steht man vor der Aufgabe, systematisch zu identifizieren, welche Kritikalitätsphänomene zu kritischen Verkehrssituationen führen können [1]. Die HazardDB ist eine Wissensbasis mit 299 vernetzten Kritikalitätsphänomenen, die entwickelt wurde, um solches sicherheitsrelevante Wissen strukturiert und wiederverwendbar bereitzustellen. Die Nutzung dieser Wissensbasis erfordert jedoch entweder eine zeitaufwendige manuelle Durchsicht oder den Einsatz generischer Graph-Visualisierungstools, die nicht auf die spezifischen Anforderungen der Domäne zugeschnitten sind. Eine intuitive, schrittweise Exploration fehlt. Das Problem ist nicht die Wissensbasis selbst, sondern die fehlenden Werkzeuge für deren effektive Nutzung.

1.1.1. Domänenkontext: Kritikalitätsanalyse für automatisierte Fahrsysteme

Die Absicherung hochautomatisierter Fahrsysteme (Automated Driving System (ADS)) erfordert die systematische Analyse sicherheitsrelevanter Einflussfaktoren. Im Rahmen des Verification & Validation Methods (VVM)-Projekts wurde hierfür eine methodische Kritikalitätsanalyse entwickelt [2, 3]. Diese identifiziert und systematisiert sogenannte *Kritikalitätsphänomene* (Criticality Phenomena) – Faktoren wie Umweltbedingungen, Sensorausfälle oder Verhaltensanomalien anderer Verkehrsteilnehmer – sowie deren kausale und hierarchische Zusammenhänge.

Kritikalitätsphänomene sind nicht isoliert zu betrachten, sondern stehen in komplexen Beziehungen zueinander. Ein abstraktes Kritikalitätsphänomen wie „widrige Umweltbedingungen“ konkretisiert sich in „Regen“, was wiederum zu „Starkregen“ oder „Hagel“ spezifiziert werden kann. Gleichzeitig können kausale Zusammenhänge existieren: „Hagel“ kann zu „Beschädigte Sensorabdeckung“ führen, was die Umgebungswahrnehmung beeinträchtigt [4]. Die Kritikalitätsphänomene sind damit mehrdimensional vernetzt – durch hierarchische Abstraktionen, kausale Einflüsse und synergetische Effekte. Diese Komplexität macht die strukturierte Erfassung und Exploration zu einer anspruchsvollen Aufgabe.

Die HazardDB ist ein zentrales Ergebnis dieser Kritikalitätsanalyse und sammelt derzeit 299 Kritikalitätsphänomene [3, S. 27] über drei Relationstypen:

- *abstraction_ of / concretization_ of* – bidirektionale hierarchische Beziehungen zwischen abstrakten und konkreten Kritikalitätsphänomenen (als inverses Paar modelliert)
- *could_ lead_ to* – kausale Zusammenhänge, die beschreiben, welche Kritikalitätsphänomene andere auslösen können
- *synergizes_ with* – synergetische Effekte, bei denen das Zusammenwirken mehrerer Kritikalitätsphänomene die Kritikalität erhöht

Eine Wissensbasis ist jedoch nur so wertvoll wie der Zugang zu ihr. Ohne ein Werkzeug, das Exploration und Kuration gleichermaßen unterstützt, bleibt das Potenzial der HazardDB ungenutzt.

1.1.2. Drei zentrale Herausforderungen

Problem 1: Exploration ist umständlich

Die Navigation durch die Graphstruktur erfordert entweder Kenntnisse in formalen Abfragesprachen oder den Einsatz generischer Graph-Visualisierungstools, die nicht auf die spezifischen Anforderungen der Domäne zugeschnitten sind. Selbst einfache Aufgaben – wie das Identifizieren aller möglichen Ursachen für einen Sensorausfall aus dem Bereich widriger Umweltbedingungen – erfordern entweder manuelle Navigation durch Dutzende Knoten oder die Formulierung komplexer Abfragen. Eine intuitive Suche und pfadbasierte Exploration fehlt, bei der Nutzer schrittweise von abstrakten zu konkreten Kritikalitätsphänomenen navigieren und dabei relevante Kontextinformationen erhalten.

Problem 2: Kuration ist fehleranfällig und skaliert nicht

Das Hinzufügen eines neuen Kritikalitätsphänomens stellt Kuratoren vor eine aufwendige Aufgabe: Welche der bestehenden Phänomene sind relevant? Handelt es sich um eine Konkretisierung eines bestehenden Phänomens? Gibt es kausale Zusammenhänge oder synergetische Effekte mit anderen? Bereits die Suche nach ähnlichen oder verwandten Kritikalitätsphänomenen erfordert manuelle Durchsicht oder das Formulieren von Abfragen – ein Prozess, der mit wachsender Datenbasis zunehmend aufwendiger wird. Ohne geführte Eingabeprozesse, intelligente Vorschläge oder Konsistenzprüfungen ist diese Aufgabe nicht nur zeitintensiv, sondern auch fehleranfällig. Die manuelle Bearbeitung der Datenbasis bietet keine prozessuale Unterstützung und erhöht das Risiko für Inkonsistenzen oder strukturelle Fehler.

Problem 3: Nachhaltigkeit ist gefährdet

Diese Hürden in Exploration und Kuration haben direkte Konsequenzen für die langfristige Nutzung der Wissensbasis. Wenn Pflege und Nutzung zu aufwendig sind, fehlen Anreize für kontinuierliche Beiträge – die Datenbasis wird nicht mehr aktualisiert, neue Erkenntnisse fließen nicht ein und die Wissensbasis verliert an

Relevanz. Die langfristige Nutzbarkeit der HazardDB hängt damit nicht nur von der Qualität der Daten ab, sondern auch von niedrigschwelligen Werkzeugen, die sowohl Exploration als auch Kuration unterstützen und damit Anreize für nachhaltige Pflege schaffen.

1.2. Forschungsfrage und Zielsetzung

Aus der identifizierten Problemstellung ergibt sich die zentrale Forschungsfrage dieser Arbeit:

> *Wie muss ein interaktives Webtool gestaltet sein, um Nutzer (a) bei einer intuitiven, pfadbasierten Exploration der HazardDB zu leiten und (b) eine effiziente und konsistente Pflege der Wissensbasis zu gewährleisten?*

Zur Beantwortung dieser Frage verfolgt die Arbeit folgende Ziele:

1. **Anforderungsanalyse**: Systematische Erhebung funktionaler, nicht-funktionaler und technischer Anforderungen in Abstimmung mit der Projektgruppe. Die Anforderungen werden durch User Stories illustriert und dienen als Grundlage für die Konzeption und Implementierung.
2. **Architekturentwurf**: Konzeption einer modularen Drei-Schicht-Architektur mit klarer Trennung von Präsentations-, Logik- und Datenschicht. Der Entwurf berücksichtigt die Anforderungen an Erweiterbarkeit, Performance und semantische Konsistenz.
3. **Prototypische Implementierung**: Entwicklung eines funktionsfähigen Webtools mit Kernfunktionalitäten für **intuitive Navigation** (Tree-View, Graph-Visualisierung, globale Volltextsuche mit Typo-Toleranz), **geführte Dateneingabe** (Formulare mit Validierung, Combobox-Filterung und LLM-Assistenz) und **Qualitätssicherung** (Health Checks, RDF-Export).
4. **Evaluation**: Validierung der entwickelten Lösung durch szenariobasierte Evaluation anhand repräsentativer Aufgabenstellungen. Im Fokus steht die zentrale Frage: Ermöglicht das Tool intuitive Exploration und effiziente Kuration?

Diese Arbeit ist als praxisorientierte Software-Engineering-Arbeit konzipiert. Der Fokus liegt auf der Entwicklung eines funktionsfähigen Prototyps, der die identifizierten Probleme adressiert und als Grundlage für den weiteren produktiven Einsatz dienen kann. Die Arbeit dokumentiert dabei nicht nur das Endergebnis, sondern auch die getroffenen Architektur- und Implementierungsentscheidungen, sodass zukünftige Entwickler das System verstehen, erweitern und warten können.

1.3. Aufbau der Arbeit

Die vorliegende Arbeit ist wie folgt strukturiert:

Kapitel 2 (Anforderungsanalyse) beschreibt das methodische Vorgehen zur Anforderungserhebung und identifiziert Stakeholder sowie deren Bedarfe. Es werden funktionale, nicht-funktionale und technische Anforderungen an das Webtool erhoben und anhand von User Stories illustriert.

Kapitel 3 (Grundlagen und Related Work) führt in die relevanten technologischen Grundlagen ein: RDF als Datenmodell, RESTful API-Design sowie Large Language Models für Kurations-Assistenz. Zudem werden verwandte Arbeiten analysiert – Unfall- und Szenariodatenbanken, Knowledge Management Tools, spezialisierte Wissensbasen – und die Forschungslücke abgeleitet.

Kapitel 4 (Systemarchitektur) präsentiert die Gesamtarchitektur des Systems – Frontend-, Backend- und Datenarchitektur – sowie zentrale Designentscheidungen zur Technologie-Auswahl und Systemstruktur. Besonderes Augenmerk liegt auf der Evolution der Datenhaltung und der Begründung der Architekturentscheidungen.

Kapitel 5 (Implementierung) dokumentiert die technische Umsetzung der Kernfunktionalitäten: CRUD-Operationen, Graphvisualisierung, LLM-gestützte Kuration, statische Analyse (Health Checks) und RDF-Export. Der Fokus liegt auf Erkenntnissen, Herausforderungen und Implementierungsentscheidungen im Kontext eines Prototyps.

Kapitel 6 (Evaluation) validiert die entwickelte Lösung durch szenariobasierte Evaluation anhand repräsentativer Aufgabenstellungen, diskutiert die Ergebnisse hinsichtlich der Forschungsfrage und benennt Limitationen der Arbeit.

2. Anforderungsanalyse

2.1. Methodisches Vorgehen

Die Anforderungserhebung erfolgte in mehreren Schritten. In einem initialen Kick-off-Workshop mit der Projektgruppe wurden in einem strukturierten Brainstorming grundlegende Funktionalitäten für das HazardDB-Webtool diskutiert und priorisiert. Ziel war es, die zentralen Funktionalitäten zu identifizieren, die sowohl die Exploration als auch die Kuration der Wissensbasis unterstützen. Der Workshop brachte dabei Perspektiven aus verschiedenen Bereichen zusammen - von der technischen Umsetzbarkeit bis hin zu den praktischen Bedürfnissen potentieller Nutzer.

Im Anschluss an den Workshop erfolgte eine iterative Feinjustierung und Konkretisierung der Anforderungen durch wöchentliche Online-Betreuungsgespräche sowie regelmäßige Vor-Ort-Abstimmungen in Oldenburg während der gesamten Entwicklung. Die Anforderungen wurden in enger Abstimmung mit der Projektgruppe erhoben, die sowohl technische Expertise als auch domänenspezifisches Wissen einbrachte. Diese Vorgehensweise ermöglichte es, Anforderungen während der Implementierung kontinuierlich anzupassen, zu validieren und zu erweitern.

2.2. Stakeholder und Bedarfe

Die HazardDB richtet sich an unterschiedliche Nutzergruppen, die alle ein gemeinsames Interesse teilen: die **Wiederverwendbarkeit von Kritikalitätsphänomenen für Sicherheitsargumentationen**. Die Wissensbasis soll sicherheitsrelevantes Wissen strukturiert aufbereiten und in verschiedenen Kontexten nutzbar machen.

Aus funktionaler Sicht lassen sich die Stakeholder in zwei Hauptrollen einteilen:

Nutzer explorieren und verwenden die Wissensbasis für ihre jeweiligen Anwendungskontexte. Typische Nutzer sind Safety-Professionals, die Sicherheitsargumentationen erstellen, Forscher und Unfallforscher, die Relationen zwischen Kritikalitätsphänomenen analysieren und Hypothesen überprüfen, sowie Zulieferer und Hersteller (OEMs), die auf strukturiertes Sicherheitswissen für Entwicklungs- und Zulassungsprozesse angewiesen sind. Nutzer benötigen effiziente Navigation, Exploration und Exportmöglichkeiten.

Kuratoren sind für die laufende Pflege, Qualitätssicherung und Erweiterung der

Wissensbasis verantwortlich. Sie erstellen neue Kritikalitätsphänomene, pflegen Relationen und sichern die Datenqualität.

Die Grenzen zwischen diesen Rollen sind nicht starr - je nach Kontext können Personen beide Rollen einnehmen.

2.3. Use Cases und User Stories

Um die Anforderungen an das Webtool zu konkretisieren, wurden typische Anwendungsszenarien in Form von User Stories formuliert. Diese illustrieren die zentralen Nutzungskontexte und dienen als Orientierung für die Entwicklung.

US1: Exploration von Kritikalitätsphänomenen

> Als Nutzer möchte ich ausgehend von einem abstrakten Kritikalitätsphänomen (z.B. „widrige Umweltbedingungen“) schrittweise zu konkreteren Ausprägungen navigieren können, um relevante Einflussfaktoren für meine Sicherheitsargumentation zu identifizieren.

US2: Anlegen neuer Kritikalitätsphänomene mit LLM-Assistenz

> Als Kurator möchte ich neue Kritikalitätsphänomene über ein geführtes Formular anlegen können, das mich bei der Eingabe von Name, Beschreibung, Tags und Relationen unterstützt. Dabei soll ein KI-Assistent mir Vorschläge für fehlende oder unvollständige Felder generieren können, um die Vollständigkeit der Einträge zu erhöhen und den Kurations-Aufwand zu reduzieren.

US3: Dokumentation kausaler Zusammenhänge

> Als Nutzer möchte ich Kausalgraphen erstellen und verwalten können, die komplexe Wirkungsketten zwischen Kritikalitätsphänomenen und Metriken dokumentieren, um Hypothesen über kausale Zusammenhänge strukturiert zu erfassen und auf den jeweiligen Detailseiten sichtbar zu machen.

US4: Qualitätsüberwachung der Wissensbasis

> Als Kurator möchte ich einen Überblick über die Datenqualität aller Kritikalitätsphänomene erhalten, der fehlende Beschreibungen, unvollständige Übersetzungen oder fehlende Metriken-Verknüpfungen identifiziert, um gezielt die größten Qualitätslücken priorisieren und beheben zu können.

US5: Export der Wissensbasis

> Als Nutzer möchte ich die Wissensbasis exportieren können, um diese in externe Tools zu integrieren, Snapshots für Versionierung zu erstellen oder Daten mit anderen Projekten auszutauschen.

Die nachfolgenden funktionalen Anforderungen (Abschnitt 2.4) leiten sich direkt aus den User Stories ab: US1 wird durch F1–F4 adressiert (Exploration und Navigation), US2 durch F5 und F7 (geführtes Formular mit LLM-Assistenz), US3 durch F10 (Kausalgraphen), US4 durch F8 (Qualitätsüberwachung) und US5 durch den Im-/Export in F6. Die Metriken-Anbindung (F9) unterstützt sowohl US3 als auch US4.

2.4. Funktionale Anforderungen

Basierend auf den Workshop-Ergebnissen und der iterativen Abstimmung während der Entwicklung wurden folgende funktionale Anforderungen identifiziert und umgesetzt.

2.4.1. Exploration und Navigation

F1: Hierarchische Baumansicht

Persistente Tree-View zur Navigation durch die Abstraktionshierarchie der Kritikalitätsphänomene. Die Baumansicht bleibt beim Wechsel zwischen Detailseiten geöffnet.

F2: Graphvisualisierung

Interaktive Visualisierung der Nachbarschaft eines Kritikalitätsphänomens. Die Graphdarstellung zeigt alle direkten Relationen und deren Typen.

F3: Detailansicht

Zentrale Detailseite bündelt alle Informationen zu einem Kritikalitätsphänomen: Beschreibungen, Tags, ausgehende und eingehende Relationen zu anderen Kritikalitätsphänomenen, verknüpfte Metriken und referenzierende Kausalgraphen.

F4: Globale Volltextsuche

Schnelle Suche über alle Kritikalitätsphänomene für bekannte Begriffe oder explorative Suchen. Durchsuchbare Felder umfassen Namen, Beschreibungen und weitere Attribute.

2.4.2. Kuration und Datenpflege

F5: Geführtes Formular zur Erstellung von Kritikalitätsphänomenen

Das System muss ein geführtes Formular mit Validierung für das Anlegen neuer Kritikalitätsphänomene bereitstellen. Das Formular muss bei der Auswahl von Relationstypen und bestehenden Kritikalitätsphänomenen unterstützen und die Eindeutigkeit der Namen sicherstellen.

F6: Bearbeitung bestehender Kritikalitätsphänomene

Das System muss die Bearbeitung bestehender Kritikalitätsphänomene direkt auf der Detailseite ermöglichen. Zur Versionierung muss das System Snapshots der Wissensbasis exportieren und reimportieren können.

F7: Intelligente Kurations-Assistenz

Das System muss einen automatisierten Vorschlagsmechanismus bereitstellen, der Kuratoren bei der Vervollständigung von Kritikalitätsphänomenen unterstützt. Der Mechanismus muss:

- Vorschläge für *nicht ausgefüllte* Beschreibungen, Tags und Relationen generieren
- Den Kontext bestehender Kritikalitätsphänomene berücksichtigen
- Strukturierte, validierbare Vorschläge liefern
- Das Human-in-the-Loop-Prinzip einhalten (alle Vorschläge erfordern explizite Bestätigung)

2.4.3. Qualitätssicherung und Erweiterungen

F8: Qualitätsüberwachung

Das System muss die Datenqualität jedes Kritikalitätsphänomens anhand definierter Kriterien bewerten (Labels, Beschreibungen, Übersetzungen, Verknüpfungen) und die Ergebnisse nach Qualitätslücken priorisiert darstellen.

F9: Kritikalitätsmetriken-Anbindung

Das System muss die Anbindung externer Kritikalitätsmetriken ermöglichen und deren Verknüpfung mit Kritikalitätsphänomenen unterstützen. Die gegenseitige Referenzierung zwischen Kritikalitätsphänomenen und Metriken muss auf den jeweiligen Detailseiten sichtbar sein.

F10: Kausalgraphen

Das System muss die Erstellung und Verwaltung von Kausalgraphen ermöglichen, um komplexe Ursache-Wirkungs-Ketten zwischen Kritikalitätsphänomenen und Metriken zu dokumentieren.

2.5. Nicht-funktionale Anforderungen

Neben den funktionalen Anforderungen wurden folgende Qualitätsanforderungen als wesentlich identifiziert:

NF1: Intuitive Bedienbarkeit

Das Webtool soll selbsterklärend und ohne umfangreiche Einarbeitung nutzbar sein. Die Benutzeroberfläche orientiert sich an etablierten Interaktionsmustern und reduziert kognitive Last durch klare Strukturierung und konsistente Navigation.

NF2: Performance und schnelle Reaktionszeiten

Häufig durchgeführte Aktionen (Navigation, Suche, Ansichtswechsel) sollen ohne spürbare Verzögerung ausgeführt werden. Die Anwendung soll sich flüssig und performant anfühlen, um produktives Arbeiten zu ermöglichen.

NF3: Modulare Erweiterbarkeit

Die Systemarchitektur soll eine klare Trennung von Verantwortlichkeiten aufweisen und die spätere Erweiterung um neue Funktionen ermöglichen, ohne bestehende Komponenten grundlegend umbauen zu müssen.

NF4: Datenqualität und Konsistenz

Die Wissensbasis soll durch geeignete Mechanismen konsistent gehalten werden. Inkonsistente oder unvollständige Daten sollen möglichst früh erkannt und verhindert werden.

2.6. Technische Anforderungen

Die folgenden technischen Anforderungen definieren die grundlegenden technologischen Rahmenbedingungen:

T1: Plattformunabhängige Web-Anwendung

Das System soll als Web-Anwendung realisiert werden, die plattformunabhängig über moderne Webbrowser zugänglich ist.

T2: Drei-Schicht-Architektur

Die Systemarchitektur soll eine klare Trennung in Präsentations-, Logik- und Datenschicht aufweisen. Frontend und Backend sollen unabhängig voneinander entwickelbar sein.

T3: REST API

Die Kommunikation zwischen Frontend und Backend soll über eine REST-API erfolgen, die CRUD-Operationen sowie Kurations- und Explorationsfunktionen bereitstellt.

T4: Typisierte Relationen mit automatischer Inferenz

Die Wissensbasis muss verschiedene Arten von Beziehungen zwischen Kritikalitätsphänomenen unterscheiden und abfragen können (Abstraktionen, kausale Zusammenhän-

ge, Synergien). Für bidirektionale Relationen (z.B. *abstraction_of* / *concretization_of*) muss das System inverse Relationen automatisch ableiten können, sodass nur eine Richtung manuell gepflegt werden muss.

T5: Standardisierte Austauschformate

Die Datenhaltung soll standardisierte RDF-Formate verwenden und Im-/Export über offene Standards (z.B. RDF/Turtle) unterstützen, um Interoperabilität und Datenaustausch zu ermöglichen.

3. Grundlagen und Related Work

Dieses Kapitel führt in die technologischen Grundlagen ein, auf denen das HazardDB-Webtool aufbaut und ordnet die Arbeit in den Kontext verwandter Forschung und existierender Systeme ein. Zunächst werden die relevanten Technologien und Konzepte vorgestellt (Abschnitt 3.1), um anschließend verwandte Arbeiten einzuordnen und die Forschungslücke ableiten zu können (Abschnitt 3.2).

3.1. Technische Grundlagen

3.1.1. RDF und Semantic Web Technologien

Die HazardDB modelliert Kritikalitätsphänomene als vernetzte Struktur: „Hagel“ ist eine konkrete Ausprägung von „widrige Umweltbedingungen“ und kann zu „beschädigter Sensorabdeckung“ führen. Solche Zusammenhänge erfordern ein Datenmodell, das verschiedene Relationstypen explizit unterscheidet und maschinell verarbeitbar macht. RDF bietet diese Grundlage.

Das Triple-Modell

RDF modelliert Wissen als Menge von *Tripeln*, wobei jedes Tripel eine Aussage in der Form Subject-Predicate-Object darstellt [5]. Ein Tripel besteht aus drei Komponenten:

- **Subject:** Die Ressource, über die eine Aussage getroffen wird (z.B. das Kritikalitätsphänomen „Starkregen“)
- **Predicate:** Die Art der Beziehung (z.B. `concretization_of`)
- **Object:** Die Zielressource oder ein Literal-Wert (z.B. das Phänomen „Regen“ oder ein Text wie „Starkregen“@de)

Beispiel: Das Tripel `<CP_042> concretization_of <CP_001>` drückt aus, dass das Kritikalitätsphänomen CP_042 (Starkregen) eine Konkretisierung von CP_001 (Regen) ist.

Namespaces und Uniform Resource Identifier (URI)

RDF verwendet URIs zur eindeutigen Identifikation von Ressourcen. Namespaces vermeiden Namenskonflikte und machen RDF-Daten eindeutig:

- `https://hazarddb.org/schema#` – HazardDB-Vokabular (Abkürzung: `haz:`)
- `http://www.w3.org/2000/01/rdf-schema#` – RDFS Standard (Abkürzung: `rdfs:`)

Durch die Verwendung vollständiger URIs sind alle Ressourcen global eindeutig identifizierbar – ein Phänomen aus der HazardDB kann nicht mit einem gleichnamigen Objekt aus einer anderen Datenbank verwechselt werden.

Language Tags

Mehrsprachigkeit ist eine Kernanforderung der HazardDB (siehe Kapitel 2). RDF unterstützt mehrsprachige Literale durch Language Tags:

```
rdfs:label "Hagel"@de ;
rdfs:label "Hail"@en ;
```

Alle Kritikalitätsphänomene werden parallel auf Deutsch und Englisch gepflegt. Language Tags ermöglichen die parallele Speicherung beider Sprachversionen ohne Schema-Änderungen.

Turtle-Syntax

Turtle (Terse RDF Triple Language) ist eine kompakte, textbasierte Darstellungsform für RDF-Daten, die speziell für menschliche Lesbarkeit entworfen wurde [6]. Ein Beispiel aus der HazardDB:

```
@prefix haz: <https://hazarddb.org/schema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<https://hazarddb.org/id#CP_042>
  a haz:CriticalityPhenomenon ;
  rdfs:label "Starkregen"@de ;
  rdfs:label "Heavy Rain"@en ;
  haz:description "Intensive Niederschläge mit hoher Regenrate..."@de ;
  haz:concretization_of <https://hazarddb.org/id#CP_001> .
```

Listing 1: Turtle-Repräsentation eines Kritikalitätsphänomens

Die wichtigsten Sprachelemente:

- `@prefix` – Deklaration von Namespace-Abkürzungen für URIs
- `a` – Kurzform für `rdf:type` (Klassenzugehörigkeit)

- `;` – Verkettung mehrerer Prädikate für dasselbe Subject
- `@de`, `@en` – Language Tags für mehrsprachige Literale

SPARQL Protocol and RDF Query Language (SPARQL)

SPARQL ist die standardisierte Abfragesprache für RDF-Daten [7]. Analog zu SQL für relationale Datenbanken ermöglicht SPARQL die Formulierung komplexer Abfragen über RDF-Graphen. Eine SPARQL-Query besteht typischerweise aus:

- `SELECT` – Auswahl der zurückzugebenden Variablen
- `WHERE` – Triple Patterns, die gegen den RDF-Graph gematcht werden
- `FILTER` – Zusätzliche Filterbedingungen

Beispiel: Alle direkten Nachbarn eines Kritikalitätsphänomens abrufen (für die Graph-Visualisierung):

```
PREFIX haz: <https://hazarddb.org/schema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?neighbor ?neighborLabel ?relation ?relationLabel
WHERE {
  {
    <https://hazarddb.org/id#CP_042> ?relation ?neighbor .
  }
  UNION
  {
    ?neighbor ?inverseRelation <https://hazarddb.org/id#CP_042> .
    ?relation owl:inverseOf ?inverseRelation .
  }

  ?relation a owl:ObjectProperty ;
            rdfs:domain haz:CriticalityPhenomenon ;
            rdfs:range haz:CriticalityPhenomenon .

  ?neighbor rdfs:label ?neighborLabel .
  FILTER(lang(?neighborLabel) = "en")

  OPTIONAL {
    ?relationMetadata a haz:RelationMetadata ;
                      haz:predicate ?relation ;
                      rdfs:label ?relationLabel .
    FILTER(lang(?relationLabel) = "en")
  }
}
```

Listing 2: SPARQL-Query zur Abfrage von Nachbar-Phänomenen

Die Variablen (`?neighbor`, `?relation`, etc.) fungieren als Platzhalter. Die Query

gibt alle Phänomene zurück, für die *alle* Triple Patterns gleichzeitig matchen. `UNION` kombiniert ausgehende und eingehende Relationen, `OPTIONAL` macht bestimmte Teile optional, `FILTER` beschränkt auf englische Labels.

Triple Stores

Ein *Triple Store* ist eine Datenbank, die speziell für die Speicherung und Abfrage von RDF-Tripeln optimiert ist. Im Gegensatz zu relationalen Datenbanken speichern Triple Stores Daten nativ als Subject-Predicate-Object-Strukturen und stellen SPARQL-Endpoints für Abfragen bereit. Zentrale Eigenschaften:

- Native Unterstützung für das RDF-Datenmodell
- Effiziente Abfrage über SPARQL
- Flexible Schema-Erweiterung durch Hinzufügen neuer Tripel (keine Migrations-Skripte nötig)
- Unterstützung für Reasoning (automatisches Ableiten neuer Tripel aus bestehenden)

3.1.2. REST API

Die Kommunikation zwischen Frontend und Backend des HazardDB-Webtools erfolgt über eine REST-basierte Hypertext Transfer Protocol (HTTP)-API. REST ist ein Architekturstil für verteilte Systeme, der auf den Prinzipien des World Wide Web aufbaut [8].

Warum REST für HazardDB?

Die Drei-Schicht-Architektur der HazardDB erfordert eine standardisierte Schnittstelle zwischen Frontend und Backend (Anforderung T2 und T3). REST ist ein etablierter Architekturstil für Web-APIs, der drei entscheidende Vorteile bietet:

- **Plattformunabhängigkeit:** REST-APIs sind über HTTP zugänglich – damit kann jeder Browser, jedes externe Tool oder zukünftige Analysewerkzeug die HazardDB-Daten abrufen und bearbeiten, ohne spezielle Bibliotheken zu benötigen.
- **Cachability:** Häufig abgerufene Ressourcen (z.B. Detailseiten von Phänomenen) können vom Browser zwischengespeichert werden, was die Performance verbessert (Anforderung NF2).
- **Einfachheit:** REST nutzt Standard-HTTP-Methoden (GET zum Abrufen, POST zum Erstellen, PUT zum Aktualisieren, DELETE zum Löschen), die jedem Web-Entwickler vertraut sind.

REST-Prinzipien

Resource-Oriented: URLs identifizieren Ressourcen (z.B. `/api/entities/CP_001` für ein Kritikalitätsphänomen). Die HTTP-Methode bestimmt die Operation: `GET` zum Abrufen, `POST` zum Erstellen, `PUT` zum Aktualisieren, `DELETE` zum Löschen.

Stateless: Jeder Request enthält alle notwendigen Informationen – der Server speichert keinen Session-State.

JavaScript Object Notation (JSON) als Austauschformat

Als Datenformat für Request/Response-Payloads dient JSON, ein standardisiertes, textbasiertes Format, das aufgrund seiner einfachen Struktur und nativen JavaScript-Unterstützung für Web-APIs etabliert ist. Beispiel:

```
POST /api/entities
Content-Type: application/json
{
  "labelEn": "Hail",
  "labelDe": "Hagel",
  "tags": ["Weather", "Precipitation"]
}
```

Listing 3: POST Request zum Anlegen eines Phänomens

```
{
  "uri": "https://hazarddb.org/id#CP_001",
  "labelEn": "Hail",
  "labelDe": "Hagel"
}
```

Listing 4: Response (201 Created) mit URI des angelegten Phänomens

3.1.3. LLMs für Kurations-Assistenz

Warum LLMs für HazardDB?

Die manuelle Kuration einer umfangreichen Wissensbasis ist aufwendig: Für jedes neue Phänomen müssen relevante Relationen zu bestehenden Phänomenen identifiziert werden – ein Prozess, der mit wachsender Datenbasis überproportional skaliert. LLMs können Kuratoren durch intelligente Vorschläge unterstützen, ohne die finale Entscheidungsgewalt abzugeben (Anforderung F7).

Im Folgenden werden die Grundlagen von LLMs und Prompt Engineering eingeführt, die für die intelligente Kurations-Assistenz relevant sind. Alternative Ansätze (regelbasierte Systeme, klassisches Machine Learning, Embedding-basierte Verfahren) und die konkrete Architekturentscheidung werden in Kapitel 4 diskutiert. Die Implementierung ist in Kapitel 5 dokumentiert.

Was sind LLMs?

LLMs sind neuronale Netze, die auf großen Textmengen trainiert werden und menschenähnliche Texte generieren können [9]. Moderne LLMs basieren auf der Transformer-Architektur [10], die es ihnen ermöglicht, Kontext über lange Textpassagen hinweg zu verstehen. Der Trainingsprozess erfolgt typischerweise in zwei Phasen:

1. **Pre-Training:** Das Modell lernt Sprachmuster aus großen Textsammlungen (z.B. Wikipedia, Bücher, Web-Inhalte). Dieser Schritt läuft unüberwacht – das Modell lernt durch Vorhersage des nächsten Wortes.
2. **Fine-Tuning & Alignment:** Das vortrainierte Modell wird auf spezifische Aufgaben angepasst und durch menschliches Feedback (Reinforcement Learning from Human Feedback (RLHF)) optimiert, um hilfreiche, sichere und wahrheitsgemäße Antworten zu geben.

Zu den bekannten LLM-Anbietern zählen OpenAI (GPT-Serie), Anthropic (Claude), xAI (Grok) sowie Google (Gemini).

Prompt Engineering

Prompt Engineering bezeichnet die systematische Strukturierung von Inputs, um gewünschte Outputs zu erhalten. Das LLM-Verhalten wird durch geschickte Formulierung gesteuert, nicht durch Programmierung. Ein typischer Prompt besteht aus:

- **System Message:** Rollenzuweisung („Du bist ein Experte für Kritikalitätsanalyse in automatisierten Fahrsystemen...“)
- **Context:** Relevante Informationen (bestehende Phänomene, Relationstypen, Tags)
- **Task:** Konkrete Aufgabe („Schlage Relationen für dieses Phänomen vor“)
- **Constraints:** Einschränkungen (Format, Länge, Struktur der Antwort)

Kontext für LLMs bereitstellen

Für präzise Vorschläge muss das LLM die bestehenden Kritikalitätsphänomene kennen. Zwei Ansätze sind etabliert:

Retrieval-Augmented Generation (RAG): Das System sucht zuerst die relevantesten Phänomene aus der Wissensbasis (z.B. die 10 ähnlichsten) und gibt nur diese dem LLM als Kontext. Dies skaliert gut für sehr große Wissensbasen.

Long-Context Prompting: Alle Phänomene werden direkt in den Prompt eingebettet. Die HazardDB nutzt diesen Ansatz, da moderne LLMs große Kontextfenster

unterstützen (bis zu 200.000 Tokens, wobei ein Token etwa einem Wortfragment entspricht) und die Wissensbasis klein genug ist. Da das LLM alle Phänomene sieht und nicht nur eine algorithmisch gefilterte Vorauswahl, führt dies zu präziseren Vorschlägen.

Die Architekturentscheidung und der Vergleich beider Ansätze werden in Kapitel 4 detailliert begründet.

Human-in-the-Loop

Alle LLM-Vorschläge werden im Frontend präsentiert und müssen von Kuratoren bestätigt werden. Dieses *Human-in-the-Loop*-Prinzip verhindert:

- Halluzinationen (das LLM erfindet nicht-existierende Phänomene)
- Qualitätsverlust durch ungenaue Vorschläge
- Unkontrollierte Änderungen an der Wissensbasis

Das LLM fungiert als Assistent, nicht als autonomer Agent.

3.2. Related Work

Nachdem die technologischen Grundlagen etabliert sind, ordnet dieser Abschnitt die HazardDB in den Kontext verwandter Forschung und existierender Systeme ein. Die Analyse gliedert sich in drei Kategorien: Unfall- und Szenariodatenbanken (Abschnitt 3.2.1), generische Knowledge-Base-Tools (Abschnitt 3.2.2) und spezialisierte Wissensbasen (Abschnitt 3.2.3). Abschließend wird die Forschungslücke abgeleitet (Abschnitt 3.2.4).

3.2.1. Unfall- und Szenariodatenbanken

Unfall- und Szenariodatenbanken sind etablierte Instrumente zur Dokumentation sicherheitsrelevanter Ereignisse und Testfälle im Automobilbereich. Sie dienen als Grundlage für statistische Analysen, Safety-Regulierung und die Validierung automatisierter Fahrsysteme.

Unfalldatenbanken

GIDAS (German In-Depth Accident Study) ist seit 1999 die größte In-Depth Unfalldatenbank Deutschlands, betrieben von der TU Dresden und der Medizinischen Hochschule Hannover [11]. Jährlich werden etwa 2000 Unfälle mit ca. 3500 Datenpunkten pro Fall dokumentiert - insgesamt wurden bis 2020 über 41.000 Unfälle

vollständig erfasst. Die Stärken von GIDAS liegen in der sehr detaillierten Datenerfassung (3D-Rekonstruktion, medizinische Daten, Fahrzeugschäden), standardisierten Kategorien für statistische Analysen und langen Zeitreihen für Trendanalysen.

Weitere internationale Unfalldatenbanken wie SHARP2 und IRTAD verfolgen ähnliche Ansätze, unterscheiden sich jedoch in geografischer Abdeckung und methodischer Ausrichtung.

Limitation für HazardDB: Unfalldatenbanken dokumentieren konkrete Ereignisse, nicht abstrahierte, wiederverwendbare Konzepte. Die HazardDB adressiert diese Lücke durch Fokus auf Kritikalitätsphänomene als eigenständige Entitäten mit hierarchischen und expliziten kausalen Relationen.

Szenariodatenbanken

Szenariodatenbanken sammeln und verwalten Testszenarien für die Validierung automatisierter Fahrsysteme. **SafetyPool** ist mit 250.000 Edge Cases die weltweit größte öffentliche Szenariodatenbank und dient als Multi-Stakeholder-Plattform für transparente Tests und Zertifizierung von ADS [12]. **scenario.center** ist eine Plattform für urbane Testszenarien, die einen vereinheitlichten Parameterraum für automatisierte Szenariogenerierung bereitstellt [13]. Die Plattform ermöglicht systematische Variation von Umgebungsparametern (Wetter, Verkehrsdichte, Straßentyp) für Simulationstests.

Limitation für HazardDB: Szenariodatenbanken betten Gefährdungen in konkrete Testfälle ein („Fahrzeug überholt bei Nebel“), modellieren aber nicht die kausalen Zusammenhänge zwischen Nebel, Sichteinschränkung und Kollisionsrisiko als eigenständige, wiederverwendbare Konzepte.

3.2.2. Generische Knowledge-Base-Tools

Generische Wissensmanagement-Tools sind weit verbreitet und flexibel einsetzbar. Die folgende Analyse untersucht, warum ihre Allzweck-Natur sie für die spezialisierte HazardDB-Kuration ungeeignet macht.

Dokumenten-zentrierte Tools

Dokumenten-basierte Systeme wie **Notion** und **Confluence** dominieren das kollaborative Wissensmanagement in Unternehmen. Notion kombiniert Dokumente, Datenbanken und Kanban-Boards in einem flexiblen Workspace [14], während Confluence als etabliertes Enterprise-Wiki mit Versionskontrolle und Team-Kollaboration überzeugt [15].

Beide teilen jedoch eine strukturelle Limitation: Sie modellieren Wissen primär als Dokumente und Seiten. Relationen zwischen Entitäten werden durch einfache

Hyperlinks dargestellt – ein Link von „Hagel“ zu „Beschädigter Sensor“ transportiert keine Information darüber, *welche Art* von Beziehung besteht. Handelt es sich um eine kausale Relation („führt zu“) oder eine Hierarchie („ist Teil von“)? Die HazardDB unterscheidet explizit zwischen verschiedenen Relationstypen: *could_lead_to* (kausal), *abstraction_of* (hierarchisch) und *synergizes_with* (synergetisch). Diese Unterscheidung lässt sich in dokumenten-basierten Systemen nicht abbilden.

Graph-basierte Note-Taking Tools

Obsidian erweitert das Konzept persönlicher Notizen durch bidirektionale Links und eine Graph-Visualisierung aller Verbindungen [16]. Die Option zum Self-Hosting und die flexible Markdown-Struktur machen es zu einem beliebten Werkzeug für individuelles Wissensmanagement.

Für eine kollaborativ kuratierte Wissensbasis wie die HazardDB fehlt jedoch die notwendige Formalisierung. Obsidian erzwingt keine gemeinsamen Schemas oder Pflichtfelder - jeder Nutzer strukturiert Inhalte nach eigenen Konventionen. Was für persönliche Notizen Freiheit bedeutet, führt bei strukturierten Daten zu Inkonsistenz. Zudem setzt die gemeinsame Bearbeitung Git-Kenntnisse voraus, was für nicht-technische Nutzer eine Hürde darstellt.

Graph-Datenbank-Tools

Neo4j Bloom ist eine etablierte Visualisierungs-Plattform für Graph-Datenbanken [17]. Die Cypher Query Language ermöglicht komplexe Graph-Abfragen, und die flexible Modellierung erlaubt die Abbildung beliebiger Relationstypen. Bloom bietet ein visuelles Interface für die Exploration von Graphstrukturen.

Für die HazardDB fehlen jedoch domänenspezifische Funktionen: Bloom ist ein generisches Tool ohne spezialisierte Eingabehilfen für Kritikalitätsphänomene. Nutzer müssen Datenstrukturen manuell definieren und pflegen. Kontextsensitive Vorschläge für Tags oder Relationen fehlen ebenso wie automatische Konsistenzprüfungen. Die Kuration erfordert Graph-Datenbank-Kenntnisse, was für Safety-Engineers ohne technischen Hintergrund eine Hürde darstellt.

Semantische Wikis

Wikibase, die Software hinter Wikidata, unterstützt RDF, SPARQL und vollständige semantische Modellierung [18]. Es kombiniert die Offenheit von Wikis mit den Standards des Semantic Web und ermöglicht Multi-User-Editing mit Versionskontrolle.

Wikibase folgt dem Wikipedia-Modell: Offenes Editieren mit nachträglicher Qualitätskontrolle durch die Community. Für die HazardDB ist dieser Ansatz ungeeignet:

Der Aufbau einer spezialisierten Wissensbasis für sicherheitskritische Kritikalitätsphänomene erfordert kuratierte Datenpflege mit proaktiven Qualitätsprüfungen bereits während der Eingabe, nicht nachträgliche Korrekturen durch eine offene Community. Die HazardDB benötigt ein eigenständiges, spezialisiertes Tool mit domänenspezifischen Eingabehilfen.

Die gemeinsame Lücke

Allen analysierten Tools fehlt domänenspezifische Kurations-Assistenz: LLM-basierte Vorschläge für fehlende Beschreibungen, Tags oder Relationen existieren nicht. Automatisierte Qualitätsprüfungen (z.B. für fehlende Übersetzungen oder isolierte Knoten) fehlen. Generische Tools sind flexibel, bieten aber keine Unterstützung für spezialisierte Anwendungsfälle. Diese Lücke motiviert die Entwicklung eines spezialisierten Web-Tools, das durch gezielte Eingabehilfen die Qualität erhöht und gleichzeitig den Kurations-Aufwand reduziert.

3.2.3. Spezialisierte Wissensbasen

Spezialisierte Wissensbasen adressieren spezifische Forschungsdomänen mit maßgeschneiderten Datenmodellen. Die folgenden Beispiele aus der evolutionären Linguistik und dem Automotive-Bereich illustrieren sowohl das Potenzial als auch die Herausforderungen solcher Systeme.

CHIELD - Ein funktionierendes Beispiel

Die *Causal Hypotheses In Evolutionary Linguistics Database* (CHIELD) dokumentiert über 700 kausale Hypothesen zur Sprachevolution als vernetzte Graphen [19]. Die Plattform ermöglicht es Forschern, konkurrierende Theorien zur Entstehung von Sprache und Sprachvielfalt zu modellieren und miteinander zu vergleichen. Jede Hypothese wird mit Publikationen verknüpft, die als Evidenz dienen.

CHIELD demonstriert, dass spezialisierte Wissensbasen erfolgreich funktionieren können - vorausgesetzt, eine aktive Forschungs-Community trägt kontinuierlich bei. Gleichzeitig zeigt CHIELD die Grenzen manueller Kuration: Alle Relationen werden von Hand erfasst, was den Beitragsprozess aufwendig macht.

PerCollECT - Ein warnendes Beispiel

Die *Perception Sensor Collaborative Effect and Cause Tree* (PerCollECT) verfolgte ein ähnliches Ziel im Automotive-Kontext [20]. Die Wissensbasis sollte Ursache-Wirkungs-Ketten für Umgebungswahrnehmungssensoren (Radar, Lidar, Kamera, Ultraschall) systematisch dokumentieren - mit dem Ziel, Sensormodelle für die ADS-Validierung zu spezifizieren.

Die GitHub-Repositories zeigen jedoch seit längerer Zeit keine Aktivität mehr. PerCollECT illustriert ein Nachhaltigkeitsproblem spezialisierter Wissensbasen: Ohne kontinuierliche Pflege veralten die Daten, neue Erkenntnisse fließen nicht ein, und potenzielle Nutzer wenden sich ab. Ein Teufelskreis kann entstehen: Sinkende Datenqualität führt zu sinkender Nutzung, was wiederum die Motivation zur Pflege reduziert.

Implikationen für HazardDB

Die Analyse spezialisierter Wissensbasen verdeutlicht: Domänenspezifität allein garantiert keine Nachhaltigkeit. HazardDB benötigt integrierte Kurations-Assistenz, um langfristig pflegbar zu bleiben. Konkret bedeutet dies:

- **LLM-gestützte Kurations-Vorschläge:** Reduktion des manuellen Aufwands durch intelligente Vorschläge für fehlende Beschreibungen, Tags, Relationen oder ähnliche Phänomene - basierend auf dem bestehenden Graph-Kontext
- **Automatisierte Health Checks:** Proaktive Identifikation von Qualitätslücken (fehlende Übersetzungen, zu kurze Beschreibungen, isolierte Knoten)
- **Eingabehilfen:** Formulare mit Combobox-Filterung für die Auswahl existierender Phänomene, Autocomplete für Tags, Validierung von Relationstypen (verhindert Duplikate)
- **Transparenz und optionale Gamification:** Sichtbarmachung von Beiträgen zur Stärkung intrinsischer Motivation (z.B. Nutzungsstatistiken, optional Leaderboards)

Diese Anforderungen motivieren die in Kapitel 4 beschriebenen Architekturentscheidungen.

3.2.4. Ableitung der Forschungslücke

Die Analyse verwandter Arbeiten offenbart drei zentrale Lücken:

Lücke 1: Unfall- und Szenariodatenbanken sind zu konkret. GIDAS und SafetyPool dokumentieren einzelne Unfälle bzw. Testszenarien. HazardDB abstrahiert diese konkreten Ereignisse zu *wiederverwendbaren Kritikalitätsphänomenen* mit expliziten hierarchischen und kausalen Relationen.

Lücke 2: Generische Tools bieten keine domänenspezifische Unterstützung. Notion, Obsidian und Neo4j sind flexibel, bieten aber keine spezialisierten Eingabehilfen. HazardDB benötigt domänenspezifische Relationstypen (*could_lead_to*, *synergizes_with*) und kontextsensitive Unterstützung bei der Datenpflege.

Lücke 3: Kritikalitätsphänomene erfordern kontinuierliche Pflege. Nicht alle Wissensbasen benötigen Updates, doch Kritikalitätsphänomene entwickeln sich

weiter: Neue Sensortypen, Fahrzeugarchitekturen und Verkehrsszenarien erfordern laufende Ergänzungen. PerCollECT zeigt, dass ohne aktive Kuration Wissensbasen veralten. HazardDB benötigt Mechanismen zur Reduktion des Kurations-Aufwands (LLM-Vorschläge) und zur Sicherung der Datenqualität (automatisierte Prüfungen), um langfristig pflegbar zu bleiben.

Aus diesen Lücken ergibt sich der Bedarf für ein spezialisiertes Web-Tool, das Exploration durch intuitive Navigation ermöglicht, Kuration durch intelligente Eingabehilfen unterstützt und durch automatisierte Qualitätssicherung sowie optionale Anreizmechanismen langfristig pflegbar bleibt.

4. Systemarchitektur

4.1. Architekturüberblick

Die Architektur des HazardDB-Webtools folgt einer klassischen Drei-Schicht-Architektur mit klarer Trennung von Präsentations-, Logik- und Datenschicht. Diese Strukturierung entspricht der technischen Anforderung T2 (siehe Abschnitt 2.6) und ermöglicht die unabhängige Entwicklung und Wartung der einzelnen Komponenten. Die drei Schichten kommunizieren über definierte Schnittstellen: Das Frontend sendet HTTP-Anfragen an das Backend. Das Backend kommuniziert mit der Datenbank via SPARQL und mit der Suchmaschine via HTTP.

Die Architekturentscheidungen wurden maßgeblich durch die nicht-funktionalen Anforderungen NF1–NF4 geprägt: intuitive Bedienbarkeit, Performance, modulare Erweiterbarkeit und Datenqualität. Die konkrete Technologiewahl wird in Abschnitt 4.1.1 detailliert begründet.

4.1.1. Technologie-Stack und Architekturentscheidungen

Das HazardDB-Webtool basiert auf einem JavaScript-basierten Technologie-Stack. JavaScript/TypeScript wird durchgängig für Frontend und Backend genutzt. Dies vereinfacht die Entwicklung durch gemeinsame Sprachfeatures und wiederverwendbare Typdefinitionen.

Die **Präsentationsschicht** wird durch Next.js 15 realisiert, ein React-Framework für moderne Webanwendungen. Details zur Framework-Wahl und Frontend-Architektur werden in Abschnitt 4.4 beschrieben.

Die **Logikschicht** basiert auf NestJS, einem TypeScript-Framework für serverseitige Anwendungen. Die modulare Architektur unterstützt Anforderung NF3 (modulare Erweiterbarkeit). Details zur Framework-Wahl werden in Abschnitt 4.3 beschrieben.

Die **Datenschicht** nutzt zwei spezialisierte Datenbanksysteme: Apache Jena Fuseki speichert alle strukturierten Daten als RDF-Tripel und fungiert als zentrale, verlässliche Datenquelle. Meilisearch ergänzt Fuseki um eine schnelle Volltextsuche mit Tippfehler-Toleranz. Beide Systeme werden über Ereignisse synchron gehalten – strukturelle Änderungen in Fuseki lösen automatisch Updates in Meilisearch aus. Die Begründung dieser Architekturentscheidung wird in Abschnitt 4.2.2 detailliert erläutert.

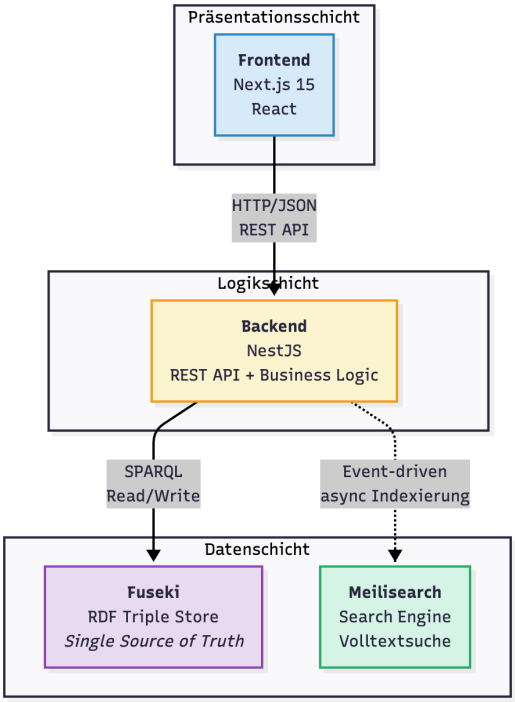

Abbildung 4.1.: Drei-Schichten-Architektur der HazardDB mit Dual-Storage-Strategie

4.2. Datenarchitektur

Die Datenhaltung der HazardDB stellt besondere Anforderungen an die Technologie: Die Wissensbasis modelliert nicht nur einzelne Phänomene, sondern auch deren Beziehungen zueinander. Ein Phänomen wie „Nebel“ steht in hierarchischer Beziehung zu „widrige Umweltbedingungen“ (als Konkretisierung) und kann kausal zu „Sichteinschränkung“ führen. Diese komplexen Zusammenhänge erfordern ein Datenmodell, das Relationen explizit repräsentiert, semantisch verarbeitet und effizient abfragbar macht.

4.2.1. Architekturentscheidung: RDF Triple Store

Die Wahl von RDF als Datenhaltungstechnologie wurde in Abschnitt 3.1.1 bereits motiviert: automatische Relationsinferenz (T4), native Mehrsprachigkeit und standardisierte Austauschformate (T5). Die folgenden Abschnitte kontrastieren RDF mit alternativen Ansätzen und begründen die konkrete Implementierungsentscheidung für Apache Jena Fuseki.

Drei Technologieansätze wurden evaluiert: Relationale Datenbanken (PostgreSQL als Stellvertreter), Property Graph Databases (Neo4j) und RDF Triple Stores (Apache

Jena Fuseki). Im Folgenden wird die Entscheidungsfindung narrativ dargestellt, einschließlich der historischen Evolution von Neo4j zu RDF.

Relationale Datenbanken: PostgreSQL

Relationale Datenbanken sind grundsätzlich für Graphstrukturen geeignet. Die CHIELD-Datenbank demonstriert in SQLite, dass Graph-Operationen via rekursive SQL-Abfragen (CTEs) realisierbar sind.

Für die HazardDB sprachen zwei Gründe gegen diese Lösung:

(i) Natives semantisches Reasoning fehlt. Inverse Relationen erfordern manuelle Implementierung via Datenbank-Trigger oder Applikationslogik, was Wartungsaufwand und Fehleranfälligkeit erhöht. RDF mit OWL-Reasoning garantiert diese Konsistenz deklarativ auf Spezifikationsebene (T4).

(ii) Die geforderte Interoperabilität ist erschwert. SQL-Dumps repräsentieren Datenstrukturen, nicht semantische Metadaten. RDF/Turtle ermöglicht verlustfreien Export mit expliziten Relationstypen und mehrsprachigen Labels (T5).

PostgreSQL ist technisch möglich, aber semantisch unpassend für die Anforderungen T4 und T5.

Die Evolution: Von Neo4j zu RDF

Die HazardDB begann mit Neo4j als Property Graph Database. Das Neosemantics-Plugin (n10s) unterstützt nur leichtes Reasoning über Klassenhierarchien, nicht jedoch vollständiges OWL-Reasoning. Beispielsweise können inverse Relationen nicht automatisch inferiert werden [21]. Da die HazardDB komplexere Relationen erfordert, fiel die Entscheidung auf Apache Jena Fuseki mit voller OWL-Unterstützung.

Apache Jena Fuseki: RDF Triple Store mit OWL Reasoning

Jena Fuseki erfüllt alle identifizierten Anforderungen und adressiert die Limitationen von PostgreSQL und Neo4j:

(i) **Automatische Relationsinferenz:** Der integrierte OWL-Reasoner inferiert inverse Relationen automatisch [22]. Kuratoren pflegen nur eine Richtung, das System garantiert Konsistenz und Verfügbarkeit beider Relationen.

(ii) **Native Mehrsprachigkeit:** RDF unterstützt mehrsprachige Literale nativ durch Language Tags. Im Gegensatz zu relationalen Datenbanken können neue Sprachen ohne Schema-Migration hinzugefügt werden.

(iii) **Flexible Schema-Evolution:** Neue Properties können ohne Schema-Migration hinzugefügt werden. Dies ist besonders wertvoll für iterativ evolvierendes Schema.

(iv) **Interoperabilität:** RDF und SPARQL sind W3C-Standards mit etabliertem Ökosystem. Die HazardDB exportiert Daten im Turtle-Format für die weitere Bearbeitung, beispielsweise in Protégé.

4.2.2. Dual-Storage-Strategie: Fuseki und Meilisearch

Meilisearch wurde als spezialisierte Such-Engine gewählt [23]. Es ermöglicht schnellere Entwicklung (weniger Konfiguration, bessere Developer-Experience als Elasticsearch) und bietet Out-of-the-Box Zukunfts-Features (Synonyme, Relevanz-Ranking, Typo-Toleranz). Als etablierte Open-Source-Lösung (MIT-Lizenz) erfüllt Meilisearch die Anforderung NF1 (intuitive Bedienbarkeit).

Architektur: Fuseki als Source of Truth, Meilisearch für Suche

Die HazardDB implementiert eine Dual-Storage-Architektur: Jena Fuseki dient als Single Source of Truth für alle CRUD-Operationen und Graph-Queries. Meilisearch fungiert als spezialisierte Such-Engine für performante Volltextsuche. Alle strukturellen Änderungen werden ausschließlich in Fuseki persistiert. Meilisearch repliziert diese Daten asynchron für Such-Zwecke.

Die Synchronisation zwischen beiden Systemen erfolgt ereignisgesteuert: Fuseki-Operationen emittieren Domain-Events (z.B. EntityCreatedEvent), auf die Meilisearch asynchron reagiert. Die technische Implementierung wird in Abschnitt 5.4 detailliert beschrieben.

4.3. Backend-Architektur

Das Backend orchestriert die Business-Logik zwischen Frontend und Dateninfrastruktur. Es stellt eine HTTP-API bereit für CRUD-Operationen, Volltextsuche, Graph-Queries, LLM-gestützte Kurations-Assistenz und Qualitätschecks.

4.3.1. Framework-Wahl: NestJS

NestJS wurde als Backend-Framework gewählt, da es eine strukturierte, modulare Architektur erzwingt und dadurch die Anforderung NF3 (modulare Erweiterbarkeit) unterstützt. Das Framework basiert auf TypeScript und bietet Dependency Injection, Decorators für Routing und Validierung sowie eine klare Trennung von Verantwortlichkeiten durch das Controller-Service-Repository-Pattern.

Im Vergleich zu minimalistischen Frameworks wie Express.js bietet NestJS eine opinionierte Struktur, die Boilerplate-Code reduziert. Express erfordert manuelle

Einrichtung von Routing, Validierung, Dependency Injection und Fehlerbehandlung – Aspekte, die NestJS standardisiert. Für eine Bachelorarbeit mit begrenztem Zeitbudget war diese Out-of-the-Box-Funktionalität entscheidend.

TypeScript als gemeinsame Sprache für Frontend und Backend ermöglicht wiederverwendbare Typdefinitionen und verbessert die Wartbarkeit durch statische Typisierung.

4.3.2. LLM-Integration-Architektur

Die LLM-gestützte Kurations-Assistenz ist ein Alleinstellungsmerkmal der HazardDB und adressiert die funktionale Anforderung F7 (intelligente Kurations-Assistenz). Manuelle Kuration skaliert nicht: Für jedes neue Phänomen müssen Kuratoren relevante Relationen zu allen bestehenden Phänomenen identifizieren – ein Problem, das mit wachsender Datenbankgröße zunehmend aufwendiger wird. Large Language Models können Kuratoren durch intelligente Vorschläge unterstützen, ohne die finale Entscheidungsgewalt abzugeben.

Warum LLMs?

Die Wahl fiel auf LLMs statt trainierbare Klassifikatoren aus zwei Gründen: Erstens würde das Training spezialisierter Modelle den zeitlichen Rahmen dieser Bachelorarbeit übersteigen. Zweitens erfordern semantische Relationen kausales Verständnis, das über statistische Korrelation hinausgeht – LLMs bringen dieses Weltwissen bereits mit.

Die Implementierung bettet die IDs und Labels (deutsch und englisch) aller Kritikalitätsphänomene ein. Moderne LLMs verfügen über Context Windows von mittlerweile 200.000 Tokens oder mehr – die Wissensbasis der HazardDB passt damit problemlos in einen einzigen Prompt.

Jüngste Forschung zu Optical Character Recognition (OCR) zeigt, dass Context Windows effektiv um das Zehnfache vergrößert werden können: DeepSeek-OCR [24], ein im Oktober 2025 veröffentlichtes System des chinesischen Forschungslabs DeepSeek, erreicht bei zehnfacher Komprimierung eine Genauigkeit von 97 Prozent. Dies adressiert potentielle zukünftige Probleme bei wachsender Anzahl von Kritikalitätsphänomenen.

Architektur-Übersicht: 5-Schritt-Flow

Die LLM-Kurations-Assistenz orchestriert fünf Schritte:

1. **Context-Aggregation:** Laden aller Phänomene aus Fuseki
2. **Prompt-Generierung:** Einbettung in strukturierten Prompt

3. **LLM-Kommunikation:** Anfrage an Provider
4. **Validation & Grounding:** Filterung halluzinierter Vorschläge
5. **Human-in-the-Loop:** Frontend-Präsentation mit Accept/Reject

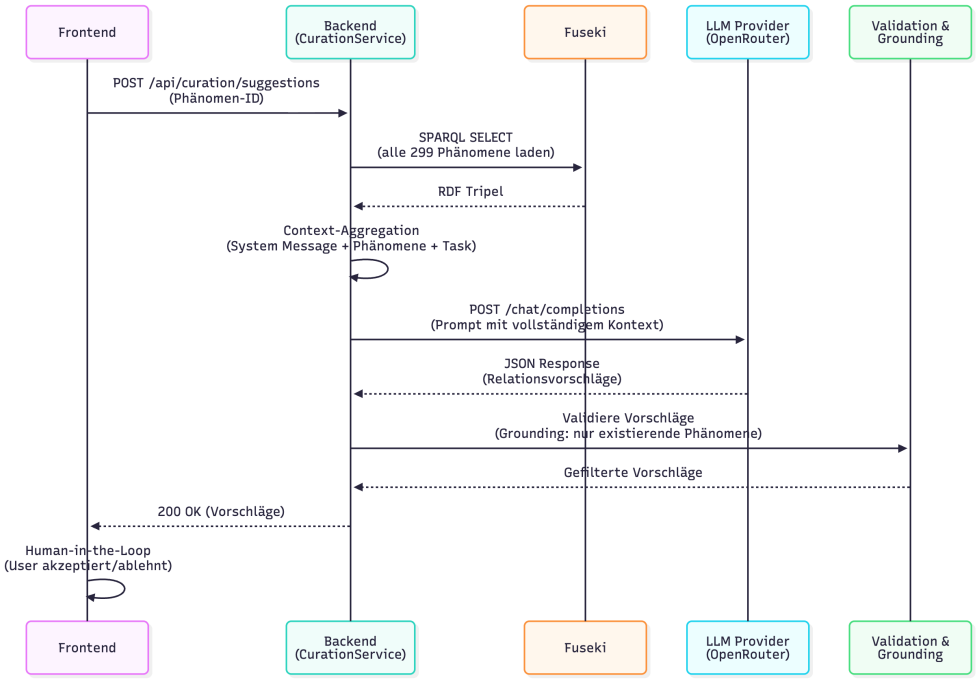

Abbildung 4.2.: LLM-Kurations-Pipeline mit 5-Schritt-Flow und Human-in-the-Loop-Validierung

Die direkte Einbettung aller Phänomene ist einfach und präzise: Alle Daten stehen dem LLM vollständig zur Verfügung, ohne dass Informationen durch Vorfilterung verloren gehen. Details zur Prompt-Struktur und Validation finden sich in Abschnitt 5.3.

Provider-Abstraktion

Die LLM-Integration nutzt eine Abstraktion basierend auf dem Strategy Pattern und Factory Pattern [25, 26]. Eine abstrakte Basisklasse definiert das Interface, konkrete Provider-Implementierungen sind austauschbar ohne Änderung der Business-Logik. Dies ermöglicht den Wechsel zwischen verschiedenen LLM-Anbietern (OpenAI, Anthropic, Google, Meta) und zukünftige Integration lokaler Modelle (z.B. via Ollama).

Die konkrete Implementierung wird in Kapitel 5 detailliert beschrieben.

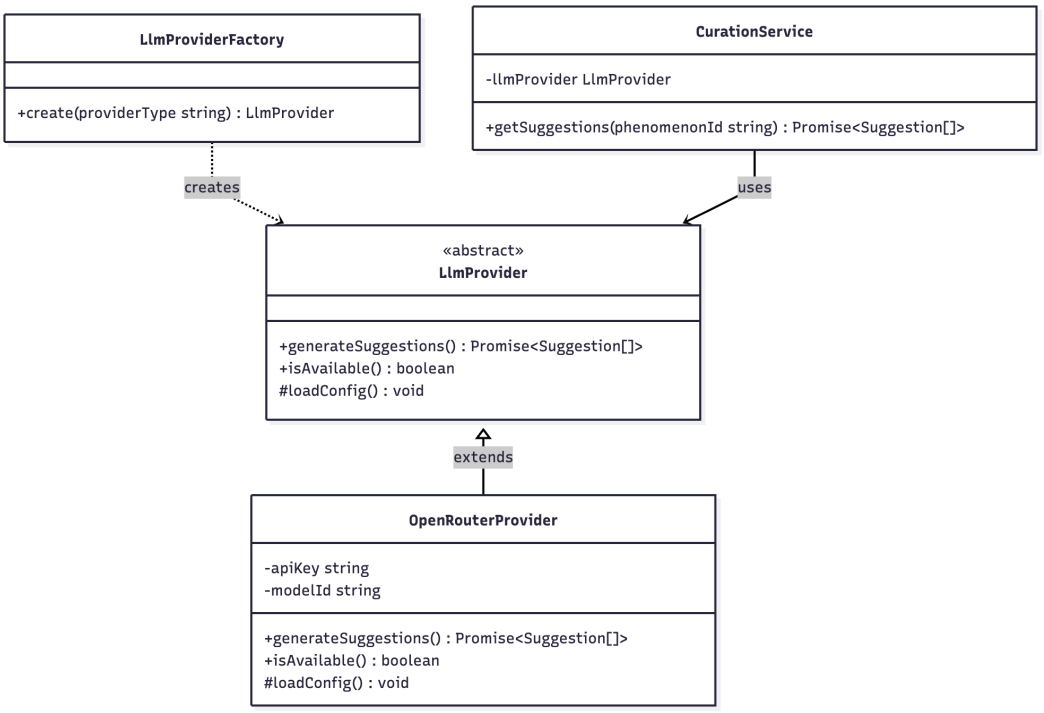

Abbildung 4.3.: LLM-Provider-Architektur mit Strategy Pattern und Factory Pattern

4.3.3. Qualitätssicherungs-Architektur

Die HazardDB implementiert einen regelbasierten Qualitätssicherungs-Mechanismus, um Kuratoren bei der Identifikation von Qualitätslücken zu unterstützen. Jedes Kritikalitätsphänomen wird automatisch bewertet:

- Sind englische und deutsche Labels vorhanden?
- Sind Beschreibungen ausreichend dokumentiert?
- Ist das Phänomen vernetzt (Relationen, Tags, Kritikalitätsmetriken)?

Ein Scoring-System bewertet diese Kriterien und gibt Feedback über Qualitätslücken. Die Implementierung nutzt SPARQL-Aggregationen für effiziente Metadaten-Prüfung (Details in Abschnitt 5.6).

4.4. Frontend-Architektur

Das Frontend des HazardDB-Webtools nutzt eine hybride Rendering-Strategie, die Server- und Client-Rendering kombiniert. Die Architektur priorisiert schnelle Navigation und interaktive Nutzererfahrung, um Anforderung NF2 (Performance

und schnelle Reaktionszeiten) zu erfüllen. Der Technologie-Stack basiert auf Next.js, React und TypeScript.

4.4.1. Framework-Wahl: Next.js 15

Next.js wurde als Basis-Framework gewählt. Es ist ein etabliertes React-Framework für moderne Webanwendungen und bietet dateibasiertes Routing, optimierte Client-seitige Navigation ohne Full-Page-Reloads und gute Developer-Experience – wichtig für die interaktive Exploration der Wissensbasis (Anforderung NF2).

Die Verwendung von TypeScript als gemeinsame Sprache für Frontend und Backend ermöglicht wiederverwendbare Typdefinitionen und verbessert die Wartbarkeit durch statische Typisierung über die gesamte Anwendung hinweg.

4.4.2. State Management: TanStack Query

Die Verwaltung von Server-Daten erfolgt über TanStack Query. Sie bietet automatisches Caching, Fehlerbehandlung und Optimistic Updates für CRUD-Operationen. Da die HazardDB primär Backend-Daten darstellt und kaum lokalen UI-State benötigt, ist TanStack Query die richtige Wahl.

4.4.3. Komponenten-Architektur und UI-Bibliothek

Die UI-Komponenten basieren auf Shadcn/UI (Radix UI + Tailwind CSS). Diese etablierte Open-Source-Lösung bietet wiederverwendbare, produktionserprobte Komponenten (Dialogs, Dropdowns, Tooltips) mit anpassbarem Styling und beschleunigt die Frontend-Entwicklung erheblich.

Die interaktive Graph-Visualisierung der Relationen nutzt D3.js für flexible Positionierung. Nutzer können Knoten per Drag-and-Drop verschieben und die Exploration schrittweise durchführen. Details zur Implementierung dokumentiert Abschnitt 5.5.2.

4.5. Kausalgraphen

Die HazardDB ermöglicht die Modellierung von Kausalgraphen, welche Kritikalitätsphänomene mit Kritikalitätsmetriken und Kausalvariablen verbinden [4, 27]. Ein Kausalgraph modelliert, wie ein Phänomen andere Phänomene beeinflusst.

4.5.1. Drei Entity-Typen als Knoten

Kausalgraphen können drei Arten von Knoten enthalten:

- **Kritikalitätsphänomene:** Referenzen auf bestehende Phänomene der Wissensbasis (z.B. `CP_13`)

- **Kritikalitätsmetriken:** Referenzen auf Metriken zur Quantifizierung (z.B. Time To Brake, siehe Abschnitt 4.6)
- **Kausalvariablen:** Abstrakte Variablen ohne Datenbankentität, die Expertenwissen repräsentieren. Diese existieren nur zur Visualisierung.

4.5.2. DOT-Format

Die Eingabe erfolgt über die Graphviz DOT-Sprache [28], ein etabliertes Format für Graphspezifikationen.

```
digraph "Safety_Scenario" {
  rankdir=LR;

  Fog [label="Fog", class="CriticalityPhenomenon",
       uri="https://hazarddb.org/id#CP_13"];
  LowVisibility [label="Low Visibility", class="CausalVariable"];
  TimeToBreak [label="Time To Brake", class="CriticalityMetric",
               uri="https://hazarddb.org/id#CM_time-scale__TTB"];
  BrakingDecision [label="Emergency Braking", class="CausalVariable"];

  Fog -> LowVisibility [label="causes"];
  LowVisibility -> TimeToBreak [label="reduces"];
  TimeToBreak -> BrakingDecision [label="triggers"];
}
```

4.5.3. Automatische Entitäts-Erkennung

Das System erkennt automatisch, welche Kausalgraphen welche Entitäten (Phänomene, Metriken) verwenden. Dies ermöglicht bidirektionale Navigation: Auf der Detailseite eines Phänomens werden alle Kausalgraphen angezeigt, die dieses Phänomen referenzieren. Die technische Umsetzung nutzt die RDF-Property `haz:includesEntity`, die beim Speichern automatisch befüllt wird (Details in Abschnitt 5.7).

4.6. Kritikalitätsmetriken

Die HazardDB integriert externe Kritikalitätsmetriken zur Quantifizierung von Kritikalitätsphänomenen [29] – beispielsweise Time To Brake (TTB) zur Messung der verfügbaren Reaktionszeit bei Notbremsungen.

4.6.1. ReadTheDocs als Datenquelle

Die Kritikalitätsmetriken stammen aus der zugehörigen ReadTheDocs-Dokumentation[1] und werden über eine API abgefragt. Das System fetcht den Metriken-Katalog ad-hoc und cached ihn für 6 Stunden. Vorteile dieser Architektur:

[1] https://criticality-metrics.readthedocs.io/en/latest/index.html

- Aktualisierungen der Dokumentation werden automatisch übernommen
- Keine manuelle Pflege von Metadaten

4.6.2. Bidirektionale Verlinkung

Metriken können mit Kritikalitätsphänomenen verlinkt werden. Die Verlinkung ist bidirektional in RDF gespeichert:

```
<CM_time-scale__TTB> haz:assessesPhenomenon <CP_13> .  # TTB bewertet Fog
<CP_13> haz:isAssessedByMetric <CM_time-scale__TTB> .
```

Dies ermöglicht Queries in beide Richtungen: „Welche Metriken bewerten dieses Phänomen?“ und „Welche Phänomene werden von dieser Metrik bewertet?“

4.6.3. Minimal-Speicherung in RDF

Nur essenzielle Metadaten werden in der Datenbank gespeichert. Die vollständige Dokumentation wird on-demand von ReadTheDocs geladen. Die Implementierung wird in Abschnitt 5.8 detailliert beschrieben.

4.7. Von der Architektur zur Implementierung

Die in diesem Kapitel beschriebene Architektur bildet das konzeptuelle Fundament der HazardDB. Kapitel 5 dokumentiert die praktische Umsetzung: Wie werden SPARQL-Queries gebaut? Wie sieht ein konkreter LLM-Prompt aus? Wie reagiert das System auf Halluzinationen? Welche Implementierungsentscheidungen mussten getroffen werden?

5. Implementierung

5.1. Einleitung

Dieses Kapitel dokumentiert die praktische Umsetzung der in Kapitel 4 beschriebenen Architektur. Während die Architektur das konzeptuelle Fundament liefert, stehen hier die konkreten Implementierungsentscheidungen, technischen Details und Umsetzungserfahrungen im Mittelpunkt. Der Fokus liegt auf der Darstellung, *wie* die Kernfunktionalitäten technisch realisiert wurden – von CRUD-Operationen über LLM-gestützte Kurations-Assistenz bis hin zur Graph-Visualisierung.

Die Entwicklung erfolgte über einen Zeitraum von sechs Monaten in einem iterativen Prozess: Features wurden prototypisch implementiert, in Abstimmung mit der Projektgruppe evaluiert und anschließend verfeinert. Diese agile Vorgehensweise ermöglichte es, Anforderungen während der Entwicklung zu konkretisieren und anzupassen.

Das Kapitel ist feature-orientiert strukturiert: Jeder Abschnitt behandelt eine Kernfunktionalität der HazardDB. Für jedes Feature wird die technische Umsetzung anhand repräsentativer Code-Beispiele illustriert. Die präsentierten Code-Snippets stammen direkt aus dem implementierten Prototyp und wurden zur besseren Lesbarkeit gekürzt und kommentiert.

Hinweis zur Lesart: Die folgenden Abschnitte enthalten vereinzelt Code-Beispiele zur Illustration der technischen Umsetzung. Leser ohne Programmiererfahrung können diese überspringen – die konzeptionellen Erklärungen sind eigenständig verständlich. Vollständige Code-Beispiele mit Kontext finden sich in Anhang A.

5.2. CRUD-Operationen mit SPARQL

Die Verwaltung von Kritikalitätsphänomenen – das Anlegen, Abrufen und Bearbeiten von Entities – bildet die Grundlage der HazardDB. Die Implementierung folgt dem Controller-Service-Repository-Pattern, das eine klare Trennung von HTTP-Handling, Business-Logik und Datenzugriff erzwingt.

5.2.1. Architekturpattern: Controller → Service → Repository

Die Backend-Architektur folgt dem in Abschnitt 4.3 beschriebenen Controller-Service-Repository-Pattern. Im Folgenden wird die konkrete Implementierung in NestJS illustriert.

5.2.2. Entity-Update: Ein repräsentatives Beispiel

Die `updateEntity`-Operation illustriert das Zusammenspiel der drei Schichten (siehe Listing A.1.1 im Anhang für Details). Ein Kurator bearbeitet ein bestehendes Phänomen – etwa die Ergänzung einer deutschen Beschreibung oder das Hinzufügen neuer Relationen. Die Service-Methode koordiniert mehrere Repository-Operationen:

1. **Validierung:** Labels müssen eindeutig sein. Um zu vermeiden, dass zwei Kuratoren gleichzeitig dasselbe Label vergeben (*Race Condition* – eine Situation, in der die Korrektheit von Operationen von der zeitlichen Abfolge abhängt), erfolgt die Prüfung vor der Persistierung.
2. **Atomare Updates:** Properties (Labels, Beschreibungen) werden zuerst aktualisiert, dann Relationen ersetzt, schließlich Tags. Diese sequenzielle Abarbeitung gewährleistet, dass bei Fehlern keine inkonsistenten Teildaten in der Datenbank verbleiben.
3. **Event-Emission:** Nach erfolgreichem Update wird ein `EntityUpdatedEvent` emittiert – dieses Event triggert asynchron die Re-Indexierung in Meilisearch (siehe Abschnitt 5.4).

Dieser Ablauf demonstriert die Trennung von Verantwortlichkeiten: Der Controller empfängt die Anfrage, der Service orchestriert die Logik, das Repository kommuniziert mit der Datenbank.

5.2.3. SPARQL-Query-Generierung in der Repository-Schicht

Die Repository-Schicht übersetzt Operationen in SPARQL-Queries. Die `replaceOutgoingRelations`-Methode ersetzt alle ausgehenden Relationen eines Phänomens in drei Schritten:

1. **Delete:** Bestehende Relationen werden mit SPARQL DELETE entfernt
2. **Deduplizierung:** Neue Relationen werden nach Kombination aus Relationstyp und Zielphänomen dedupliziert
3. **Insert:** Die Relationen werden als RDF-Tripel eingefügt

Die Methode `buildRelationTriple` normalisiert URIs: Kurzformen wie `CP_042` werden zu vollständigen URIs expandiert (`https://hazarddb.org/id#CP_042`). Dies ermöglicht kürzere API-Requests – statt der vollen URI genügt die ID. Die vollständige Implementierung findet sich in Listing A.1.2 im Anhang.

Die SPARQL-Queries werden über Template-Funktionen generiert (z.B. `ENTITY_DELETE_OUTGOING_RELATIONS(uri)`), die parametrisierte SPARQL-UPDATE-Statements zurückgeben. Dieser Ansatz wurde gegenüber Query-Builder-Bibliotheken gewählt, da er bei überschaubarer Komplexität direktere Kontrolle bietet.

5.2.4. SPARQL-Client: Kommunikation mit Apache Fuseki

Der `SparqlClient` kapselt die HTTP-Kommunikation mit Apache Fuseki und bietet zwei Hauptmethoden:

- `executeQuery()` – Sendet SPARQL SELECT-Queries an den `/sparql`-Endpunkt (Lesezugriffe)
- `update()` – Sendet SPARQL UPDATE-Queries an den `/update`-Endpunkt (Schreibzugriffe)

5.3. LLM-gestützte Kurations-Assistenz

Die LLM-gestützte Kurations-Assistenz (konzeptionell beschrieben in Abschnitt 4.3.2) kombiniert Prompt Engineering, Validation und Human-in-the-Loop-Integration. Die folgenden Abschnitte dokumentieren die praktische Umsetzung der in Kapitel 4 eingeführten 5-Schritt-Pipeline.

5.3.1. Prompt Engineering: Kontext-Injektion und strukturierte Ausgabe

Der Kern der LLM-Assistenz liegt im Prompt-Design. Der Prompt muss drei Ziele erfüllen: Erstens muss er dem LLM den gesamten Kontext der Wissensbasis vermitteln (alle Phänomene, Relationstypen, Tags). Zweitens muss er klare Anweisungen zur Generierung von Vorschlägen geben. Drittens muss er eine strukturierte, maschinenlesbare Ausgabe erzwingen (JSON-Format mit definierten Feldern).

Ein vereinfachter Ausschnitt des generierten Prompts:

```
You are a critical phenomena relationship expert. Analyse the given
phenomenon, suggest relationships, and improve missing or weak metadata.

**Phenomenon Overview:**
English label: "Heavy Rain"
German label: "Starkregen"
English description: "Intensive precipitation..." (120 chars)
German description: Not provided

**Available Relationship Types:** could_lead_to, synergizes_with, ...

**All Available Phenomena (ID and label):**
{ id: "CP_001", label: "Rain" }
{ id: "CP_002", label: "Fog" }
... (297 weitere)

**Required JSON Response Format:**
{ "reasoning": "...",
  "suggested_relationships": [
    {"relationUri": "concretization_of", "targetNode": "CP_001",
     "confidence": 0.85}
  ],
  "suggested_tags": [{"tagId": "Weather", "confidence": 0.9}]
}
```

Listing 5: LLM-Prompt-Struktur (gekürzt, vollständige Version in Anhang A.2.1)

Der Prompt enthält zudem Qualitätshinweise: Das LLM soll nur Relationen vorschlagen, wenn es hohe Konfidenz hat (Confidence ≥ 0.5). Labels und Beschreibungen sollen bestimmte Mindestlängen erfüllen. Diese Hinweise reduzieren spekulative Vorschläge und erhöhen die Präzision.

5.3.2. Validation & Grounding: Halluzinations-Filterung

LLMs können halluzinieren – sie erfinden plausibel klingende, aber nicht existierende Phänomene oder Relationen. Die Validation-Logik im `CurationService` filtert solche Halluzinationen durch systematischen Abgleich mit der Wissensbasis (siehe Listing A.2.2 im Anhang für Details).

Die Validation erfolgt in drei Schritten:

1. **Lookup-Tabellen-Generierung:** Für effizienten Abgleich werden Lookup-Tabellen für Relationstypen und Phänomene aufgebaut. Diese ermöglichen fehlertolerante Validierung: Das LLM kann Phänomene wahlweise über ihre ID (z.B. `CP_013`), URI oder Labels (englisch/deutsch) referenzieren – alle Varianten werden auf die kanonische ID aufgelöst.

2. **Einzelvalidierung:** Jeder LLM-Vorschlag wird gegen die Lookup-Tabellen validiert:

- Existiert der Relationstyp in der Wissensbasis?
- Existiert das Zielphänomen (unabhängig davon, ob per ID oder Label referenziert)?
- Ist der Confidence-Score valide (zwischen 0 und 1)?

Nur Vorschläge, die alle Checks bestehen, werden akzeptiert.

3. **Deduplizierung:** Doppelte Vorschläge (gleiche Relation + Zielphänomen) werden herausgefiltert.

Diese Validation ist entscheidend für die Datenqualität: Ohne sie könnten halluzinierte Vorschläge in die Wissensbasis gelangen und Inkonsistenzen verursachen. Die fehlertolerante Referenzierung kompensiert die Variabilität in LLM-Ausgaben – das Modell muss nicht exakt das technische Format treffen, sondern kann Phänomene natürlich benennen.

5.3.3. Confidence-Gruppierung für Human-in-the-Loop

Die validierten Vorschläge werden nach *Confidence Scores* gruppiert – numerischen Vertrauenswerten zwischen 0 und 1, die angeben, wie sicher das LLM von einem Vorschlag ist. Ein Score von 0.9 bedeutet „sehr sicher“, 0.5 bedeutet „unsicher“. Die Vorschläge werden in drei Kategorien eingeteilt:

- **High Confidence (> 0.8):** Vorschläge mit hoher Sicherheit
- **Medium Confidence (0.5–0.8):** Vorschläge mit mittlerer Sicherheit
- **Low Confidence (< 0.5):** Vorschläge mit niedriger Sicherheit (werden herausgefiltert)

Im Frontend werden High-Confidence-Vorschläge automatisch zur Relation-Liste hinzugefügt und visuell als LLM-generiert gekennzeichnet. Sie können jederzeit manuell abgelehnt werden. Medium-Confidence-Vorschläge werden in einer separierten Sektion angezeigt. Low-Confidence-Vorschläge werden nicht angezeigt.

Dieses Confidence-basierte Filtering gilt sowohl bei der Node-Creation als auch beim Update auf der Detailseite – das Human-in-the-Loop-Prinzip gewährleistet, dass kein Vorschlag ohne Möglichkeit zur Ablehnung in die Wissensbasis übernommen wird.

5.3.4. Provider-Implementierung

Die in Abschnitt 4.3.2 beschriebene Provider-Abstraktion wurde als abstrakte Klasse mit drei Kernmethoden implementiert:

- `generateSuggestions()` – LLM-Anfragen
- `isAvailable()` – Health-Checks

- `loadConfig()` – Konfiguration

Die vollständige Implementierung findet sich in Listing A.2.3 im Anhang.

Die konkrete Implementierung `OpenRouterProvider` kommuniziert mit dem OpenRouter-Service, der Zugriff auf mehrere LLM-Anbieter bietet (OpenAI, Anthropic, Google, Meta).

Die Provider-Auswahl erfolgt zur Laufzeit über eine Umgebungsvariable (`CURATION_LLM_PROVIDER`). Die Factory liest diese Variable beim Server-Start und instantiiert den entsprechenden Provider. Dadurch kann der LLM-Anbieter gewechselt werden, ohne den TypeScript-Code neu zu kompilieren – lediglich ein Neustart des Servers ist erforderlich.

Ein neuer Provider (z.B. für direkte OpenAI-API oder lokale Modelle via Ollama) erfordert lediglich die Implementierung der drei abstrakten Methoden und Registrierung in der Factory. Diese Flexibilität war essenziell für die iterative Entwicklung – während der Prototypen-Phase wurden verschiedene State-of-the-Art-Modelle evaluiert.

5.4. Event-gesteuerte Such-Synchronisation

Die in Abschnitt 4.2.2 beschriebene Dual-Storage-Architektur wird durch ereignisgesteuerte Synchronisation implementiert. Die folgenden Abschnitte dokumentieren die technische Umsetzung.

5.4.1. Domain-Events: EntityCreatedEvent und EntityUpdatedEvent

Domain-Events sind typisierte Objekte, die strukturelle Änderungen in der Wissensbasis repräsentieren:

```
export class EntityCreatedEvent {
  constructor(
    public readonly uri: string,
    public readonly entityType: string,
    public readonly data: any
  ) {}
}

export class EntityUpdatedEvent {
  constructor(
    public readonly uri: string,
    public readonly entityType: string,
    public readonly changes: any
  ) {}
}
```

Listing 6: Domain-Event-Definition

Diese Events werden nach erfolgreicher Persistierung in Fuseki emittiert (siehe Listing A.1.1 im Anhang, Zeile 28). Der EventEmitter-Pattern in NestJS ermöglicht asynchrone, entkoppelte Kommunikation zwischen Modulen.

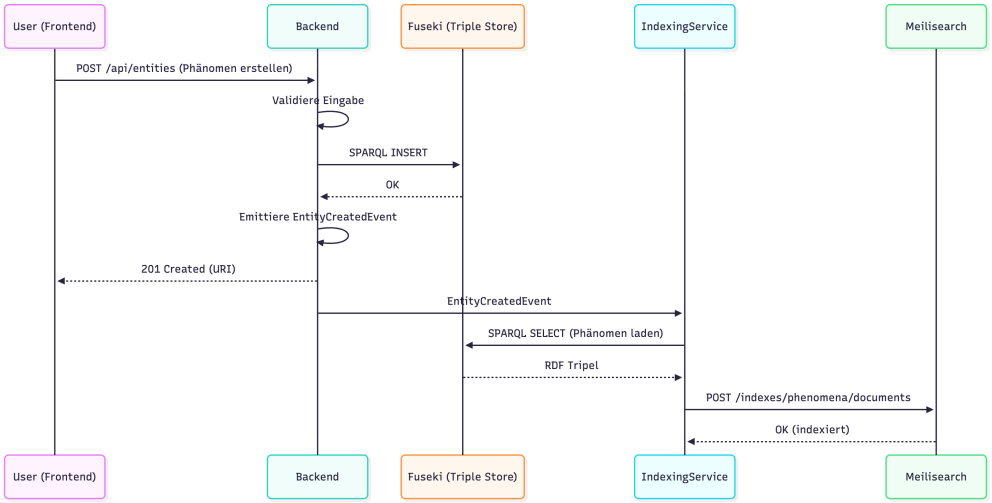

Abbildung 5.1.: Ereignisgesteuerte Synchronisation zwischen Fuseki und Meilisearch beim Anlegen eines Phänomens

5.4.2. Indexing-Service: Event-Listener und Meilisearch-Synchronisation

Der `IndexingService` hört auf Domain-Events und aktualisiert den Meilisearch-Index in drei Schritten:

1. **Event-Empfang:** Die `@OnEvent`-Decorators registrieren Listener für `entity.created` und `entity.updated` Events
2. **Datenabruf:** Das betroffene Phänomen wird via SPARQL SELECT aus Fuseki geladen
3. **Transformation & Indexierung:** Das SPARQL-Ergebnis wird in ein Meilisearch-Dokument transformiert und indexiert

Die Transformation fügt ein `searchableText`-Feld hinzu, das alle durchsuchbaren Felder kombiniert (Labels, Beschreibungen, Tags, Relationstypen) – dies ermöglicht die bilinguale Volltextsuche über alle Felder hinweg. Die vollständige Implementierung findet sich in Listing A.3.2 im Anhang.

Beim ersten System-Start wird der Meilisearch-Index vollständig aufgebaut: Alle Phänomene werden aus Fuseki geladen, transformiert und in einem Bulk-Operation indexiert. Dies gewährleistet initiale Konsistenz zwischen beiden Systemen.

5.5. Interaktive Graph-Exploration

Die Visualisierung der Nachbarschaft eines Phänomens ist ein Kernfeature der HazardDB. Sie ermöglicht Kuratoren, die Vernetzung eines Phänomens zu erkunden und durch verwandte Phänomene zu navigieren. Die Implementierung kombiniert automatisches Layout mit manueller Positionierung.

5.5.1. Nachbarschafts-Graph und Layout

Beim Öffnen der Detailseite eines Phänomens lädt das Frontend die direkten Nachbarn (alle Phänomene, die über ausgehende oder eingehende Relationen verbunden sind) und visualisiert sie als interaktiven Graphen. Abbildung 5.2 zeigt ein Beispiel für das Phänomen *Fog*.

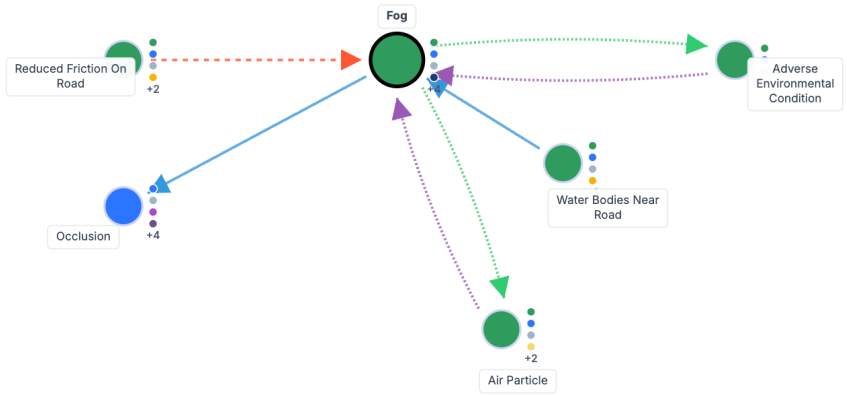

Abbildung 5.2.: Graph-Visualisierung der Nachbarschaft von *Fog*. Das zentrale Phänomen (dunkel) ist mit verwandten Phänomenen verbunden. Relationstypen werden als Kantenfarben kodiert.

5.5.2. Interaktive Funktionen

Die Graph-Visualisierung unterstützt fünf Interaktionsformen:

(1) Hover – Tooltip mit Details: Beim Hovern über einen Knoten wird ein Tooltip angezeigt, das das englische Label enthält. Dies ermöglicht schnelle Kontextinformation ohne Navigation (siehe Abbildung 5.3).

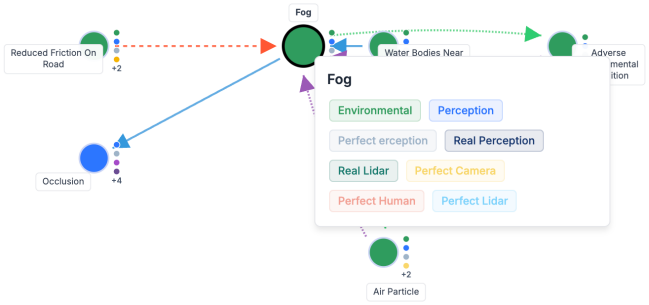

Abbildung 5.3.: Tooltip beim Hovern über einen Nachbar-Knoten im Graphen von *Fog*.

(2) Linksklick – Node-Selektion: Ein Linksklick selektiert einen Knoten. Bei geöffnetem Side-Panel auf der rechten Seite werden die Informationen der selektierten Node angezeigt (Label, Beschreibung, Tags, Relationen) ohne Navigation zur Detailseite (siehe Abbildung 5.4).

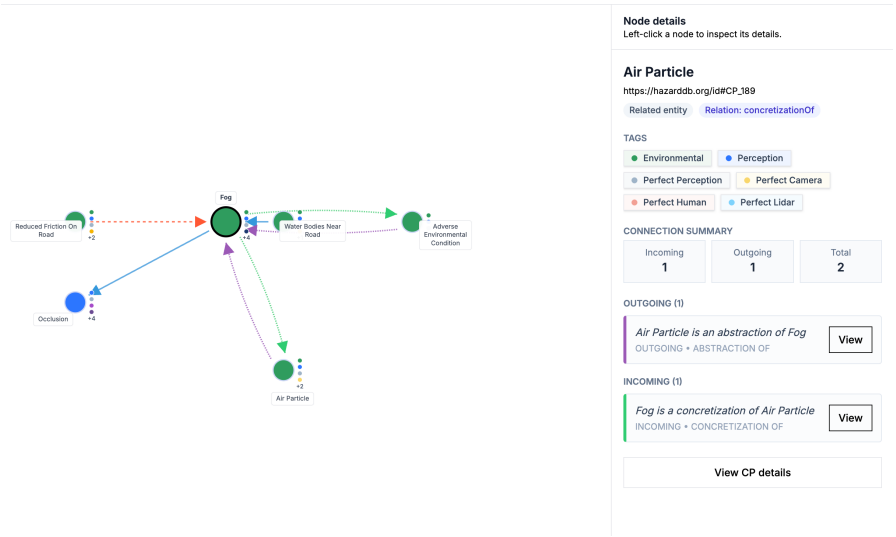

Abbildung 5.4.: Preview-Panel nach Linksklick auf *Air Particle*. Das Panel zeigt Schnellinformationen ohne Navigation zur Detailseite.

(3) Shift + Drag – Manuelle Positionierung: Knoten können mit gedrückter Shift-Taste per Drag-and-Drop verschoben werden. Dies ermöglicht Kuratoren, eigene Layouts zu erstellen – etwa zur Gruppierung von Phänomenen nach Kategorien oder zur Optimierung der Lesbarkeit bei dichten Graphen.

(4) Doppelklick – Graph expandieren: Ein Doppelklick auf einen Knoten

lädt dessen Nachbarn und fügt sie zum aktuellen Graphen hinzu. Dies ermöglicht schrittweise Exploration der Wissensbasis: Kuratoren starten bei einem Phänomen und expandieren sukzessive interessante Zweige (siehe Abbildung 5.5).

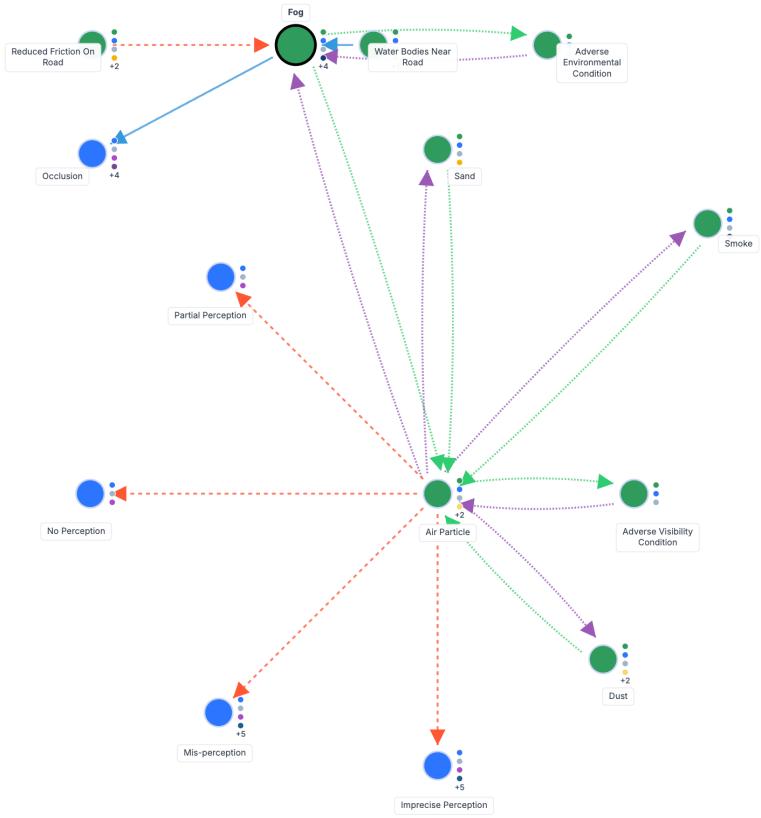

Abbildung 5.5.: Graph-Expansion via Doppelklick. Oben: Initiale Nachbarschaft von *Fog*. Unten: Nach Doppelklick auf *Air Particle* wurden dessen Nachbarn hinzugefügt (neu erschienene Knoten hervorgehoben).

(5) Expandierte Nachbarn verschieben: Auch nach der Expansion können alle Knoten (inklusive der neu hinzugefügten) mit Shift + Drag verschoben werden. Dies ermöglicht kontinuierliche Layout-Anpassung während der Exploration.

Diese Kombination aus manueller Kontrolle und interaktiver Exploration balanciert Benutzerfreundlichkeit und Flexibilität für komplexe Analysen.

5.5.3. Technische Umsetzung

Die Visualisierung nutzt D3.js, eine etablierte JavaScript-Bibliothek für datengetriebene Visualisierungen. D3.js berechnet das automatische Layout über physikalische

Simulationen (Knoten stoßen sich ab, Verbindungen ziehen sich an) und bietet Event-Handler für Interaktionen. Die vollständigen Code-Beispiele zur D3-Initialisierung und Tooltip-Generierung finden sich in Anhang A.4.1 und A.4.2.

5.6. Qualitätssicherung: Health Checks

Die Health-Check-Funktionalität bewertet die Datenqualität jedes Kritikalitätsphänomens anhand definierter Regeln. Ziel ist es, Kuratoren bei der Identifikation von Qualitätslücken zu unterstützen – etwa fehlende Beschreibungen, unvollständige Übersetzungen, isolierte Knoten ohne Relationen oder fehlende Kritikalitätsmetriken.

5.6.1. Regelbasiertes Scoring-System

Die Qualitätsbewertung basiert auf einem Punktesystem: Jedes Phänomen startet mit 100 Punkten. Für jede nicht erfüllte Regel werden Punkte abgezogen. Die Regeln sind nach Kritikalität gewichtet:

- **Labels (kritisch):** Fehlendes englisches Label: -25 Punkte, fehlendes deutsches Label: -20 Punkte
- **Beschreibungen:** Fehlende englische Beschreibung: -20 Punkte, zu kurze Beschreibung (<80 Zeichen): -10 Punkte
- **Vernetzung:** Keine Relationen: -20 Punkte, keine Tags: -10 Punkte, keine Metrik: -10 Punkte

Die Regeln prüfen systematisch Vorhandensein und Qualität von Labels, Beschreibungen, Relationen in beiden Sprachen, Tags und Metriken. Jede Regel besitzt einen Priority-Wert zur Sortierung im Frontend – Issues mit niedrigerer Priority-Nummer (höhere Dringlichkeit) werden zuerst angezeigt. Das Regelsystem ist schematisch aufgebaut und kann im Code jederzeit um zusätzliche Qualitätskriterien erweitert werden. Die vollständige Regel-Definition findet sich in Listing A.5.1 im Anhang.

Abbildung 5.6 zeigt die Health-Check-Übersicht im Frontend.

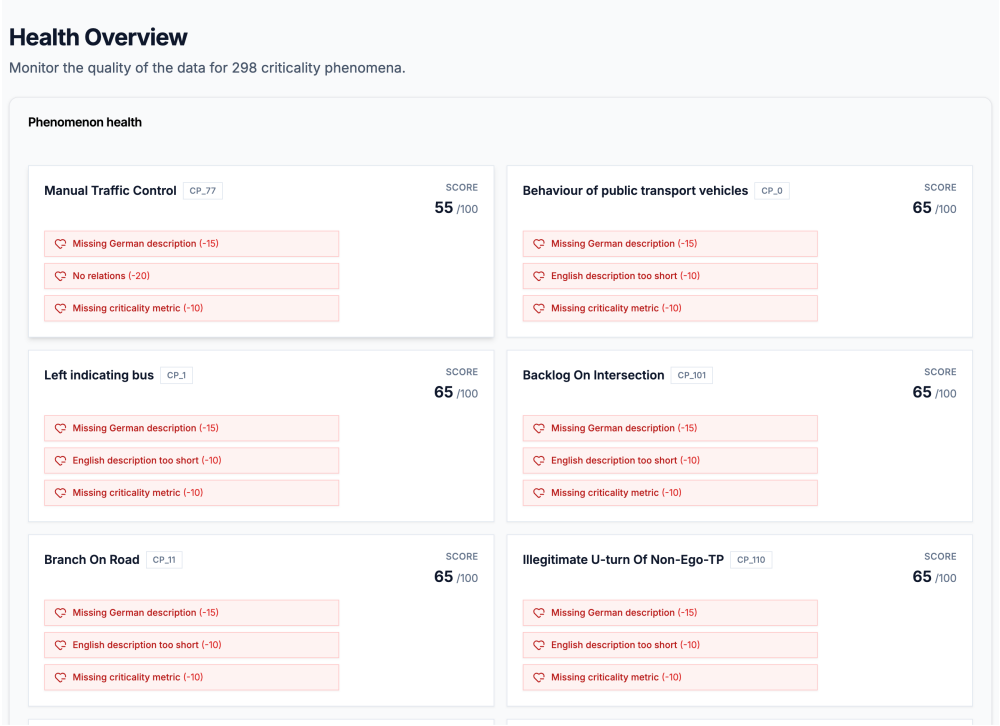

Abbildung 5.6.: Health-Check-Übersicht. Die Liste zeigt alle Phänomene mit Scores und erkannten Issues zur Identifikation von Qualitätslücken.

5.6.2. Health-Check-Ausführung

Der `HealthCheckService` führt die Qualitätsbewertung in drei Schritten durch:

1. **Metadaten-Abfrage:** Eine aggregierende SPARQL-Query (`ENTITY_HEALTH_OVERVIEW`) lädt alle relevanten Metadaten: Anzahl englischer/deutscher Labels, Länge der Beschreibungen, Anzahl Relationen, Tags und Metriken
2. **Regel-Evaluation:** Jede definierte Health-Check-Regel wird gegen die Metadaten geprüft. Fehlgeschlagene Checks werden gesammelt und deren Penalties summiert
3. **Score-Berechnung:** Der finale Score wird berechnet (100 Punkte - Summe der Penalties), begrenzt auf minimal 0 Punkte

Die Metadaten-Aggregation erfolgt effizient in einer einzigen SPARQL-Query via `COUNT`- und `STRLEN`-Funktionen. Issues werden nach Priority sortiert zurückgegeben – das Frontend zeigt kritische Mängel zuerst an. Die vollständige Implementierung findet sich in Listing A.5.2 im Anhang.

Abbildung 5.7 zeigt das Health-Check-Ergebnis für ein einzelnes Phänomen.

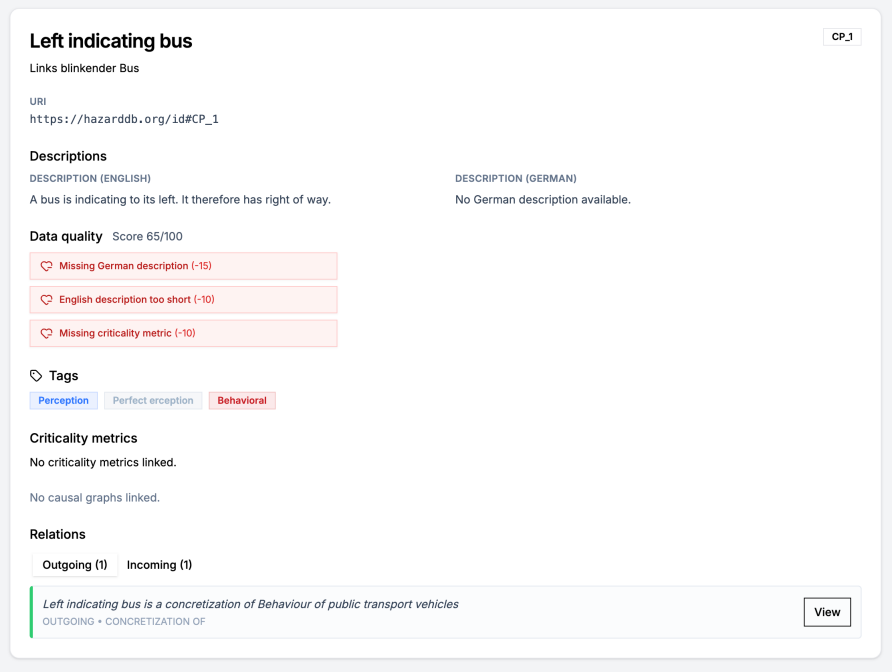

Abbildung 5.7.: Health-Check-Detail für ein Phänomen. Issues werden priorisiert angezeigt, der Score visualisiert die Gesamtqualität.

5.7. Kausalgraphen-Implementierung

Die in Abschnitt 4.5 beschriebene Kausalgraphen-Architektur wird durch einen DOT-Parser, RDF-Speicherlogik und Frontend-Visualisierung umgesetzt.

5.7.1. DOT-Parsing und Validation

Der `parseDotSubset`-Parser extrahiert Knoten und Kanten aus DOT-Quellcode:

- **Node-Syntax:** `id [label="...", uri="...", class="NodeKind"]`
- **Edge-Syntax:** `source -> target [label="...", predicate="..."]`

Die Validation prüft, ob URIs auf existierende Entities zeigen und ob `class` valide ist (`CriticalityPhenomenon`, `CriticalityMetric`, `CausalVariable`).

5.7.2. RDF-Speicherung

Causal Graphs werden in RDF mit folgender Struktur gespeichert:

```
<https://hazarddb.org/id#CG_safety_scenario>
  a haz:CausalGraph ;
```

```
  rdfs:label "Safety Scenario"@en ;
  haz:dotSource "digraph { ... }" ;
  haz:hasCausalEdge <CG_1_E1>, <CG_1_E2>, <CG_1_E3> ;
  haz:includesEntity <CP_13> ;              # Fog (AUTO-DETECTED)
  haz:includesEntity <CM_time-scale__TTB> . # Time To Brake (AUTO-DETECTED)

<CG_1_E1>
  a haz:CausalEdge ;
  haz:source <CP_13> ;                      # Fog
  haz:target <CG_1_N1> ;                    # Low Visibility
  rdfs:label "causes" .

<CG_1_N1>  # CausalVariable (synthetisch)
  a haz:CausalNode ;
  rdfs:label "Low Visibility" ;
  haz:inGraph <CG_safety_scenario> .
```

5.7.3. Automatische `haz:includesEntity` Logik

Die `writeGraph`-Methode iteriert über alle Nodes:

```
nodes.forEach((n) => {
  if (n.uri && (n.kind === 'CriticalityPhenomenon' ||
                n.kind === 'CriticalityMetric')) {
    includesEntities.add(n.uri);  // ← AUTO-DETECTION
  }
});
```

Dies ermöglicht SPARQL-Queries wie:

```
SELECT ?graph WHERE {
  ?graph haz:includesEntity <CP_13>  # Findet alle Graphs mit "Fog"
}
```

5.7.4. Frontend-Visualisierung

Die Visualisierung nutzt `d3-graphviz` für Live-Preview beim Editieren. Änderungen im DOT-Quellcode werden on-the-fly in SVG-Graphen gerendert. Die Implementierung unterstützt Export als DOT-Datei und SVG-Download.

5.8. Kritikalitätsmetriken-Implementierung

Die Integration von Criticality Metrics erfolgt über den `ReadTheDocsCatalogService`, das `MetricRepository` und Frontend-Komponenten für Embedded Documentation.

5.8.1. ReadTheDocsCatalogService

Der Service fetcht den Metriken-Katalog von der ReadTheDocs-API:

```
async listAll(): Promise<CatalogMetric[]> {
  const cached = this.cache.get('catalog');
  if (cached && !this.isExpired(cached.timestamp)) {
    return cached.data;  // 6h TTL
  }
  const catalog = await this.fetchCatalog();
  this.cache.set('catalog', { data: catalog, timestamp: Date.now() });
  return catalog;
}
```

Die API `searchindex.js` liefert `docnames` und `titles`, aus denen `CatalogMetric`-Objekte konstruiert werden.

5.8.2. Metric Repository

Das Repository speichert Minimal-Metadaten in RDF:

```
<https://hazarddb.org/id#CM_time-scale__TTB>
  a haz:CriticalityMetric ;
  dcterms:identifier "time-scale/TTB" ;        # docSlug
  haz:slug "time-scale__TTB" ;                 # routeSlug (URL-safe)
  rdfs:label "Time To Brake (TTB)"@en ;        # displayName
  haz:metricCategorySlug "time-scale" ;        # category
  schema:url "https://criticality-metrics.readthedocs.io/en/latest/time-scale/TTB.html" .
```

5.8.3. Frontend: Embedded Documentation

Die Detail-Seite einer Kritikalitätsmetrik lädt die ReadTheDocs-Dokumentation via Embed-API:

```
const embedUrl = `https://criticality-metrics.readthedocs.io/_/api/v3/embed/?url=${docUrl}`;
const response = await fetch(embedUrl);
const html = await response.json();
```

LaTeX-Formeln werden mit KaTeX clientseitig gerendert.

5.9. RDF-Export und -Import

Die HazardDB muss die Wissensbasis als RDF/Turtle exportieren und reimportieren können. Dies dient mehreren Zwecken: Versionierung (Snapshots für Git), Interoperabilität mit Ontologie-Editoren (z.B. Protégé) und Backup/Restore.

5.9.1. Export-Funktionalität

Der Export erfolgt über die n3-JavaScript-Bibliothek, die W3C-konformes RDF/Turtle generiert. Mit einem Klick wird die gesamte Wissensbasis als Turtle-Datei heruntergeladen (siehe Abbildung 5.8).

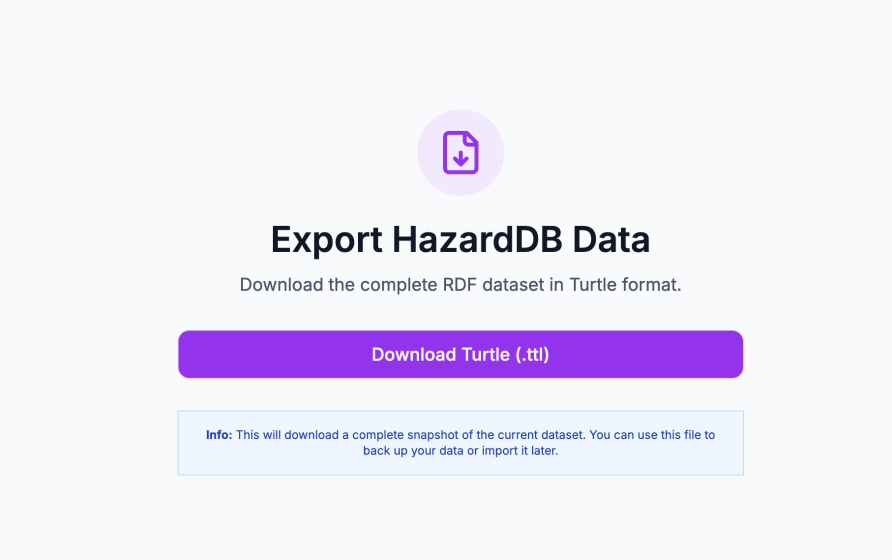

Abbildung 5.8.: Export-Interface. Kuratoren können die gesamte Wissensbasis als Turtle-Datei (.ttl) exportieren.

Die Export-Implementierung lädt alle Tripel aus Fuseki via SPARQL SELECT, serialisiert sie als Turtle und preserviert Language Tags (`@de`, `@en`) sowie Typed Literals (z.B. `xsd:integer`). Das Ergebnis ist ein Turtle-Dokument (siehe Listing A.6.1 im Anhang für technische Details).

5.9.2. Import und Index-Rebuild

Der Import löscht zunächst den bestehenden Fuseki-Dataset und lädt die neuen RDF-Daten. Nach erfolgreichem Upload wird automatisch der Meilisearch-Index vollständig neu aufgebaut – dies gewährleistet Konsistenz zwischen beiden Systemen (siehe Abbildung 5.9).

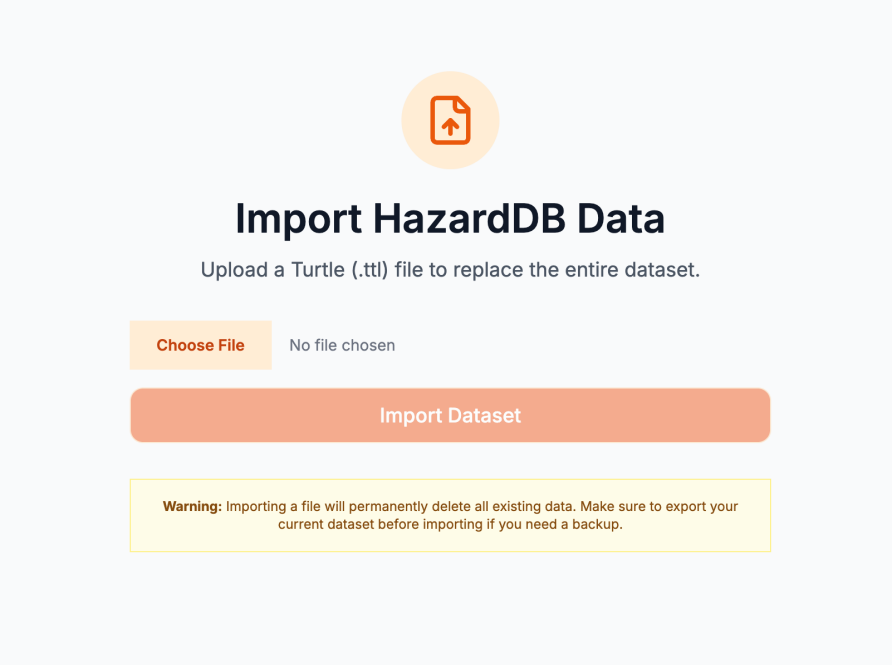

Abbildung 5.9.: Import-Interface. Nach erfolgreicher Validierung wird die Wissensbasis ersetzt und der Suchindex neu aufgebaut.

Die Operation ist idempotent – mehrfache Imports mit denselben Daten führen zum gleichen Endzustand (siehe Listing A.6.2 im Anhang für Implementierungsdetails).

Anwendungsfall: Ein Kurator exportiert die Wissensbasis am Freitagabend, bearbeitet sie über das Wochenende mit Protégé (z.B. Hinzufügen neuer Relationstypen), und importiert sie am Montagmorgen zurück. Der automatische Index-Rebuild gewährleistet, dass die Suche sofort alle Änderungen berücksichtigt.

5.10. Reflexion und Erkenntnisse

Die Implementierung der HazardDB lieferte mehrere Erkenntnisse über die praktische Umsetzung semantischer Wissensbasen mit modernen Web-Technologien.

Was hat gut funktioniert?

- **Provider-Abstraktion für LLMs:** Der Wechsel zwischen verschiedenen Modellen erforderte nur Konfigurationsänderungen, keine Code-Anpassungen. Diese Flexibilität war essenziell für die iterative Entwicklung – während der Prototypen-Phase wurden verschiedene State-of-the-Art-Modelle evaluiert.
- **RDF Schema-Evolution mit OPTIONAL-Patterns:** Neue Entities und Klassen wurden während der Entwicklung mehrfach hinzugefügt (z.B. Criticali-

ty Metrics, Causal Graphs), ohne dass bestehende SPARQL-Queries angepasst werden mussten. `OPTIONAL`-Patterns machen Queries robust gegen fehlende Properties – ein Vorteil gegenüber strikten relationalen Schemas.

Was war herausfordernd?

- **SPARQL Query Debugging:** Fuseki bietet keine IDE-Unterstützung, kein Syntax-Highlighting und kryptische Fehlermeldungen. Für komplexe Queries (z.B. Graph-Nachbarschaft mit Inverse Relations) war trial-and-error unvermeidbar. Eine SPARQL Query Builder-Library (z.B. rdflib.js) hätte die Entwicklung beschleunigt.
- **LLM-Modellauswahl:** Lokale, selbst-gehostete Modelle mit 8 Milliarden Parametern (z.B. Llama 3.1 8B, Mistral 7B) lieferten zu viele Halluzinationen und unzureichende Reasoning-Fähigkeiten für die Kurations-Aufgabe. Nur State-of-the-Art-Modelle (GPT, Claude, Gemini) erreichten gute Qualität. Dies verhindert Offline-Betrieb, da diese Modelle Cloud-APIs erfordern.
- **LLM Prompt Engineering:** Der initiale Prompt lieferte viele spekulative Vorschläge mit niedriger Confidence. Durch iterative Optimierung wurden Grounding-Mechanismen eingefügt, Output-Beispiele im Prompt integriert und das LLM instruiert, eigene Confidence-Scores zu generieren. Dies reduzierte halluzinierte Vorschläge signifikant. Die parallel entwickelte Validation-Logik filtert zusätzlich Edge-Cases – etwa wenn das LLM englische Labels statt URIs verwendet oder nicht-existierende Relationstypen vorschlägt.

Übertragbarkeit

Die entwickelte Architektur ist domänenunabhängig: Die Kombination aus RDF Triple Store (für semantische Konsistenz), spezialisierter Suchinfrastruktur (für performante Volltextsuche) und LLM-Assistenz (für skalierbare Kuration) funktioniert für beliebige Wissensbasen mit strukturierten, vernetzten Daten.

6. Evaluation

6.1. Einleitung

Die vorangegangenen Kapitel haben die Anforderungen, Architektur und Implementierung des HazardDB-Webtools dokumentiert. Dieses Kapitel validiert die entwickelte Lösung durch szenariobasierte Evaluation anhand repräsentativer Aufgabenstellungen. Die Evaluation fokussiert sich auf die zentrale Forschungsfrage dieser Arbeit:

> *Wie muss ein interaktives Webtool gestaltet sein, um Anwender (a) bei einer intuitiven, pfadbasierten Exploration der HazardDB zu leiten und (b) eine effiziente und konsistente Pflege der Wissensbasis zu gewährleisten?*

Die Beantwortung erfolgt durch praktische Demonstration der implementierten Funktionalitäten entlang typischer Nutzungsszenarien. Dabei werden die in Kapitel 2 formulierten User Stories als Grundlage für realistische Aufgabenstellungen herangezogen.

Methodisches Vorgehen

Die Evaluation folgt einem aufgabenbasierten Ansatz: Für jede User Story wird eine konkrete Aufgabe definiert und schrittweise durchgeführt. Die Workflows werden durch Screenshots dokumentiert, die die Benutzeroberfläche und Interaktionsmuster illustrieren. Die Bewertung erfolgt qualitativ anhand der Evaluationskriterien Intuitivität, Effizienz und Vollständigkeit.

Evaluationskriterien

Aus der Forschungsfrage lassen sich zwei zentrale Evaluationskriterien ableiten:

(a) Intuitive, pfadbasierte Exploration: Ermöglicht das Tool Nutzern, ausgehend von einem abstrakten Kritikalitätsphänomen schrittweise zu konkreteren Ausprägungen zu navigieren und dabei relevante Relationen zu verstehen? Sind die verschiedenen Relationstypen klar unterscheidbar? Funktioniert die Kombination aus Tree View, Detailseiten und Graph-Visualisierung zusammen?

(b) Effiziente und konsistente Kuration: Unterstützt das Tool Kuratoren durch geführte Formulare, intelligente Vorschläge und Qualitätsprüfungen? Reduziert

die LLM-gestützte Assistenz den manuellen Aufwand? Werden Qualitätslücken systematisch identifiziert und priorisiert?

Die folgenden Szenarien adressieren diese Kriterien systematisch.

6.2. Evaluationsmethodik

Die Evaluation erfolgt entlang von vier Szenarien, die jeweils eine User Story (siehe Abschnitt 2.3) operationalisieren. Jedes Szenario besteht aus:

1. **Aufgabenbeschreibung**: Konkrete Aufgabenstellung mit Begründung der gewählten Beispiele
2. **Workflow-Dokumentation**: Schrittweise Durchführung mit Screenshot-Dokumentation
3. **Bewertung**: Qualitative Einschätzung anhand der Evaluationskriterien

Die Screenshots werden direkt im Fließtext platziert, da sie essentiell für das Verständnis der Workflows sind. Jeder Screenshot wird mit einer Caption versehen, die den dargestellten Schritt erläutert.

Die Szenarien wurden bewusst so gewählt, dass sie die Kernfunktionalitäten des Systems abdecken und gleichzeitig die identifizierten Probleme aus Kapitel 1 adressieren:

- **Szenario 1** adressiert Problem 1 (umständliche Exploration)
- **Szenario 2** adressiert Problem 2 (fehleranfällige Kuration)
- **Szenario 3** adressiert Problem 3 (gefährdete Nachhaltigkeit durch Qualitätsverlust)
- **Szenario 4** demonstriert erweiterte Funktionalität für Dokumentation kausaler Hypothesen

6.3. Szenario 1: Pfadbasierte Exploration

6.3.1. Aufgabenstellung

Dieses Szenario operationalisiert User Story US1 (siehe Abschnitt 2.3):

> *Als Nutzer möchte ich ausgehend von einem abstrakten Kritikalitätsphänomen (z.B. „widrige Wetterbedingungen“) schrittweise zu konkreteren Ausprägungen navigieren können, um relevante Einflussfaktoren für meine Sicherheitsargumentation zu identifizieren.*

Konkrete Aufgabe: Exploriere das Kritikalitätsphänomen `CP_273` „schlechte Sichtverhältnisse“ und identifiziere dessen Relationen zu anderen Phänomenen.

Begründung der Wahl: `CP_273` wurde bewusst gewählt, da es eine sinnvolle Abstraktionsebene repräsentiert und mehrere direkte Konkretisierungen besitzt (z.B. „Air Particle“, das wiederum in „Fog“, „Smoke“, „Dust“ konkretisiert wird). Die hierarchische Struktur eignet sich gut zur Demonstration der schrittweisen Navigation durch die Abstraktionshierarchie und des Nachladens von Nachbar-Phänomenen in der Graph-Visualisierung.

6.3.2. Workflow-Dokumentation

Der Explorationsprozess besteht aus mehreren aufeinander aufbauenden Schritten, die verschiedene Navigationsmechanismen kombinieren.

Schritt 1: Navigation über Tree View

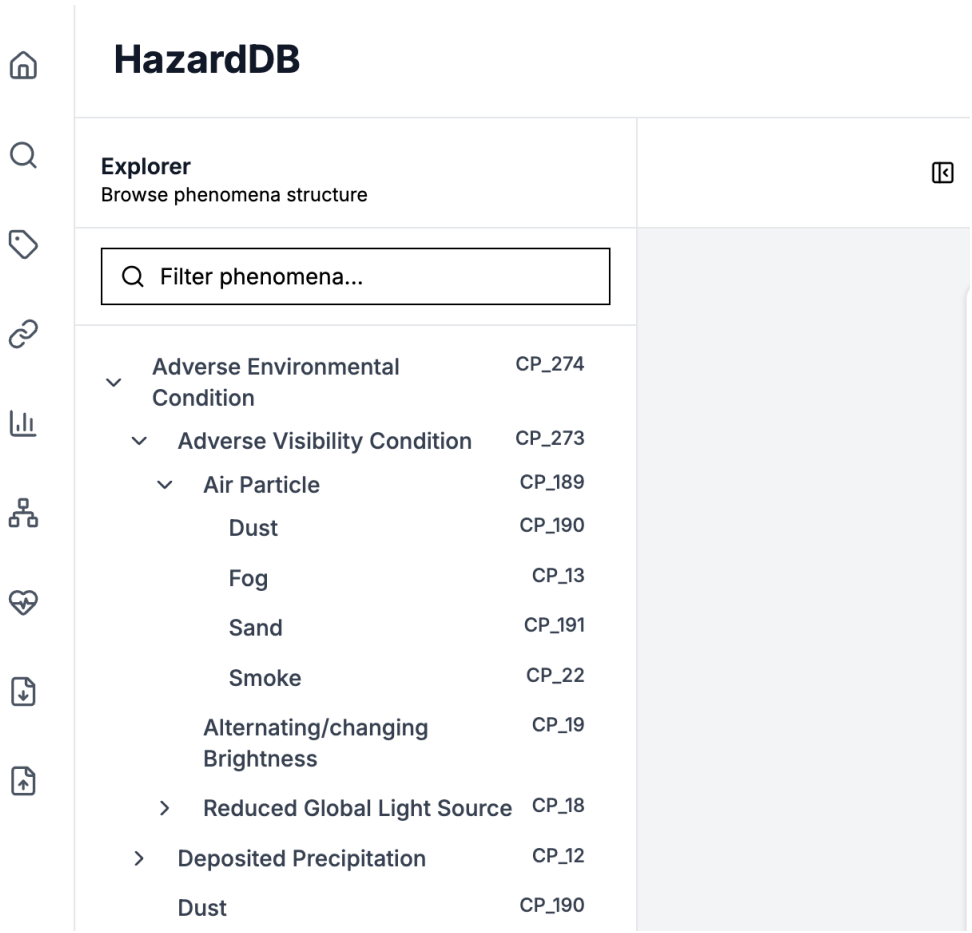

Abbildung 6.1.: Hierarchische Navigation über die persistente Tree View. Die Abstraktionshierarchie ermöglicht schrittweises Durchklicken von abstrakten zu konkreten Phänomenen.

Die persistente Tree View im linken Seitenbereich zeigt die Abstraktionshierarchie der Kritikalitätsphänomene. Durch Aufklappen der Hierarchie-Ebenen lässt sich CP_273 identifizieren und durch Klick auf den Knoten zur Detailseite navigieren.

Schritt 2: Detailseite mit Relationsübersicht

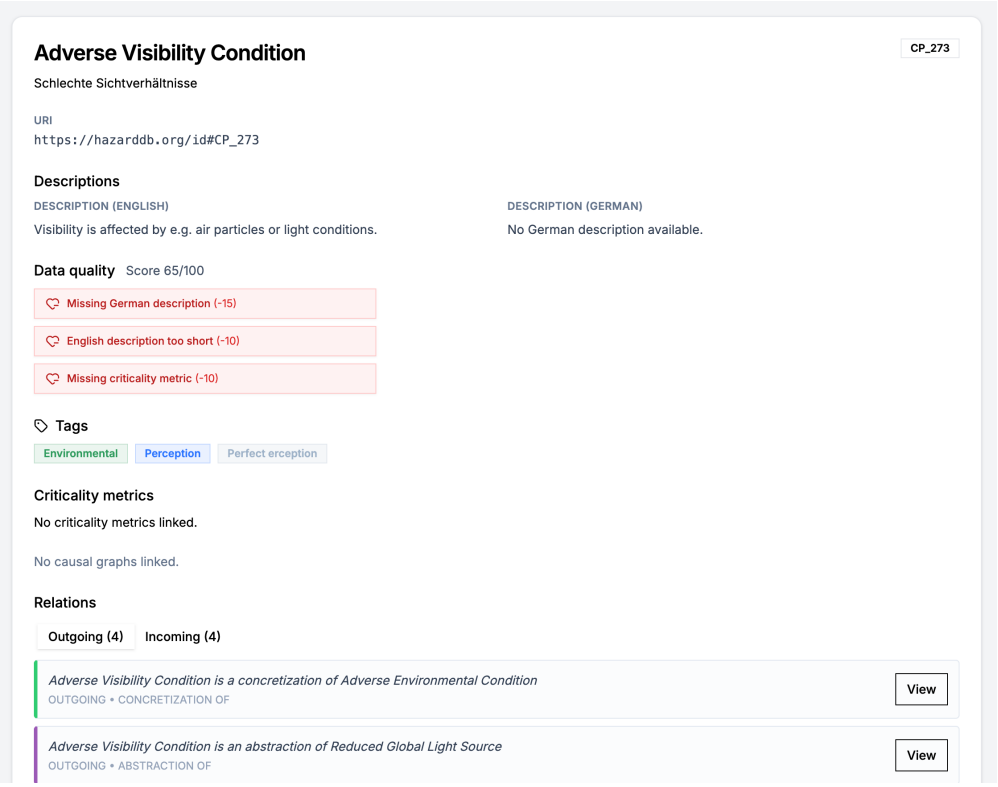

Abbildung 6.2.: Detailseite von CP_273 mit strukturierter Darstellung aller ausgehenden und eingehenden Relationen. Relationstypen sind farblich unterschieden.

Die Detailseite bündelt alle Informationen zu `CP_273`: deutsche und englische Labels, Beschreibungen, Tags sowie alle Relationen gruppiert nach Typ und Richtung. Die tabellarische Darstellung ermöglicht schnelles Erfassen der Nachbarschaft.

Schritt 3: Graph-Visualisierung mit Nachbarn-Nachladen

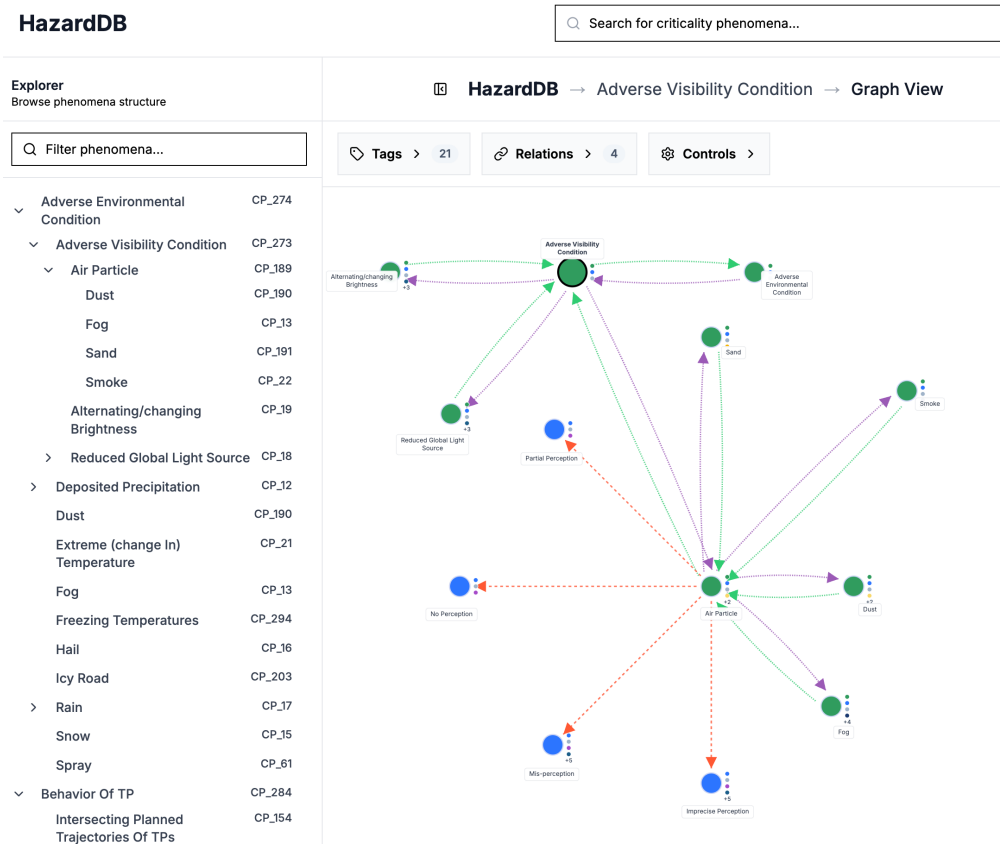

Abbildung 6.3.: Interaktive Graph-Visualisierung zeigt `CP_273` (zentral) und alle direkt verbundenen Phänomene. Durch Doppelklick auf den Knoten „Air Particle“ wurden dessen Nachbar-Phänomene nachgeladen („Fog“, „Smoke“, „Dust“, „Sand“). Relationstypen sind durch unterschiedliche Kantenfarben/-stile visualisiert.

Die Graph-Visualisierung bietet eine alternative, visuelle Perspektive auf die Relationen. Knoten können per Hovering inspiziert und per Shift-Drag neu positioniert werden. Kanten zeigen den Relationstyp. Ein **Doppelklick auf einen Knoten** lädt dessen Nachbar-Phänomene nach und erweitert so schrittweise den sichtbaren Graphausschnitt. Im Beispiel wurde „Air Particle“ expandiert, wodurch dessen Konkretisierungen („Fog“, „Smoke“, „Dust“, „Sand“) sichtbar werden – dies demonstriert die hierarchische Verfeinerung: *schlechte Sichtverhältnisse* → *Air Particle* → *Fog/Smoke/Dust*.

Schritt 4: Node-Details-Panel

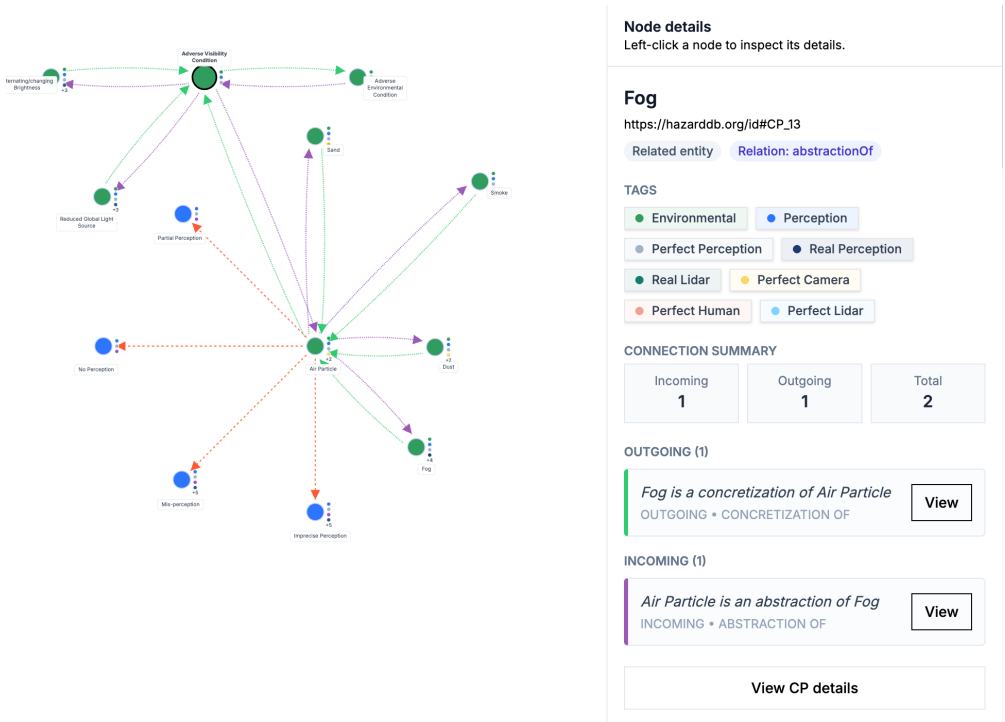

Abbildung 6.4.: Das Node-Details-Panel (rechts) wird über einen Button in der Topbar ein-/ausgeblendet. Durch Linksklick auf einen Graphknoten wird dieser im Panel angezeigt. Es zeigt Labels, Beschreibungen und Relationen des ausgewählten Phänomens, ohne die aktuelle Seite zu verlassen.

Ein **ein-/ausklappbares Details-Panel** kann über die Topbar geöffnet werden. Durch Linksklick auf einen Knoten im Graph wird dieser im Panel angezeigt und zeigt Labels, Beschreibungen und Relationen des ausgewählten Phänomens. Weitere Klicks auf andere Knoten aktualisieren das Panel, ohne die Graph-Visualisierung zu verlassen. Die Tree View im linken Bereich bleibt dabei unverändert geöffnet und markiert das Haupt-Phänomen der Seite – dies ermöglicht durchgängige Orientierung in der Hierarchie.

6.3.3. Bewertung

Intuitivität: Die Exploration ist selbsterklärend und folgt etablierten Interaktionsmustern. Die persistente Tree View ermöglicht hierarchische Navigation ohne Verlust des Kontexts. Die Detailseite strukturiert Informationen klar nach Typ (Labels, Beschreibungen, Relationen). Die Graph-Visualisierung bietet eine visuelle Ergänzung, die vor allem bei der schrittweisen Verfeinerung durch Nachbarn-Nachladen hilft.

Das Node-Details-Panel ermöglicht schnelle Inspektion mehrerer Phänomene ohne Seitenwechsel.

Relationstypen: Die verschiedenen Relationstypen (*abstraction_of*, *concretization_of*, *could_lead_to*, *synergizes_with*) sind sowohl textuell als auch visuell klar unterscheidbar. Auf der Detailseite erfolgt die Gruppierung nach Typ, im Graph werden unterschiedliche Kantenfarben oder -stile verwendet.

Zusammenspiel der Komponenten: Tree View, Detailseite, Graph-Visualisierung und Node-Details-Panel funktionieren nahtlos zusammen. Die persistente Tree View verhindert Desorientierung, die Detailseite liefert vollständige Information, der Graph ermöglicht visuelles Explorieren hierarchischer Strukturen und das Details-Panel erlaubt schnelle Inspektion ohne Kontextverlust. Die Navigation zwischen Phänomenen erfolgt flüssig ohne Full-Page-Reloads, was Anforderung NF2 (Performance und schnelle Reaktionszeiten) adressiert. *Hinweis:* Die Graph-Visualisierung verwendet keine automatische Cluster-Erkennung – die räumliche Anordnung wird durch ein Kraft-basiertes Layout berechnet.

Fazit zu Evaluationskriterium (a): Das Tool ermöglicht intuitive, pfadbasierte Exploration. Die schrittweise Navigation von abstrakten zu konkreten Phänomenen ist durch die hierarchische Tree View unterstützt. Die Kombination mehrerer Visualisierungsformen (hierarchisch, tabellarisch, graphisch) bietet Flexibilität für unterschiedliche Explorationsziele.

6.4. Szenario 2: Hinzufügen eines neuen Phänomens mit LLM-Assistenz

6.4.1. Aufgabenstellung

Dieses Szenario operationalisiert User Story US2 (siehe Abschnitt 2.3):

> *Als Kurator möchte ich neue Kritikalitätsphänomene über ein geführtes Formular anlegen können, das mich bei der Eingabe von Name, Beschreibung, Tags und Relationen unterstützt. Dabei soll ein KI-Assistent mir Vorschläge für fehlende oder unvollständige Felder generieren können, um die Vollständigkeit der Einträge zu erhöhen und den Kurations-Aufwand zu reduzieren.*

Konkrete Aufgabe: Lege ein neues Kritikalitätsphänomen namens „Spurrillen“ an.

Begründung der Wahl: „Spurrillen“ ist ein straßenbezogenes Kritikalitätsphänomen, das in der bestehenden Wissensbasis noch nicht vorhanden ist und sich gut zur Illustration des LLM-gestützten Kurationsprozesses eignet: Der Assistent muss basierend auf minimalen Eingaben (Labels und Beschreibungen) plausible Relationen, Tags und weitere Details generieren.

6.4.2. Workflow-Dokumentation

Schritt 1: Öffnen des Create-Formulars

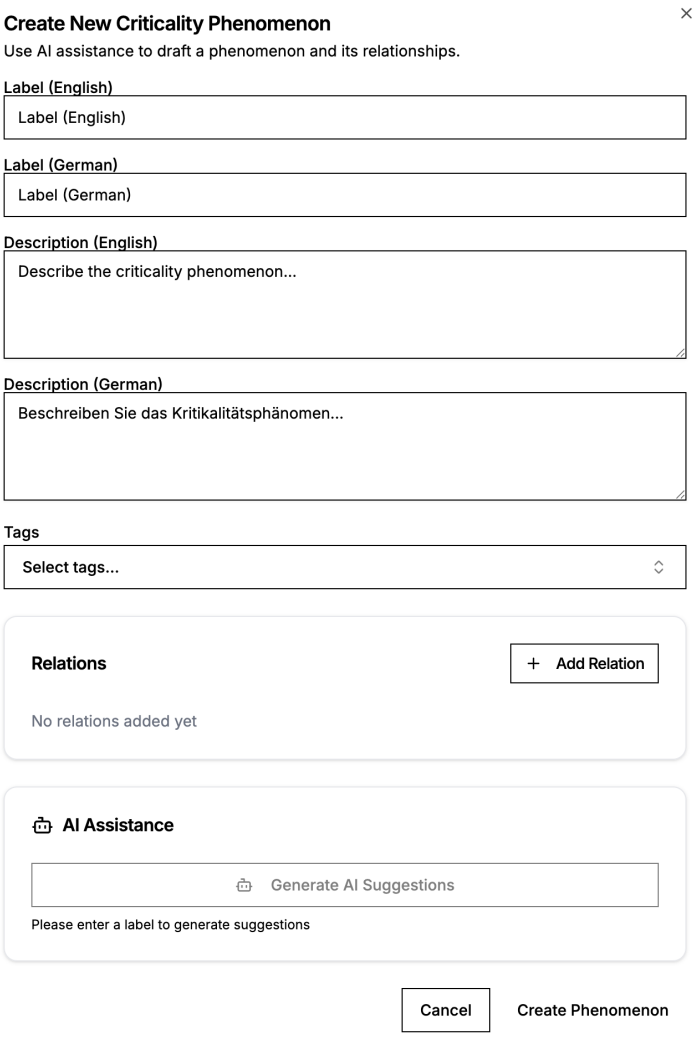

Abbildung 6.5.: Geführtes Formular zum Anlegen neuer Kritikalitätsphänomene. Alle Felder (Labels, Beschreibungen, Tags, Relationen) werden als Pflichtfelder behandelt, um eine konsistente Datenqualität zu gewährleisten. Ein LLM-Assistenz-Button ermöglicht das Anfordern intelligenter Vorschläge.

Das Create-Formular strukturiert die Eingabe in mehrere Bereiche: Labels (Deutsch/Englisch), Beschreibungen (Deutsch/Englisch), Tags und Relationen. Um eine hohe Datenqualität sicherzustellen, werden alle Felder als Pflichtfelder behandelt – bei versuchtem Absenden ohne vollständige Daten erscheint eine Fehlermeldung. Der

prominente „LLM-Vorschläge generieren“-Button signalisiert die Verfügbarkeit intelligenter Assistenz. Ein minimaler Input reicht aus, um das Kritikalitätsphänomen vollständig zu kuratieren.

Schritt 2: Auswahl der Parent-Relation mittels Combobox

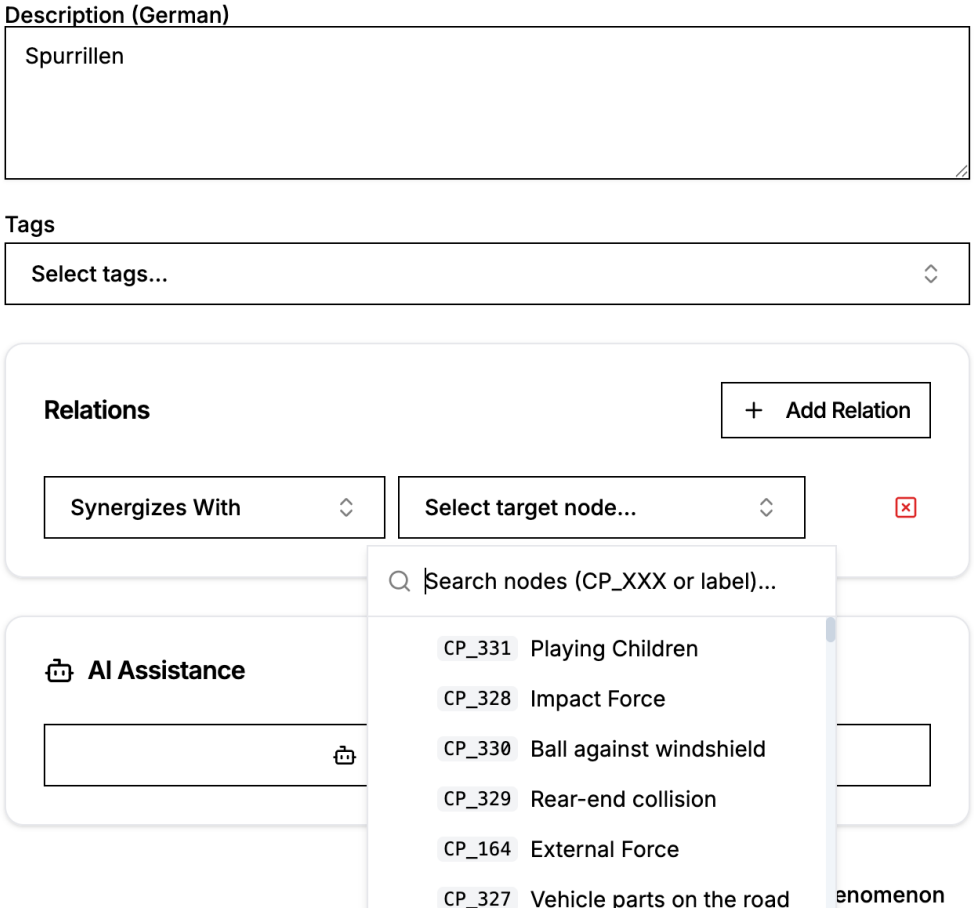

Abbildung 6.6.: Combobox-Komponente für die Auswahl von Relationen. Die Combobox lädt alle verfügbaren Kritikalitätsphänomene und ermöglicht einfaches Durchsuchen durch Scrollen oder Tippen der ersten Buchstaben.

Die Combobox-Komponente ermöglicht die manuelle Auswahl von Relationen zu anderen Kritikalitätsphänomenen. Für dieses Evaluationsszenario verzichten wir jedoch auf manuelle Eingaben und nutzen stattdessen die LLM-Assistenz, um Vorschläge für Relationen, Tags und Beschreibungen automatisch generieren zu lassen.

Schritt 3: Anfordern von LLM-Vorschlägen

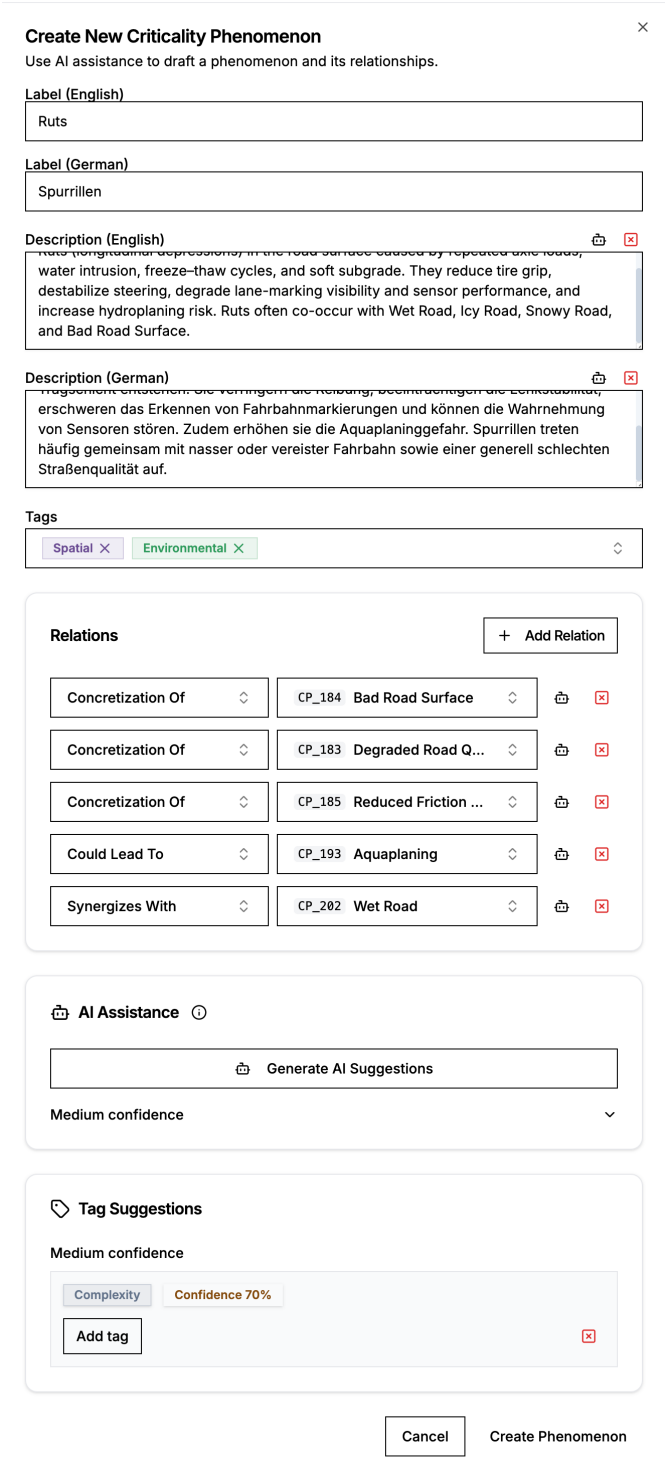

Abbildung 6.7.: LLM-generierte Vorschläge werden strukturiert präsentiert. Vorschläge mit hoher Konfidenz (Confidence $> 0{,}8$) werden automatisch in die Formularfelder übernommen und mit einem Roboter-Icon markiert. Vorschläge mit mittlerer Konfidenz sind ausgeklappt und müssen manuell geprüft werden.

Nach Klick auf den Assistenz-Button sendet das System die bereits eingegebenen Informationen (deutsches Label „Spurrillen“, englisches Label „Ruts“, sowie beide Labels jeweils in die Beschreibungsfelder kopiert) zusammen mit dem Kontext aller bestehenden Phänomene an das LLM. Nach einer Wartezeit von ca. 30–50 Sekunden werden die generierten Vorschläge zurückgegeben. Diese umfassen:

- **Beschreibungen** (Deutsch/Englisch): Vorgeschlagene Texte, die das Phänomen charakterisieren
- **Tags**: Vorgeschlagene Kategorisierungen (z.B. „Road Condition“, „Infrastructure“, „Surface“)
- **Zusätzliche Relationen**: Vorschläge für kausale Zusammenhänge oder Synergien

Confidence-basiertes Handling: Das System bewertet jeden Vorschlag mit einem Confidence-Score zwischen 0 und 1:

- **High-Confidence ($>$ 0,8):** Diese Vorschläge werden automatisch in die Formularfelder übernommen und mit einem Roboter-Icon markiert. Der Kurator kann sie durch Klick auf ein X wieder entfernen.
- **Medium-Confidence ($\leq$ 0,8):** Diese Vorschläge sind initial ausgeklappt und müssen manuell geprüft und bestätigt werden. Sie sind potentiell relevant, aber weniger sicher.

Dieses Vorgehen kombiniert Effizienz (automatisches Übernehmen sicherer Vorschläge) mit dem Human-in-the-Loop-Prinzip (manuelle Prüfung unsicherer Vorschläge).

Schritt 4: Speicherung

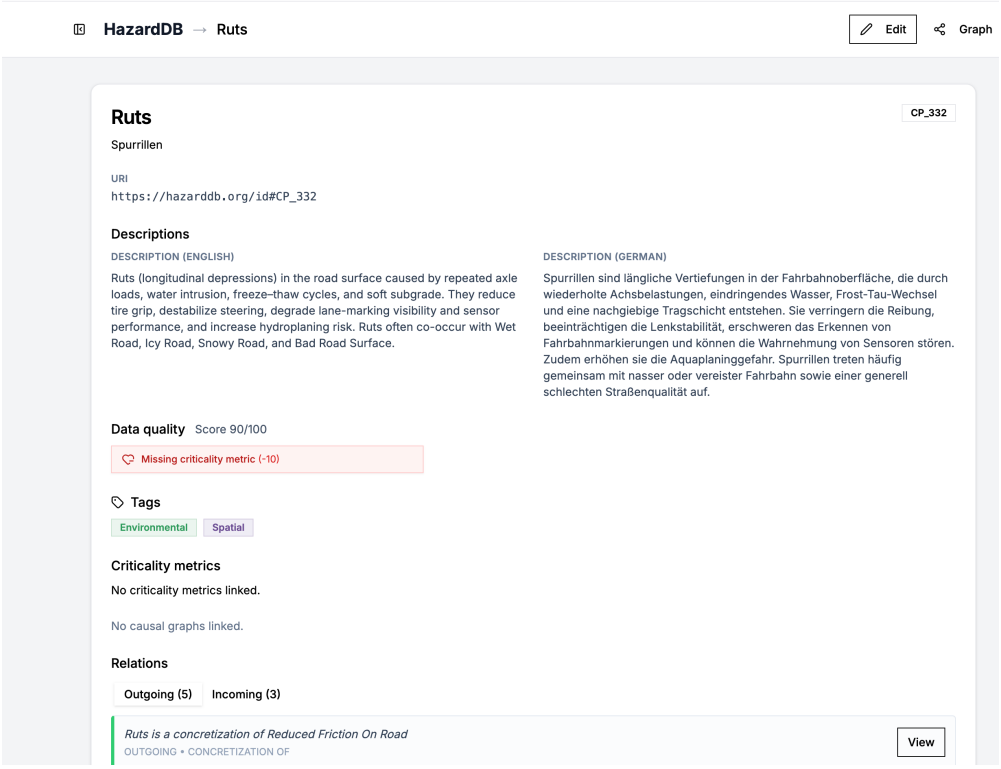

Abbildung 6.8.: Detailsicht des neu erstellten Kritikalitätsphänomens „Spurrillen“. Die Ansicht zeigt alle Labels, Beschreibungen, Tags und Relationen des soeben gespeicherten Phänomens.

Nach Übernahme der akzeptierten Vorschläge wurde das Phänomen validiert und gespeichert. Die integrierte Validierung prüfte auf Eindeutigkeit der Labels (keine Duplikate) und Vollständigkeit aller Pflichtfelder. Nach erfolgreicher Speicherung in Fuseki wird der Meilisearch-Index asynchron aktualisiert. Die Detailsicht zeigt das neu erstellte Phänomen mit allen übernommenen Labels, Beschreibungen, Tags und Relationen.

6.4.3. Bewertung

Geführter Eingabeprozess: Das Formular strukturiert den Kurationsprozess klar. Validierungsregeln verhindern häufige Fehler (z.B. doppelte Labels). Die Combobox-Komponente unterstützt die manuelle Kuration, indem sie verfügbare Relationstypen und Kritikalitätsphänomene übersichtlich darstellt und durch Suchfunktionalität schnelles Auffinden ermöglicht. In diesem Evaluationsszenario steht jedoch die LLM-gestützte Assistenz im Fokus, nicht die manuelle Eingabe.

LLM-Qualität: Die generierten Vorschläge sind – soweit ohne Domänenexpertise beurteilbar – strukturell plausibel. Beschreibungen folgen dem Stil bestehender Phänomene, Tags orientieren sich an der etablierten Taxonomie, Relationsvorschläge sind kontextuell nachvollziehbar. Eine fachliche Validierung würde jedoch die Einbindung von Safety-Professionals erfordern.

Effizienz: Der LLM-gestützte Workflow reduziert den manuellen Aufwand potenziell erheblich. Ohne Assistenz müsste der Kurator:

- Beschreibungen selbst formulieren (konsistente, bilinguale Texte)
- Tags aus der Gesamtmenge aller etablierten Tags identifizieren
- Relationen durch Durchsicht und Analyse aller bisher bestehenden Kritikalitätsphänomene manuell identifizieren

Mit LLM-Assistenz reduziert sich der Aufwand auf Bestätigung oder Anpassung der Vorschläge. Eine genaue Quantifizierung der Zeitersparnis würde eine vergleichende Studie erfordern, die außerhalb des Scopes dieser Arbeit liegt. Dies adressiert Problem 2 aus Kapitel 1 (fehleranfällige Kuration, die nicht skaliert).

Human-in-the-Loop mit Confidence-Scores: Das System balanciert Automatisierung und Kontrolle: High-Confidence-Vorschläge werden automatisch übernommen (effizient), können aber jederzeit entfernt werden. Medium-Confidence-Vorschläge erfordern explizite Bestätigung. Dies verhindert unkontrollierte Änderungen durch Halluzinationen und wahrt die Datenqualität (Anforderung NF4), ohne die Effizienzgewinne durch übermäßige manuelle Prüfungen zu untergraben.

Fazit zu Evaluationskriterium (b): Das Tool gewährleistet effiziente Kuration durch geführte Formulare und intelligente Assistenz. Die LLM-Vorschläge beschleunigen den Prozess erheblich, ohne die Kontrolle des Kurators zu untergraben.

6.5. Szenario 3: Qualitätsüberwachung und Verbesserung

6.5.1. Aufgabenstellung

Dieses Szenario operationalisiert User Story US4 (siehe Abschnitt 2.3):

> *Als Kurator möchte ich einen Überblick über die Datenqualität aller Kritikalitätsphänomene erhalten, der fehlende Beschreibungen, unvollständige Übersetzungen oder fehlende Metriken-Verknüpfungen identifiziert, um gezielt die größten Qualitätslücken priorisieren und beheben zu können.*

Konkrete Aufgabe: Nutze die Health Checks zur Identifikation problematischer Phänomene und verbessere das am höchsten priorisierte Phänomen mittels LLM-Assistenz.

Begründung der Wahl: Die Health-Check-Funktionalität adressiert Problem 3 aus Kapitel 1 – gefährdete Nachhaltigkeit durch schleichenden Qualitätsverlust. Durch proaktive Identifikation von Lücken (fehlende Beschreibungen, unvollständige Übersetzungen, isolierte Knoten) können Kuratoren gezielt priorisieren.

6.5.2. Workflow-Dokumentation

Schritt 1: Health Check Dashboard

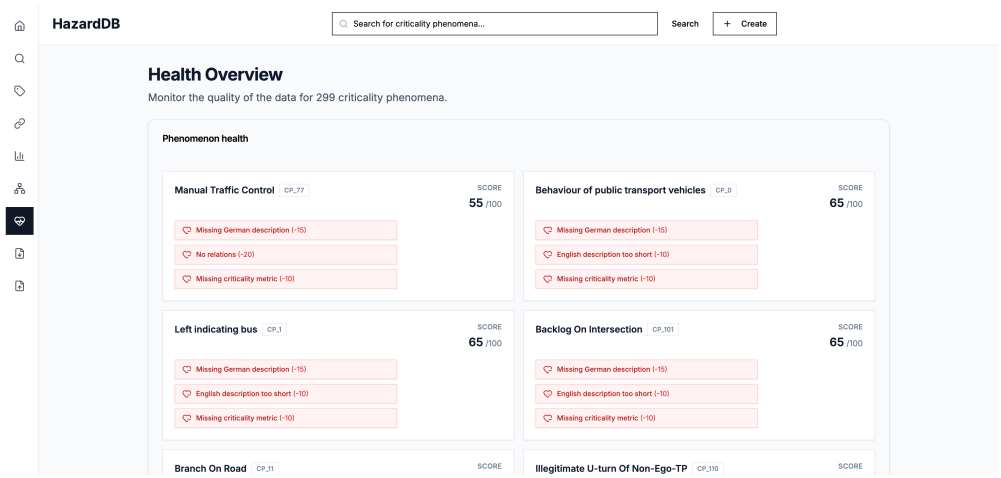

Abbildung 6.9.: Health Check Dashboard zeigt alle Kritikalitätsphänomene sortiert nach Health Score (aufsteigend). Ein Score von 100 bedeutet perfekte Datenqualität, Abzüge erfolgen für jede identifizierte Qualitätslücke. Probleme sind kategorisiert (fehlende Labels, zu kurze Beschreibungen, etc.).

Das Health Check Dashboard führt automatisierte Prüfungen für alle Phänomene durch und berechnet für jedes Phänomen einen **Health Score**. Ein Score von 100 bedeutet perfekte Datenqualität; für jede Qualitätslücke werden Punkte abgezogen. Typische Prüfungen umfassen:

- **Fehlende Labels**: Phänomene ohne deutsches oder englisches Label
- **Fehlende oder zu kurze Beschreibungen**: Beschreibungen unter 20 Zeichen oder komplett fehlend
- **Fehlende Tags**: Phänomene ohne Kategorisierung
- **Unvollständige Übersetzungen**: Phänomene mit deutscher, aber fehlender englischer Beschreibung (oder umgekehrt)
- **Fehlende Kritikalitätsmetriken**: Phänomene, die keiner Kritikalitätsmetrik zugeordnet sind

Die Sortierung nach Health Score (aufsteigend) ermöglicht die Priorisierung: Phänomene mit den niedrigsten Scores – und damit den meisten Qualitätslücken – werden zuerst angezeigt.

Schritt 2: Detailansicht eines problematischen Phänomens

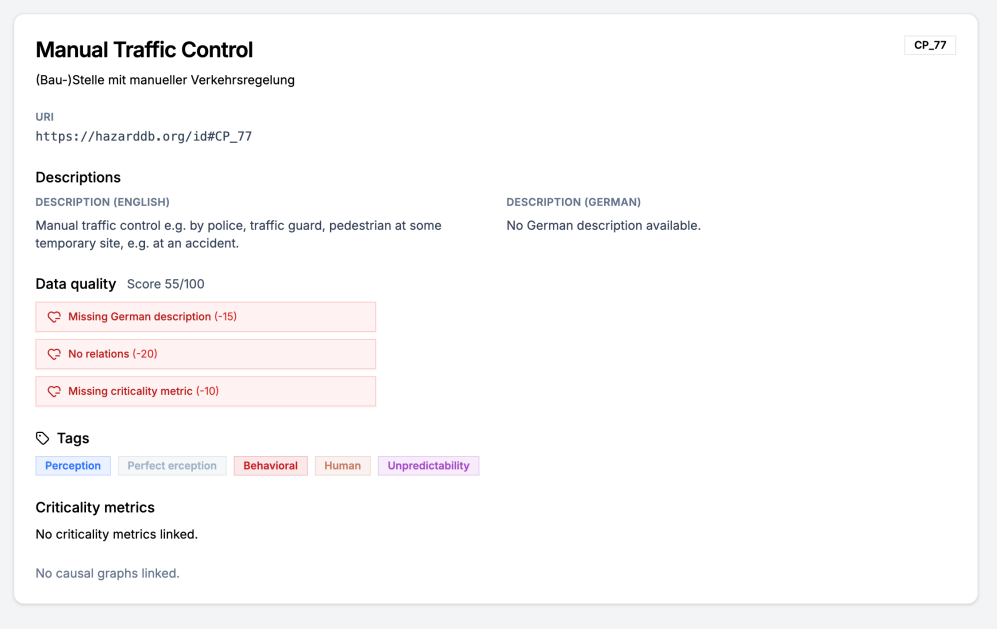

Abbildung 6.10.: Detailseite eines problematischen Phänomens mit visueller Markierung der identifizierten Lücken. Fehlende oder unzureichende Felder sind hervorgehoben.

Durch Klick auf ein Phänomen im Health Check Dashboard gelangt man zur Detailseite. Dort sind die identifizierten Qualitätslücken visuell hervorgehoben (z.B. fehlende deutsche Beschreibung rot markiert, Tags-Sektion zeigt „Keine Tags vorhanden“).

Schritt 3: Edit-Modus mit LLM-Vorschlägen

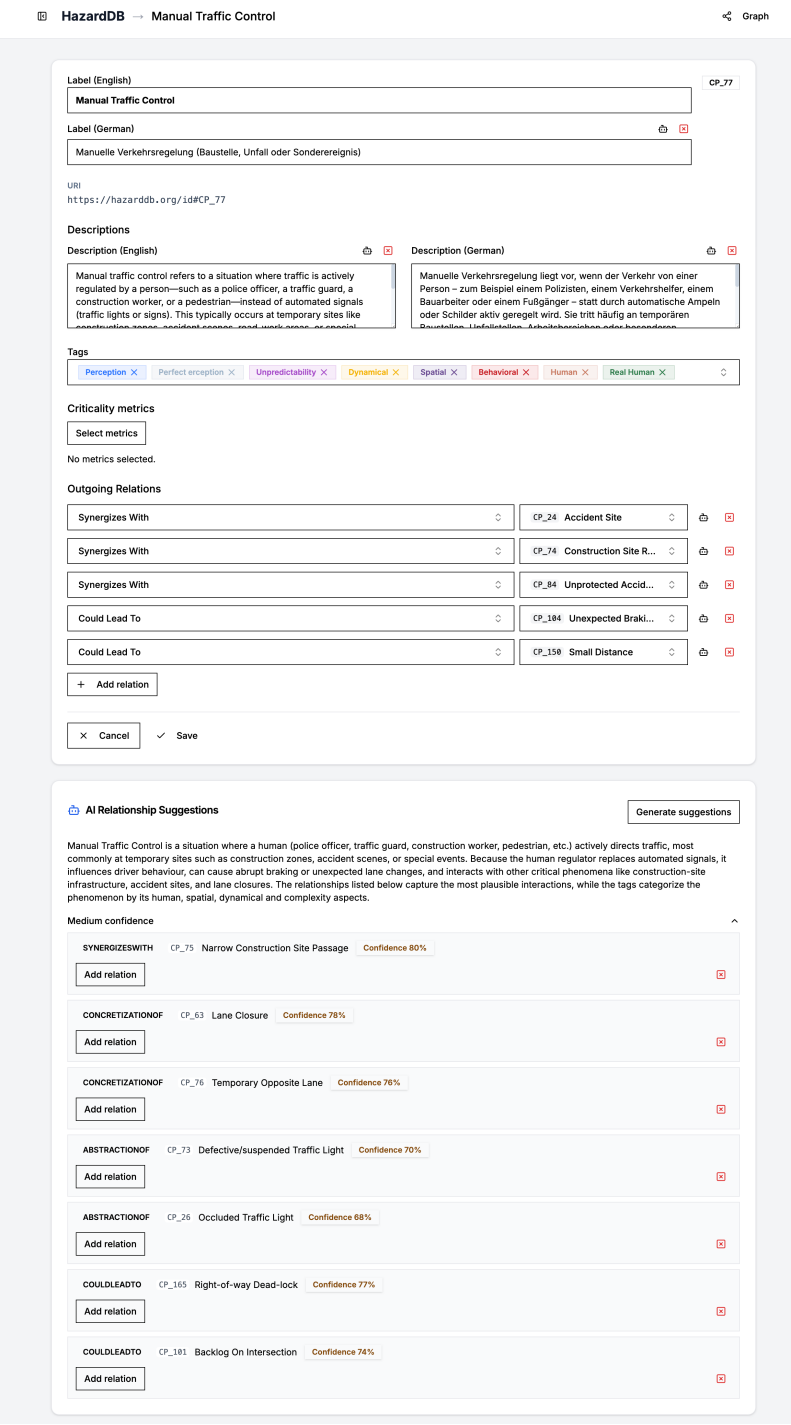

Abbildung 6.11.: Edit-Modus zeigt vorhandene Daten und bietet LLM-Assistenz für fehlende Felder. Vorschläge werden kontextbasiert generiert (bestehende Labels, Relationen, ähnliche Phänomene). *Hinweis:* Die Verknüpfung von Kritikalitätsphänomenen mit Metriken kann nicht durch die AI vorgeschlagen werden und muss manuell erfolgen.

Im Edit-Modus können die identifizierten Lücken direkt behoben werden. Der LLM-Assistenz-Button ist verfügbar und generiert Vorschläge nur für die *fehlenden* Felder – bestehende Daten werden nicht überschrieben. Dies unterscheidet sich vom Create-Workflow: Hier liegt der Fokus auf Vervollständigung, nicht auf initialer Erstellung. **Wichtige Einschränkung:** Die Verknüpfung von Kritikalitätsphänomenen mit Metriken kann nicht durch die AI vorgeschlagen werden – diese Verknüpfungen müssen manuell vom Kurator erstellt werden, da sie domänenspezifisches Fachwissen erfordern.

Schritt 4: Verbessertes Phänomen

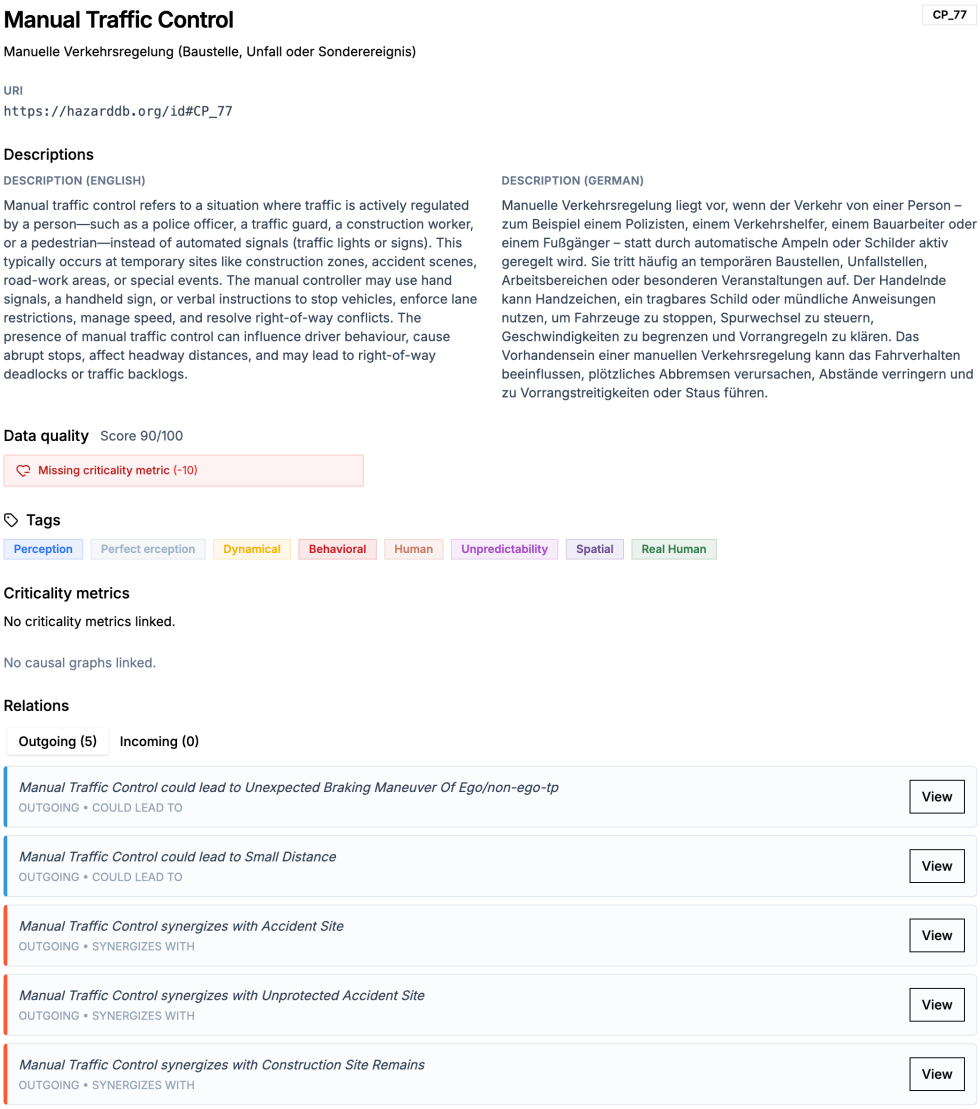

Abbildung 6.12.: Detailseite nach Übernahme der LLM-Vorschläge. Alle zuvor markierten Qualitätslücken sind behoben. Das Phänomen verschwindet aus der priorisierten Health-Check-Liste oder wird niedriger eingestuft.

Nach Speicherung der Verbesserungen ist das Phänomen vollständiger: Fehlende Beschreibungen sind ergänzt, Tags hinzugefügt, eventuell fehlende Relationen nachgetragen. Bei erneutem Ausführen der Health Checks erscheint das Phänomen nicht mehr (oder deutlich niedriger priorisiert) in der Problemliste.

6.5.3. Bewertung

Effektivität der Qualitätsprüfungen: Die automatisierten Health Checks identifizieren Qualitätslücken zuverlässig. Die Kategorisierung nach Problemtyp ermöglicht gezielte Behebung. Die Priorisierung nach Health Score ist intuitiv und quantifizierbar – Phänomene mit niedrigem Score (viele Lücken) erhalten zuerst Aufmerksamkeit.

Priorisierung: Die Sortierung nach Health Score ist hilfreich für Kuratoren mit begrenzten Ressourcen. Statt alle Phänomene manuell zu prüfen, können gezielt die problematischsten adressiert werden. Dies adressiert Problem 3 (Nachhaltigkeit) direkt: Regelmäßige Health Checks verhindern schleichenden Qualitätsverlust.

LLM-Unterstützung bei Vervollständigung: Die Fokussierung auf fehlende Felder ist sinnvoll – bestehende Daten bleiben unangetastet. Die kontextbasierte Generierung (unter Berücksichtigung vorhandener Labels, Relationen, ähnlicher Phänomene) führt zu plausibleren Vorschlägen als bei der Neuerstellung. Der Aufwand für Vervollständigung erscheint subjektiv minimal, eine quantitative Evaluierung würde jedoch eine vergleichende Nutzerstudie erfordern. *Limitation:* Die Verknüpfung mit Metriken muss manuell erfolgen, da dies domänenspezifisches Fachwissen erfordert, das außerhalb der Möglichkeiten der AI liegt.

Fazit zu Evaluationskriterium (b): Die Kombination aus automatisierten Qualitätsprüfungen und LLM-gestützter Vervollständigung gewährleistet konsistente Pflege der Wissensbasis. Die proaktive Identifikation von Lücken verhindert Verwahrlosung (Problem 3 aus Kapitel 1).

6.6. Szenario 4: Dokumentation kausaler Zusammenhänge

6.6.1. Aufgabenstellung

Dieses Szenario operationalisiert User Story US3 (siehe Abschnitt 2.3):

> *Als Nutzer möchte ich Kausalgraphen erstellen und verwalten können, die komplexe Wirkungsketten zwischen Kritikalitätsphänomenen und Metriken dokumentieren, um Hypothesen über kausale Zusammenhänge strukturiert zu erfassen und auf den jeweiligen Detailseiten sichtbar zu machen.*

Konkrete Aufgabe: Betrachte einen existierenden Kausalgraphen oder erstelle einen neuen, um kausale Hypothesen strukturiert zu erfassen.

Begründung der Wahl: Kausalgraphen erweitern die HazardDB über einfache paarweise Relationen hinaus. Sie ermöglichen die Dokumentation komplexer Hypothesen der Form „Phänomen A und B führen zusammen über Zwischenschritte C und D zu Metrik M“. Dies ist besonders relevant für wissenschaftliche Analysen und Validierungsprozesse.

6.6.2. Workflow-Dokumentation

Schritt 1: Kausalgraph-Übersicht und Editor

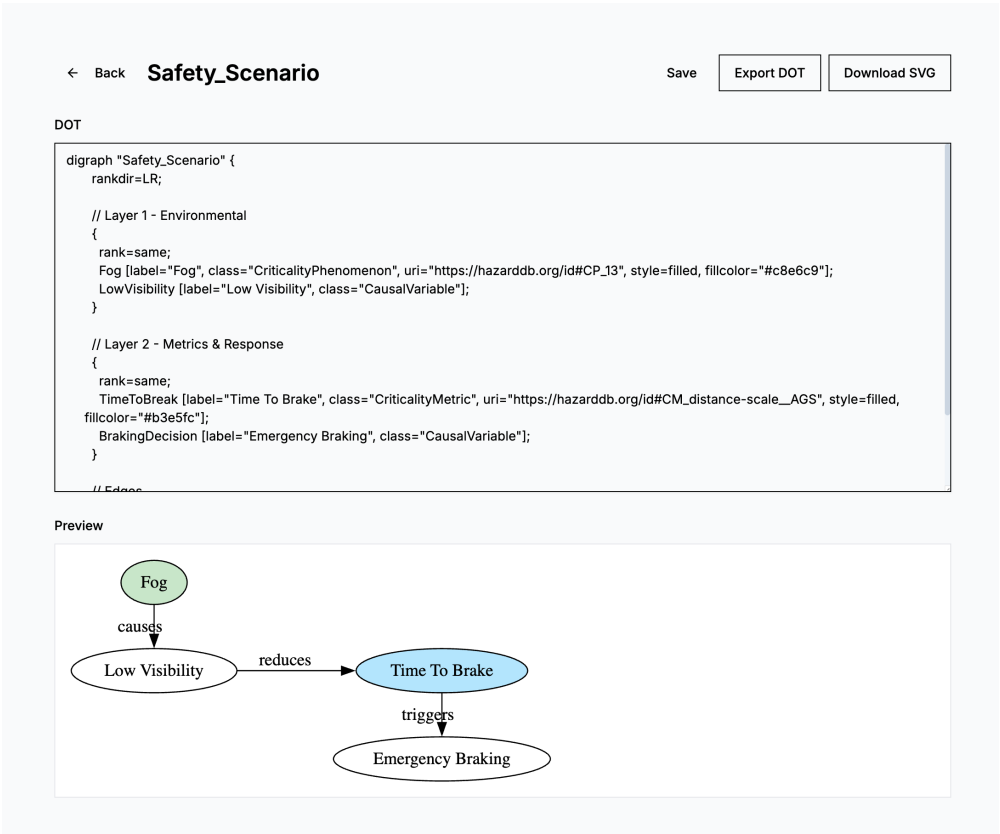

Abbildung 6.13.: Editor-Ansicht zur Erstellung und Bearbeitung von Kausalgraphen. Kausale Ketten werden in **Dot-Notation** (Graphviz-Sprache) definiert und als gerichtete Graphen mit Knoten (Phänomene, Metriken) und Kanten (kausale Einflüsse) visualisiert.

Die Kausalgraph-Funktionalität ermöglicht die Definition komplexer Ursache-Wirkungs-Ketten. Die Erstellung erfolgt durch Eingabe der Graphstruktur in **Dot-Notation** (Graphviz-Format) – ein textuelles Eingabefeld erlaubt die Spezifikation von Knoten (Phänomene, Metriken) und gerichteten Kanten (kausale Einflüsse). Die Dot-Notation ermöglicht präzise strukturelle Definition, erfordert jedoch Kenntnisse der Graphviz-Syntax. Ein visueller Drag-and-Drop-Editor ist in dieser Implementierung nicht verfügbar.

Schritt 2: Verknüpfung zu Kritikalitätsphänomenen

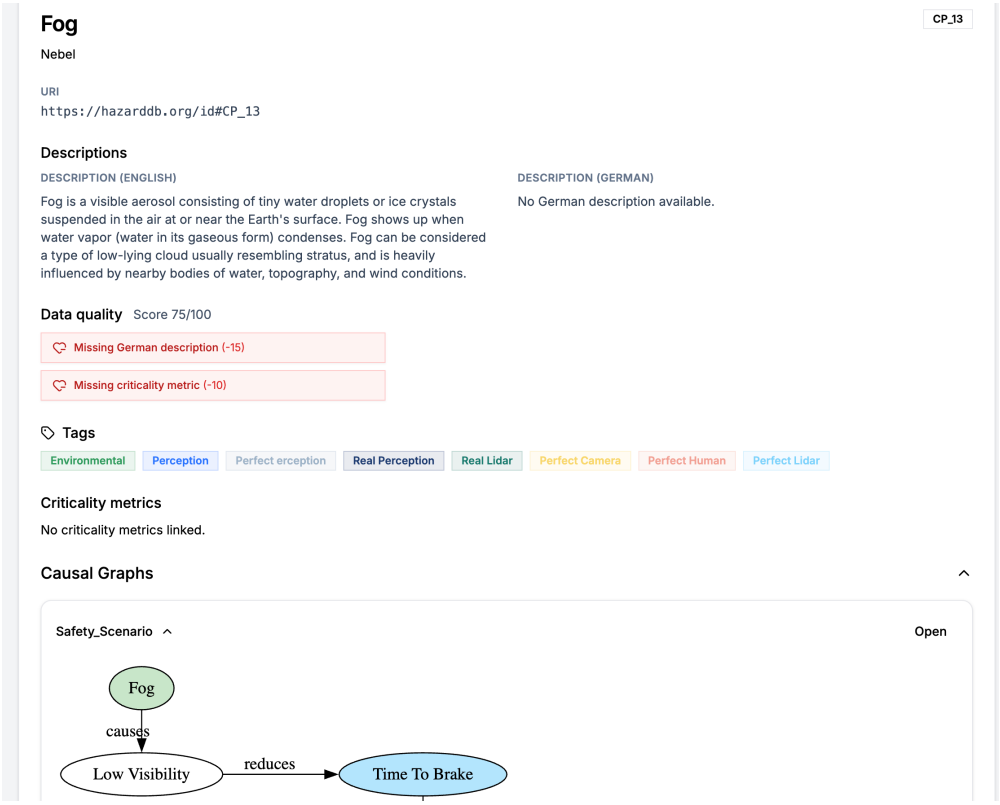

Abbildung 6.14.: Auf der Detailseite eines Kritikalitätsphänomens werden alle Kausalgraphen angezeigt, in denen das Phänomen referenziert wird. Dies ermöglicht bidirektionale Navigation zwischen Phänomenen und Hypothesen.

Die Integration in die Detailseiten der Kritikalitätsphänomene stellt sicher, dass Kausalgraphen nicht isoliert existieren. Für jedes Phänomen ist sichtbar, in welchen kausalen Hypothesen es eine Rolle spielt.

Schritt 3: Verknüpfung zu Kritikalitätsmetriken

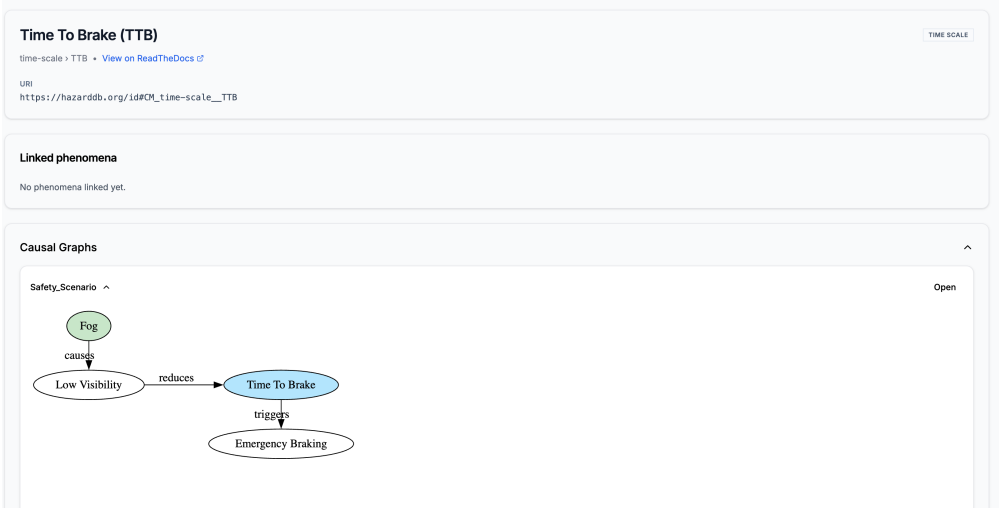

Abbildung 6.15.: Analog zu Kritikalitätsphänomenen zeigen auch die Detailseiten von Metriken alle Kausalgraphen, in denen die jeweilige Metrik vorkommt. Dies ermöglicht die Nachverfolgung, welche kausalen Hypothesen zu einer bestimmten Metrik führen.

Die gleiche bidirektionale Verknüpfung existiert auch für Kritikalitätsmetriken: Auf deren Detailseiten werden alle Kausalgraphen aufgelistet, die die entsprechende Metrik als Endpunkt oder Zwischenknoten verwenden. Dies ermöglicht die Nachverfolgung kausaler Pfade von Phänomenen zu Metriken.

6.6.3. Bewertung

Integration in Workflow: Die Kausalgraph-Funktionalität ist nahtlos in die bestehende Architektur integriert. Die bidirektionale Verknüpfung zu Phänomenen und Metriken über Detailseiten macht kausale Hypothesen sichtbar und nutzbar.

Mehrwert für Dokumentation: Kausalgraphen ermöglichen die strukturierte Erfassung komplexer Hypothesen, die über einfache paarweise Relationen (*could_lead_to*) hinausgehen. Dies ist besonders wertvoll für wissenschaftliche Analysen und Dokumentation kausaler Pfade von Phänomenen zu Metriken.

Usability-Limitation: Die Erstellung von Kausalgraphen ist in der aktuellen Implementierung **schematisch** und erfordert Kenntnisse der Dot-Notation (Graphviz-Sprache). Ein visueller Drag-and-Drop-Editor mit Click-UI ist nicht vorhanden – Nutzer müssen die Graphstruktur textuell definieren. Dies macht die Funktionalität weniger zugänglich für Nutzer ohne Erfahrung mit Graphviz. Eine zukünftige Erweiterung mit visuellem Editor würde die Usability erheblich verbessern.

Domänen-Limitation: Die Evaluation dieses Szenarios ist oberflächlicher als die vorherigen, da die tatsächliche Nutzung von Kausalgraphen domänenspezifisches Wissen erfordert. Eine vollständige Validierung würde die Erstellung realistischer Hypothesen durch Safety-Professionals erfordern.

Fazit: Die Kausalgraph-Funktionalität erweitert die HazardDB sinnvoll für wissenschaftliche Use Cases. Die technische Implementierung ist funktional, jedoch mit Usability-Einschränkungen aufgrund der textbasierten Dot-Notation. Eine umfassende Evaluation der fachlichen Nützlichkeit steht aus.

6.7. Diskussion

Die durchgeführten Szenarien ermöglichen eine fundierte Beantwortung der zentralen Forschungsfrage dieser Arbeit. Die Frage wird entlang ihrer beiden Teilaspekte beantwortet:

6.7.1. (a) Intuitive, pfadbasierte Exploration

Das entwickelte Webtool ermöglicht intuitive Exploration der HazardDB durch die Kombination mehrerer Navigationsmechanismen:

Hierarchische Navigation: Die persistente Tree View erlaubt schrittweises Durchklicken der Abstraktionshierarchie von abstrakten zu konkreten Kritikalitätsphänomenen. Die Baumstruktur bleibt beim Wechsel zwischen Detailseiten geöffnet und markiert das aktuell betrachtete Phänomen – dies verhindert Desorientierung und ermöglicht kontinuierliche Orientierung in der Struktur.

Detaillierte Informationspräsentation: Die Detailseiten bündeln alle relevanten Informationen strukturiert: Labels, Beschreibungen, Tags sowie alle ausgehenden und eingehenden Relationen gruppiert nach Typ. Die klare Strukturierung ermöglicht schnelles Erfassen der Nachbarschaft eines Phänomens.

Interaktive Graph-Visualisierung: Die Graph-Visualisierung bietet eine komplementäre, visuelle Perspektive auf die Relationen. Die pfadbasierte Exploration wird durch interaktives Nachladen von Nachbar-Phänomenen unterstützt: Ein Doppelklick auf einen Knoten lädt dessen direkte Nachbarn nach und ermöglicht so schrittweises Erweitern des sichtbaren Graphausschnitts entlang relevanter Pfade.

Relationstypen-Unterscheidbarkeit: Die verschiedenen Relationstypen (*abstraction_of*, *concretization_of*, *could_lead_to*, *synergizes_with*) sind sowohl textuell als auch visuell klar unterscheidbar. Dies adressiert eine Kernlimitation dokumentenbasierter Tools wie Notion oder Confluence, die nur generische Hyperlinks unterstützen (siehe Kapitel 3, Abschnitt 3.2.2).

Performance: Die Navigation erfolgt flüssig ohne spürbare Verzögerungen. Client-Side Rendering ermöglicht schnelle Transitions ohne Full-Page-Reloads (Anforderung

NF2). Die Kombination der Komponenten funktioniert nahtlos zusammen.

Fazit: Das Tool erfüllt Teilanforderung (a) der Forschungsfrage. Die Exploration ist intuitiv, pfadbasiert und unterstützt verschiedene Explorationsstrategien (hierarchisch, graphisch, suchbasiert). Problem 1 aus Kapitel 1 (umständliche Exploration, die Kenntnisse in formalen Abfragesprachen erfordert) ist gelöst.

6.7.2. (b) Effiziente und konsistente Pflege der Wissensbasis

Das Tool gewährleistet effiziente Kuration durch drei zentrale Mechanismen:

Geführte Eingabeprozesse: Die Formulare für Create/Edit strukturieren den Kurationsprozess klar. Validierungsregeln verhindern häufige Fehler (z.B. doppelte Labels). Combobox-Komponenten unterstützen die Auswahl von Relationstypen und die Suche nach bestehenden Kritikalitätsphänomenen. Duplikate werden durch Validierungsregeln verhindert. Dies adressiert Problem 2 (fehleranfällige Kuration) direkt.

LLM-gestützte Kurations-Assistenz: Die intelligenten Vorschläge für Beschreibungen, Tags und Relationen reduzieren den manuellen Aufwand erheblich. Der Kurator muss nicht mehr alle bestehenden Phänomene manuell durchsuchen, um relevante Relationen zu identifizieren – das LLM aggregiert den Kontext automatisch. Die Vorschläge sind strukturell plausibel und orientieren sich an etablierten Mustern der Wissensbasis. Das Human-in-the-Loop-Prinzip wahrt die Datenqualität durch explizite Bestätigungspflicht.

Die Effizienzsteigerung ist potentiell erheblich: Die LLM-Assistenz reduziert die notwendigen manuellen Schritte deutlich, indem sie Vorschläge für Beschreibungen, Tags und Relationen automatisch generiert. Eine genaue Quantifizierung der Zeitersparnis würde eine kontrollierte Vergleichsstudie mit und ohne LLM-Assistenz erfordern. Dies adressiert die Skalierungsprobleme aus Problem 2 direkt.

Automatisierte Qualitätssicherung: Die Health Checks identifizieren Qualitätslücken proaktiv und systematisch. Die Priorisierung nach Schweregrad ermöglicht gezielte Behebung der problematischsten Fälle. Die LLM-gestützte Vervollständigung erleichtert die Behebung der identifizierten Lücken. Dies adressiert Problem 3 (gefährdete Nachhaltigkeit durch Qualitätsverlust) direkt und unterscheidet die HazardDB von Wissensbasen ohne kontinuierliche Pflege wie PerCollECT (siehe Kapitel 3, Abschnitt 3.2.3).

Konsistenz: Die Kombination aus semantischer Datenhaltung (automatische Inferenz inverser Relationen über OWL Reasoning), Validierung und Health Checks gewährleistet strukturelle Konsistenz. Manuelle Duplikation bidirektionaler Relationen entfällt (Anforderung T4).

Fazit: Das Tool erfüllt Teilanforderung (b) der Forschungsfrage. Die Kuration ist effizient durch intelligente Assistenz und konsistent durch automatisierte Prüfungen.

Die identifizierten Probleme 2 und 3 aus Kapitel 1 sind gelöst.

6.7.3. Limitationen und Einschränkungen

Die Evaluation unterliegt mehreren Einschränkungen, die transparent benannt werden müssen:

Fehlende Domänenexpertise: Eine zentrale Limitation dieser Evaluation ist, dass der Autor über keine Expertise in der Domäne der Kritikalitätsanalyse für automatisierte Fahrsysteme verfügt. Die Evaluation fokussiert sich daher ausschließlich auf die **Tool-Perspektive** – Usability, Workflow-Effizienz, Funktionalität der Features – und nicht auf die **inhaltliche Qualität** der erzeugten oder verbesserten Kritikalitätsphänomene.

Konkret bedeutet dies: Die LLM-generierten Vorschläge für Beschreibungen, Tags oder Relationen können hinsichtlich ihrer technischen Plausibilität bewertet werden (Sind sie strukturiert? Passen sie zum Kontext bestehender Phänomene?), jedoch nicht hinsichtlich ihrer fachlichen Korrektheit (Ist die Beschreibung eines Kritikalitätsphänomens wissenschaftlich präzise? Sind die kausalen Zusammenhänge realistisch?). Eine fachliche Validierung würde die Einbindung von Safety-Professionals erfordern.

Fehlende vergleichende Langzeitstudie: Eine vollständige Validierung des entwickelten Web-Tools würde eine vergleichende Langzeitstudie mit dem bisherigen Neo4j-basierten System erfordern – mit parallelen Nutzergruppen, kontinuierlichen Befragungen und Erhebungen über mehrere Monate hinweg. Ein solcher Evaluationsaufwand übersteigt den Rahmen dieser Bachelorarbeit deutlich und wäre selbst für eine Masterarbeit zu umfangreich.

Entwickler-Perspektive: Die Evaluation wurde durch den Entwickler des Systems durchgeführt, nicht durch externe Endnutzer. Dies birgt das Risiko eines Confirmation Bias – Usability-Probleme, die externen Nutzern auffallen würden, könnten übersehen werden. Eine Evaluation mit tatsächlichen Nutzern würde zusätzliche Erkenntnisse liefern.

Prototypischer Charakter: Die Evaluation fokussiert sich auf Kernfunktionalitäten. Aspekte wie Authentifizierung, Security, Performance unter Last oder Multi-User-Szenarien wurden nicht evaluiert.

Diese Limitationen schmälern die Aussagekraft der Evaluation nicht fundamental, definieren jedoch klar den Scope: Die Arbeit demonstriert die *Machbarkeit* domänenspezifischer Kurations-Assistenz für spezialisierte Wissensbasen. Die *Produktionsreife* und *langfristige Validierung* sind Aufgaben für zukünftige Arbeiten.

6.8. Zusammenfassung

Die szenariobasierte Evaluation hat gezeigt, dass das entwickelte HazardDB-Webtool die identifizierten Probleme aus Kapitel 1 adressiert:

Problem 1 (umständliche Exploration) ist gelöst durch intuitive Navigation mittels Tree View, Detailseiten und Graph-Visualisierung. Die Exploration erfordert keine SPARQL-Kenntnisse und erfolgt pfadbasiert.

Problem 2 (fehleranfällige Kuration) ist gelöst durch geführte Formulare mit Validierung, Combobox-Filterung und LLM-gestützte Vorschläge. Die Effizienz der Kuration wird erheblich gesteigert.

Problem 3 (gefährdete Nachhaltigkeit) ist adressiert durch automatisierte Health Checks und LLM-gestützte Vervollständigung. Qualitätslücken werden proaktiv identifiziert und können gezielt behoben werden.

Die Forschungsfrage ist positiv beantwortet: Ein interaktives Webtool *kann* Anwender bei intuitiver Exploration leiten und effiziente, konsistente Pflege gewährleisten – vorausgesetzt, es kombiniert domänenspezifische Navigationsmechanismen, semantische Datenhaltung, intelligente Kurations-Assistenz und automatisierte Qualitätssicherung.

Die Evaluation unterliegt den benannten Limitationen (fehlende Domänenexpertise, Entwickler-Perspektive, prototypischer Charakter). Eine umfassende Validierung durch tatsächliche Nutzer ist der notwendige nächste Schritt für die Überführung in den produktiven Einsatz.

Das entwickelte System bietet eine solide Grundlage für die langfristige Nutzung der HazardDB und kann als Referenz für ähnliche spezialisierte Wissensbasen dienen.

A. Code-Listings der Implementierung

Dieser Anhang enthält die vollständigen Code-Beispiele aus Kapitel 5. Die Listings illustrieren die technische Umsetzung der beschriebenen Features und sind für vertieftes Verständnis der Implementierungsdetails gedacht.

A.1. CRUD-Operationen mit SPARQL

A.1.1. Entity-Update mit Event-Emission

```typescript
async updateEntity(uri: string, updateDto: UpdateCriticalityPhenomenonDto) {
  const { relations, tags, ...propertyUpdates } = updateDto;

  // Validierung: Labels müssen eindeutig sein
  if (updateDto.labelEn || updateDto.labelDe) {
    const existing = await this.repository.getMetadata(uri);
    if (updateDto.labelEn) {
      const currentLabelEn = existing?.labelEn?.value;
      if (updateDto.labelEn !== currentLabelEn) {
        await this.repository.ensureLabelUnique(updateDto.labelEn, 'en');
      }
    }
    // ... analog für labelDe
  }

  // Atomare Updates: Properties, Relations, Tags
  await this.repository.updateEntityProperties(uri, propertyUpdates);

  if (relations !== undefined) {
    await this.repository.replaceOutgoingRelations(uri, relations ?? []);
  }

  if (tags !== undefined) {
    await this.repository.replaceTags(uri, tags);
  }

  this.logger.log(`Updated entity ${uri}`);

  // Kritisch: Event-Emission für Indexing
  this.eventEmitter.emit(
    'entity.updated',
    new EntityUpdatedEvent(uri, 'CriticalityPhenomenon', updateDto)
  );

  return { uri, updated: true, changes: updateDto };
}
```

Listing 7: Entity-Update mit Event-Emission (Service-Schicht)

A.1.2. SPARQL-basierte Relation-Verwaltung

```typescript
async replaceOutgoingRelations(
  uri: string,
  relations: RelationInput[]
): Promise<void> {
  // 1. Delete: Alle bestehenden Relationen löschen
  await this.sparqlClient.update(
    ENTITY_DELETE_OUTGOING_RELATIONS(uri)
  );

  if (!relations?.length) return;

  // 2. Deduplizierung: Keine doppelten Relationen
  const seen = new Set<string>();
  const uniqueRelations = relations.filter((rel) => {
    const key = `${rel.relationUri}|${rel.targetNode}`;
    if (seen.has(key)) return false;
    seen.add(key);
    return true;
  });

  // 3. Insert: Neue Tripel generieren und einfügen
  const triples = uniqueRelations
    .map((relation) => this.buildRelationTriple(uri, relation))
    .filter((triple): triple is string => triple !== null);

  if (triples.length) {
    await this.helpers.insertTriples(triples);
  }
}

private buildRelationTriple(
  subjectUri: string,
  relation: RelationInput
): string | null {
  // URI-Normalisierung
  const predicateUri = relation.relationUri.startsWith('http')
    ? relation.relationUri
    : buildHazardSchemaUri(relation.relationUri);

  const targetUri = relation.targetNode.startsWith('http')
    ? relation.targetNode
    : buildHazardDataUri(relation.targetNode);

  return `<${subjectUri}> <${predicateUri}> <${targetUri}> .`;
}
```

Listing 8: SPARQL-basierte Relation-Verwaltung (Repository-Schicht)

A.1.3. SPARQL-Client-Implementierung

```typescript
@Injectable()
export class SparqlClient {
  async executeQuery(query: string): Promise<SparqlResult> {
    const response = await fetch(
      `${this.fusekiUrl}/${this.dataset}/sparql`,
      {
        method: 'POST',
        headers: this.buildHeaders(),
        body: `query=${encodeURIComponent(query)}`,
      }
    );

    if (!response.ok) {
      throw new Error(`SPARQL query failed: ${response.statusText}`);
    }

    return await response.json() as SparqlResult;
  }

  async update(updateQuery: string): Promise<void> {
    const response = await fetch(
      `${this.fusekiUrl}/${this.dataset}/update`,
      {
        method: 'POST',
        headers: this.buildHeaders(),
        body: `update=${encodeURIComponent(updateQuery)}`,
      }
    );

    if (!response.ok) {
      throw new Error(`SPARQL update failed: ${response.statusText}`);
    }
  }

  private buildHeaders(): Record<string, string> {
    const headers: Record<string, string> = {
      'Content-Type': 'application/x-www-form-urlencoded',
    };

    // Basic Authentication
    if (this.fusekiUser && this.fusekiPassword) {
      const credentials = Buffer.from(
        `${this.fusekiUser}:${this.fusekiPassword}`
      ).toString('base64');
      headers.Authorization = `Basic ${credentials}`;
    }

    return headers;
  }
}
```

Listing 9: SPARQL-Client-Implementierung

A.2. LLM-gestützte Kurations-Assistenz

A.2.1. LLM-Prompt-Struktur

```
You are a critical phenomena relationship expert. Analyse the given
phenomenon, suggest relationships, and improve missing or weak metadata.

# Task
Suggest relationships and tags for this criticality phenomenon:

Label (EN): Heavy Rain
Label (DE): Starkregen
Description (EN): Intensive precipitation with high rainfall rates...
Description (DE): Intensive Niederschläge mit hoher Regenrate...

# Context: All Available Phenomena
- CP_001 (Rain | Regen)
- CP_002 (Fog | Nebel)
- CP_003 (Adverse Weather | Widrige Wetterbedingungen)
- CP_004 (Sensor Malfunction | Sensorfehlfunktion)
... (295 more)

# Available Relation Types
- abstraction_of: Hierarchical (this is more abstract than target)
- concretization_of: Hierarchical (this is more specific than target)
- could_lead_to: Causal (this can cause target)
- synergizes_with: Synergistic (combined with target increases criticality)

# Available Tags
- Weather (Wetter)
- Precipitation (Niederschlag)
- Sensor (Sensor)
... (20 more)

# Output Format (JSON)
{
  "reasoning": "Brief explanation...",
  "suggested_relationships": [
    {
      "relationUri": "concretization_of",
      "targetNode": "CP_001",
      "confidence": 0.95
    }
  ],
  "suggested_tags": [
    {
      "tagUri": "https://hazarddb.org/id#Weather",
      "confidence": 0.9
    }
  ]
}
```

Listing 10: LLM-Prompt-Struktur (vereinfacht)

A.2.2. LLM-Suggestion-Validation

```
private validateSuggestions(
  suggestions: unknown[],
  context: GraphContext
): RelationshipSuggestion[] {
  if (!Array.isArray(suggestions) || !suggestions.length) {
    return [];
  }

  // Lookup-Maps für schnellen Abgleich
  const relationLookup = new Map<string, string>();
  context.relation_types.forEach((rel) => {
    relationLookup.set(rel.uri.toLowerCase(), rel.uri);
    relationLookup.set(rel.name.toLowerCase(), rel.uri);
  });

  const nodeLookup = new Map<string, string>();
  context.all_nodes.forEach((node) => {
    nodeLookup.set(node.uri.toLowerCase(), node.uri);
    nodeLookup.set(node.enLabel.toLowerCase(), node.uri);
    nodeLookup.set(node.deLabel.toLowerCase(), node.uri);
  });

  const valid: RelationshipSuggestion[] = [];
  const seen = new Set<string>();

  for (const raw of suggestions) {
    const result = this.validateSingleSuggestion(
      raw,
      relationLookup,
      nodeLookup
    );

    if (!result) continue; // LLM hallucination

    // Deduplizierung
    const dedupeKey = `${result.relationUri}|${result.targetNode}`;
    if (seen.has(dedupeKey.toLowerCase())) continue;

    seen.add(dedupeKey.toLowerCase());
    valid.push(result);
  }

  return valid;
}
```

Listing 11: LLM-Suggestion-Validation

```
private validateSingleSuggestion(
  raw: unknown,
  relationLookup: Map<string, string>,
  nodeLookup: Map<string, string>
): RelationshipSuggestion | null {
  if (!raw || typeof raw !== 'object') return null;

  const candidate = raw as Record<string, unknown>;

  // Validate relation type
  const relationInput = (candidate.relationUri as string)?.trim() || '';
  const canonicalRelation = relationLookup.get(
    relationInput.toLowerCase()
  );
  if (!canonicalRelation) return null; // Unknown relation type

  // Validate target node
  const targetNode = (candidate.targetNode as string)?.trim() || '';
  const canonicalTarget = nodeLookup.get(targetNode.toLowerCase());
  if (!canonicalTarget) return null; // Non-existent phenomenon

  // Validate confidence
  const confidence = this.parseConfidence(candidate.confidence);
  if (confidence === null) return null;

  return {
    relationUri: canonicalRelation,
    targetNode: canonicalTarget,
    confidence: Math.min(1, Math.max(0, confidence)),
  };
}
```

Listing 12: LLM-Single-Suggestion-Validation

A.2.3. LLM-Provider-Abstraktion

```
export abstract class LlmProvider {
  protected config: LlmConfig;

  abstract generateSuggestions(prompt: string): Promise<LlmResponse>;
  abstract isAvailable(): Promise<boolean>;
  protected abstract loadConfig(): LlmConfig;
}
```

Listing 13: LLM-Provider-Abstraktion

A.3. Event-gesteuerte Such-Synchronisation

A.3.1. Domain-Event-Definition

```
export class EntityCreatedEvent {
  constructor(
    public readonly uri: string,
    public readonly entityType: string,
    public readonly data: any
  ) {}
}

export class EntityUpdatedEvent {
  constructor(
    public readonly uri: string,
    public readonly entityType: string,
    public readonly changes: any
  ) {}
}
```

Listing 14: Domain-Event-Definition

A.3.2. Event-Driven Indexing

```typescript
@Injectable()
export class IndexingService {
  constructor(
    private sparqlClient: SparqlClient,
    private meilisearchClient: MeilisearchClient
  ) {}

  @OnEvent('entity.created')
  async handleEntityCreated(event: EntityCreatedEvent): Promise<void> {
    try {
      if (event.entityType === 'CriticalityPhenomenon') {
        this.logger.log(`Indexing created entity: ${event.uri}`);
        await this.updateSinglePhenomenon(event.uri);
      }
    } catch (error) {
      this.logger.error(`Indexing failed for ${event.uri}`, error);
    }
  }

  @OnEvent('entity.updated')
  async handleEntityUpdated(event: EntityUpdatedEvent): Promise<void> {
    try {
      if (event.entityType === 'CriticalityPhenomenon') {
        this.logger.log(`Re-indexing updated entity: ${event.uri}`);
        await this.updateSinglePhenomenon(event.uri);
      }
    } catch (error) {
      this.logger.error(`Re-indexing failed for ${event.uri}`, error);
    }
  }

  private async updateSinglePhenomenon(uri: string): Promise<void> {
    // 1. Fetch entity from Fuseki
    const query = INDEXING_SINGLE_SUMMARY(uri);
    const result = await this.sparqlClient.executeQuery(query);

    // 2. Map SPARQL result to Meilisearch document
    const summary = mapIndexingSummary(result.results.bindings[0]);
    const document = this.summaryToDocument(summary);

    // 3. Index in Meilisearch
    await this.meilisearchClient.addDocuments([document]);
  }
}
```

Listing 15: Event-Driven Indexing

A.3.3. Bulk-Indexing beim System-Start

```
async indexAllPhenomena(): Promise<void> {
  await this.meilisearchClient.clearIndex();

  const query = INDEXING_BULK_SUMMARY();
  const result = await this.sparqlClient.executeQuery(query);

  const summaries = mapIndexingSummary(result.results.bindings);
  const documents = summaries.map((s) => this.summaryToDocument(s));

  await this.meilisearchClient.addDocuments(documents);
  this.logger.log(`Indexed ${documents.length} phenomena`);
}
```

Listing 16: Bulk-Indexing beim System-Start

A.4. Graph-Visualisierung

A.4.1. D3-Graph-Rendering

```
const simulation = d3.forceSimulation(nodes)
  .force('link', d3.forceLink(links)
    .id((d) => d.id)
    .distance(150))
  .force('charge', d3.forceManyBody().strength(-300))
  .force('center', d3.forceCenter(width / 2, height / 2))
  .on('tick', updatePositions);

function updatePositions() {
  // Update node positions
  nodeElements
    .attr('cx', (d) => d.x)
    .attr('cy', (d) => d.y);

  // Update link positions
  linkElements
    .attr('x1', (d) => d.source.x)
    .attr('y1', (d) => d.source.y)
    .attr('x2', (d) => d.target.x)
    .attr('y2', (d) => d.target.y);

  // Update tag badge positions
  tagBadges.attr('transform', (d) => `translate(${d.x}, ${d.y})`);
}
```

Listing 17: D3-Graph-Rendering (Frontend)

A.4.2. Tooltip-Generierung

```
function generateNodeTooltipHTML(
  node: Node,
  relationTypeName?: string,
  tagColorMap?: Map<string, string>
): string {
  const tags = Array.isArray(node.tags) ? node.tags : [];

  const tagBadgesHTML = tags
    .map((tagUri: string) => {
      const color = tagColorMap?.get(tagUri) || '#6b7280';
      const label = formatTagLabel(tagUri);

      return `<span style="
        padding: 2px 8px;
        border-radius: 4px;
        background-color: ${color}1a;
        border: 1px solid ${color}33;
        color: ${color};
      ">${label}</span>`;
    })
    .join('');

  return `
    <div style="background: white; padding: 12px; ...">
      <div style="font-weight: 600;">${node.label}</div>
      ${tags.length ? `<div>${tagBadgesHTML}</div>` : ''}
    </div>
  `;
}
```

Listing 18: Tooltip-Generierung mit Tags

A.5. Qualitätssicherung

A.5.1. Health-Check-Regeln

```
export const HEALTH_CHECK_RULES: readonly HealthCheckRule[] = [
  {
    code: 'missing_english_label',
    penalty: 25,
    priority: 1,
    check: (data) => data.labelEnCount === 0,
  },
  {
    code: 'missing_german_label',
    penalty: 20,
    priority: 2,
    check: (data) => data.labelDeCount === 0,
  },
  {
    code: 'missing_english_description',
    penalty: 20,
    priority: 3,
    check: (data) => data.descriptionEnCount === 0,
  },
  {
    code: 'english_description_too_short',
    penalty: 10,
    priority: 5,
    check: (data) =>
      data.descriptionEnCount > 0 &&
      data.descriptionEnLength < 80,
  },
  {
    code: 'no_relations',
    penalty: 20,
    priority: 7,
    check: (data) => data.relationCount === 0,
  },
  {
    code: 'no_tags',
    penalty: 10,
    priority: 8,
    check: (data) => data.tagCount === 0,
  },
];
```

Listing 19: Health-Check-Regeln

A.5.2. Health-Check-Evaluation

```
async checkHealth(iri: string): Promise<HealthReport> {
  const query = ENTITY_HEALTH_OVERVIEW([iri]);
  const results = await this.sparqlClient.executeQuery(query);

  const binding = results.results.bindings[0];
  const data: HealthQueryData = {
    labelEnCount: binding?.labelEnCount ?? 0,
    labelDeCount: binding?.labelDeCount ?? 0,
    descriptionEnCount: binding?.descriptionEnCount ?? 0,
    descriptionEnLength: binding?.descriptionEnLength ?? 0,
    relationCount: binding?.relationCount ?? 0,
    tagCount: binding?.tagCount ?? 0,
  };

  const { issues, score } = this.evaluateHealthRules(data);

  return {
    iri,
    status: issues.length === 0 ? 'healthy' : issues[0],
    score,
    issues,
  };
}

private evaluateHealthRules(data: HealthQueryData) {
  let score = 100;
  const failedRules = HEALTH_CHECK_RULES.filter((rule) => rule.check(data));

  for (const rule of failedRules) {
    score -= rule.penalty;
  }

  const issues = failedRules
    .sort((a, b) => a.priority - b.priority)
    .map((rule) => rule.code);

  return { issues, score: Math.max(0, score) };
}
```

Listing 20: Health-Check-Evaluation

A.6. RDF-Export und -Import

A.6.1. RDF-Export als Turtle

```
async exportAsTurtle(): Promise<string> {
  const results = await this.sparqlClient.executeQuery(
    EXPORT_TURTLE_TRIPLES_QUERY()
  );

  const writer = new Writer({
    format: 'Turtle',
    prefixes: {
      haz: 'https://hazarddb.org/schema#',
      hazd: 'https://hazarddb.org/id#',
      rdfs: 'http://www.w3.org/2000/01/rdf-schema#',
      owl: 'http://www.w3.org/2002/07/owl#',
    },
  });

  for (const binding of results.results.bindings) {
    const s = this.namedNode(binding.s.value);
    const p = this.namedNode(binding.p.value);

    let o;
    if (binding.o.type === 'uri') {
      o = this.namedNode(binding.o.value);
    } else if (binding.o['xml:lang']) {
      // Language-tagged literal
      o = this.literal(binding.o.value, binding.o['xml:lang']);
    } else if (binding.o.datatype) {
      // Typed literal
      o = this.literal(binding.o.value, this.namedNode(binding.o.datatype));
    } else {
      o = this.literal(binding.o.value);
    }

    writer.addQuad(this.quad(s, p, o));
  }

  return new Promise((resolve, reject) => {
    writer.end((error, result) => {
      if (error) reject(error);
      else resolve(result);
    });
  });
}
```

Listing 21: RDF-Export als Turtle

A.6.2. RDF-Import mit Index-Rebuild

```
async importFromTurtle(turtleData: string): Promise<ImportResult> {
  try {
    // 1. Clear existing data
    await this.fusekiService.clearDataset(this.datasetName);

    // 2. Upload new Turtle data
    await this.fusekiService.uploadData(
      this.datasetName,
      turtleData,
      'text/turtle'
    );

    // 3. Rebuild search index
    await this.indexingService.indexAllPhenomena();

    return { success: true, message: 'Import successful' };
  } catch (error) {
    return {
      success: false,
      message: `Import failed: ${error.message}`,
    };
  }
}
```

Listing 22: RDF-Import mit Index-Rebuild

Literatur

[1] International Organization for Standardization. *ISO 21448: Road vehicles – Safety of the intended functionality*. Standard. 2022.

[2] Christian Neurohr, Lukas Westhofen, Martin Butz, Martin Herbert Bollmann, Ulrich Eberle und Roland Galbas. „Criticality Analysis for the Verification and Validation of Automated Vehicles“. In: *IEEE Access* 9 (2021), S. 18016–18041. DOI: 10.1109/ACCESS.2021.3053159.

[3] Christian Neurohr, Lukas Westhofen, Martin Butz, Martin Bollmann, Lina Putze, Tjark Koopmann, Roman Gansch, Michael Knoop, Armin Rasch, Bogdan Cojocaru und Johannes Daube. „Advances on the Criticality Analysis for Automated Driving Systems“. In: (Feb. 2024). DOI: 10.5281/zenodo.10815308.

[4] Tjark Koopmann, Lina Putze, Lukas Westhofen, Roman Gansch, Ahmad Adee und Christian Neurohr. „Grasping Causality for the Explanation of Criticality for Automated Driving“. In: *IEEE Access* 13 (2025), S. 54739–54756. DOI: 10.1109/ACCESS.2025.3555177.

[5] Frank Manola und Eric Miller. *RDF Primer*. W3C Recommendation. Zugriff am: 23.10.2025. W3C, 2004. URL: https://www.w3.org/TR/rdf-primer/.

[6] Eric Prud'hommeaux und Gavin Carothers. *RDF 1.1 Turtle: Terse RDF Triple Language*. W3C Recommendation. Zugriff am: 23.10.2025. W3C, 2014. URL: https://www.w3.org/TR/turtle/.

[7] Steve Harris und Andy Seaborne. *SPARQL 1.1 Query Language*. W3C Recommendation. Zugriff am: 23.10.2025. W3C, 2013. URL: https://www.w3.org/TR/sparql11-query/.

[8] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Diss. University of California, Irvine, 2000. URL: https://www.epai-ict.ch/nexus/repository/course-material/m133/fielding_dissertation.pdf.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever und Dario Amodei. „Language Models are Few-Shot Learners“. In: *Advances in Neural Information Processing Systems*. Hrsg. von H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan und H. Lin. Bd. 33. Curran Associates,

Inc., 2020, S. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser und Illia Polosukhin. „Attention is All you Need“. In: *Advances in Neural Information Processing Systems*. Hrsg. von I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan und R. Garnett. Bd. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[11] Stefan Babisch, Christian Neurohr, Lukas Westhofen, Stefan Schoenawa und Henrik Liers. „Leveraging the GIDAS Database for the Criticality Analysis of Automated Driving Systems“. In: *Journal of Advanced Transportation* (2023). DOI: https://doi.org/10.1155/2023/1349269.

[12] SafetyPool. *SafetyPool*. Zugriff am: 23.10.2025. 2024. URL: https://www.safetypool.ai/.

[13] Michael Schuldes, Christoph Glasmacher und Lutz Eckstein. „scenario.center: Methods from Real-world Data to a Scenario Database“. In: *2024 IEEE Intelligent Vehicles Symposium (IV)*. 2024, S. 1119–1126. DOI: 10.1109/IV55156.2024.10588866.

[14] Notion Labs Inc. *Notion*. Zugriff am: 23.10.2025. 2025. URL: https://www.notion.so/.

[15] Atlassian. *Confluence*. Zugriff am: 23.10.2025. 2025. URL: https://www.atlassian.com/software/confluence.

[16] Obsidian. *Obsidian*. Zugriff am: 23.10.2025. 2025. URL: https://obsidian.md/.

[17] Neo4j Inc. *Neo4j*. Zugriff am: 02.11.2025. 2025. URL: https://neo4j.com/product/bloom/.

[18] Wikimedia Deutschland. *Wikibase*. Zugriff am: 23.10.2025. 2024. URL: https://wikiba.se/.

[19] Seán G Roberts, Anton Killin, Angarika Deb, Catherine Sheard, Simon J Greenhill, Kaius Sinnemäki, José Segovia-Martín, Jonas Nölle, Aleksandrs Berdicevskis, Archie Humphreys-Balkwill, Hannah Little, Christopher Opie, Guillaume Jacques, Lindell Bromham, Peeter Tinits, Robert M Ross, Sean Lee, Emily Gasser, Jasmine Calladine, Matthew Spike, Stephen Francis Mann, Olena Shcherbakova, Ruth Singer, Shuya Zhang, Antonio Benítez-Burraco, Christian Kliesch, Ewan Thomas-Colquhoun, Hedvig Skirgård, Monica Tamariz, Sam Passmore, Thomas Pellard und Fiona Jordan. „CHIELD: the causal hypotheses in evolutionary linguistics database“. In: *Journal of Language Evolution* 5.2 (Apr. 2020), S. 101–120. ISSN: 2058-458X. DOI: 10.1093/jole/lzaa001.

[20] Clemens Linnhoff, Philipp Rosenberger, Simon Schmidt, Lukas Elster, Rainer Stark und Hermann Winner. „Towards Serious Perception Sensor Simulation for Safety Validation of Automated Driving - A Collaborative Method to Specify Sensor Models“. In: *2021 IEEE International Intelligent Transportation*

Systems Conference (ITSC). 2021, S. 2688–2695. DOI: 10.1109/ITSC48978.2021.9564661.

[21] Neo4j Inc. *Neo4j*. Zugriff am: 29.10.2025. 2025. URL: https://neo4j.com/labs/neosemantics/4.0/inference/.

[22] Apache. *Apache Jena*. Zugriff am: 29.10.2025. 2025. URL: https://jena.apache.org/documentation/inference/.

[23] Meilisearch Team. *Meilisearch*. Zugriff am: 01.11.2025. 2025. URL: https://www.meilisearch.com/docs/home.

[24] Haoran Wei, Yaofeng Sun und Yukun Li. „DeepSeek-OCR: Contexts Optical Compression“. In: *arXiv* (2025). DOI: 10.48550/arXiv.2510.18234.

[25] Refactoring.Guru. *Strategy Design Pattern*. Zugriff am: 02.11.2025. 2025. URL: https://refactoring.guru/design-patterns/strategy.

[26] Refactoring.Guru. *Factory Method Design Pattern*. Zugriff am: 02.11.2025. 2025. URL: https://refactoring.guru/design-patterns/factory-method.

[27] Roman Gansch, Lina Putze, Tjark Koopmann, Jan Reich und Christian Neurohr. „Causal Bayesian Networks for Data-Driven Safety Analysis of Complex Systems“. In: *Model-Based Safety and Assessment*. Cham: Springer Nature Switzerland, 2026, S. 222–237. ISBN: 978-3-032-05073-1.

[28] Graphviz. *The DOT Language*. Zugriff am: 02.11.2025. 2025. URL: https://graphviz.org/doc/info/lang.html.

[29] Lukas Westhofen, Christian Neurohr, Tjark Koopmann, Martin Butz, Barbara Schütt, Fabian Utesch, Birte Neurohr, Christian Gutenkunst und Eckard Böde. „Criticality Metrics for Automated Driving: A Review and Suitability Analysis of the State of the Art“. In: *Archives of Computational Methods in Engineering* 30.1 (2023), S. 1–35. DOI: 10.1007/s11831-022-09788-7.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Oldenburg, den 04. November 2025

Lars Streekmann
6419007