HAW HAMBURG

BACHELORARBEIT

Prototypische Evaluation eines LLM-basierten Multi-Agenten-Systems zur Root Cause Analysis unter eingeschränkten Observability-Bedingungen in einer Kubernetes-Testumgebung

vorgelegt am 21.10.2026

Moritz Baron

Erstprüferin: Prof. Dr. Larissa Putzar
Zweitprüfer: Thorsten Wagener

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Fakultät Informatik und Digitale Gesellschaft
Finkenau 35
22081 Hamburg

Kurzfassung

Die Root Cause Analysis (RCA) in Microservice-Systemen wird durch die Abhängigkeiten zwischen den Komponenten erschwert. Diese Arbeit untersucht, wie die Verfügbarkeit von Anwendungslogs und Runbook-/Domänenwissen die Qualität eines lokal ausgeführten, LLM-basierten Multi-Agenten-Systems (MAS) zur RCA beeinflusst. In einer kontrollierten Kubernetes-Testumgebung werden vier reproduzierbare Incidents unter vier Informationsbedingungen untersucht. Anwendungslogs und Runbook-/Domänenwissen bilden ein 2×2-Design. Kubernetes-Zustände, Konfigurationen, Events und HTTP-Probes bleiben konstant verfügbar. Spezialisierte Agenten analysieren die verfügbaren Quellen und führen ihre Befunde zu einer Diagnose zusammen. Für die 16 Testfälle werden jeweils drei RCA-Durchläufe durchgeführt, insgesamt 48 Ausgaben. Bewertet werden fünf Qualitätsdimensionen sowie Halluzinationen und Stabilität. Der mittlere Gesamtwert beträgt 7,58 von 10 Punkten. In 75,0 % der Ausgaben wird die Root Cause korrekt identifiziert, 54,2 % erreichen den Gesamterfolg und 29,2 % den evidenzgestützten Erfolg. Die Ergebnisse zeigen, dass der Nutzen zusätzlicher Informationsquellen vom jeweiligen Incident, ihrem diagnostischen Zusatznutzen und der korrekten Evidenzverarbeitung abhängt. Eine korrekte Root Cause allein gewährleistet daher keine belastbare RCA. Die Aussagekraft ist auf das untersuchte System und die kontrollierten Szenarien begrenzt.

Abstract

Root Cause Analysis (RCA) in microservice systems is complicated by dependencies between individual components. This thesis investigates how the availability of application logs and runbook/domain knowledge affects the quality of Root Cause Analysis performed by a locally deployed, LLM-based multi-agent system (MAS). Four reproducible incidents are examined under four information conditions in a controlled Kubernetes test environment. Application logs and runbook/domain knowledge form a 2×2 experimental design, while Kubernetes states, configurations, events, and HTTP probes remain consistently available. Specialized agents analyze the available information sources and combine their findings into a final diagnosis. Each of the 16 test cases is executed three times, resulting in 48 RCA outputs. These outputs are evaluated across five quality dimensions, as well as hallucinations and stability. The mean quality score is 7.58 out of 10 points. The root cause is correctly identified in 75.0 % of the outputs, 54.2 % achieve overall success, and 29.2 % achieve evidence-supported success. The results show that the benefit of additional information sources depends on the specific incident, their additional diagnostic value, and the correct processing of available evidence. Correctly identifying the root cause alone therefore does not guarantee a reliable RCA. The findings are limited to the investigated system and the controlled scenarios.

Inhaltsverzeichnis

Kurzfassung II

Abstract II

Abbildungsverzeichnis V

Tabellenverzeichnis V

1 Einleitung 1
- 1.1 Motivation und Problemstellung . . . 1
- 1.2 Zielsetzung, Forschungsfrage und Abgrenzung . . . 2
- 1.3 Aufbau der Arbeit . . . 3

2 Fachlicher Hintergrund und Stand der Forschung 4
- 2.1 Root Cause Analysis in verteilten Systemen . . . 4
- 2.2 Informationsgrundlagen der RCA in Kubernetes-Systemen . . . 5
- 2.3 LLM-basierte Agenten und Multi-Agenten-Systeme . . . 7
- 2.4 Stand der Forschung zur LLM-basierten Root Cause Analysis . . . 8
- 2.5 Untersuchungsbedarf und Einordnung der eigenen Arbeit . . . 10

3 Konzeption und Implementierung des Untersuchungssystems 12
- 3.1 Gesamtarchitektur des Untersuchungssystems . . . 12
- 3.2 Kubernetes-basierte Testumgebung . . . 14
- 3.3 Datenerhebung und Aufbereitung der Agenteneingaben . . . 17
- 3.4 LLM-basiertes Multi-Agenten-System . . . 20
- 3.5 Technische Validierung und Reproduzierbarkeit . . . 28

4 Methodik und Versuchsdesign 31
- 4.1 Forschungsdesign und Operationalisierung . . . 31
- 4.2 Incident-Szenarien und Evidenzprofile . . . 33
- 4.3 Testfallbildung und kontrollierte Durchführung . . . 34
- 4.4 Ground Truth und Bewertungsverfahren . . . 35
- 4.5 Aggregation und Vergleich der Informationsbedingungen . . . 39

5 Ergebnisse 41
- 5.1 Überblick über die Versuchsdaten . . . 41

5.2 Ergebnisse nach Informationsbedingungen . . . 42
5.3 Ergebnisse nach Incident-Szenarien . . . 44
5.4 Begründungsqualität, Unsicherheitskommunikation und Halluzination . . . 46
5.5 Stabilität der RCA-Ausgaben . . . 48

6 Diskussion 50
6.1 Zusammenwirken von Informationsverfügbarkeit und Evidenzprofilen . . . 50
6.2 Qualität und Verlässlichkeit der RCA-Ausgaben . . . 54
6.3 Beantwortung der Forschungsfrage . . . 57
6.4 Einordnung in den Forschungsstand . . . 59
6.5 Limitationen und Gefährdungen der Validität . . . 59

7 Fazit und Ausblick 61

Literaturverzeichnis 62

A Runbook: Betrieb und Diagnose der BA-RCA-Testanwendung 64
A.1 Zweck und Abgrenzung . . . 64
A.2 Systemübersicht . . . 64
A.3 Service- und Abhängigkeitskatalog . . . 65
A.4 Verifizierter Baseline-Zustand . . . 66
A.5 Bedeutung der Evidenzquellen . . . 68
A.6 Allgemeiner Diagnoseablauf . . . 69
A.7 Verifikation und Eskalation . . . 70
A.8 Experimentelle Abgrenzung . . . 71

B Ground Truth der Incident-Szenarien 72
B.1 Incident I1 – Fehlendes `DB_HOST` im `order-service` . . . 73
B.2 Incident I2 – Falsches Datenbankpasswort im `order-service` . . . 74
B.3 Incident I3 – Selector-Mismatch beim `Service/postgres` . . . 76
B.4 Incident I4 – Falscher Readiness-Pfad im `api-service` . . . 77

C Bewertungsmatrizen und Entscheidungsregeln 78
C.1 Bewertung einzelner RCA-Ausgaben . . . 78
C.2 Bewertung und Aggregation der Wiederholungen . . . 81
C.3 Bewertungsbogen pro RCA-Ausgabe . . . 82

Eigenständigkeitserklärung 84

Abbildungsverzeichnis

3.1 Gesamtarchitektur des Untersuchungssystems 13

5.1 Ergebnisse der Informationsbedingungen im 2×2-Versuchsdesign 44

Tabellenverzeichnis

3.1 Kubernetes-Baseline der Untersuchungsanwendung 16
3.2 Zusammensetzung der Eingabebündel B1–B4 19
3.3 Verantwortungsverteilung innerhalb der LLM-basierten Verarbeitung . . . 22
3.4 Konfiguration der LLM-Inferenz im Untersuchungssystem 23
3.5 Verpflichtende Initialquellen der Spezialagenten 24
3.6 Zentrale Felder des RCA-Ausgabeschemas 27

4.1 Faktorbelegung der Informationsbedingungen B1–B4 32
4.2 Fehlerklassen, Ursachen und relevante Evidenz der Incident-Szenarien I1–I4 33
4.3 Paarvergleiche zur getrennten Betrachtung der beiden Informationsfaktoren 40

5.1 Umfang der ausgewerteten Versuchsdaten 42
5.2 Gesamtergebnisse nach Informationsbedingungen 42
5.3 Mittlere Bewertung der Qualitätsdimensionen nach Informationsbedingung 43
5.4 Gesamtergebnisse nach Incident-Szenarien 45
5.5 Mittlere Bewertung der Qualitätsdimensionen nach Incident-Szenario . . . 45
5.6 Verteilung der Halluzinationsklassen 47
5.7 Verteilung der Halluzinationstypen nach Informationsbedingung 47
5.8 Beispielhafte Ausprägungen der Halluzinationsbewertung 48
5.9 Stabilität der Testfälle nach Informationsbedingung 48

A.1 Systemübersicht der BA-RCA-Testanwendung 65
A.2 Komponenten und Abhängigkeiten der Testanwendung 65
A.3 Fachliche Endpunkte und ihre Abhängigkeitsketten 66
A.4 Erwartete Baseline-Konfiguration der Anwendungskomponenten 67

A.5 Aussagekraft und Grenzen der verfügbaren Evidenzquellen 68

B.1 Kurzüberblick über die Ground Truth der vier Incident-Szenarien 72
B.2 Stufen zur Beschreibung der Diagnostizierbarkeit 73
B.3 Ground Truth und Evidenzgrenzen für Incident I1 74
B.4 Ground Truth und Evidenzgrenzen für Incident I2 75
B.5 Ground Truth und Evidenzgrenzen für Incident I3 76
B.6 Ground Truth und Evidenzgrenzen für Incident I4 77

C.1 Bewertungsmatrix für die direkte Bewertung einer RCA-Ausgabe 79
C.2 Abgeleitete primäre Erfolgskennzahlen 80
C.3 Schweregrade der Halluzinationsbewertung 80
C.4 Typisierung von Halluzinationen 81
C.5 Regeln zur Stabilitätsbewertung eines Testfalls 81
C.6 Ergänzende Merkmale zur Einordnung der drei Wiederholungen 81
C.7 Aggregationsregeln für die Bewertungskennzahlen 82
C.8 Bewertungsbogen für eine einzelne RCA-Ausgabe 83

1 Einleitung

Verteilte Microservice-Architekturen gewinnen bei der Entwicklung und dem Betrieb moderner Anwendungen zunehmend an Bedeutung. Diese Systeme bestehen meist aus mehreren miteinander interagierenden und voneinander abhängigen Services. Bei komplexen Systemen können sich Fehler aufgrund von Abhängigkeiten auf mehrere Komponenten auswirken, wodurch der Fehlerursprung nicht zwangsläufig aus dem Fehlerbild erkennbar ist. Die Ursachenanalyse erweist sich in solchen Systemen aufgrund der verteilten Struktur oft als herausfordernd (Cornacchia et al., 2025).

1.1 Motivation und Problemstellung

Tritt in einer der nachgelagerten Komponenten ein Fehler auf, können auch die von dieser Komponente abhängigen Services Fehler erzeugen, obwohl sie den eigentlichen Fehler nicht verursachen. Diese Folgefehler zeigen die Auswirkungen und lassen meist nicht direkt auf den Fehlerursprung schließen. Für eine zuverlässige Ursachenanalyse müssen deshalb die Abhängigkeiten zwischen den betroffenen Komponenten betrachtet werden und die beobachteten Symptome im Zusammenhang mit diesen Abhängigkeiten untersucht werden. Eine alleinige Betrachtung des direkt sichtbaren Fehlerzustandes reicht für die Diagnose nicht aus (Cornacchia et al., 2025; Roy et al., 2024).

Unterschiedliche Informationsquellen liefern verschiedene Arten von Diagnosegrundlagen. Für die Ursachenanalyse müssen also unterschiedliche betriebliche Informationen zusammengeführt werden, da eine einzige nicht zwangsläufig ein vollständiges Fehlerbild liefert. Die Verfügbarkeit unterschiedlicher Informationen kann die Qualität einer RCA beeinflussen (Pei et al., 2025; Riddell et al., 2026). Im Kontext dieser Arbeit sind deshalb Anwendungslogs, Kubernetes-Zustände und -Konfigurationen, Events, HTTP-Probes sowie Runbook-/Domänenwissen relevant.

In der Praxis stehen relevante Informationsquellen nicht immer vollständig zur Verfügung, was die Ursachenanalyse erschwert. Fehlen relevante Informationen, kann die Unterscheidung zwischen Symptomen und Ursache erschwert werden. Bei LLM-basierten RCA-Systemen kann dies die Qualität, Nachvollziehbarkeit und Verlässlichkeit der RCA beeinflussen. Es ist daher entscheidend, wie belastbar die RCA auf Basis der tatsächlich verfügbaren Informationen ist (Riddell et al., 2026).

LLM-basierte Systeme werden als Ansatz zur Unterstützung oder sogar Automatisierung von Root Cause Analysis untersucht. Sie bieten die Möglichkeit, unterschiedliche textuelle und technische Informationen gemeinsam auszuwerten (Roy et al., 2024). In Multi-Agenten-Ansätzen können zusätzlich die unterschiedlichen Aufgaben auf spezialisierte Agenten verteilt werden, welche jeweils bestimmte Informationsquellen oder Teilprobleme bearbeiten. Die dadurch entstehenden Teilergebnisse werden anschließend von einer koordinierenden Instanz zu einer Diagnose zusammengeführt (Di et al., 2026).

Neben der Verfügbarkeit der Diagnoseinformationen ist auch die technische Bereitstellung LLM-basierter RCA-Systeme für den betrieblichen Einsatz relevant. Da bei der Ursachenanalyse Anwendungslogs, Konfigurationen und weitere betriebliche Informationen verarbeitet werden, kann die Übermittlung dieser Daten an externe LLM-Dienste Datenschutzrisiken mit sich bringen. Z. Wang et al. (2024) und Sun et al. (2026) berücksichtigen diese Problematik durch die interne Bereitstellung eines Sprachmodells. Die lokale LLM-Ausführung stellt damit eine mögliche Architekturentscheidung dar, um die Übertragung betrieblicher Diagnoseinformationen an externe Modelldienste zu vermeiden.

Besonders relevant ist hierbei die Frage, wie sich ein LLM-basiertes RCA-System verhält, wenn nicht alle grundsätzlich verfügbaren Informationen bereitstehen. Da sich die Verfügbarkeit einzelner Informationsquellen auf die resultierende Diagnose auswirken kann, ergibt sich daraus ein Untersuchungsbedarf für einen kontrollierten Vergleich unterschiedlicher Informationsbedingungen.

1.2 Zielsetzung, Forschungsfrage und Abgrenzung

In dieser Arbeit wird eine prototypische Evaluation eines lokal ausgeführten LLM-basierten Multi-Agenten-Systems zur Root Cause Analysis durchgeführt. Die Untersuchung erfolgt in einer kontrollierten Kubernetes-Testumgebung, wobei der Einfluss einer unterschiedlichen Informationsverfügbarkeit auf die Qualität der RCA analysiert wird. Als Informationsquellen für das MAS dienen Kubernetes-Zustände und -Konfigurationen, Events, HTTP-Probes, Anwendungslogs sowie Runbook-/Domänenwissen. Ausschließlich Anwendungslogs und Runbook-/Domänenwissen werden variiert. Das Ziel ist nicht, die generelle Leistungsfähigkeit von LLMs oder Multi-Agenten-Systemen zu bewerten, sondern vielmehr den Einfluss der genannten Informationsfaktoren unter kontrollierten Bedingungen vergleichend zu untersuchen. Daraus ergibt sich folgende Forschungsfrage:

> Wie beeinflusst die Verfügbarkeit von Anwendungslogs und Runbook-/Domänenwissen die Qualität der Root Cause Analysis eines lokal ausgeführten LLM-basierten Multi-Agenten-Systems in kontrollierten Kubernetes-Incident-Szenarien?

Die Arbeit ist keine Untersuchung von realen Groß- oder Produktionssystemen. Da reale Cloud-Umgebungen Einflussfaktoren aufweisen, die im Rahmen dieser Arbeit nicht kontrolliert werden, wurde eine lokale Testumgebung gewählt, weshalb die Ergebnisse nur eingeschränkt auf produktive Cloud-Umgebungen übertragbar sind. Die lokale Ausführung des Multi-Agenten-Systems stellt dabei eine konstante technische Rahmenbedingung dar und wird nicht als eigenständiger experimenteller Faktor untersucht. Ein Vergleich zwischen lokal ausgeführten und extern bereitgestellten Sprachmodellen ist daher nicht Bestandteil dieser Arbeit. Es werden keine Sprachmodelle trainiert und auch ein umfangreicher Vergleich von Single- und Multi-Agenten-Systemen wird nicht behandelt. Von den Agenten werden zwar Maßnahmen zur Incident-Behebung empfohlen, aber eine automatische Ausführung dieser Maßnahmen ist nicht Teil des Versuchsaufbaus. Der Fokus dieser Arbeit liegt auf den festgelegten Informationsquellen und nicht auf einer vollständigen Observability-Evaluation.

1.3 Aufbau der Arbeit

Der Aufbau der folgenden Kapitel orientiert sich an der IMRAD-Struktur. Zuerst werden in Kapitel 2 fachliche Hintergründe und der Stand der Forschung dargestellt, um eine wissenschaftliche Grundlage zu schaffen. Kapitel 3 beschreibt die Architektur der Kubernetes-Testumgebung und des Multi-Agenten-Systems. Es zeigt, welches System für die Untersuchung geschaffen wurde. In Kapitel 4 wird erklärt, welche Methoden zur Versuchsdurchführung und Auswertung verwendet wurden. Darauf folgt Kapitel 5, in dem die Ergebnisse dargestellt werden. In Kapitel 6 werden diese Ergebnisse interpretiert und eingeordnet. Abschließend folgen in Kapitel 7 ein Fazit sowie ein Ausblick auf mögliche Erweiterungen und weitere Forschung.

2 Fachlicher Hintergrund und Stand der Forschung

Riddell et al. (2026) beschreiben RCA als Bestimmung des zugrunde liegenden Fehlerursprungs eines Incidents und behandeln sie als Aufgabe, bei der der Ursprung eines Fehlers identifiziert und näher klassifiziert wird. Sie betrachten RCA im Kontext LLM-basierter Diagnosesysteme als reasoning-orientierte Aufgabe und unterscheiden dabei drei Diagnoseziele. Das erste ist die *root cause localization*, um den Ursprungsort des Fehlers zu bestimmen. Das zweite Ziel ist die *fault type classification*. Hier geht es darum, die Art des Fehlers zu klassifizieren und einer Fehlerklasse zuzuordnen. Als Letztes wird *propagation path identification* genannt, bei der der Ausbreitungspfad des Fehlers im System bestimmt wird (Riddell et al., 2026). Für die Diagnose sind damit mindestens Fehlertyp, Ursprungsort und Ausbreitungspfad voneinander unterscheidbare Aspekte. Diese Diagnoseaufgaben werden vor allem dann anspruchsvoll, wenn die Fehlerwirkung über mehrere voneinander abhängige Komponenten weitergegeben wird und somit die Bestimmung des Fehlerursprungs erschwert.

2.1 Root Cause Analysis in verteilten Systemen

Diese Fehlerfortpflanzung wird von Riddell et al. (2026) als *multi-hop fault propagation* bezeichnet. Die eindeutige Rückführung des Fehlers auf den Ursprung wird schwieriger, je mehr Abhängigkeiten zwischen Ursache und Symptom liegen. Daher müssen Hinweise aus verschiedenen Komponenten und Datenquellen gemeinsam betrachtet werden. Um prüfen zu können, welche Hinweise auf die eigentliche Ursache hindeuten und welche lediglich Folgefehler dokumentieren, müssen diese Informationen miteinander in Beziehung gesetzt werden. Entscheidend ist daher nicht allein die Verfügbarkeit verschiedener Informationen, sondern deren korrekte Verknüpfung und Interpretation (Cornacchia et al., 2025).

Roy et al. (2024) beschreiben, dass ähnliche Symptome durch unterschiedliche Ursachen entstehen, und nennen Hardware-, Netzwerk- oder Softwarefehler als unterschiedliche Ursachendomänen. Ein beobachtetes Symptom erlaubt für sich allein noch keine eindeutige Zuordnung zur Ursache. Treten aufgrund eines vorgelagerten Fehlers zusätzliche Störungen auf, dürfen diese Folgefehler nicht automatisch mit der Root Cause gleichgesetzt werden. Das bedeutet, dass für eine RCA erst Symptome gesammelt und auf ihre Zusammenhänge geprüft werden müssen, bevor unterschieden werden kann, was Ursache und was Folge des

Fehlers ist. Ist dieser Zusammenhang klar, kann der Fehler auf den eigentlichen Ursprung zurückgeführt werden. Es reicht also nicht, eine mögliche Ursache allein anhand eines beobachtbaren Symptoms zu vermuten. Vielmehr muss die Fehlerzuordnung durch die vorhandenen Informationen nachvollziehbar begründet werden.

Diese Unterscheidung zwischen dem eigentlichen Diagnoseergebnis und dessen Begründung wird auch von Riddell et al. (2026) aufgegriffen. Sie betrachten die Korrektheit der identifizierten Root Cause und die Qualität des zugrunde liegenden Reasonings getrennt. Der erste Aspekt, die Root Cause Accuracy, beschreibt, ob die tatsächliche Ursache korrekt bestimmt wurde. Der zweite Aspekt betrifft die Frage, ob die Qualität der Zwischenschritte, die zur Herleitung der Diagnose geführt haben, nachvollziehbar ist. Nur weil die richtige Root Cause gefunden wurde, bedeutet das nicht, dass auch die verwendete Argumentation korrekt war. Umgekehrt kann auch eine schlüssige, nachvollziehbare Argumentation zu einer falschen Root-Cause-Diagnose führen. Für die Nachvollziehbarkeit ist relevant, welche Hinweise verwendet und wie sie zur Diagnose in Beziehung gesetzt werden. Zusätzlich sollte erkennbar werden, wieso diese Ursache durch die beobachteten Informationen gestützt wird. Dadurch wird zusätzlich zum eigentlichen Diagnoseergebnis auch dessen Informationsgrundlage wichtig.

2.2 Informationsgrundlagen der RCA in Kubernetes-Systemen

Als Grundlage für eine RCA dienen unterschiedliche Arten von Systeminformationen. Diese stammen aus unterschiedlichen Kategorien und sollten deshalb begrifflich getrennt werden. Eine dieser Kategorien ist die Telemetrie. OpenTelemetry (2026b) unterscheidet vor allem Logs, Metrics und Traces als Telemetriesignale, welche dazu dienen, unterschiedliche Aspekte des laufenden Systems zu beschreiben. Des Weiteren gibt es Kubernetes-spezifische Informationen, die von Kubernetes bereitgestellt werden. Darunter fallen Zustandsinformationen, Konfigurationen und Events. Diese Informationen beschreiben Pod-Zustände, gewünschte Konfigurationen oder relevante Ereignisse innerhalb eines Clusters. Zusätzlich existieren Kubernetes-Probes, welche periodisch bestimmte Aspekte von Containerzuständen überprüfen (Kubernetes, 2025b, 2026a, 2026c, 2026d). Eine weitere Informationsgrundlage bildet das Runbook-/Domänenwissen. Es besteht aus betrieblichem Wissen und bekannten Prüf- und Vorgehensweisen (Las-Casas et al., 2024).

Logs entstehen während der Ausführung einer Anwendung und halten Ereignisse bzw. Meldungen aus dem Laufzeitverhalten fest. Typische Inhalte sind Meldungen über Fehlerzustände oder Ausführungsereignisse, die Einblick in das Verhalten einer Anwendung während eines Incidents geben (OpenTelemetry, 2026a). Dadurch können Hinweise auf konkrete Fehler wie Exceptions oder fehlgeschlagene Zugriffe geliefert werden. Allerdings zeigen Logs nur die Perspektive der jeweiligen Komponente und nicht die des vollständigen

Systemzusammenhangs, weshalb eine dort sichtbare Fehlermeldung nicht automatisch die Root Cause ist. Logs sind daher nicht als alleinige Ursachennachweise zu verstehen, sondern müssen für eine RCA mit anderen Informationen in Beziehung gesetzt werden (Cornacchia et al., 2025).

Kubernetes kann Informationen über den aktuellen Zustand seiner Ressourcen bereitstellen. Anwendungen werden in Kubernetes in Pods ausgeführt, die mehrere Container enthalten können. Ein Pod bildet somit die kleinste deploybare Einheit für Anwendungen und besitzt Spezifikationen und Statusinformationen, die von Kubernetes verwaltet werden. Die Spezifikation beschreibt den gewünschten Zustand des Pods, während der Status den von Kubernetes beobachteten Zustand beschreibt. Zum Pod-Status gehören Conditions, die verschiedene Zustandsaspekte beschreiben, sowie eine Phase, die den aktuellen Abschnitt des Pod-Lebenszyklus zusammenfasst. Statusinformationen zeigen den von Kubernetes beobachteten Betriebszustand, woraus abgeleitet werden kann, ob ein Pod läuft oder bereit ist (Kubernetes, 2025b, 2026d, 2026e). Für die RCA wiederum ist wichtig, dass ein auffälliger Zustand zunächst als ein Symptom behandelt wird. So sagt beispielsweise der Diagnosehinweis „Not Ready“ etwas über den Zustand des Pods aus, aber noch nicht, warum dieser Zustand eingetreten ist (Cornacchia et al., 2025). Neben den Statusinformationen stellt Kubernetes auch die Konfiguration seiner Objekte bereit, wodurch sich betrachten lässt, welcher gewünschte Zustand festgelegt ist (Kubernetes, 2025b). Zu diesen Konfigurationsinformationen gehören unter anderem Container- und Deployment-Konfigurationen, Umgebungsvariablen, Labels, Service-Selektoren, Ports, Probe-Konfigurationen und Referenzen auf andere Kubernetes-Objekte wie z. B. Secrets (Kubernetes, 2026b, 2026c, 2026f, 2026g). Diese Informationen können Hinweise darauf geben, welche Einstellungen als mögliche Ursache für den beobachteten Zustand infrage kommen. Wenn die Statusinformationen beispielsweise einen auffälligen Zustand anzeigen, können die Konfigurationen Hinweise auf den Ursprung dieses Zustands liefern (Cornacchia et al., 2025).

Des Weiteren betrachtet Kubernetes nicht nur einzelne Pods, sondern auch deren Erreichbarkeit über Services. Ein Pod ist die laufende Instanz einer Anwendung und kann als Backend eines Services fungieren. Pods sind mit einem oder mehreren Labels versehen, die mit dem Selector eines Services abgeglichen werden. Bei einer Übereinstimmung kann ein entsprechender Pod dem Service als Backend zugeordnet werden. Diese Backend-Zuordnung wird von Kubernetes in EndpointSlices abgebildet, wodurch dem Service passende Endpunkte zur Weiterleitung des Traffics zur Verfügung stehen. Damit lässt sich prüfen, ob einem Service tatsächlich passende Pod-Endpunkte zugeordnet sind (Kubernetes, 2025a, 2026b, 2026g).

Da Kubernetes seine Ressourcen nicht nur verwaltet, sondern auch überwacht, entstehen bei bestimmten Aktionen oder Fehlern sogenannte Events. Diese Events beziehen sich auf konkrete Kubernetes-Objekte wie beispielsweise einen Pod und können Hinweise auf Fehlerzustände oder Vorgänge wie fehlgeschlagene Starts oder Probe-Fehler liefern.

Dadurch ergänzen Events den reinen aktuellen Zustand um aufgetretene Ereignisse. Sie sind keine dauerhafte Ereignishistorie wie klassische Logs, können aber trotzdem zusätzlich Kontext liefern, der in Kombination mit der Konfiguration zur Eingrenzung der Ursache beitragen kann (Kubernetes, 2026a).

Eine weitere Informationsquelle sind Kubernetes-Probes und aktive HTTP-Prüfungen. Eine Probe wird in der Pod-/Container-Konfiguration festgelegt und von Kubernetes regelmäßig ausgeführt. Die Probe prüft einen Zustand, wobei das Ergebnis unterschiedliche Reaktionen von Kubernetes auslösen kann. Hierbei wird zwischen drei Arten von Probes unterschieden. Die Readiness Probe prüft, ob der Container aktuell bereit ist, Anfragen zu verarbeiten. Schlägt diese fehl, wird der Pod als nicht bereit behandelt und kann aus der regulären Service-Weiterleitung herausfallen. Die Liveness Probe hingegen prüft, ob der Container weiterhin funktionsfähig ist. Scheitert die Probe wiederholt, kann ein Containerneustart ausgelöst werden. Die Startup Probe prüft, ob eine Anwendung erfolgreich startet. Sie schützt zusätzlich langsam startende Anwendungen davor, zu früh durch Liveness- oder Readiness-Prüfungen bewertet zu werden (Kubernetes, 2026c). Die aktiven HTTP-Probes hingegen sind keine Kubernetes-Health-Probes im obigen Sinn, sondern vielmehr gezielte HTTP-Anfragen an einen konkreten Service-Endpunkt. Hierbei werden lediglich die Erreichbarkeit und HTTP-Antwort des Services beobachtet. Die Probes liefern nur zusätzliche Diagnosehinweise über das beobachtbare Verhalten (Tundo et al., 2023).

Zusätzlich zu Telemetrie und Kubernetes-spezifischen Informationen gibt es das sogenannte Runbook-/Domänenwissen. Dieses Wissen liefert Informationen über den betrieblichen Kontext bzw. das erwartete Verhalten. Darin werden typische Prüf- und Diagnoseabläufe sowie bekannte Vorgehensweisen zur Fehlerbehandlung dokumentiert. Pei et al. (2025) nutzen hierfür Standard Operating Procedures (SOPs) als betriebliche Wissensquelle, um Diagnoseabläufe bei der RCA zu unterstützen. Solche prozeduralen Vorgaben können den Diagnoseablauf strukturieren und den Aktionsraum der Agenten begrenzen. Allerdings sind Runbooks bzw. dokumentierte Vorgehensweisen nicht immer eindeutig. Las-Casas et al. (2024) thematisieren vor allem ungenaue Sprache als Qualitätsproblem für die RCA.

Die genannten Informationsquellen liefern jeweils nur einen Ausschnitt des Incidentzustands und keine dieser Informationen muss für sich allein die Root Cause eindeutig belegen. Deswegen müssen die Hinweise über Komponenten und Informationsarten hinweg miteinander in Beziehung gesetzt werden. Dadurch können die Symptome von möglichen Ursachen abgegrenzt werden. Diese Zusammenführung ist in der RCA in verteilten Microservice-Systemen zentral (Cornacchia et al., 2025).

2.3 LLM-basierte Agenten und Multi-Agenten-Systeme

Ein LLM verarbeitet einen gegebenen Kontext und erzeugt daraus eine Ausgabe. Eine Agentenarchitektur erweitert das Modell um Mechanismen, welche die Bearbeitung einer

Aufgabe strukturieren. Zu diesen Mechanismen gehören unter anderem Profiling, Memory und Planning. Das Profiling legt die Rolle und das Verhalten eines Agenten fest, während das Planning es ermöglicht, komplexe Aufgaben in Teilschritte zu zerlegen und diese Aufgaben strukturiert zu bearbeiten. Memory sorgt zusätzlich dafür, dass Informationen gespeichert und verwaltet werden. Dadurch kann ein Agent während verschiedener Teilaufgaben darauf zurückgreifen (L. Wang et al., 2024).

Hinzu kommt, dass Agenten sogenannte Aktionen durchführen können. Solche Aktionen ermöglichen die Interaktion mit externen Werkzeugen bzw. Tools, die weitere Informationen liefern können. Yao et al. (2023) beschreiben hierfür einen Action-Observation-Loop. Ein durch das Reasoning abgeleiteter Bearbeitungsschritt wird mit einer Action ausgeführt. Das Ergebnis dieser Aktion wird als neue Beobachtung (Observation) in den Kontext aufgenommen. Diese neuen Informationen dienen anschließend wieder als Grundlage für weitere Reasoning- und Aktionsschritte. Dadurch entsteht eine iterative Aufgabenbearbeitung, was bedeutet, dass ein Agent seine weiteren Schritte nicht ausschließlich aus dem ursprünglichen Eingabekontext ableiten muss.

Um komplexere Aufgaben zu bearbeiten, können Multi-Agenten-Systeme eingesetzt werden. Die Agenten bearbeiten hierbei jeweils abgegrenzte Aspekte der Gesamtaufgabe und liefern entsprechend ihrer Spezialisierung getrennte Ergebnisse (Di et al., 2026). Diese Ergebnisse müssen anschließend koordiniert und zusammengeführt werden, woraus eine gemeinsame Lösung einer komplexen Aufgabe entstehen kann. Die Koordination kann dabei von einer zentralen Instanz erfolgen (Z. Wang et al., 2024). Damit verbinden Multi-Agenten-Systeme die spezialisierte Bearbeitung einzelner Teilaufgaben mit der Zusammenführung der jeweiligen Teilergebnisse.

2.4 Stand der Forschung zur LLM-basierten Root Cause Analysis

LLMs und LLM-basierte Agenten werden bereits in der Informatik verwendet und auch konkret zur Analyse technischer Incidents sowie zur Bestimmung der Fehlerursache eingesetzt. So untersuchen Roy et al. (2024) die LLM-basierte Root Cause Analysis unter realen betrieblichen Bedingungen bei Microsoft. Sie beschreiben, dass reine LLM-Verfahren auf bereits vorhandene Incidentinformationen beschränkt sind. Fehlende Informationen lassen sich hierbei nicht einfach durch weitere Sprachmodell-Reasonings vervollständigen. Die Arbeit thematisiert die Möglichkeit, während der Diagnose zusätzliche Informationen aus der Umgebung abzurufen. Dies wird mit einem ReAct-Agenten, der zusätzliche Daten abfragen und in die weitere Analyse einbeziehen kann, realisiert. Ein wesentliches Ergebnis ist die Reduktion von Halluzinationen im Vergleich zu anderen Methoden wie Retrieval-Augmented Generation (RAG) oder reinem Chain-of-Thought. Bei RAG werden relevante

Informationen aus externen Wissensquellen abgerufen und dem Sprachmodell als zusätzlicher Kontext bereitgestellt. In einer Fallstudie bei Microsoft konnte der ReAct-Agent bei vergleichsweise einfachen Incidents mithilfe von Knowledge Base Articles (KBAs) und zusätzlichen Datenbankabfragen selbstständig Diagnoseschritte ausführen. Bei komplexeren Incidents, die Informationen aus mehreren KBAs erforderten, stieß der Agent dagegen an Grenzen. Eine dieser Grenzen ist, dass bei langen Diagnosepfaden die Kapazität des Kontextfensters überschritten wird und der Agent dadurch überlastet werden kann. Damit zeigt die Studie, dass Agenten die Limitierungen statischer LLM-Modelle durch die Nutzung externen Wissens und externer Tools überwinden und eine fundierte Ursachenanalyse liefern können.

Anders als Roy et al. (2024) nutzen Z. Wang et al. (2024) nicht nur einen einzelnen Agenten, sondern ein MAS für die Diagnose. Sie nutzen für die RCA einen Controller-Agenten zur Koordination und mehrere spezialisierte Experten-Agenten, die jeweils abgegrenzte Teilaufgaben der RCA bearbeiten. Zur Verwaltung des Diagnosekontextes werden sogenannte Observation Snapshots verwendet, die relevante Beobachtungen dokumentieren und diese Informationen für die weitere Analyse zur Verfügung stellen können. Di et al. (2026) nutzen auch einen MAS-Ansatz und erweitern ihn um eine Multi-Specialist-Architektur, die darauf ausgelegt ist, unterschiedliche Informationen bzw. heterogene Daten einzubeziehen. Damit zeigen die Arbeiten, dass neben einem Single-Agent auch stärker strukturierte Multi-Agent-RCA-Systeme untersucht werden.

Neben der Agentenarchitektur spielt auch die Bereitstellung des verwendeten Sprachmodells eine Rolle für den betrieblichen Einsatz LLM-basierter RCA-Systeme. Z. Wang et al. (2024) verwenden bei RCAgent ein intern bereitgestelltes Sprachmodell, da die Übertragung vertraulicher Betriebsdaten an externe LLM-APIs Datenschutzrisiken mit sich bringen kann. Die Autoren weisen zugleich darauf hin, dass mit der Wahl eines lokal bereitgestellten Modells Einschränkungen der Modellleistung einhergehen können, weshalb sie zusätzliche Verfahren zur Stabilisierung der Agenten einsetzen. Auch Sun et al. (2026) berücksichtigen diese Anforderungen bei ihrem Kubernetes-Diagnoseframework MetaKube. Sie verwenden mit KubeLLM ein lokal bereitstellbares, auf Qwen3-8B basierendes Sprachmodell, das durch domänenspezifisches Fine-Tuning für die Kubernetes-Diagnose angepasst wurde. Die Arbeiten zeigen damit, dass die Modellbereitstellung eine relevante Architekturentscheidung darstellt, bei der betriebliche Anforderungen und die Leistungsfähigkeit des Diagnosesystems gemeinsam berücksichtigt werden müssen.

Pei et al. (2025) untersuchen ebenfalls ein LLM-basiertes MAS zur RCA. Die Besonderheit hierbei liegt darin, dass sie SOPs und historische Incidentinformationen integrieren, wodurch der Diagnoseablauf strukturiert wird und den Agenten konkrete Handlungsschritte vorgegeben werden. Durch eine Ablationsstudie untersuchen die Autoren den Beitrag des SOP-Wissens. Allerdings wird beim Entfernen dieses Wissens zugleich der darauf aufbauende SOP-Flow unwirksam, weshalb sich der beobachtete Effekt nicht ausschließlich auf die Verfügbarkeit des Wissens zurückführen lässt. Die Studie zeigt, dass betriebli-

ches Wissen die RCA unterstützen und insbesondere unstrukturierte bzw. halluzinierte Agentenaktionen reduzieren kann.

Trotzdem sagt der Einsatz stärker strukturierter RCA-Systeme noch nichts darüber aus, wie zuverlässig solche Systeme insgesamt arbeiten. Cornacchia et al. (2025) vergleichen deswegen Multi-Agenten- und Single-Agenten-Ansätze mit kontrollierten Failure-Injection-/Microservice-Szenarien. Diese werden wiederholt durchgeführt und mit dem Framework AIOpsLab evaluiert. Dabei wird nicht nur auf die finale Diagnose geachtet, sondern auch auf das Verhalten der Agenten während der Analyse. Ein besonderes Augenmerk wird hier auf fehlerhafte oder halluzinierte API-Aufrufe gelegt. Dabei zeigte sich, dass die untersuchten Multi-Agenten-Konfigurationen stärker zu halluzinierten API-Aufrufen neigen als die betrachteten Single-Agenten-Konfigurationen. Daraus folgt, dass stärkere Arbeitsteilung nicht automatisch auch zu einer zuverlässigeren RCA führt. Die Evaluation berücksichtigt neben der finalen Diagnose auch Fehlverhalten, Halluzinationen und die Ergebnisse wiederholter Ausführungen.

Komplexe Multi-Agenten-Pipelines garantieren keine zuverlässige RCA und können zusätzlich die eigentliche Reasoning-Leistung eines Modells verschleiern. Hier setzen Riddell et al. (2026) an und untersuchen systematisch die Argumentationsleistung von LLMs bei der RCA. Ihr Ziel ist es, die RCA-Schlussfolgerung der Modelle getrennt von komplexen Pipeline- oder Umgebungseinflüssen zu untersuchen. Dazu wurden sechs Open-Source-LLMs unter den Workflows Straight-Shot, ReAct und Plan-and-Execute mit 48.000 simulierten Fehlerszenarien getestet. Untersucht wurde hierbei, wie sich unterschiedlich verfügbare Informationsquellen auf die Diagnose auswirken. Es zeigte sich, dass Metriken insbesondere die Lokalisierung unterstützen, während Logs nützlicher für die Klassifizierung sind. Traces bzw. Ablaufverfolgungen haben teilweise sogar ablenkend gewirkt und die Genauigkeit des Modells verschlechtert. Neben dem Einfluss von Informationen untersuchten die Autoren auch Fehler im eigentlichen Reasoning-Prozess, wobei 16 wiederkehrende Arten von Fehlern festgestellt wurden, die sie in prozedurale, RCA-spezifische und allgemeine logische Fehler unterteilt haben. Das zeigt, dass die Qualität einer LLM-basierten RCA nicht allein von Architektur oder Informationsmenge abhängt, sondern auch davon, welche Informationen verfügbar sind und wie sie im Reasoning verarbeitet werden.

2.5 Untersuchungsbedarf und Einordnung der eigenen Arbeit

Die Ergebnisse aus Abschnitt 2.4 zeigen, dass die Qualität einer LLM-basierten RCA nicht nur von Modell oder von der Agentenarchitektur abhängt, sondern auch von den verfügbaren Informationen. Riddell et al. (2026) untersuchen diesen Einfluss für technische Informationsquellen wie Logs, Metriken und Traces, wobei Teile dieser Informationen gezielt zurückgehalten werden. Pei et al. (2025) greifen einen anderen Aspekt auf, indem Logs als

Informationsquelle verfügbar bleiben und SOP-Wissen gezielt entfernt werden, wodurch der Einfluss betrieblichen Wissens untersucht wird. Da dadurch zugleich die SOP-basierte Ablaufsteuerung beeinflusst wird, lässt sich der isolierte Einfluss der Wissensverfügbarkeit jedoch nicht bestimmen. Weitere Arbeiten wie Z. Wang et al. (2024), Cornacchia et al. (2025) oder Roy et al. (2024) betrachten ebenfalls Logs und Wissensquellen, variieren diese aber entweder über Agenteneinschränkungen oder in getrennten Experimentteilen. In den betrachteten Arbeiten werden Anwendungslogs und Runbook-/Domänenwissen nicht als zwei unabhängige Faktoren im gleichen Versuchsdesign variiert. Daraus ergibt sich ein spezifischer Untersuchungsbedarf bezüglich des getrennten und kombinierten Einflusses dieser Informationsarten auf die Qualität einer LLM-basierten RCA.

Diese Arbeit greift den Untersuchungsbedarf auf und untersucht ein lokal ausgeführtes LLM-basiertes Multi-Agenten-System zur RCA in einer kontrollierten Kubernetes-Testumgebung. Hierbei werden ausschließlich die zwei Faktoren Anwendungslogs und Runbook-/Domänenwissen gezielt variiert. Daraus entsteht ein 2×2-Versuchsdesign mit vier Bedingungen. Weitere technische Informationen wie Kubernetes-Zustände, Konfigurationen, Events und HTTP-Probes bleiben konstant. Durch dieses Versuchsdesign kann der Einfluss der beiden Informationsarten auf dieselben RCA-Fälle getrennt und kombiniert betrachtet werden. Die Ground Truth bleibt dabei unverändert, damit Unterschiede zwischen den Informationsbedingungen untersucht werden können. Das Ziel ist es, zu bewerten, wie sich die Qualität der RCA unter diesen Informationsbedingungen verändert.

Der Beitrag dieser Arbeit ist die kontrollierte Evaluation der Informationsverfügbarkeit, nicht die Entwicklung einer grundsätzlich neuen Agentenarchitektur. Die Qualität der RCA-Ausgaben wird dabei nicht nur anhand der identifizierten Root-Cause bewertet, sondern zusätzlich anhand von mehreren Faktoren wie Lokalisierung, Evidenzbezug, Halluzinationen, Handlungsempfehlungen, Umgang mit unvollständigen Informationen und Stabilität betrachtet. Dadurch ist es möglich, einzuordnen, wie sich eingeschränkte Informationsbedingungen auf die Diagnose auswirken. Die Ergebnisse beziehen sich auf die kontrollierte Kubernetes-Testumgebung und erlauben weder eine allgemeine Bewertung von MAS gegenüber Single-Agenten noch eine Verallgemeinerung auf Produktionssysteme.

3 Konzeption und Implementierung des Untersuchungssystems

Für die Untersuchung wird ein technisches Untersuchungssystem benötigt, mit dem definierte System- und Fehlerzustände kontrolliert hergestellt und analysiert werden können. Als Ausgangspunkt dient eine fehlerfreie Kubernetes-Baseline. Durch gezielte Incident-Injektionen kann die Testumgebung in einen definierten Fehlerzustand versetzt und anschließend wieder auf den Baseline-Zustand zurückgesetzt werden. Dabei müssen erzeugte Zustände durch Kubernetes-Zustände und -Konfigurationen, HTTP-Beobachtungen und Anwendungslogs systematisch erfasst werden können. Zusätzlich muss das Runbook-/Domänenwissen als ergänzender Kontext genutzt werden können. Die Ablage der Informationen soll in voneinander getrennten Gruppen realisiert werden. Die Grundlage für die weitere Verarbeitung bilden dabei die Rohdaten, welche für jeden Incident einmalig als Snapshots gespeichert werden. Die Rohdaten-Snapshots dürfen nach ihrer Erfassung nicht mehr verändert werden, da daraus die späteren Agenteneingaben abgeleitet werden. Außerdem sollten die verschiedenen Informationsquellen voneinander abgrenzbar sein, damit sie bei der Erzeugung der Agenteneingaben vollständig ein- oder ausgeschlossen werden können. All diese Anforderungen sollen dazu beitragen, dass die Verarbeitung der Daten durch das Untersuchungssystem nachvollziehbar wird. Dies umfasst die Erfassung des Incidentzustands, die Speicherung der Rohdaten als Snapshot, die Ableitung der Agenteneingaben sowie die anschließende Verarbeitung durch die einzelnen Agenten bis zur abschließenden RCA-Ausgabe. Zusätzlich sollen Maßnahmen zum Schutz der erhobenen und abgeleiteten Daten verhindern, dass diese unbemerkt im Nachhinein geändert werden. Als zentrale Anforderungen ergeben sich damit Kontrollierbarkeit, Reproduzierbarkeit, Quellentrennung, Datenintegrität und Nachvollziehbarkeit. Diese beziehen sich sowohl auf die gezielte Herstellung und Wiederholung definierter Systemzustände als auch auf die getrennte, unveränderte und eindeutige Verarbeitung der erhobenen Informationen.

3.1 Gesamtarchitektur des Untersuchungssystems

Die in Abbildung 3.1 dargestellte Gesamtarchitektur des Untersuchungssystems besteht aus drei funktional getrennten technischen Bereichen. Die Kubernetes-Testumgebung dient als ausführbare Systemebene. Hier wird die Microservice-Anwendung betrieben und der jeweils erzeugte Systemzustand bereitgestellt. Sie ist der Ausgangspunkt für die

anschließende Datenerhebung. Die Datenerhebungs- und Aufbereitungsebene verbindet die Testumgebung mit dem MAS. Hier werden die Informationen aus der laufenden Testumgebung übernommen und als Rohdaten abgespeichert, woraus dann die Eingaben für die Agenten des MAS aufbereitet werden. Das MAS bildet hierbei die Analyseebene. Es erhält die zuvor aufbereiteten Eingaben aus der Datenpipeline und verarbeitet die verschiedenen Informationsbereiche über spezialisierte Agenten. Durch diesen Aufbau ergibt sich eine klare Verantwortungsverteilung. Die Testumgebung erzeugt den zu untersuchenden Zustand, die Datenpipeline erfasst und bereitet Informationen auf und das Agentensystem analysiert die bereitgestellten Informationen. Diese Bereiche sind funktional voneinander getrennt, aber über definierte Datenübergaben miteinander verbunden.

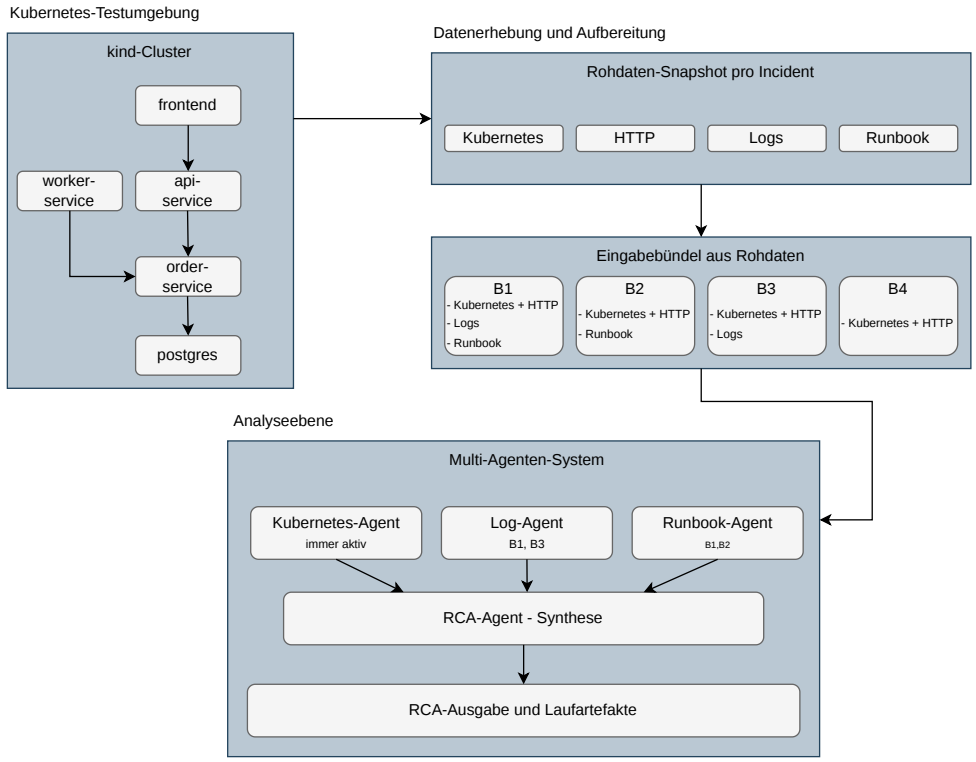


Abbildung 3.1: Gesamtarchitektur des Untersuchungssystems

Die Basis der Datenerhebung ist ein zuvor hergestellter Incidentzustand in der laufenden Kubernetes-Testumgebung. Die Datenerhebung greift auf diesen Zustand zu und fragt Kubernetes-Ressourcen und deren Zustände sowie Anwendungslogs ab. Zusätzlich wird das beobachtete Verhalten der Anwendung durch aktive HTTP-Anfragen überprüft. Diese Informationen werden mit dem Runbook-Kontext gemeinsam als Rohdaten-Snapshot gespeichert, wobei die Informationen nach Quellgruppen getrennt bleiben. Nach abgeschlossener Datenerhebung wird der Live-Zustand des Clusters nicht weiter herangezogen und lediglich der Snapshot als Grundlage für die Weiterverarbeitung der Daten genutzt.

Dadurch werden die Zustandserzeugung und die spätere RCA-Verarbeitung technisch voneinander getrennt.

Dieser Snapshot ist allerdings noch nicht die unmittelbare Eingabe des MAS. Zwischen Rohdaten und Analyse liegt eine eigene Aufbereitungsstufe, damit der ursprüngliche Datenbestand und die tatsächlich verwendeten Agenteneingaben voneinander getrennt werden. In der Datenpipeline werden aus dem Snapshot separate Eingaben für die RCA-Pipeline abgeleitet. Diese Eingabebündel bilden den Übergabepunkt zwischen Datenpipeline und MAS. Das MAS greift dadurch weder auf den laufenden Cluster zu noch führt es erneute Datenerhebungen durch.

Die bereitgestellten Informationen werden nicht von einer einzelnen Analysekomponente verarbeitet, sondern auf spezialisierte Agenten verteilt. Jeder Agent ist für einen unterschiedlichen Informationsbereich zuständig und erzeugt daraus ein Teilergebnis. Diese Teilergebnisse werden anschließend vom RCA-Agenten zu einer abschließenden Root-Cause-Analyse zusammengeführt. Die während der Analyse erzeugten Ergebnisse werden getrennt voneinander gespeichert, sodass die RCA-Ausgabe eindeutig auf die zuvor bereitgestellten Agenteneingaben zurückzuführen ist.

3.2 Kubernetes-basierte Testumgebung

Die Testumgebung wird lokal auf einem Kubernetes-Cluster betrieben, der von kind bereitgestellt wird. Der kind-Cluster bildet dabei zusammen mit einer lokalen Container-Registry die technische Infrastruktur der Untersuchungsumgebung. Die Registry ist unter `localhost:5001` erreichbar und wird getrennt vom Kubernetes-Anwendungszustand betrieben. In `kind-config.yaml` wird containerd so konfiguriert, dass zusätzliche Registry-Konfigurationen für die Anbindung an den Cluster verwendet werden können. Diese Registry-Konfigurationen werden dann in den kind-Node eingebracht und die separat laufende Registry mit dem kind-Netzwerk verbunden. Dadurch kann der Cluster auf die lokal bereitgestellten Containerimages zugreifen. Eine externe Cloud-Plattform ist nicht Bestandteil der Testumgebung. Für die selbst definierten Services `frontend`, `api-service`, `order-service` und `worker-service` werden die Containerimages aus dem jeweiligen Quellcode lokal gebaut, für die lokale Registry getaggt und in die Registry gepusht. Die Kubernetes-Manifeste zeigen auf diese Images, damit eine durchgängige technische Kette von Quellcode über das Docker-Image und die lokale Registry zu dem Kubernetes-Deployment gebildet wird. Einzig Postgres verwendet das vorhandene Image `postgres:16`. Die Ressourcen der Untersuchungsanwendung, insbesondere Deployments, Services, Secrets und daraus erzeugte Pods, werden in dem Namespace `ba-rca` innerhalb des bestehenden kind-Clusters zusammengefasst. Der Cluster und die Registry befinden sich außerhalb des Namespaces, wodurch eine Trennung von bestehender Infrastruktur und veränderbarem Anwendungszustand der Untersuchung entsteht. Die Kubernetes-Ressourcen werden über

jeweilige YAML-Manifeste beschrieben. Darin sind unter anderem verwendete Images, Replikzahlen, Ports, Service-Selektoren, Umgebungsvariablen, Secret-Referenzen und vorhandene Readiness-Probes festgelegt. Damit beschreiben diese Manifeste den gewünschten Soll-Zustand der Ressourcen, während der tatsächliche Laufzeitzustand erst nach der Anwendung dieser Manifeste im Cluster entsteht. Weil sich alle anwendungsbezogenen Ressourcen im Namespace befinden, kann dieser Namespace später entfernt und durch erneutes Ausrollen der Baseline-Konfiguration wiederhergestellt werden. Dies schafft die technische Grundlage für eine kontrollierbare Baseline und spätere Incidentzustände.

Auf dem lokalen kind-Cluster läuft eine Microservice-Anwendung, die aus vier selbst entwickelten Services und einer Datenbank besteht. Die für die Untersuchung relevanten Service-Abhängigkeiten sind in Abbildung 3.1 dargestellt. Die Services `frontend`, `api-service`, `order-service`, `worker-service` und die PostgreSQL-Datenbank sind keine voneinander unabhängigen Dienste, sondern besitzen eine feste Abhängigkeitsstruktur. Das `frontend` bildet den Einstieg für einen nutzernahen Checkout-Vorgang, bei dem das Frontend den `api-service` aufruft. Dieser übernimmt die Anfrage und leitet die eigentliche Bestellerstellung an den `order-service` weiter. Frontend und API besitzen selbst keinen direkten Datenbankzugriff, weshalb der `order-service` die zentrale fachliche Komponente für Bestellungen darstellt. Er besitzt Funktionen zum Erstellen und Auslesen von Orders und greift allein auf PostgreSQL zu. Die Datenbankverbindung wird über `DB_HOST`, `DB_PORT`, `DB_NAME`, `DB_USER`, `DB_PASSWORD` konfiguriert. Dadurch entsteht eine mehrstufige Abhängigkeitsstruktur, was dazu führt, dass sich Fehler in tieferen Komponenten auch auf andere Services auswirken können. Die Services erhalten jeweils die Host- und Portangaben der nachgelagerten Komponenten über Umgebungsvariablen, während der Order-Service die Datenbankparameter erhält.

Die zuvor beschriebene Microservice-Architektur wird in Kubernetes durch jeweils eigene Deployments für `frontend`, `api-service`, `order-service`, `worker-service` und `postgres` umgesetzt. Jedes Deployment definiert dabei unter anderem das zu verwendende Containerimage, die Anzahl der Replikate und die Laufzeitkonfiguration der daraus erzeugten Pods. Die zugehörigen Kubernetes-Services bilden davon getrennte Ressourcen und dienen als stabile Kommunikationspunkte innerhalb des Clusters. Die einzelnen Anwendungen kommunizieren dabei nicht über konkrete Pod-IP-Adressen, sondern adressieren die jeweils nachgelagerten Komponenten über deren Kubernetes-Service-Namen und Ports. Die Zuordnung eines Services zu den zugehörigen Pods erfolgt über Labels und Selektoren. Die durch ein Deployment erzeugten Pods erhalten definierte Labels, während der jeweilige Service über einen passenden Selektor die entsprechenden Pods adressiert. Kubernetes kann daraus die dem Service zugehörigen Endpoints beziehungsweise EndpointSlices bilden. Stimmen Selektor und Label nicht überein, kann der Service weiterhin bestehen, besitzt jedoch keine passenden Pods als Backend für die Kommunikation. Neben den internen Serviceverbindungen wird der Service `frontend` als `NodePort` auf Port `30082` bereitgestellt, während `api-service`, `order-service`, `worker-service` und `postgres` als `ClusterIP`-Services konfiguriert sind. Für den Datenbankzugriff benötigt

der `order-service` neben Host, Port, Datenbankname und Benutzername auch ein Passwort. Dieses wird nicht direkt im Deployment hinterlegt, sondern über eine `secretKeyRef` aus dem Kubernetes-Secret `order-service-db-secret` bezogen. PostgreSQL verwendet mit `postgres-secret` ein getrenntes Kubernetes-Secret. In der Baseline enthalten beide Secrets denselben Passwortwert, sodass die Zugangsdaten des `order-service` mit der PostgreSQL-Konfiguration übereinstimmen. Zusätzlich sind für den `api-service` und PostgreSQL Readiness-Probes konfiguriert. Der `api-service` wird dabei über eine HTTP-Probe auf dem Pfad `/ready` geprüft, während PostgreSQL eine `exec`-Probe mit `pg_isready` verwendet. Damit werden zentrale technische Abhängigkeiten der Anwendung über explizite Kubernetes-Konfigurationen wie Service-Selektoren, Umgebungsvariablen, Secrets und Readiness-Probes umgesetzt. Die dadurch entstehende Baseline wird in Tabelle 3.1 dargestellt.

Tabelle 3.1: Kubernetes-Baseline der Untersuchungsanwendung

Komponente	Replikate	Service-Typ	Port	Readiness-Probe
`frontend`	1	`NodePort`	8002 (30082)	–
`api-service`	1	`ClusterIP`	8000	HTTP `/ready`
`order-service`	1	`ClusterIP`	8001	–
`worker-service`	1	`ClusterIP`	8003	–
`postgres`	1	`ClusterIP`	5432	`pg_isready`

Der Ausgangspunkt für die Baseline sind die zuvor beschriebenen Baseline-Manifeste, welche die Konfiguration der fünf Komponenten enthalten. Durch das Ausrollen dieser Konfiguration entsteht die Baseline im Cluster. Für das Deployment wurde ein zentrales Skript `deploy.sh` erstellt, das den gesamten Deployment-Ablauf übernimmt, wodurch ein fester technischer Ablauf entsteht. Das Skript übernimmt die Imagebereitstellung, indem es die vier Images der entwickelten Services baut, für `localhost:5001` taggt und in die lokale Registry pusht. Danach werden Namespace und Secrets angewendet und PostgreSQL bereitgestellt. Erst wenn der Rollout von PostgreSQL abgeschlossen ist, startet das Ausrollen der anderen vier Services. Nachdem die Manifeste der einzelnen Services angewendet wurden, werden die Anwendungsdeployments neu gestartet und auf die jeweiligen Rollouts gewartet. Zum Abschluss werden Pods, Services, Endpoints und EndpointSlices ausgegeben. Um die Anwendungsressourcen nach einem Incident zu löschen bzw. zurückzusetzen, wurde ein weiteres Skript erstellt. Das Skript `reset.sh` sorgt dafür, dass der Namespace `ba-rca` gelöscht und alle darin enthaltenen anwendungsbezogenen Kubernetes-Ressourcen entfernt werden. Der kind-Cluster und die lokale Registry bleiben erhalten. Nachdem `reset.sh` erfolgreich durchgelaufen ist, kann `deploy.sh` erneut ausgeführt werden, wodurch Namespace, Secrets, Deployments und Services wieder aus der Baseline-Konfiguration aufgebaut werden. Dadurch wird nicht ein veränderter Zustand rückgängig gemacht, sondern der Zustand wird entfernt und anschließend aus der definierten Baseline neu erzeugt. Damit wird die geforderte Kontrollierbarkeit technisch umgesetzt. Sobald die Baseline wieder-

hergestellt ist, kann die Testumgebung gezielt in einen definierten Fehlerzustand versetzt werden.

Die Incident-Injektion erfolgt über das zentrale Skript `inject-incident.sh`. Beim Ausführen des Skripts wird zusätzlich eine Incident-ID `i1` bis `i4` mitgegeben, wodurch je nach ID eine definierte, incidentspezifische Kubernetes-Konfiguration angewendet wird. Danach werden wie bei `deploy.sh` Pods, Services und Endpoints ausgegeben, um den entstandenen Clusterzustand sichtbar zu machen. Bei Incident 1 wird eine erforderliche Umgebungsvariable im `order-service` nicht gesetzt. Dazu skaliert das Skript zunächst das Deployment `order-service` auf null Replikate und wartet auf den Abschluss dieser Skalierung. Anschließend wird das Incidentmanifest angewendet, in dem die in der Baseline enthaltene Umgebungsvariable `DB_HOST=postgres` fehlt. Die übrigen Deployment-Konfigurationen bleiben gegenüber der Baseline unverändert. Nach der Anwendung wartet das Skript 10 Sekunden, damit sich der fehlerhafte Zustand ausbilden kann. Für Incident 2 wird ein falsches Datenbankpasswort gesetzt. Hierbei bleibt das PostgreSQL-Secret unverändert, während der Wert des Schlüssels `DB_PASSWORD` im Kubernetes-Secret `order-service-db-secret` durch `wrong_ba_password` ersetzt wird. Im Anschluss wird das Deployment `order-service` neu gestartet, wodurch der neu erzeugte Pod den veränderten Secret-Wert erhält. Anschließend wartet das Skript auf den Rollout des Deployments `order-service`. Für Incident 3 wurde ein Selector-Mismatch des PostgreSQL-Services umgesetzt. Der PostgreSQL-Pod behält sein Baseline-Label, während der Selector des Services von `app: postgres` auf `app: postgres-broken` gesetzt wird. Dadurch passen Selector und Pod-Label nicht mehr zueinander. Am Ende wartet das Skript fünf Sekunden auf die Aktualisierung der Endpoints. Als letzter Incident wurde eine fehlerhafte Readiness-Probe des Services `api-service` definiert. In der Baseline ist für den `api-service` eine Readiness-Probe mit dem Pfad `/ready` gesetzt. Das Incidentmanifest ändert diesen Pfad auf `/invalid-ready`. Vor der Anwendung des fehlerhaften Manifests wird der `api-service` noch auf 0 Replikate reduziert, um sicherzustellen, dass kein alter, funktionierender Baseline-Pod übrig bleibt. Das Skript wartet, bis der Scale-Down der Replikate abgeschlossen ist, wendet das Incidentmanifest an und wartet dann zehn Sekunden, damit Kubernetes den neu erzeugten Pod starten und die fehlerhafte Readiness-Probe ausführen kann. Die Incident-Injektionen greifen ausschließlich in definierte Kubernetes-Konfigurationen beziehungsweise Secrets ein. Eine zusätzliche Fehlerlogik im Anwendungscode wird dafür nicht verwendet.

3.3 Datenerhebung und Aufbereitung der Agenteneingaben

Nachdem eine Incident-Injektion stattgefunden hat, befindet sich der Cluster in dem zu untersuchenden Fehlerzustand. Dieser Zustand bildet den Ausgangspunkt der Datenerhebung und das Skript `collect-data.sh` wird mit genau einer Incident-ID aufgerufen.

Dadurch wird der Zielpfad `data/snapshots/<incident-id>` für den späteren Snapshot festgelegt. Um sicherzustellen, dass die Datenerhebung möglich ist, prüft das Skript zuerst, ob `kubectl`, `sha256sum` und das Runbook vorhanden sind und ob der Namespace `ba-rca` über `kubectl` abfragbar ist. Schlägt eine dieser Voraussetzungen fehl, beginnt die Snapshot-Erhebung gar nicht erst. Zusätzlich wird geprüft, ob für die mitgegebene Incident-ID bereits ein Snapshot-Verzeichnis existiert. Ist das der Fall, beendet sich das Skript. Damit wird sichergestellt, dass bestehende Snapshots nicht einfach ersetzt oder überschrieben werden. Sind alle Prüfungen ohne Probleme abgeschlossen, wird ein temporäres Verzeichnis unter `data/snapshots` erzeugt und die spätere Rohdatenstruktur mit `kubernetes`, `http`, `logs` und `runbook` erzeugt. Hier findet die Datenerhebung statt und es wird geprüft, ob alle vorgesehenen Snapshot-Bestandteile vorhanden sind. Erst nach erfolgreichem Abschluss dieser Prüfungen wird der Snapshot finalisiert und in das endgültige Incident-Verzeichnis verschoben. Damit liegt der Snapshot nach erfolgreichem Abschluss als abgeschlossener Datenbestand unter `data/snapshots/<incident-id>` vor.

Im temporären Snapshot-Verzeichnis werden unter `raw/` vier separate Quellgruppen angelegt: `kubernetes/`, `http/`, `logs/` und `runbook/`. Unter `kubernetes/` werden Namespace-weite Übersichten zu Pods, Pod-Labels, Deployments und EndpointSlices sowie Secret-Metadaten ohne Secret-Werte erfasst. Ergänzend werden die Deployment-Konfigurationen von `api-service`, `order-service` und `postgres` als YAML, die Pod-`describe`-Ausgaben von `api-service` und `order-service` sowie die YAML- und `describe`-Ausgaben des PostgreSQL-Service gespeichert. Kubernetes-Events werden dabei über die Pod-`describe`-Ausgaben erfasst. Dadurch enthält der Snapshot sowohl allgemeine Zustandsübersichten als auch gezielt ausgewählte Konfigurations- und Zustandsinformationen. Zusätzlich werden HTTP-Probes ausgeführt und unter `http/` gespeichert. Für jede Probe werden Name, HTTP-Methode, Ziel-URL, Antwort und `curl`-Exit-Code in einer eigenen Datei gespeichert. Für alle fünf Komponenten versucht das Skript, aktuelle Containerlogs unter `logs/` zu erfassen. Zusätzlich wird versucht, die Logs der vorherigen Containerinstanz abzurufen. Bei beiden Abrufen werden Zeitstempel ausgegeben und die Ausgabe jeweils auf die letzten 200 Zeilen begrenzt. Sind vorherige Logs nicht verfügbar, wird dies protokolliert, ohne die Erhebung abzubrechen. Sind für eine Komponente weder aktuelle noch vorherige Logs verfügbar, wird die Erhebung mit einem Fehler beendet. Abschließend wird das Runbook aus `docs/runbooks/system-runbook.md` unverändert nach `runbook/` kopiert. Inhaltlich umfasst das Runbook die System- und Abhängigkeitsstruktur der Testanwendung, den erwarteten Baseline-Zustand mit relevanten Konfigurationswerten sowie allgemeine Diagnoseprinzipien zur Untersuchung von Systemzuständen und zur Einordnung verfügbarer Evidenz. Es enthält hingegen keinen eigenständigen Katalog fehlerbildspezifischer Diagnose- und Behebungsabläufe. Das Runbook stellt somit sowohl betriebliches Systemwissen als auch allgemeine methodische Hinweise für die RCA bereit. Das Runbook wurde unter fachlicher Anleitung des Verfassers mithilfe von ChatGPT (OpenAI) ausgearbeitet und strukturiert. Als fachliche Grundlage dienten der Aufbau und die Konfiguration der implementierten Testumgebung. Die im Experiment verwendete Fassung ist vollständig in

Anhang A dokumentiert.

Nach der Datenerhebung wird von `collect-data.sh` `validate_snapshot()` aufgerufen. Hierbei wird geprüft, ob unter `kubernetes/` genau die zwölf in der experimentellen Dateiliste festgelegten Kubernetes-Dateien vorhanden, nicht leer und reguläre Dateien sind sowie ob das Runbook unter `runbook/` vorhanden und nicht leer ist. Außerdem muss für jede Komponente mindestens eine Logdatei und unter `http/` müssen genau zwölf Probe-Dateien vorliegen. Bei Abweichungen wird die Snapshot-Erstellung abgebrochen. Nach erfolgreicher Prüfung wird eine `metadata.txt` erzeugt, die Incident-ID, Namespace, Erhebungszeitraum, verwendeten `kubectl`-Kontext, Git-Commit, Runbook-Pfad und Raw-Datenverzeichnis dokumentiert. Zusätzlich werden für sämtliche Dateien unter `raw/` SHA-256-Prüfsummen als Grundlage der späteren Integritätsprüfung gespeichert. Die Datenerhebung wird durch zeitgestempelte Protokolle der einzelnen Schritte und möglicher Fehler dokumentiert. Abschließend wird das temporäre Arbeitsverzeichnis nach `data/snapshots/<incident-id>` verschoben. Der finalisierte Snapshot dient anschließend als Grundlage für die Ableitung der Agenteneingaben.

Die Ableitung der Agenteneingaben erfolgt durch das Skript `derive-inputs.sh`, das mit der jeweiligen Incident-ID aufgerufen wird. Es arbeitet ausschließlich mit den gespeicherten Dateien unter `data/snapshots/<incident-id>/raw` und den zugehörigen SHA-256-Prüfsummen. Vor der Ableitung prüft das Skript, ob alle vier Quellverzeichnisse `kubernetes/`, `http/`, `logs/` und `runbook/` vorhanden sind und Dateien enthalten. Außerdem werden die gespeicherten Snapshot-Prüfsummen überprüft. Wie bei der Datenerhebung wird ein temporäres Arbeitsverzeichnis unter `data/experiment-inputs/` erstellt, das erst nach erfolgreicher Prüfung finalisiert wird. Besteht das Zielverzeichnis `data/experiment-inputs/<incident-id>` bereits, wird die Ableitung abgebrochen, um ein Überschreiben bestehender Eingaben zu verhindern. Im temporären Verzeichnis werden die vier Eingabebündel `b1/` bis `b4/` sowie ein `_audit/`-Verzeichnis für Prüf- und Auditdaten angelegt. Zunächst werden die konstanten Quellgruppen `kubernetes/` und `http/` in alle vier Eingabebündel kopiert. Anschließend erfolgt die gezielte Verteilung der optionalen Quellgruppen. Die Zusammensetzung der Eingabebündel ist in Tabelle 3.2 dargestellt.

Tabelle 3.2: Zusammensetzung der Eingabebündel B1–B4

Eingabebündel	Kubernetes	HTTP	Logs	Runbook
B1	enthalten	enthalten	enthalten	enthalten
B2	enthalten	enthalten	–	enthalten
B3	enthalten	enthalten	enthalten	–
B4	enthalten	enthalten	–	–

Die letztendliche Prüfung, dass alle Eingabebündel die richtigen Kopien besitzen und die ausgeschlossenen Quellen vollständig fehlen, wird in folgendem Abschnitt behandelt. Nach

dieser Kopieroperation liegen in dem temporären Arbeitsverzeichnis vier unterschiedlich zusammengesetzte Eingabebündel vor.

Nach der Kopieroperation kontrolliert `derive-inputs.sh` die oberste Verzeichnisstruktur für jedes Eingabebündel entsprechend Tabelle 3.2. Anschließend verwendet das Skript rekursive `diff`-Vergleiche. B1 muss dabei vollständig byte-identisch zum gesamten Raw-Verzeichnis sein. Zusätzlich werden die Quellgruppen `kubernetes/` und `http/` für B1 bis B4 jeweils mit den entsprechenden Raw-Dateien verglichen. Außerdem müssen die vorhandenen Logs in B1 und B3 sowie das `runbook/` in B1 und B2 ebenfalls byte-identisch zu `raw/logs/` bzw. `raw/runbook/` sein. Des Weiteren prüft das Skript nicht nur, ob vorhandene Quellen korrekt sind, sondern auch, ob die ausgeschlossenen Quellen entsprechend 3.2 tatsächlich fehlen. Erst nach diesen Prüfungen werden für jedes Eingabebündel Auditdateien erzeugt und unter `_audit/` abgelegt. Hierbei entstehen für B1 bis B4 jeweils eigene Checksumdateien, in denen für jede Datei eines Bündels ein SHA-256-Hash gespeichert wird. Zusätzlich zu der Erstellung der Checksums werden die vier Checksumdateien selbst gehasht und gemeinsam in der Datei `input-hashes.txt` gespeichert. Dadurch wird für jedes Eingabebündel ein prüfbarer Bezug zu seinem vollständigen Dateibestand hergestellt. Außerdem wird im `_audit/`-Verzeichnis eine `derivation.txt` erzeugt, die unter anderem Incident-ID, verwendeten Quell-Snapshot, den Pfad der Checksums des verwendeten Quell-Snapshots, den SHA-256-Hash dieser Checksumdatei, den Zeitpunkt der Ableitung und die Zusammensetzung von B1 bis B4 dokumentiert, um die Zuordnung von Snapshot und Agenteneingaben nachvollziehbar zu machen. Erst nach erfolgreicher Strukturprüfung, erfolgreichen Byte-Vergleichen und der Erzeugung der Auditdaten wird das temporäre Arbeitsverzeichnis nach `data/experiment-inputs/<incident-id>` verschoben. Damit liegen B1 bis B4 als finalisierte Eingabebündel für den RCA-Prozess bereit.

3.4 LLM-basiertes Multi-Agenten-System

Für das Untersuchungssystem wurde eine lokale Ausführung des LLM-basierten Multi-Agenten-Systems gewählt. Dadurch können das verwendete Sprachmodell, die Modellruntime und die Modellparameter für die Versuchsdurchführung festgelegt und dokumentiert werden. Gleichzeitig müssen die bereitgestellten Incident- und Observability-Daten für die Modellinferenz nicht an einen externen LLM-Dienst übertragen werden. Die betriebliche Relevanz dieser Bereitstellungsform wird auch von Z. Wang et al. (2024) und Sun et al. (2026) aufgegriffen. Die technische Umsetzung erfolgt mit Ollama als Modellruntime und Qwen3:8b als verwendetem Sprachmodell. Die Modellanbindung erfolgt über die Ollama-Chat-API, deren Tool-Calling-Funktionalität in einer offiziellen Veröffentlichung dokumentiert wird (Ollama, 2025). Die Sampling-Konfiguration orientiert sich an den dokumentierten Empfehlungen der offiziellen Qwen3-8B-Modellkarte (Qwen, 2025). Das MAS besteht aus einem Kubernetes-Agenten, einem Log-Agenten, einem Runbook-Agenten und einem nachgelagerten RCA-Agenten. Die drei spezialisierten Agenten untersuchen

jeweils ihren eigenen Informationsraum und ihre Ergebnisse werden anschließend von dem RCA-Agenten zusammengeführt. Eine Peer-to-Peer-Kommunikation zwischen den Agenten ist nicht Bestandteil der Architektur. Die Einordnung des Multi-Agenten-Systems in das Untersuchungssystem sowie die Aufteilung in spezialisierte Agenten und den abschließenden RCA-Agenten sind in Abbildung 3.1 dargestellt. Bei der Implementierung des Multi-Agenten-Systems wurde ChatGPT (OpenAI) als unterstützendes Werkzeug eingesetzt.

Die vier Agentenrollen verwenden hierbei nicht vier unterschiedliche Modelle, sondern basieren alle auf derselben konfigurierten Qwen3:8b-Laufzeit. Die funktionale Trennung der vier Rollen entsteht durch unterschiedliche Systeminstruktionen, unterschiedliche Aufgaben, unterschiedliche Informationsräume und unterschiedliche Verarbeitungsschritte, wodurch ein Agent in dieser Implementierung als separat orchestrierte Modellrolle und nicht als eigenständiger Modellserver oder LLM-Prozess zu verstehen ist. Somit wird Ollama als LLM-Laufzeit verwendet und Python kommuniziert mit der lokal bereitgestellten Ollama-Instanz. Hierbei bildet `llm_client.py` die technische Verbindung zwischen Python-Pipeline und Ollama. Modellanfragen werden über HTTP an die Ollama-Chat-API übertragen. Ollama stellt dabei die lokale Laufzeit für Qwen3:8b sowie die Chat-, Thinking- und Tool-Calling-Schnittstelle bereit und übernimmt die Verarbeitung der Nachrichtenhistorie sowie die Tokengenerierung. Die bereitgestellten Tool-Definitionen werden dem Modell übergeben, sodass Qwen3:8b bei Bedarf strukturierte Tool-Calls erzeugen kann. Deren Verarbeitung sowie die tatsächliche Ausführung der zugelassenen Werkzeuge übernimmt anschließend die Python-Pipeline. Sie übernimmt darüber hinaus die Auswahl des auszuführenden Agenten, die Erstellung seines Informationsraums, die Übergabe der Agenteninstruktionen, die Rückgabe der Tool-Ergebnisse an das Modell, die Kontrolle der Abschlussbedingungen und die Weiterleitung der Spezialergebnisse an den RCA-Agenten. Damit erhält das Modell keinen freien Systemzugriff und ein Tool-Call ist zunächst lediglich eine strukturierte Anforderung. Erst die Python-Pipeline entscheidet anhand der implementierten Werkzeug- und Zugriffslogik, wie dieser Aufruf verarbeitet wird. Dieser Ablauf wird von Ollama als Tool-Calling-Prinzip dokumentiert. Die Verantwortungsverteilung zwischen Sprachmodell, Modellruntime und Python-Pipeline ist in Tabelle 3.3 zusammengefasst.

Zusätzlich zum Tool-Calling wird der Thinking-Modus verwendet, welcher von Ollama explizit für Qwen3 unterstützt wird. Diese Einstellung wird von `llm_client.py` bei Modellaufrufen an Ollama übergeben. Zusätzlich werden zentrale Sampling-Parameter an Qwen3:8b übergeben. Die Parameter `temperature`, `top_p`, `top_k` und `min_p` entsprechen den dokumentierten Empfehlungen für Qwen3-8B im Thinking-Modus (Qwen, 2025). Die Größe des Kontextfensters wurde mit `num_ctx=16384` an die verfügbare Hardware angepasst, sodass die Modellausführung überwiegend auf der GPU erfolgen konnte. Die konfigurierte maximale Ausgabelänge mit `num_predict=32768` orientiert sich an der von Qwen für Qwen3 verwendeten Generierungslänge. Zusätzlich wurden mit `presence_penalty=1.5` und `repeat_penalty=1.0` weitere Generierungsparameter fest vorgegeben, für deren konkrete Werte keine separate Optimierung durchgeführt wurde. Die Modellparameter sind in

Tabelle 3.3: Verantwortungsverteilung innerhalb der LLM-basierten Verarbeitung

Komponente	Aufgabe
Qwen3:8b	Sprachmodell-Inferenz, Tokengenerierung und Erzeugung strukturierter Tool-Calls auf Grundlage der übergebenen Tool-Definitionen.
Ollama	Lokale Bereitstellung und Ausführung des Modells sowie Bereitstellung der Chat-, Thinking- und Tool-Calling-Schnittstelle.
Python-Pipeline	Orchestrierung der Agenten, Bereitstellung der jeweiligen Informationsräume und Instruktionen, Verarbeitung und Ausführung der Tool-Calls, Rückgabe der Tool-Ergebnisse, Kontrolle der Abschlussbedingungen sowie Weiterleitung der Spezialanalysen an den RCA-Agenten.

der Tabelle 3.4 aufgelistet.

Die Ausführung der Spezialagenten innerhalb eines RCA-Laufs erfolgt sequenziell. Zuerst wird der Kubernetes-Agent ausgeführt, dann der Log-Agent, dann der Runbook-Agent und letztendlich der RCA-Agent. Ist die jeweilige Quellgruppe nicht vorhanden, wird der Log- beziehungsweise Runbook-Agent nicht ausgeführt und für die spätere RCA-Synthese stattdessen ein fest definierter Hinweistext verwendet. Die Spezialagenten werden in voneinander getrennten Sessions ausgeführt, wobei ihre Zwischenergebnisse nicht an die jeweils anderen Sessions weitergegeben werden. Nach Abschluss der Spezialagenten sammelt `run_rca.py` die einzelnen Teilergebnisse und kombiniert sie zu einem neuen Eingabetext für den RCA-Agenten. Der RCA-Agent erhält nicht erneut die ursprünglichen Primärdateien, sondern ausschließlich die zuvor erzeugten Spezialanalysen. Dadurch lässt sich die Verarbeitung in zwei Stufen gliedern: die agentenspezifische Verarbeitung der Primärdaten und die anschließende Synthese der Spezialanalysen zur finalen RCA.

Die vorbereiteten Quelldateien werden durch `file_loader.py` eingelesen und unter ihrem relativen Dateipfad im Speicher abgelegt. Unterstützt werden dabei die Quellgruppen `kubernetes`, `http`, `logs` und `runbook`. Pro Datei werden maximal 20.000 Zeichen übernommen. Überschreitet eine Datei diese Grenze, wird der weitere Inhalt abgeschnitten und die Kürzung im geladenen Text gekennzeichnet. Anschließend teilt die Pipeline die geladenen Daten auf die drei Spezialagenten auf. Der Kubernetes-Agent erhält ausschließlich Kubernetes- und HTTP-Daten, der Log-Agent ausschließlich Anwendungslogs und der Runbook-Agent ausschließlich die Runbook-Daten. Für jeden Spezialagenten erzeugt die Pipeline eine eigene `SourceTools`-Instanz, die nur die für diesen Agenten ausgewählten Dateien enthält. Die Agenten greifen damit nicht direkt auf das Dateisystem zu, sondern ausschließlich auf die bereits geladenen Inhalte ihres jeweiligen Informationsraums. Beim Aufbau wird zusätzlich geprüft, ob alle enthaltenen Dateien einer für den Agenten zugelassenen Quellgruppe zugeordnet sind. Aus den verfügbaren Dateien erzeugt die Pipeline anschließend ein Source Manifest, das dem Agenten die vorhandenen relativen Dateipfade

Tabelle 3.4: Konfiguration der LLM-Inferenz im Untersuchungssystem

Parameter	Wert	Funktion und Einordnung
Thinking	aktiviert	Aktiviert den Thinking-Modus von Qwen3 für die Modellaufrufe.
`temperature`	0.6	Beeinflusst die Streuung bei der Tokenauswahl und entspricht der dokumentierten Qwen3-Empfehlung für den Thinking-Modus.
`top_p`	0.95	Begrenzt die Tokenauswahl anhand ihrer kumulierten Wahrscheinlichkeit und entspricht der dokumentierten Qwen3-Empfehlung für den Thinking-Modus.
`top_k`	20	Begrenzt die Auswahl auf die wahrscheinlichsten Token und entspricht der dokumentierten Qwen3-Empfehlung für den Thinking-Modus.
`min_p`	0.0	Legt eine relative Mindestwahrscheinlichkeit für berücksichtigte Token fest und entspricht der dokumentierten Qwen3-Empfehlung für den Thinking-Modus.
`presence_penalty`	1.5	Beeinflusst die erneute Verwendung bereits aufgetretener Inhalte. Der konkrete Wert wurde nicht separat optimiert.
`repeat_penalty`	1.0	Gewichtet die Wiederholung bereits erzeugter Token. Der Wert 1.0 entspricht keiner zusätzlichen Wiederholungsstrafe und wurde nicht separat optimiert.
`num_ctx`	16384	Legt die bereitgestellte Kontextgröße fest und wurde an die verfügbare Hardware angepasst, um die Modellausführung überwiegend auf der GPU zu ermöglichen.
`num_predict`	32768	Legt die konfigurierte Obergrenze der Generierung fest und orientiert sich an der für Qwen3 dokumentierten maximalen Generierungslänge.

auflistet. Dateiinhalte werden darin nicht übermittelt. Soll ein Agent den Inhalt einer Quelle untersuchen, muss er die entsprechende Datei mit `read_file` aufrufen. Da nur Dateien aus der zuvor erzeugten `SourceTools`-Instanz gelesen werden können, bleibt der Zugriff eines Spezialagenten auf seinen festgelegten Informationsraum beschränkt.

Um diesen Informationsraum nun zu analysieren, wird für jeden Spezialagenten eine eigene Multi-Turn-Session gestartet. Diese Session enthält die Systeminstruktionen des jeweiligen Agenten, seine konkrete Aufgabe sowie die für ihn freigegebene Tool-Definition. Zusätzlich bleibt die Nachrichtenhistorie während der gesamten Session erhalten. Innerhalb einer Session kann ein Modellaufruf entweder einen oder mehrere Tool-Calls anfordern oder eine abschließende Analyse zurückgeben. Nach ausgeführtem Tool-Call werden die Ergebnisse wieder in dieselbe Session eingebracht und ein weiterer Modellaufruf durchgeführt, wodurch ein mehrstufiger Agentenablauf aus Modellentscheidung, Tool-Call, Tool-Ergebnis und nächstem Modellschritt entsteht. Den Agenten stehen für den Tool-Call ausschließlich drei Werkzeuge zur Verfügung. `list_files` listet die Dateien des eigenen Informationsraums auf, `read_file` liest eine konkrete Datei vollständig aus dem geladenen Informationsraum und `search_files` sucht nach einem konkreten Textbegriff innerhalb der verfügbaren Daten. Zusätzlich enthalten Suchergebnisse durch `search_files` Dateipfade, Zeilennummer und die gefundene Zeile. Trotzdem wird in den Agenteninstruktionen deutlich festgelegt, dass ein Suchtreffer nicht das Lesen einer Datei ersetzt. Zu Beginn eines jeden Agentenlaufs sind bestimmte Quellen verpflichtend zu lesen und solange diese nicht vollständig mit `read_file` gelesen wurden, blockiert die Pipeline andere Tool-Calls. Dadurch beginnt die Analyse nicht mit einer vollständig freien Werkzeugwahl. Erst nach erfolgreichem Lesen aller Pflichtquellen wird die weitere Exploration des Informationsraums freigegeben. Die Initialquellen wurden als deterministischer Einstieg in die Spezialanalysen festgelegt, damit jeder Lauf mit denselben grundlegenden Beobachtungen beginnt. Gleichzeitig bleibt die anschließende Exploration offen, sodass die vorgegebenen Startquellen den weiteren Untersuchungsverlauf nicht auf diese Dateien beschränken. Die Initialquellen unterscheiden sich je nach Agent, wobei die genauen Quellen in der Tabelle 3.5 dargestellt sind.

Tabelle 3.5: Verpflichtende Initialquellen der Spezialagenten

Agent	Verpflichtende Initialquellen
Kubernetes-Agent	`kubernetes/pod-order-service-describe.txt`
	`http/order-service-ready.txt`
	`kubernetes/endpointslices.txt`
	`kubernetes/pod-api-service-describe.txt`
Log-Agent	`logs/order-service-current.txt`
	`logs/postgres-current.txt`
	`logs/api-service-current.txt`
Runbook-Agent	Alle im Informationsraum verfügbaren Runbook-Dateien

Nach den Pflichtquellen unterscheiden sich die Abschlussanforderungen der Spezialagenten.

So muss der Kubernetes-Agent nach dem Bootstrap anschließend `list_files` ausführen und mindestens eine zuvor noch nicht gelesene zusätzliche Quelle untersuchen. Dies soll den Agenten dazu ermutigen, einen beobachteten Auffälligkeitspfad weiter zu verfolgen und ihn mit weiteren Zuständen oder Signalen abzugleichen. Der Log-Agent muss nach dem Bootstrap `list_files` ausführen und alle Dateien mindestens einmal lesen. Dies verhindert eine vorzeitige Finalisierung und dadurch das Übersehen von wichtigen Events. Der Runbook-Agent besitzt alle verfügbaren Daten bereits als Initialquelle und benötigt deswegen keine zusätzliche Explorationsphase. Nachdem die verpflichtenden Einstiege abgeschlossen sind, entscheidet das Modell selbst, welche weiteren zulässigen Daten oder Suchbegriffe für die Analyse relevant sind. Hierbei können mehrere Tool-Calls innerhalb eines Modellschritts angefordert werden, welche dann in der vom Modell vorgegebenen Reihenfolge ausgeführt werden. Wiederholte oder inhaltlich gleichwertige Tool-Aufrufe ohne erwartbaren Informationsgewinn werden bereits in den Agenteninstruktionen untersagt. Diese Regel wurde eingeführt, nachdem sich Agenten in einzelnen Läufen wiederholt in Schleifen aus vergleichbaren Tool-Aufrufen verfangen hatten. Dadurch soll verhindert werden, dass eine Session ohne zusätzlichen Erkenntnisgewinn fortgesetzt wird. Wird am Ende `final` angefordert und es kommt zu einer abschließenden Textanalyse, wird diese nicht unmittelbar akzeptiert, sondern zunächst von einem formalen Evidence Gate zurückgehalten. Das Gate prüft unter anderem, ob alle verpflichtenden Dateien gelesen wurden, ob die für den jeweiligen Agenten festgelegten zusätzlichen Prüfschritte durchgeführt wurden und ob die in der finalen Analyse genannten Dateien zuvor mit `read_file` gelesen wurden. Hierbei wird nicht die fachliche Richtigkeit geprüft, sondern lediglich, ob die zuvor festgelegten technischen Anforderungen zur Quellenprüfung erfüllt wurden. Wird eine finale Analyse erzeugt, obwohl noch formale Abschlussbedingungen offen sind, wird sie nicht als Ergebnis akzeptiert. Die Pipeline gibt dem Agenten daraufhin eine rein formale Kontrollmeldung zurück, die auf die noch nicht erfüllte Abschlussbedingung verweist und gegebenenfalls die betroffenen Dateipfade nennt. Diese Rückmeldung enthält weder neue Domäneninformationen noch eine fachliche Bewertung oder konkrete Diagnoseanweisung. Anschließend kann der Agent seine Analyse innerhalb derselben Session fortsetzen. Erst wenn das Evidence Gate vollständig erfüllt ist, wird seine Textanalyse als Ergebnis des Spezialagenten übernommen. Die Agentenschleife ist zusätzlich auf maximal 32 Modellschritte begrenzt, um ausufernde oder nicht abschließende Agentenläufe zu verhindern. Ein Erreichen des Limits erzwingt keine künstliche Antwort, sondern wird mit einem Fehler beendet. Nach erfolgreichem Abschluss liegen für jeden Spezialagenten seine textuelle Analyse sowie die Menge der tatsächlich erfolgreich gelesenen Dateien vor. Die Spezialanalysen bilden anschließend die Eingabe für den RCA-Agenten. Die gelesenen Dateipfade dienen zusätzlich als Grundlage dafür, die zulässigen Evidenzverweise der finalen RCA auf tatsächlich untersuchte Quellen zu beschränken.

Nachdem die Einzelanalysen der Spezialisten durch `run_rca.py` gesammelt wurden, wird daraus ein neuer Eingabetext für den RCA-Agenten aufgebaut. Dieser enthält die Kubernetes-Analyse, die Log-Analyse, die Runbook-Analyse und das für diesen Lauf

erzeugte JSON-Schema. Der RCA-Agent besitzt keine Tools und erhält dabei nicht erneut die ursprünglichen Primärdateien. Seine Aufgabe ist ausschließlich die Synthese der bereits erzeugten Spezialanalysen. Durch die Systeminstruktionen soll der RCA-Agent insbesondere die Soll-/Baseline-Zustände von beobachteten Zuständen unterscheiden, die Konfigurationsabweichungen mit Laufzeitverhalten und Folgewirkung verbinden, widersprüchliche oder gesunde Beobachtungen zur Eingrenzung der Diagnose berücksichtigen und plausible Alternativen prüfen. Außerdem soll er die Ursache, die technische Wirkung und die nachgelagerten Symptome auseinanderhalten. Damit soll verhindert werden, dass ein sichtbares Symptom vorschnell als Root Cause übernommen wird. Vor dem Aufruf des RCA-Agenten werden alle Dateipfade, die im Vorhinein mit `read_file` gelesen wurden, gesammelt und das JSON-Schema für den jeweiligen Lauf auf diese zulässigen Evidenzpfade eingeschränkt. Ausschließlich diese Quellen sind dann als Evidenznachweis zulässig. Anders als bei den Spezialagenten liefert der RCA-Agent keinen freien Markdown-Text, sondern die Ausgabe muss ein festes JSON-Schema einhalten. Die zentralen Felder des Schemas und ihre Inhalte sind in Tabelle 3.6 dargestellt.

Nach der Antwort des Modells wird das erzeugte JSON nicht direkt als finales Ergebnis übernommen, sondern vorher geprüft, ob es der vorgegebenen Struktur folgt. Zusätzlich wird geprüft, ob die vom RCA-Agenten gewählten Evidenzen auch tatsächlich zur Menge der zuvor gelesenen Quellen gehören. Die Prüfung bewertet nicht, ob die Diagnose korrekt ist. Es werden lediglich Schema und Herkunft der angegebenen Evidenzen geprüft. Nach der erfolgreichen Validierung wird das strukturierte RCA-Ergebnis als `rca-summary.json` gespeichert und daraus deterministisch der Markdown-Bericht `rca-output.md` erzeugt. Damit wird die Struktur nicht noch einmal von dem LLM formuliert, sondern aus dem validierten JSON erzeugt. Der Bericht enthält festgelegte Abschnitte für Root Cause, betroffenen Service, betroffenes Kubernetes-Objekt, Evidenz, Kausalkette, Alternativerklärungen, Unsicherheit, Handlungsempfehlung und Annahmen. Zusätzlich werden Laufmetadaten und die Traces der Spezialagenten gespeichert.

Tabelle 3.6: Zentrale Felder des RCA-Ausgabeschemas

Feld	Inhalt
`root_cause`	Spezifischste durch die verfügbare Evidenz gestützte Root Cause. Ist keine eindeutige Ursache bestimmbar, muss dies ausdrücklich angegeben werden.
`affected_service`	Betroffener beziehungsweise verantwortlicher Service, sofern dieser eindeutig bestimmt werden kann.
`affected_kubernetes_object`	Kubernetes- oder Konfigurationsobjekt, das die diagnostizierte ursächliche Abweichung enthält oder trägt.
`evidence`	Konkrete Evidenz mit dem exakten Dateipfad sowie getrennter Angabe von Beobachtung und Interpretation.
`causal_chain`	Geordnete Ursache-Wirkungs-Kette von der Root Cause über unmittelbare Symptome bis zu nachgelagerten Auswirkungen.
`alternative_explanations`	Plausible Alternativerklärungen und deren evidenzbasierte Bewertung.
`uncertainty`	Konfidenzeinstufung (`low`, `medium` oder `high`) sowie konkrete Einschränkungen der Diagnose.
`recommended_action`	Konkrete Handlungsempfehlung zur Behebung der diagnostizierten Root Cause.
`assumptions`	Explizite Annahmen, die über die direkt gestützte Evidenz hinausgehen.

3.5 Technische Validierung und Reproduzierbarkeit

Wie in den vorherigen Kapiteln beschrieben, besteht das Untersuchungssystem aus der Kubernetes-Testumgebung und der RCA-Pipeline. Beide müssen vor der Datenerhebung und der Versuchsdurchführung auf ihre technische Korrektheit geprüft werden. Zum einen muss die Kubernetes-Testumgebung aus einer funktionsfähigen Baseline reproduzierbar in die vorgesehenen Incidentzustände überführt werden können. Zum anderen muss die anschließend verwendete RCA-Pipeline die vorbereiteten Eingaben technisch korrekt verarbeiten und die vorgesehenen Agenten- und Ausgabeprozesse vollständig durchlaufen.

Die Baseline ist der Ausgangspunkt aller vier Incidents und muss dementsprechend auf ihre Funktionsfähigkeit im fehlerfreien Zustand geprüft werden. Nach dem erfolgreichen Rollout wurden dazu die relevanten fachlichen Abhängigkeiten der Microservice-Kette geprüft. Hierzu gehörten laufende und im Baselinezustand bereite Pods, gültige Service-Backend-Zuordnungen, der Datenbankzugriff des `order-service`, die Kommunikation von `api-service` zu `order-service`, ein vollständiger Checkout über `frontend`, `api-service`, `order-service` und PostgreSQL. Zusätzlich wurden Health-, Readiness- und fachliche Endpunkte der Services zur Validierung verwendet. Nachdem die Korrektheit der Baseline bestätigt wurde, wurden die Incidents erzeugt. Nach der Injektion wurde geprüft, ob der jeweilige erwartete Fehlerzustand tatsächlich vorlag. Dazu wurde der Zustand je nach Incident anhand der relevanten Kubernetes-Zustände und -Konfigurationen, Logs sowie HTTP-Beobachtungen geprüft. Erst wenn der Zustand bestätigt wurde, wurde die Datenerhebung eingeleitet und der jeweilige Snapshot erstellt. Durch diese Prüfungen wurde verhindert, dass ein technisch falsch aufgebauter Baselinezustand oder eine nicht wirksam gewordene Injektion unbemerkt in die Datenerhebung übernommen werden.

Nachdem die korrekte Herstellung der Baseline- und Incidentzustände geprüft wurde, wurde die nachgelagerte RCA-Pipeline schrittweise validiert. Dabei wurde zunächst die gemeinsame technische Mechanik des Tool-Agent-Loops isoliert geprüft, bevor das Zusammenspiel der finalen Pipelinekomponenten und anschließend vollständige RCA-Läufe mit realen Experimentinputs betrachtet wurden. Für die Prüfung des generischen Tool-Agent-Loops wurde das separate Skript `scripts/smoke_test_tool_agent.py` verwendet. Ziel war es, das Zusammenspiel aus LLM, Tool-Ausführung, Rückgabe der Observation und anschließender Modellfortsetzung zu überprüfen. Dazu wurde über `SourceTools` ein kontrollierter In-Memory-Informationsraum mit drei synthetischen Quellen erzeugt, welche einen einfach erkennbaren Readiness-Fehler zeigen. Der Informationsraum war auf die Quellgruppen `kubernetes` und `http` begrenzt, wodurch zugleich die technische Grundlage der Informationstrennung geprüft wurde. Der Agent erhielt dieselben grundlegenden Source-Tools wie die späteren Spezialagenten. Als erste Aktion musste zwingend `read_file("kubernetes/events.txt")` ausgeführt werden, um zu prüfen, ob vorgegebene Tool-Aufrufe durch die Orchestrierung erzwungen werden können. Im anschließenden Agentenablauf erzeugte das Modell einen Tool-Call, Python führte diesen lokal aus und gab das Ergebnis an dieselbe Modellkonversation zurück. Daraufhin konnte das Modell

weitere Tool-Calls durchführen und anschließend in eine finale Analyse wechseln. Damit wurde ein vollständiger Multi-Turn-Zyklus geprüft. Nach Abschluss des Laufs wurden die Trace-Daten automatisch ausgewertet. Der Smoke-Test galt nur dann als bestanden, wenn mindestens ein Tool-Call ausgeführt wurde, der erste Tool-Call dem vorgeschriebenen `read_file`-Aufruf entsprach und keine Tool-Ausführung einen Fehler zurücklieferte. Zusätzlich wurden Modellschritte, Tool-Calls, finale Analyse und vollständiger Trace ausgegeben. Nachdem die grundlegende Agentenmechanik getestet wurde, wurde ein End-To-End-Smoke-Test zur Agenten- und Pipelinevalidierung durchgeführt. Ziel hierbei war es, die vorgesehenen Spezialagenten mit tatsächlichen Eingabebündeln zu prüfen, die bedingungsabhängige Zusammensetzung der Pipeline zu testen und die Abläufe der einzelnen Agenten zu validieren. Das Skript `scripts/e2e_test_rca.py` wird mit einem konkreten Incident und einer Informationsbedingung ausgeführt. Bei der Ausführung verwendet es die bereits abgeleiteten Eingabebündel und prüft vor Testbeginn, ob die entsprechenden Eingabebündel und deren Auditdateien vorhanden sind. Zusätzlich werden Ausgaben und fehlerhafte Läufe in einem separaten Validierungsverzeichnis abgelegt. Dadurch können Validierungsläufe unabhängig von den eigentlichen Versuchsläufen durchgeführt werden, ohne deren Datenbestand zu verändern. Innerhalb dieser Smoke-Tests wurden die einzelnen Spezialagenten anhand ihrer Ausgaben und Traces geprüft. Dabei wurde darauf geachtet, dass der Agent seine vorgegebenen Quellen liest, die initialen Reads funktionieren, die zusätzlichen Quellen entsprechend der Agentenregeln untersucht werden, nur der vorgegebene Informationsraum genutzt wird, der Agent regulär abschließen kann und das Evidence-Gate sowie die Tool-Schleife mit realen Inputs funktioniert. Neben der Agentenvalidierung diente das Skript auch dazu, die Pipelinezusammensetzung zu prüfen, indem unterschiedliche Bedingungen ausgewählt und getestet wurden. Auf Grundlage der Spezialagenten-Ausgaben und -Traces, der Tool-Aufrufe, der verwendeten Evidenzpfade und der Pipelinezusammensetzung konnten Probleme in der Agenten- und Evidenzlogik erkannt und vor dem Freeze angepasst werden. Damit diente der Smoke-Test nicht nur als Pass/Fail-Test, sondern auch der schrittweisen technischen Weiterentwicklung und Absicherung der Agentenarchitektur und Pipeline. Nach den iterativen E2E-Smoke-Tests wurden vollständige Validierungsläufe durchgeführt. Diese Läufe dienten als Kontrolle des Gesamtverhaltens und wurden als gesamte Läufe dokumentiert. Hierbei wurden alle Incidents unter der Bedingung B1 getestet, da in dieser Bedingung alle im Versuchsdesign vorgesehenen Informationsquellen verfügbar waren. Nach den Läufen wurden insbesondere die erzeugten Ausgaben betrachtet und bei Bedarf die Traces und Laufmetadaten zur Analyse herangezogen. Es wurde geprüft, ob die Agenten die tatsächlich vorhandenen Evidenzen nutzen, die Spezialanalysen sinnvoll an die RCA-Stufe übergeben werden, die finale RCA mit den Spezialanalysen konsistent ist, keine nicht gelesenen Daten als Evidenzen benutzt werden, erwartete Agenten- und RCA-Ausgaben vollständig erzeugt werden und die Traces für die spätere Nachvollziehbarkeit erhalten bleiben. Mit den vollständige Validierungsläufen wurde abschließend geprüft, ob die zuvor einzeln und in unterschiedlichen Informationsbedingungen validierten Komponenten als zusammenhängende RCA-Pipeline mit realen Experimentinputs technisch vollständig und nachvollziehbar ausgeführt werden

können.

Nach erfolgreicher technischer Validierung wurde der für die Versuchsdurchführung verwendete Stand festgelegt und der Implementierungsstand eingefroren. Der verwendete Freeze ist `freeze/experiment-2026-09-24` mit dem Commit `bd10aa54320caeeda10cb794e10dc4afbd9a7368`. Nach dem Freeze wurden keine funktionalen Änderungen mehr an der Implementierung vorgenommen. Der Quellcode des Untersuchungssystems sowie die zugehörigen Versuchs- und Ergebnisdaten werden als digitale Beilage auf dem mit der Bachelorarbeit eingereichten USB-Datenträger bereitgestellt. Um die Nachvollziehbarkeit und Reproduzierbarkeit zu gewährleisten, besitzt jedes Experiment-Eingabebündel eine SHA-256-basierte Auditdatei. Vor jedem Lauf wird geprüft, ob die erwarteten Dateien vollständig vorhanden sind, keine zusätzlichen Daten übernommen wurden, die Dateipfade gültig sind und der Hash jeder Datei mit dem festgelegten Auditbestand übereinstimmt. Zusätzlich wird aus der Auditdatei ein Input-Hash abgeleitet und in den Laufmetadaten gespeichert, wodurch sich später nachvollziehen lässt, welcher konkrete Datenbestand für einen Lauf verwendet wurde. Des Weiteren wurden für jeden Lauf die technischen Identitätsdaten erfasst. Diese beinhalten unter anderem den Git-Commit, den Git-Dirty-State, die Python-Version, Ollama-Laufzeitinformationen, das verwendete Model-Digest und die in Tabelle 3.4 aufgeführten Modellparameter. Damit wird dokumentiert, unter welchem Software- und Modellstand die jeweilige RCA erzeugt wurde. Außerdem werden relevante Implementierungsdateien mittels SHA-256 gehasht und im Reproduzierbarkeitsdatensatz festgehalten. Ergänzend werden unter anderem Prompt-Hashes, Agentenregeln, Tooldefinitionen, Initial-Read-Konfiguration, Evidence-Gate-Regeln, Agentenaktivierung und das RCA-Schema dokumentiert. Diese Informationen werden zusammengeführt und daraus der Hash `configuration_sha256` berechnet. Er bildet einen kompakten technischen Fingerabdruck der verwendeten RCA-Konfiguration und ermöglicht den Vergleich, ob zwei Läufe unter demselben technischen Konfigurationsstand ausgeführt wurden. Neben diesen Metadaten werden auch Zwischenartefakte wie Traces und Laufartefakte gespeichert. Diese dokumentieren die Tool-Aufrufe und den Verlauf der Spezialagenten, wodurch später nachvollzogen werden kann, welche Dateien tatsächlich gelesen wurden, welche Tool-Aufrufe stattgefunden haben und ob ein Spezialagent regulär abgeschlossen hat. Sollte es zu einem Fehler in einem Lauf kommen, werden vorhandene Teilartefakte unter `_failures` gespeichert, um nachvollziehen zu können, wo die Probleme aufgetreten sind. Trace-Checkpoints erhalten dabei den letzten bekannten Agentenzustand auch bei technischen Abbrüchen. Dieses Verhalten wurde bereits während der Validierung praktisch genutzt. Die Reproduzierbarkeit bezieht sich in dieser Arbeit auf die kontrollierte Wiederherstellung der technischen Ausgangsbedingungen und nicht auf identische Modellantworten. Innerhalb der drei Wiederholungen eines Testfalls werden deshalb dieselben Eingabedaten, Prompts und Modellparameter verwendet. Die beschriebenen Maßnahmen ermöglichen es, Eingaben, Softwarestand und Laufkonfiguration eines RCA-Laufs nachzuvollziehen und dessen technische Ausgangsbedingungen wiederherzustellen. Eine byteidentische Reproduktion der LLM-Ausgabe wird nicht vorausgesetzt.

4 Methodik und Versuchsdesign

Nachdem in Kapitel 3 die technische Untersuchungsumgebung, die Datenerhebung und Aufbereitung der Agenteneingaben sowie das MAS beschrieben wurden, behandelt dieses Kapitel die methodische Nutzung dieses Systems. Ziel ist es, zu untersuchen, welchen Einfluss die unterschiedliche Verfügbarkeit von Anwendungslogs und Runbook-/Domänenwissen auf die RCA-Qualität hat. Diese beiden Informationsarten bilden die beiden Faktoren des Versuchsdesigns und werden in vier Informationsbedingungen B1–B4 kombiniert, während die Kubernetes-/HTTP-Basisdaten konstant bleiben. Die Informationsbedingungen werden innerhalb von vier vorab definierten Incident-Szenarien untersucht. Die daraus entstehenden RCA-Ausgaben werden anhand einer vorab festgelegten Ground Truth und eines einheitlichen Bewertungsverfahrens beurteilt. Das Kapitel erläutert zunächst das Forschungsdesign und die Operationalisierung der untersuchten Informationsverfügbarkeit und RCA-Qualität. Anschließend werden die Incident-Szenarien und ihre unterschiedlichen Evidenzprofile beschrieben. Darauf folgen die Testfallbildung und die kontrollierte Durchführung der RCA-Läufe mit dem MAS. Danach werden die vorab festgelegte Ground Truth und das Bewertungsverfahren erläutert. Abschließend wird beschrieben, wie die Einzelbewertungen aggregiert und die Informationsbedingungen miteinander verglichen werden.

4.1 Forschungsdesign und Operationalisierung

Für die Untersuchung werden Informationsverfügbarkeit und RCA-Qualität als zentrale Begriffe der Forschungsfrage operationalisiert. Die Informationsverfügbarkeit wird operationalisiert, indem die beiden festgelegten Informationsarten Anwendungslogs und Runbook-/Domänenwissen gezielt variiert werden. Diese beiden Informationsarten werden deshalb als getrennte experimentelle Faktoren behandelt und sind jeweils entweder verfügbar oder nicht verfügbar. Nicht verfügbar bedeutet hierbei, dass die jeweilige Informationsquelle dem RCA-System für den aktuellen Versuchslauf vollständig vorenthalten wird. Die technische Erzeugung und Prüfung der daraus resultierenden Eingabebündel sind in Kapitel 3 beschrieben. Gleichzeitig bleibt eine gemeinsame Informationsbasis aus Kubernetes-Zuständen und -Konfigurationen, Events sowie HTTP-Probes in allen Bedingungen erhalten. Durch das Konstanthalten dieser Basisdaten wird die experimentelle Variation auf die Verfügbarkeit der beiden untersuchten Informationsarten begrenzt. Auch die lokale Ausführung des LLM-basierten Multi-Agenten-Systems mit Ollama und Qwen3:8b stellt eine über alle 48

RCA-Durchläufe konstant gehaltene technische Rahmenbedingung dar. Sie wird nicht als eigenständiger experimenteller Faktor untersucht.

Aus den beiden Faktoren ergibt sich das in Tabelle 4.1 dargestellte 2×2-Design mit den vier Informationsbedingungen B1–B4.

Tabelle 4.1: Faktorbelegung der Informationsbedingungen B1–B4

Bedingung	Anwendungs-logs	Runbook-/ Domänenwissen	Funktion
B1	vorhanden	vorhanden	Referenzbedingung mit vollständiger Verfügbarkeit der beiden variierten Informationsarten
B2	nicht vorhanden	vorhanden	Bedingung ohne Anwendungslogs
B3	vorhanden	nicht vorhanden	Bedingung ohne Runbook-/Domänenwissen
B4	nicht vorhanden	nicht vorhanden	Kombinierte Einschränkung beider variierten Informationsarten

Die Informationsbedingungen verändern weder den zugrunde liegenden Incidentzustand noch dessen Ground Truth. Experimentell variiert ausschließlich die Verfügbarkeit von Anwendungslogs und Runbook-/Domänenwissen, während die Kubernetes-/HTTP-Basisdaten konstant bleiben.

Die RCA-Qualität wird auf Ebene der einzelnen Ausgabe über fünf voneinander getrennte Bewertungsdimensionen operationalisiert: Root-Cause-Korrektheit, Service-/Objekt-Lokalisierung, Evidenzbezug und Nachvollziehbarkeit, Umgang mit unvollständigen Informationen sowie Handlungsempfehlung. Die getrennte Betrachtung von Root-Cause-Korrektheit, Lokalisierung und Begründungsqualität orientiert sich an vergleichbaren Evaluationsansätzen für LLM-basierte RCA-Systeme (Cornacchia et al., 2025; Riddell et al., 2026; Roy et al., 2024). Aus diesen direkten Bewertungen werden anschließend definierte Erfolgskennzahlen für den Vergleich der Informationsbedingungen abgeleitet. Halluzinationen werden separat erfasst, da unbelegte oder den verfügbaren Eingangsdaten widersprechende Aussagen eine eigene Eigenschaft der RCA-Ausgabe darstellen und nicht mit der inhaltlichen Korrektheit gleichgesetzt werden (Ji et al., 2023; Roy et al., 2024). Die konkreten Bewertungs- und Ableitungsregeln werden in Abschnitt 4.4 erläutert.

Der Einfluss der Informationsverfügbarkeit wird innerhalb der vier definierten Incident-Szenarien untersucht. Die Incidents stellen dabei keinen zusätzlichen experimentellen Informationsfaktor dar, sondern bilden unterschiedliche Fehler- und Evidenzprofile ab. Jede Kombination aus Incident und Informationsbedingung bildet einen Testfall. Wiederholte Agentenausführungen werden auch in vergleichbaren RCA-Evaluationen eingesetzt (Cornacchia et al., 2025). In dieser Arbeit wird jeder Testfall dreimal ausgeführt, um die

Variation der RCA-Ausgaben unter identischen experimentellen Ausgangsbedingungen sichtbar zu machen.

4.2 Incident-Szenarien und Evidenzprofile

Da der Einfluss der Informationsbedingungen nicht nur an einem einzelnen Fehlerfall, sondern an mehreren kontrollierten Incident-Szenarien untersucht werden soll, werden vier unterschiedliche Incidents genutzt, die verschiedene Fehlerklassen innerhalb der Kubernetes-Testumgebung abbilden. Die Fehlerklassen sind Anwendungskonfiguration/Startfehler, Zugangskonfiguration/Authentifizierungsfehler, Kubernetes-Service-Konfiguration/Backend-Zuordnung und Kubernetes-Betriebs-/Readiness-Konfiguration. Die vier Incidents wurden so ausgewählt, dass für ihre Diagnose unterschiedliche Informationsquellen besonders relevant sind. Dadurch kann untersucht werden, ob sich die Verfügbarkeit von Anwendungslogs und Runbook-/Domänenwissen je nach Incident unterschiedlich auf die RCA auswirkt. Die technische Umsetzung und Injektion der vier Incident-Szenarien sind in Abschnitt 3.2 beschrieben. Die technischen Ursachen und die jeweils besonders relevanten Evidenzen sind in Tabelle 4.2 gegenübergestellt.

Tabelle 4.2: Fehlerklassen, Ursachen und relevante Evidenz der Incident-Szenarien I1–I4

Incident	Fehlerklasse	Technische Ursache	Evidenzprofil
I1	Anwendungs-konfiguration / Startfehler	Fehlende Pflichtvariable `DB_HOST` im `order-service`	Anwendungslogs, Deployment-Konfiguration und Runbook-/Domänenwissen
I2	Zugangskonfiguration / Authentifizierungs-fehler	Passwort-Mismatch durch `DB_PASSWORD` im `order-service-db-secret`	Anwendungslogs und HTTP-Probes mit Hinweisen auf den Authentifizierungsfehler
I3	Kubernetes-Service-Konfiguration / Backend-Zuordnung	Selector-Mismatch zwischen `Service/postgres` und dem PostgreSQL-Pod	Service-Selector, Pod-Labels, Endpoints und EndpointSlices
I4	Kubernetes-Betriebs- / Readiness-Konfiguration	Falscher Readiness-Pfad `/invalid-ready` im `api-service`	Probe-Konfiguration, Kubernetes-Events und Pod-Readiness

Die vier Incidents unterscheiden sich damit hinsichtlich der Evidenz, die zur Diagnose ihrer konkreten Ursache besonders relevant ist. Bei I1 und I2 können die variierten Informationsquellen zusätzliche Diagnosehinweise liefern, während I3 und I4 bereits anhand der in

allen Bedingungen verfügbaren Kubernetes-/HTTP-Basisdaten eindeutig diagnostizierbar sind. Diese Einordnung beschreibt ausschließlich die vorab festgelegte Evidenzlage und nimmt keine späteren Qualitätsunterschiede zwischen B1–B4 vorweg. Die Ground Truth eines Incidents bleibt über alle vier Informationsbedingungen hinweg unverändert. Die zentralen incidentspezifischen Ursachen, Evidenzgrenzen und Bewertungsgrenzen sind in Anhang B zusammengefasst.

4.3 Testfallbildung und kontrollierte Durchführung

Aus der Kombination eines Incidents mit einer Informationsbedingung entsteht jeweils ein Testfall. Die vier Incidents und vier Informationsbedingungen ergeben somit insgesamt 16 Testfälle, die jeweils dreimal ausgeführt werden. Diese Wiederholungen sind wichtig, da auch bei identischen Eingangsdaten unterschiedliche RCA-Ausgaben entstehen können, weshalb ein einzelner RCA-Lauf nicht stellvertretend für einen gesamten Testfall betrachtet werden sollte. Die Wiederholungen dienen dazu, sichtbar zu machen, wie stark die resultierenden Diagnosen und Qualitätsbewertungen innerhalb desselben Testfalls variieren. Durch vier Incidents, vier Informationsbedingungen und drei Wiederholungen entstehen damit insgesamt 48 einzeln beobachtbare RCA-Ausgaben, ohne dass die drei Wiederholungen als eigenständige Testfälle behandelt werden.

Die drei Wiederholungen eines Testfalls werden unter denselben experimentellen Ausgangsbedingungen durchgeführt. Die dabei entstehenden RCA-Ausgaben werden zunächst unabhängig voneinander bewertet und nicht zu einer gemeinsamen Diagnose zusammengeführt. Erst nach dieser Einzelbewertung werden die drei Wiederholungen gemeinsam betrachtet, um ihre Stabilität innerhalb des Testfalls einzuordnen. Stabilität und inhaltliche Korrektheit bleiben dabei getrennte Eigenschaften, da auch übereinstimmende Diagnosen fachlich falsch sein können. Die konkrete Stabilitätsbewertung wird in Abschnitt 4.4 beschrieben.

Jeder einzelne der 48 Durchläufe ist ein vollständig neuer RCA-Lauf und verwendet genau das Eingabebündel, das zur jeweiligen Kombination aus Incident und Informationsbedingung gehört. Von den Spezialagenten wird der Kubernetes-Agent in allen vier Bedingungen ausgeführt, da Kubernetes- und HTTP-Daten durchgängig verfügbar sind. Der Log-Agent wird ausschließlich in B1 und B3, der Runbook-Agent ausschließlich in B1 und B2 ausgeführt. Ist eine Quellgruppe nicht verfügbar, wird der zugehörige Spezialagent nicht ausgeführt und anstelle eines Spezialagentenergebnisses wird ein in der Implementierung festgelegter neutraler Hinweistext an die RCA-Synthese übergeben. Der RCA-Agent wird in jedem Durchlauf neu ausgeführt und synthetisiert die im aktuellen Lauf entstandenen Spezialanalysen bzw. neutralen Hinweise. Ergebnisse früherer Wiederholungen oder Testfälle werden nicht in einen neuen Lauf übernommen. Die technische Umsetzung der

Agentenarchitektur, die Tool-Nutzung und das Evidence Gate werden in Abschnitt 3.4 beschrieben.

Auch bei einer Wiederholung eines Testfalls werden die Agenten vollständig neu ausgeführt und keine Zwischenergebnisse oder Gesprächshistorien werden zwischen den Wiederholungen übernommen. Hierbei bleiben allerdings unter anderem Eingabedaten, Prompts/-Systeminstruktionen, Agentenregeln, Tooldefinitionen, Modell, Modellparameter und die Pipelinekonfiguration konstant. Incident-ID, Bedingungs-ID und Run-ID dürfen technisch für die Dateipfade und Metadaten übernommen werden, aber dürfen nicht als diagnostisch verwertbare Information in den Modellinput eingehen. Alle Versuchsläufe verwenden den in Abschnitt 3.5 dokumentierten eingefrorenen Implementierungsstand. Die technische Sicherung dieser konstanten Ausgangsbedingungen durch Hashprüfungen, Provenienz- und Identitätsdaten sowie die zugrunde gelegte Definition von Reproduzierbarkeit sind in Abschnitt 3.5 beschrieben.

4.4 Ground Truth und Bewertungsverfahren

Für die Bewertung der RCA-Ausgaben wird eine vom jeweiligen RCA-Ergebnis unabhängige Referenz benötigt, anhand welcher entschieden werden kann, ob Ursache und Lokalisierung fachlich korrekt sind. Dafür wird für jeden der vier Incidents vor den RCA-Läufen eine Ground Truth festgelegt, welche die kanonische Root Cause, die betroffene oder verantwortliche Komponente bzw. das relevante Konfigurationsobjekt und die korrekte Behebung beschreibt. Die Ground Truth wird aus der kontrollierten Implementierung der Incident-Szenarien abgeleitet. Als fachliche Primärnachweise dienen insbesondere Incident- und Baseline-Konfigurationen, Anwendungscode und Injektionslogik (vgl. Kapitel 3). Die Ground Truth wurde unter fachlicher Anleitung des Verfassers mithilfe von ChatGPT (OpenAI) aus den technischen Primärinformationen ausgearbeitet und strukturiert. ChatGPT (OpenAI) diente dabei als unterstützendes Werkzeug zur Dokumentenerstellung und nicht als eigenständige fachliche Wahrheitsquelle. Die Ground Truth wurde anschließend vor den finalen RCA-Läufen eingefroren und nicht anhand von späteren Modellantworten verändert. Dadurch wird verhindert, dass sich der Bewertungsmaßstab nachträglich an die zu bewertenden Ergebnisse anpasst. Außerdem wird die Ground Truth dem RCA-System nicht als Eingangsinformation zur Verfügung gestellt, sondern ausschließlich für die anschließende Bewertung verwendet. Die wesentlichen Festlegungen der incidentspezifischen Ground Truth werden zur Nachvollziehbarkeit in Anhang B zusammengefasst und im Haupttext referenziert.

Da die Ground Truth eines Incidents für alle vier Informationsbedingungen unverändert bleibt, bleibt auch die fachlich korrekte Ursache eines Incidents unabhängig von B1–B4 gleich. Eine andere Informationsbedingung verändert somit nicht die fachlich korrekte Root Cause, sondern lediglich, wie gut diese anhand der jeweils bereitgestellten Daten

begründet werden kann. Deshalb werden in der Ground Truth neben der Ursache auch das verantwortliche Objekt, korrekte Behebung, erwartbare Symptome, relevante Evidenz und Incident-spezifische diagnostische Grenzen festgehalten. Dadurch lässt sich einordnen, welche Aussagen eindeutig, stark begründbar oder nur eingeschränkt durch die jeweils verfügbaren Daten gestützt werden. Das ist notwendig, weil die Incidents unterschiedlich stark von den variierten Informationsquellen abhängen. Daraus folgt, dass eine RCA-Ausgabe nicht allein danach bewertet werden kann, ob ihre Aussage mit der Ground Truth übereinstimmt. Zusätzlich muss berücksichtigt werden, ob diese Aussage mit den im jeweiligen Lauf verfügbaren Daten begründet werden kann. Eine fachlich richtige Vermutung kann hinsichtlich der Root Cause korrekt sein, ohne durch die verfügbaren Daten ausreichend gestützt zu sein (Riddell et al., 2026; Roy et al., 2024). Umgekehrt darf aber das Fehlen einer Informationsquelle nicht automatisch zu einer schlechteren Bewertung führen, wenn die verbleibenden Daten trotzdem eine belastbare Diagnose ermöglichen. Damit erfüllt die Ground Truth zwei getrennte Funktionen. Zum einen definiert sie die unveränderte Soll-Ursache des Incidents und andererseits beschreibt sie die Evidenzgrenzen, also wie sicher und nachvollziehbar eine RCA unter den jeweiligen Bedingungen begründet werden kann. Diese Trennung bildet die Grundlage für die anschließende Bewertung der einzelnen RCA-Ausgaben, insbesondere für die getrennte Betrachtung von Root-Cause-Korrektheit, Evidenzbezug und Umgang mit unvollständigen Informationen.

Um die RCA-Ausgaben systematisch bewerten zu können, umfasst das Bewertungsverfahren vier Bestandteile: die direkte Bewertung, die abgeleiteten Erfolgskennzahlen, die Halluzinationsbewertung und die Stabilitätsbetrachtung der drei Wiederholungen. Für die direkte Bewertung wird jede RCA-Ausgabe zunächst einzeln manuell bewertet. Die Bewertung erfolgt nach dem Inhalt der RCA-Ausgabe und nicht nach ihrer sprachlichen Ähnlichkeit zur Ground Truth (Roy et al., 2024). Neben der konkreten Ursache werden deshalb auch Lokalisierung, Evidenzbezug, Umgang mit Informationsgrenzen und Handlungsempfehlung getrennt berücksichtigt. Die getrennte Betrachtung von Root-Cause-Korrektheit, Lokalisierung, Begründungsqualität und Handlungsempfehlung orientiert sich an den Evaluationsansätzen von Riddell et al. (2026), Roy et al. (2024) und Cornacchia et al. (2025). Genau diese Trennung beschreibt das Bewertungsdokument im Anhang C.1. Die direkte Bewertung erfolgt anhand der bereits in Abschnitt 4.1 operationalisierten fünf Dimensionen, Root-Cause-Korrektheit, Service-/Objekt-Lokalisierung, Evidenzbezug und Nachvollziehbarkeit, Umgang mit unvollständigen Informationen und Handlungsempfehlung. Jede dieser Dimensionen wird mit 0, 1 oder 2 Punkten bewertet. Die konkrete 0–2-Punkte-Skala und ihre Entscheidungsgrenzen stellen eine für diese Arbeit festgelegte Operationalisierung dar. Die konkreten Bewertungsanker unterscheiden sich je nach Dimension: 0 steht für eine nicht erfüllte oder fehlerhafte Ausprägung, 1 für eine teilweise erfüllte oder unvollständige Ausprägung und 2 für die vollständige Erfüllung des jeweiligen Kriteriums. Die detaillierten Kriterien werden im Anhang C dokumentiert. Die Root-Cause-Korrektheit prüft die Übereinstimmung der genannten konkreten Ursache mit der Ground Truth. Davon getrennt prüft die Service-/Objekt-Lokalisierung, ob auch der

tatsächliche Ursprungsort korrekt bestimmt wurde. Dadurch wird getrennt erfasst, ob die konkrete Ursache fachlich korrekt erkannt und ob ihr verantwortlicher Ursprungsort korrekt lokalisiert wird. Evidenz und Nachvollziehbarkeit bewerten, ob die Diagnose durch konkrete Signale begründet und nachvollziehbar hergeleitet wird. Der Umgang mit unvollständigen Informationen bewertet dagegen, ob Aussagekraft und Grenzen der verfügbaren Daten korrekt eingeschätzt und die Sicherheit der Diagnose entsprechend angepasst werden. Die Handlungsempfehlung bewertet schließlich, ob die vorgeschlagene Maßnahme die korrekte Root Cause adressiert, technisch passend und operativ nachvollziehbar ist. Aus den fünf Einzelwerten ergibt sich durch Addition ein Gesamtwert zwischen 0 und 10 Punkten pro RCA-Ausgabe. Dieser Wert fasst die direkte Qualitätsbewertung zusammen, ist jedoch nicht mit dem Gesamterfolg gleichzusetzen. Die Grenzwerte werden vor der Versuchsauswertung festgelegt und danach nicht an die beobachteten RCA-Ausgaben angepasst. Bei Grenzfällen wird die Bewertungsentscheidung kurz begründet und dieselbe Auslegungsregel anschließend auf vergleichbare RCA-Ausgaben angewendet.

Für den späteren Vergleich der Informationsbedingungen werden aus den Punkten der direkten Bewertung zusätzlich binäre Erfolgskennzahlen abgeleitet. Diese Kennzahlen werden nicht noch einmal unabhängig beurteilt, sondern ausschließlich aus den bereits entstandenen Einzelwerten der direkten Bewertung berechnet. Die Kennzahl *Root Cause korrekt* erhält den Wert 1, wenn die Dimension Root-Cause-Korrektheit mit 2 Punkten bewertet wurde. Damit zählt nur eine vollständig und konkret identifizierte kanonische Ursache als korrekter Root-Cause-Treffer. Ebenso erhält die Kennzahl *Lokalisierung korrekt* den Wert 1, wenn die Service-/Objekt-Lokalisierung mit 2 Punkten bewertet wurde. Dadurch wird erfasst, ob sowohl der verantwortliche Service als auch das relevante Objekt vollständig lokalisiert wurden. Aus diesen beiden Kriterien wird der Gesamterfolg abgeleitet. Dieser liegt nur dann vor, wenn sowohl Root Cause als auch Ursprungsort vollständig korrekt bestimmt wurden. Zusätzlich wird ein etwas strengerer evidenzgestützter Erfolg definiert. Hierfür müssen neben dem Gesamterfolg Evidenzbezug und Nachvollziehbarkeit mit 2 Punkten bewertet worden sein und es darf keine Halluzination vorliegen. Damit wird zwischen einer inhaltlich richtigen Diagnose und einer zusätzlich durch die verfügbaren Eingangsdaten getragenen Diagnose unterschieden (Riddell et al., 2026). Diese Trennung ist insbesondere für die Forschungsfrage relevant, da eine RCA-Ausgabe fachlich korrekt sein kann, obwohl ihre Aussage durch die unter der jeweiligen Informationsbedingung verfügbaren Daten nur unzureichend getragen wird. Der evidenzgestützte Erfolg macht diesen Unterschied im Vergleich von B1–B4 sichtbar. Die abgeleiteten Kennzahlen reduzieren die direkte Bewertung also nicht auf einen einzelnen Qualitätsbegriff, sondern ergänzen sie um klar definierte Erfolgskriterien. Der Gesamtwert bleibt als differenzierte Qualitätsbewertung erhalten, die binären Kennzahlen bilden hingegen klar definierte Erfolgskriterien hinsichtlich Korrektheit, Lokalisierung und Evidenzstützung ab.

Da Halluzinationen die Qualität einer RCA-Ausgabe beeinflussen können, werden sie zusätzlich erfasst, jedoch getrennt von der direkten Bewertung festgehalten, damit dieselbe Schwäche nicht mehrfach negativ in die Bewertung einfließt (Cornacchia et al., 2025;

Riddell et al., 2026; Roy et al., 2024). Die Halluzinationsbewertung dient dazu, Art und Schwere unbelegter oder widersprechender Aussagen sichtbar zu machen, ohne den direkten 0–10-Punkte-Gesamtwert nachträglich zu verändern. Hierfür wird zwischen drei Schweregraden unterschieden: keine, leichte oder schwere Halluzination. Die vollständigen Entscheidungsanker werden im Anhang in Tabelle C.3 dokumentiert. Keine Halluzination liegt vor, wenn Tatsachenbehauptungen durch die bereitgestellten Daten gedeckt oder ausdrücklich als Hypothese beziehungsweise Unsicherheit gekennzeichnet sind. Eine leichte Halluzination bezeichnet einzelne unbelegte oder übermäßig spezifische Annahmen, die die zentrale Root Cause, Lokalisierung oder Handlungsempfehlung nicht wesentlich verändern. Eine schwere Halluzination liegt hingegen insbesondere dann vor, wenn Logs, Events, Objekte oder Systemverhalten erfunden oder den Eingangsdaten widersprechend dargestellt werden, unbelegt sichere Kausalketten behauptet oder nicht existierende Komponenten genannt werden sowie wenn darauf falsche oder potenziell schädliche Handlungsempfehlungen beruhen. Die Einteilung in keine, leichte und schwere Halluzination stellt eine projektspezifische Schweregradskala dar. Unabhängig vom Schweregrad wird zusätzlich der Halluzinationstyp dokumentiert. Die Typisierung orientiert sich an der Unterscheidung intrinsischer und extrinsischer Halluzinationen bei Ji et al. (2023). Dabei wird zwischen Aussagen unterschieden, die den bereitgestellten Daten widersprechen, und solchen, die durch die verfügbaren Daten nicht verifiziert werden können. Diese Unterscheidung ist für das Versuchsdesign besonders relevant, weil der Umfang der verfügbaren Evidenz zwischen B1–B4 gezielt variiert wird. Eine Aussage kann daher fachlich plausibel sein, unter einer bestimmten Informationsbedingung aber trotzdem nicht durch die tatsächlich verfügbaren Eingangsdaten abgesichert sein. Die Halluzinationsbewertung dient damit nicht der allgemeinen Bewertung der Halluzinationsanfälligkeit des MAS, sondern ergänzt die Qualitätsbewertung hinsichtlich unbelegter oder widersprechender Aussagen und des Umgangs mit den Grenzen der jeweils verfügbaren Evidenz.

Nachdem die drei RCA-Ausgaben eines Testfalls jeweils einzeln bewertet wurden, werden sie zusätzlich gemeinsam betrachtet, um ihre Stabilität einzuordnen. Wiederholte Agentenausführungen zur Erfassung variierender Ergebnisse werden auch von Cornacchia et al. (2025) eingesetzt. Die Stabilitätsbewertung bezieht sich immer auf drei Wiederholungen der gleichen Incident-Bedingungs-Kombination und nicht auf einzelne RCA-Ausgaben oder das MAS insgesamt. Für die Einordnung wird die inhaltlich zentrale Root-Cause-Diagnose der drei Wiederholungen miteinander verglichen. Hohe Stabilität liegt vor, wenn alle drei Läufe dieselbe zentrale Root Cause nennen. Mittlere Stabilität liegt vor, wenn genau zwei der drei Läufe in ihrer zentralen Root-Cause-Diagnose übereinstimmen. Niedrige Stabilität liegt vor, wenn keine zwei Läufe dieselbe zentrale Root Cause nennen. Die Stabilitätsstufe bewertet damit ausschließlich die Konsistenz der Diagnose zwischen den Wiederholungen und nicht deren fachliche Korrektheit. Drei übereinstimmende, aber falsche Diagnosen können daher hoch stabil sein, ohne einen erfolgreichen Testfall darzustellen. Auch hier verändert die Stabilitätsbewertung weder die bereits vergebenen 0–10-Punkte-Werte noch die binären Erfolgskennzahlen. Sie ist eine Ergänzung zu den beiden Bewertungen, damit die Ergebnisse

eines Testfalls später bezüglich ihrer Variation zwischen den drei Ausführungen eingeordnet werden können. Zusätzlich werden ohne weitere Gewichtung die Anzahl erfolgreicher Läufe von 0 bis 3, die Spannweite des Gesamtwerts sowie gegebenenfalls Unterschiede in der Halluzinationsschwere zwischen den drei Läufen dokumentiert. Dadurch bleibt sichtbar, worin sich die Wiederholungen unterscheiden. Die vollständigen Entscheidungsregeln für die Stabilitätsstufen werden gemeinsam mit den übrigen Bewertungsmatrizen im Anhang in der Tabelle C.5 dokumentiert.

4.5 Aggregation und Vergleich der Informationsbedingungen

Die Auswertung erfolgt auf mehreren Ebenen. Zunächst wird jede RCA-Ausgabe anhand der in Abschnitt 4.4 definierten Kriterien einzeln bewertet. Für einen Testfall wird anschließend der Mittelwert der drei Wiederholungen gebildet. Auf Ebene einer Informationsbedingung wird der Mittelwert der Gesamtwerte der zwölf zugehörigen RCA-Ausgaben, also vier Incidents mit jeweils drei Wiederholungen, berechnet. Zusätzlich werden die abgeleiteten Erfolgskennzahlen als Raten ausgewiesen. Die Erfolgsrate ergibt sich aus dem Anteil der Ausgaben mit Gesamterfolg an den zwölf Ausgaben einer Bedingung, die evidenzgestützte Erfolgsrate entsprechend aus dem Anteil evidenzgestützt erfolgreicher Ausgaben. Ergänzend werden die Halluzinationsrate sowie die Rate schwerer Halluzinationen je Informationsbedingung bestimmt. Die exakten Berechnungsregeln sind in Tabelle C.7 des Anhangs dokumentiert.

Auf Grundlage dieser aggregierten Qualitäts- und Erfolgskennzahlen werden die Informationsbedingungen anschließend paarweise gegenübergestellt. Da B1–B4 ein 2×2-Design aus der Verfügbarkeit von Anwendungslogs und Runbook-/Domänenwissen bilden, reicht ein alleiniger Gesamtvergleich der vier Bedingungen nicht aus. Deshalb wird der Einfluss der Verfügbarkeit von Anwendungslogs untersucht, indem Bedingungen mit konstantem Runbookzustand verglichen werden. Entsprechend wird der Einfluss der Verfügbarkeit von Runbook-/Domänenwissen bei konstantem Logzustand betrachtet. Die daraus entstehenden Paarvergleiche sind in Tabelle 4.3 dargestellt. Dadurch unterscheidet sich innerhalb jedes Paarvergleichs ausschließlich die Ausprägung einer der beiden variierten Informationsarten, während die jeweils andere konstant bleibt.

Neben den aggregierten Werten der Informationsbedingungen werden die Paarvergleiche zusätzlich incidentspezifisch betrachtet. Dies ist erforderlich, da die vier Incidents unterschiedliche Fehler- und Evidenzprofile aufweisen und Veränderungen der Informationsverfügbarkeit daher unterschiedlich auf die RCA-Ausgaben wirken können. Die incidentspezifische Betrachtung macht sichtbar, ob aggregierte Unterschiede von einzelnen Szenarien

Tabelle 4.3: Paarvergleiche zur getrennten Betrachtung der beiden Informationsfaktoren

Untersuchter Faktor	Konstant gehaltener Faktor	Paarvergleich
Anwendungslogs	Runbook-/Domänenwissen vorhanden	B1 gegenüber B2
Anwendungslogs	Runbook-/Domänenwissen nicht vorhanden	B3 gegenüber B4
Runbook-/Domänenwissen	Anwendungslogs vorhanden	B1 gegenüber B3
Runbook-/Domänenwissen	Anwendungslogs nicht vorhanden	B2 gegenüber B4

geprägt werden oder sich gegenläufige Ergebnisse zwischen den Incidents teilweise ausgleichen. Die Ergebnisse werden dabei mit den in Abschnitt 4.2 beschriebenen Evidenzprofilen in Beziehung gesetzt, ohne daraus bereits kausale Erklärungen abzuleiten.

Die Vergleiche zwischen B1–B4 werden deskriptiv ausgewertet. Ziel hierbei ist es, beobachtete Unterschiede der Qualitäts- und Erfolgskennzahlen innerhalb der vier kontrollierten Incident-Szenarien darzustellen. Unterschiede zwischen den Informationsbedingungen werden deshalb als beobachtete Muster innerhalb dieses Versuchsdesigns beschrieben und nicht als statistisch nachgewiesene Effekte auf Kubernetes-RCA allgemein. Die Stabilitätsangaben werden dabei ergänzend zur Einordnung der Testfallergebnisse verwendet. Hierbei führt eine geringe Stabilität nicht zum Ausschluss, sondern zeigt, dass die zentralen Root-Cause-Diagnosen zwischen den drei Wiederholungen voneinander abweichen. Die Ergebnisse erlauben daher Aussagen darüber, wie sich die RCA-Ergebnisse unter unterschiedlicher Informationsverfügbarkeit innerhalb der vier untersuchten Incident-Szenarien und der kontrollierten Testumgebung unterscheiden, nicht jedoch eine statistische Verallgemeinerung auf andere Kubernetes-Systeme, Incidenttypen oder LLM-basierte RCA-Systeme insgesamt.

5 Ergebnisse

Nach der Versuchsdurchführung liegen insgesamt 48 RCA-Ausgaben vor, die im Folgenden nach Informationsbedingungen, Incident-Szenarien sowie weiteren Qualitätsmerkmalen ausgewertet und gegenübergestellt werden. Die 48 RCA-Ausgaben wurden manuell bewertet und die Bewertungen zur besseren Übersicht in eine vorbereitete Excel-Vorlage eingetragen. Die ausgefüllte Excel-Datei sowie die RCA-Ausgaben sind auf dem beigefügten USB-Stick enthalten. Über alle Ausgaben hinweg ergibt sich ein mittlerer Gesamtwert von 7,58 bei maximal 10 Punkten. Die Root Cause wurde in 75,0 % der Ausgaben vollständig korrekt erkannt, während der Ursprungsort in 62,5 % der Ausgaben korrekt lokalisiert wurde. Ein Gesamterfolg liegt bei 54,2 % der Ausgaben vor, ein evidenzgestützter Erfolg bei 29,2 %. In 43,8 % der Ausgaben wurden Halluzinationen festgestellt. Die folgenden Abschnitte differenzieren diese Gesamtergebnisse nach Informationsbedingungen, Incident-Szenarien, Begründungsqualität, Halluzinationen und Stabilität der wiederholten RCA-Durchläufe.

5.1 Überblick über die Versuchsdaten

Das geplante Versuchsdesign wurde vollständig durchgeführt, sodass alle 48 vorgesehenen RCA-Ausgaben vorliegen. Jede RCA-Ausgabe wurde anhand der fünf definierten Bewertungsdimensionen bewertet. Daraus wurden anschließend die in Kapitel 4 definierten Erfolgs- und Qualitätskennzahlen abgeleitet. Sämtliche Ausgaben konnten in die Auswertung aufgenommen werden. Damit liegen je Informationsbedingung und je Incident-Szenario zwölf RCA-Ausgaben vor. Jede Kombination aus Incident und Informationsbedingung umfasst drei Ausgaben. Die folgende Tabelle 5.1 fasst den Umfang der unterschiedlichen Versuchsmerkmale zusammen.

Tabelle 5.1: Umfang der ausgewerteten Versuchsdaten

Versuchsmerkmal	Umfang
Incident-Szenarien	4
Informationsbedingungen	4
Testfälle	16
Wiederholungen je Testfall	3
RCA-Ausgaben gesamt	48
RCA-Ausgaben je Informationsbedingung	12
RCA-Ausgaben je Incident-Szenario	12

5.2 Ergebnisse nach Informationsbedingungen

Die vier Informationsbedingungen werden entsprechend dem in Kapitel 4 beschriebenen Versuchsdesign auf Grundlage von jeweils zwölf RCA-Ausgaben miteinander verglichen. Die folgende Tabelle 5.2 beschreibt die Gesamtergebnisse der unterschiedlichen Bedingungen über alle Incidents. Hierbei werden sowohl der Gesamtwert als auch die Erfolgsraten dargestellt, welche aber voneinander getrennt betrachtet werden müssen. Der Gesamtwert fasst die fünf Bewertungsdimensionen zu einem Wert zwischen 0 und 10 Punkten zusammen und wird für jede Informationsbedingung über zwölf RCA-Ausgaben gemittelt. Die Erfolgsraten beruhen dagegen auf den zuvor definierten binären Erfolgskriterien. Zusätzlich wird die Halluzinationsrate, also der Anteil mit leichter oder schwerer Halluzination, dargestellt, wobei die schwere Halluzination noch einmal separat aufgeführt wird. Die Prozentwerte beziehen sich jeweils auf zwölf RCA-Ausgaben, die sich aus vier Incidents mit jeweils drei Wiederholungen pro Informationsbedingung ergeben.

Tabelle 5.2: Gesamtergebnisse nach Informationsbedingungen

Kennzahl	B1	B2	B3	B4
Mittlerer Gesamtwert	8,75	6,83	8,08	6,67
Root Cause korrekt	91,7 %	66,7 %	83,3 %	58,3 %
Lokalisierung korrekt	66,7 %	58,3 %	75,0 %	50,0 %
Gesamterfolg	58,3 %	50,0 %	66,7 %	41,7 %
Evidenzgestützter Erfolg	41,7 %	25,0 %	33,3 %	16,7 %
Halluzinationsrate	33,3 %	58,3 %	50,0 %	33,3 %
Schwere Halluzinationsrate	0,0 %	33,3 %	25,0 %	25,0 %

B1 erreicht in dem Versuch mit durchschnittlich 8,75 von 10 Punkten den höchsten mittleren Gesamtwert, während B4 mit 6,67/10 den niedrigsten Wert aufweist. B3 liegt bei 8,08/10 und somit über B2 mit 6,83. Daraus ergibt sich die Reihenfolge B1 > B3 > B2 > B4. Bei der Root-Cause-Korrektheit ergibt sich die gleiche Reihenfolge, wobei B1 die höchste Rate mit 91,7 % besitzt, während B4 bei 58,3 % liegt. Damit wird die Root Cause häufiger

vollständig korrekt erkannt als ein vollständiger Gesamterfolg erreicht wird. Bei der Lokalisierung unterscheidet sich erstmals die Reihenfolge. Hier erreicht B3 mit 75,0 % die höchste Lokalisierungsrate, gefolgt von B1 mit 66,7 %, B2 mit 58,3 % und B4 mit 50,0 %. B1 hat zwar den höchsten Gesamtwert, aber nicht die höchste Lokalisierungsrate. Auch beim Gesamterfolg liegt B3 vor B1, wodurch sich dieselbe Reihenfolge wie bei der Lokalisierung bildet. B3 besitzt mit 66,7 % die höchste Erfolgsrate, während B4 mit 41,7 % den niedrigsten Wert besitzt. Beim evidenzgestützten Erfolg sinken die Werte in allen Bedingungen, weil zusätzlich vollständiger Evidenzbezug und das Fehlen von Halluzinationen verlangt wird. Dadurch erreicht B1 mit 41,7 % den höchsten Wert. Darauf folgen B3 mit 33,3 %, B2 mit 25,0 % und B4 mit 16,7 %. Die höchste Halluzinationsrate weist B2 mit 58,3 % auf, während B1 als einzige Bedingung keine schweren Halluzinationen enthält.

Da der Gesamtwert nicht zeigt, wo die Unterschiede zwischen den Bedingungen entstehen, werden die Mittelwerte der fünf 0–2-Punkte-Dimensionen in Tabelle 5.3 dargestellt. Auch hier basiert jeder Mittelwert auf den zwölf RCA-Ausgaben pro Bedingung. Die hier aufgelisteten Kriterien sind keine neuen Kennzahlen, sondern die Bestandteile des Gesamtwertes.

Tabelle 5.3: Mittlere Bewertung der Qualitätsdimensionen nach Informationsbedingung

Bewertungsdimension	**B1**	**B2**	**B3**	**B4**
Root-Cause-Korrektheit	1,92	1,50	1,83	1,33
Service-/Objekt-Lokalisierung	1,67	1,58	1,75	1,50
Evidenzbezug und Nachvollziehbarkeit	2,00	1,33	1,75	1,25
Umgang mit unvollständigen Informationen	1,75	1,50	1,83	1,67
Handlungsempfehlung	1,42	0,92	0,92	0,92

Der größte Unterschied zwischen den Bedingungen zeigt sich beim Evidenzbezug. Hier erreicht B1 volle Punktzahl, während B4 nur 1,25 Punkte erreicht. Auch die Root-Cause-Werte unterscheiden sich deutlich, wobei auch hier B1 vor B3, B2 und B4 liegt. Die Werte beim Umgang mit unvollständigen Informationen sind bei allen vier Bedingungen relativ hoch und besitzen mit denen der Lokalisierung eine der geringsten Unterschiede innerhalb ihrer Bewertungsdimension. Bei der Handlungsempfehlung besitzen B2, B3 und B4 den gleichen Wert von 0,92, während B1 mit 1,42 abschneidet.

Da die vier Bedingungen ein 2×2-Design beschreiben, werden B1–B4 nicht nur insgesamt verglichen, sondern gezielt gegenübergestellt. Die Ergebnisse der Gegenüberstellung werden in der Abbildung 5.1 dargestellt. Hierbei wird der Vergleich der Logverfügbarkeit bei konstantem Runbookzustand horizontal dargestellt und der Vergleich der Runbookverfügbarkeit bei konstantem Logzustand vertikal dargestellt.

Bei dem Vergleich zwischen B1 und B2, also Logs bei verfügbarem Runbook, schneidet B1 in allen vier dargestellten Kennzahlen besser ab als B2. Dasselbe zeigt sich beim Vergleich

	Logs verfügbar	**Logs nicht verfügbar**
Runbook verfügbar	**B1** Gesamtwert: 8,75 RC korrekt: 91,7 % Gesamterfolg: 58,3 % Evidenzgest. Erfolg: 41,7 %	**B2** Gesamtwert: 6,83 RC korrekt: 66,7 % Gesamterfolg: 50,0 % Evidenzgest. Erfolg: 25,0 %
Runbook nicht verfügbar	**B3** Gesamtwert: 8,08 RC korrekt: 83,3 % Gesamterfolg: 66,7 % Evidenzgest. Erfolg: 33,3 %	**B4** Gesamtwert: 6,67 RC korrekt: 58,3 % Gesamterfolg: 41,7 % Evidenzgest. Erfolg: 16,7 %

Abbildung 5.1: Ergebnisse der Informationsbedingungen im 2×2-Versuchsdesign

von B3 und B4, also bei nicht verfügbarem Runbook. Auffällig ist der Unterschied der Root-Cause-Korrektheit, der in beiden Log-Vergleichen 25,0 Prozentpunkte beträgt. Beim Vergleich von B1 und B3 zeigt sich beim Gesamterfolg eine gegenläufige Reihenfolge zu den übrigen dargestellten Kennzahlen. B3 erreicht mit 66,7 % einen höheren Gesamterfolg als B1 mit 58,3 %. Dieser Unterschied entspricht bei zwölf RCA-Ausgaben einem zusätzlichen erfolgreichen Lauf. Beim Vergleich von B2 und B4 zeigt sich diese Umkehr nicht. Hier liegt B2 mit 50,0 % vor B4 mit 41,7 %, wobei die Differenz ebenfalls einem erfolgreichen RCA-Lauf entspricht.

5.3 Ergebnisse nach Incident-Szenarien

Ergänzend zur Auswertung nach Informationsbedingungen werden die Gesamtergebnisse auch nach Incident-Szenarien betrachtet. Auch hier werden pro Incident zwölf RCA-Ausgaben zusammengeführt und zu den entsprechenden Mittelwerten und Raten verdichtet. Tabelle 5.4 zeigt die aggregierten Ergebnisse der vier Incident-Szenarien.

Die Ergebnisse zeigen, dass I4 mit 8,17/10 den höchsten mittleren Gesamtwert erreicht, während I1 den niedrigsten Wert von 6,50 aufweist. Incident I2 folgt I4 knapp mit 8,08 und I3 liegt bei 7,58/10. Damit ergibt sich eine Reihenfolge mit I4 > I2 > I3 > I1. Bei der Root-Cause-Korrektheit liegt I4 mit 91,7 % an erster und I1 mit 50 % an letzter Stelle,

Tabelle 5.4: Gesamtergebnisse nach Incident-Szenarien

Kennzahl	I1	I2	I3	I4
Mittlerer Gesamtwert	6,50	8,08	7,58	8,17
Root Cause korrekt	50,0 %	75,0 %	83,3 %	91,7 %
Lokalisierung korrekt	41,7 %	91,7 %	25,0 %	91,7 %
Gesamterfolg	33,3 %	66,7 %	25,0 %	91,7 %
Evidenzgestützter Erfolg	25,0 %	25,0 %	25,0 %	41,7 %
Halluzinationsrate	58,3 %	33,3 %	25,0 %	58,3 %
Schwere Halluzinationsrate	33,3 %	16,7 %	8,3 %	25,0 %

während I3 83,3 % und I2 75,0 % aufweisen. Bei der Lokalisierung erreichen I2 und I4 jeweils 91,7 %, gefolgt von I1 mit 41,7 % und I3 mit 25,0 %. Der Gesamterfolg wird von I4 mit 91,7 % am meisten erreicht. Besonders auffällig ist I3: Obwohl die Root Cause in 83,3 % der Ausgaben korrekt erkannt wird, beträgt die Gesamterfolgsrate lediglich 25,0 %. Auch beim strengeren evidenzgestützten Erfolg liegt I4 mit 41,7 % über den anderen drei Incidents, welche alle 25 % erreichen. Auch die Halluzinationsrate ist auffällig. Hier besitzen I1 und I4 mit jeweils 58,3 % die höchste Rate, während die Raten bei I2 bei 33,3 % und I3 bei 25,0 % liegen. Bei schweren Halluzinationen weist I1 mit 33,3 % die höchste Rate auf.

Zur differenzierten Betrachtung zeigt Tabelle 5.5 die Mittelwerte der fünf Bewertungsdimensionen für die einzelnen Incidents. Hierbei werden die jeweiligen Mittelwerte für I1–I4 separat betrachtet. Jeder Mittelwert basiert wieder auf den zwölf RCA-Ausgaben des jeweiligen Incidents.

Tabelle 5.5: Mittlere Bewertung der Qualitätsdimensionen nach Incident-Szenario

Bewertungsdimension	I1	I2	I3	I4
Root-Cause-Korrektheit	1,08	1,75	1,83	1,92
Service-/Objekt-Lokalisierung	1,42	1,92	1,25	1,92
Evidenzbezug und Nachvollziehbarkeit	1,25	1,58	1,83	1,67
Umgang mit unvollständigen Informationen	1,58	1,83	1,50	1,83
Handlungsempfehlung	1,17	1,00	1,17	0,83

Bei der Root-Cause-Korrektheit erreicht I4 mit 1,92 den höchsten Mittelwert, gefolgt von I3 mit 1,83 und I2 mit 1,75. I1 liegt hier mit 1,08 deutlich darunter. Bei der Lokalisierung erreichen I2 und I4 jeweils 1,92, während I3 mit 1,25 den niedrigsten Wert besitzt. Die Auffälligkeit aus den Gesamtergebnissen zeigt sich auch hier. Auch in den Dimensionsmittelwerten zeigt I3 eine hohe Root-Cause-Korrektheit bei gleichzeitig niedriger Bewertung der Service-/Objekt-Lokalisierung. Beim Evidenzbezug erreicht I3 mit 1,83 den höchsten Wert, während I1 mit 1,25 am niedrigsten liegt. Beim Umgang mit unvollständigen Informationen liegen I2 und I4 mit jeweils 1,83 vorne, gefolgt von I1 mit 1,58 und I3 mit 1,50.

Die Handlungsempfehlung fällt insgesamt niedriger aus als die anderen Dimensionen. Hier erreichen I1 und I3 jeweils 1,17, I2 liegt bei 1,00 und I4 mit 0,83 am niedrigsten.

Bei der Betrachtung der Incidents über die vier Informationsbedingungen zeigen sich unterschiedlich starke Abweichungen der mittleren Gesamtwerte. I1 weist dabei die größten Unterschiede auf. Während B1 und B3 jeweils 9,00 Punkte erreichen, liegen B2 bei 3,67 und B4 bei 4,33 Punkten. Bei den anderen Incidents fallen die Unterschiede zwischen den Bedingungen geringer aus. So liegt die größte Differenz bei I2 zwischen B1 mit 8,67 und B4 mit 7,67 Punkten, bei I3 zwischen B1 mit 8,33 und B3 mit 7,00 Punkten und bei I4 zwischen B1 mit 9,00 und B4 mit 7,00 Punkten.

5.4 Begründungsqualität, Unsicherheitskommunikation und Halluzination

Begründungsqualität und Unsicherheitskommunikation werden in dieser Arbeit durch die Dimensionen Evidenzbezug und Nachvollziehbarkeit sowie Umgang mit unvollständigen Informationen operationalisiert. Begründungsqualität steht hierbei für die Qualität des Evidenzbezugs und der Nachvollziehbarkeit einer RCA-Ausgabe. Die Unsicherheitskommunikation beschreibt dagegen den Umgang einer RCA-Ausgabe mit nicht verfügbaren oder nicht direkt überprüfbaren Informationen. Zusätzlich wird das Auftreten von Halluzinationen anhand unbelegter oder den verfügbaren Daten widersprechender Tatsachenbehauptungen bewertet. Sie werden separat erfasst und nicht vom 0–10-Punkte-Gesamtwert abgezogen. Die drei Aspekte werden zunächst auf Ebene der einzelnen RCA-Ausgaben erfasst. Zusammengefasste Mittelwerte und Häufigkeiten beschreiben daher das beobachtete Verhalten des untersuchten RCA-Systems innerhalb des Versuchsaufbaus und keine allgemeine Eigenschaft des zugrunde liegenden Sprachmodells.

Bei der Begründungsqualität erreicht die Dimension Evidenzbezug und Nachvollziehbarkeit über alle 48 RCA-Ausgaben einen Mittelwert von 1,58 von maximal 2 Punkten. Die Bedingungsmittelwerte reichen von 1,25 bei B4 bis 2,00 bei B1. Beim Umgang mit unvollständigen Informationen liegt der Gesamtmittelwert bei 1,69/2 Punkten. Die Bedingungswerte liegen hier zwischen 1,50 bei B2 und 1,83 bei B3. Damit fallen die Unterschiede zwischen den Bedingungen beim Evidenzbezug größer aus als beim Umgang mit unvollständigen Informationen.

Tabelle 5.6 zeigt die Verteilung der Halluzinationsschwere nach Informationsbedingungen. Die Zeile Halluzination gesamt fasst leichte und schwere Halluzinationen zusammen und bildet keine zusätzliche Halluzinationsklasse.

Hierbei besitzt B2 die höchste Halluzinationsrate mit 58,3 %, während B1 und B4 bei 33,3 % liegen. B1 und B4 besitzen zwar die gleiche Rate, aber eine unterschiedliche Verteilung der Schweregrade der Halluzinationen. Bei B3 liegt die Halluzinationsrate mit

Tabelle 5.6: Verteilung der Halluzinationsklassen

Halluzinationsklasse	B1	B2	B3	B4	Gesamt	Anteil gesamt
Keine	8	5	6	8	27	56,3 %
Leicht	4	3	3	1	11	22,9 %
Schwer	0	4	3	3	10	20,8 %
Halluzination gesamt	4	7	6	4	21	43,8 %

sechs betroffenen Ausgaben (50,0 %) zwischen B2 mit sieben Ausgaben (58,3 %) und B1/B4 mit jeweils vier Ausgaben (33,3 %). Die dargestellten Halluzinationen wurden zusätzlich typisiert. Dabei wurde zwischen zwei Halluzinationstypen unterschieden: widersprechend und nicht verifizierbar. Von den 21 Ausgaben mit Halluzination gab es 17 mit dem Typ widersprechend und 4 mit dem Typ nicht verifizierbar. Die Verteilung der 21 RCA-Ausgaben mit Halluzinationen auf die vier Informationsbedingungen und die beiden Halluzinationstypen ist in Tabelle 5.7 dargestellt.

Tabelle 5.7: Verteilung der Halluzinationstypen nach Informationsbedingung

Bedingung	Widersprechend	Nicht verifizierbar
B1	3	1
B2	6	1
B3	4	2
B4	4	0
Gesamt	17	4

Ergänzend werden ausgewählte RCA-Ausgaben betrachtet, um die zuvor quantitativ erfassten Halluzinationskategorien anhand konkreter Beispiele zu verdeutlichen. Die in Tabelle 5.8 dargestellten Beispiele dienen ausschließlich der Veranschaulichung der Bewertungsfälle und stellen keine zusätzliche Auswertung der Incident-Szenarien dar.

Diese Beispiele zeigen, dass Halluzinationen sowohl als zusätzliche unbelegte Annahmen als auch als Widersprüche zu vorhandenen Daten auftreten können. Außerdem zeigt I4/B3/Run2, dass eine korrekte Root Cause und Lokalisierung das Auftreten einer Halluzination nicht ausschließen. Der Fall verdeutlicht damit die getrennte Erfassung von Diagnoseerfolg und Halluzination.

Tabelle 5.8: Beispielhafte Ausprägungen der Halluzinationsbewertung

RCA-Ausgabe	Bewertung	Beobachtete Auffälligkeit
I1/B1/Run3	leicht, nicht verifizierbar	Eine mögliche fehlerhafte Secret-Referenz wird zusätzlich angenommen, obwohl die verfügbaren Daten diese Annahme nicht ausreichend belegen.
I2/B2/Run2	schwer, widersprechend	Ein CrashLoopBackOff-Zustand wird in die Kausalkette aufgenommen, obwohl dieser Zustand in den bereitgestellten Daten nicht vorliegt.
I4/B3/Run2	schwer, widersprechend	Root Cause und Lokalisierung werden korrekt bestimmt, die Handlungsempfehlung ordnet die Readiness-Probe jedoch fälschlich dem order-service zu.

5.5 Stabilität der RCA-Ausgaben

Zum Schluss wird die Stabilität der 16 Testfälle anhand der RCA-Wiederholungen ausgewertet. Die Stabilität wird dabei nicht für einzelne RCA-Ausgaben, sondern anhand der jeweils drei RCA-Durchläufe eines Testfalls bestimmt. Mit vier Incidents und vier Bedingungen ergeben sich somit 16 Testfälle. Tabelle 5.9 zeigt die Verteilung der Stabilitätsstufen nach Informationsbedingung. Die 16 Testfälle werden für die tabellarische Übersicht nach Informationsbedingung zusammengefasst.

Tabelle 5.9: Stabilität der Testfälle nach Informationsbedingung

Bedingung	Hoch	Mittel	Niedrig
B1	4	0	0
B2	3	0	1
B3	4	0	0
B4	3	1	0
Gesamt	14	1	1
Anteil	87,5 %	6,3 %	6,3 %

Die überwiegende Mehrheit der Testfälle, nämlich 87,5 %, war hoch stabil. Das bedeutet, dass in diesen Testfällen alle drei Wiederholungen dieselbe zentrale Root Cause nannten. In B1 und B3 waren jeweils alle vier Testfälle hoch stabil, während in B4 ein Testfall mittlere und in B2 ein Testfall niedrige Stabilität aufwies. Auffällig ist, dass beide nicht hoch stabilen

Testfälle bei der Auswertung von I1 auftreten. Bei I1/B4 nannten zwei Wiederholungen dieselbe zentrale Root Cause, während die dritte davon abwich. Bei I1/B2 wurden dagegen in allen drei Wiederholungen unterschiedliche zentrale Root-Cause-Diagnosen genannt. Die Stabilität beschreibt ausschließlich die Übereinstimmung der zentralen Root-Cause-Diagnosen zwischen den Wiederholungen und ist von deren Korrektheit zu unterscheiden.

6 Diskussion

In Kapitel 5 wurden die Ergebnisse der 48 RCA-Ausgaben deskriptiv dargestellt. Im Folgenden werden die beobachteten Unterschiede innerhalb des untersuchten lokal ausgeführten LLM-basierten Multi-Agenten-Systems interpretiert und in Bezug auf das Versuchsdesign und die Forschungsfrage eingeordnet. Zunächst werden der Einfluss der Informationsverfügbarkeit und die incidentabhängige Diagnostizierbarkeit diskutiert. Anschließend werden die Ergebnisse hinsichtlich der verschiedenen Qualitätsdimensionen und der Verlässlichkeit der RCA-Ausgaben betrachtet, bevor die Forschungsfrage beantwortet, die Befunde in den Forschungsstand eingeordnet und die Limitationen der Untersuchung aufgezeigt werden.

6.1 Zusammenwirken von Informationsverfügbarkeit und Evidenzprofilen

Der Ausgangspunkt der Interpretation ist das in Kapitel 4 dargestellte 2×2-Design, wobei die Paarvergleiche die methodische Grundlage bilden. Trotzdem kann die Bedeutung der Paarvergleiche nur incidentabhängig interpretiert werden. Hierzu müssen das tatsächlich verfügbare Evidenzprofil, die laut Ground Truth erwartete Diagnostizierbarkeit und die tatsächlich beobachtete RCA-Leistung miteinander verglichen werden. Zusätzlich muss unterschieden werden zwischen fehlenden diagnostisch notwendigen Informationen und vorhandenen, aber nicht korrekt verarbeiteten Informationen. Die daraus abgeleiteten Erklärungen bleiben auf die kontrollierten Untersuchungsszenarien beschränkt und können nicht ohne Weiteres auf andere RCA-Systeme übertragen werden. Die Wirkmechanismen können nur innerhalb der kontrollierten Szenarien diskutiert werden.

Für Incident 1 wurde die fehlende Pflichtvariable `DB_HOST` als Root Cause festgelegt. Die Abweichung befindet sich im `Deployment/order-service`. Die Logs benennen diese fehlende Variable direkt, während die übrigen Quellen die Ursache nur indirekter erschließbar machen. Laut Ground Truth sind B1–B3 eindeutig diagnostizierbar, während B4 nur eingeschränkt bis stark begründbar ist. Die Ergebnisse aus Kapitel 5 zeigen hierzu einen deutlichen Unterschied: B1 und B3 identifizieren die Root Cause in allen Durchläufen korrekt, während B2 und B4 keine vollständig korrekte Root-Cause-Diagnose erreichen. Dieser starke Unterschied zwischen den Bedingungen lässt sich zunächst dadurch erklären, dass die Logs eine besonders ursachennahe Evidenz liefern, indem sie die fehlende Variable

unmittelbar benennen. B1 und B3 zeigen, dass die direkte Laufzeitevidenz in allen Durchläufen zur korrekten Bestimmung der Root Cause genutzt wurde. Entscheidend für die Interpretation ist allerdings B2. Das schlechte Ergebnis kann nicht allein mit fehlender Information erklärt werden, denn laut Ground Truth reichen Deployment-Konfigurationen und Runbook für eine eindeutige Diagnose. Stattdessen wurden vorhandene Hinweise teilweise falsch gewichtet und alternative Ursachen konstruiert. Für I1 ergibt sich daraus eine Trennung zwischen Informationsverfügbarkeit und Informationsverarbeitung. B4 wiederum ist anders einzuordnen. Hier fehlt die direkte Bestätigung, dass `DB_HOST` zwingend erforderlich ist. Dass das System hier stärker bei sichtbaren Symptomen wie Crash oder Deployment-Status bleibt, lässt sich zumindest teilweise durch die eingeschränkte Evidenzlage erklären. Trotzdem kann aufgrund der Konfigurationsdaten auch hier nicht von einer völligen Nicht-Diagnostizierbarkeit gesprochen werden. Damit lässt sich für Incident 1 zwar feststellen, dass Logs in diesem Fall einen hohen diagnostischen Wert besitzen, weil sie den Fehlmechanismus konkret benennen, aber ihr Fehlen erklärt nicht vollständig die schlechten Ergebnisse.

Bei Incident 2 ist ein Passwort-Mismatch die Root Cause. Das vom `order-service` verwendete `DB_PASSWORD` stimmt nicht mit dem von PostgreSQL erwarteten Passwort überein. Das Passwort wird in einem separaten Objekt `Secret/order-service-db-secret` gespeichert, während der konkrete Secret-Wert in keiner Bedingung sichtbar ist. Allerdings ist der Authentifizierungsfehler nicht nur in den Logs, sondern auch in den konstant verfügbaren HTTP-Antworten erkennbar. Aufgrund dessen sind B1–B4 laut Ground Truth jeweils stark begründbar. Die zentrale Evidenzgrenze über alle Bedingungen ist, dass der konkrete Passwortwert nicht beobachtbar ist, wodurch besonders unter B4 die tatsächlich veränderte Seite des Credential-Mismatches nicht direkt beweisbar ist. Im Vergleich zu I1 fällt die Root-Cause-Korrektheit über die vier Bedingungen deutlich gleichmäßiger aus und B2 erreicht trotz fehlender Logs sogar drei vollständig korrekte Diagnosen. Die nicht vollständig korrekten Ausgaben bleiben überwiegend im richtigen Credential-/Secret-Bereich, formulieren die Ursache jedoch zu unspezifisch, etwa als „missing or incorrect“. Anders als bei I1 stellt das Entfernen der Logs bei I2 keinen vollständigen Verlust des ursachennahen Fehlersignals dar. Da der PostgreSQL-Authentifizierungsfehler sowohl über HTTP-Probes als auch über Logs erkennbar ist, erklärt das plausibel, warum die Log-Verfügbarkeit bei I2 keinen vergleichbaren Unterschied wie bei I1 erzeugt. Besonders B2 zeigt, dass Logs für die konkrete Diagnose von I2 nicht zwingend erforderlich sind und Runbook, HTTP-Fehler und Kubernetes-/Secret-Kontext für die Herleitung der Root Cause ausreichen. Auch B4 bleibt trotz fehlender Logs und fehlendem Runbook diagnostisch tragfähig. Trotzdem bleibt die Ground-Truth-Grenze wichtig. Ohne Kenntnis des Secret-Werts ist nicht unmittelbar beweisbar, dass das clientseitige `order-service-db-secret` verändert wurde. Die hohe diagnostische Sicherheit muss deshalb von der Sicherheit über den nicht einlesbaren Secret-Inhalt getrennt werden. Bei Incident I2 entspricht die beobachtete Diagnoseleistung weitgehend der anhand der Ground Truth erwarteten Diagnostizierbarkeit und die Variationen der Informationsquellen verändern weniger die grundsätzliche Diagnostizierbarkeit

als vielmehr die Präzision und Evidenzabsicherung der konkreten Root-Cause-Aussagen. I2 zeigt damit, dass das Entfernen einer Informationsquelle nicht zwangsläufig zu einem starken Qualitätsverlust führt, wenn die verbleibenden Quellen den Fehlermechanismus weiterhin ausreichend abbilden.

Incident 3 beschreibt ein Selector-Mismatch bei `Service/postgres`, bei dem der Service-Selector und das Pod-Label nicht übereinstimmen. Hierbei liegt die entscheidende Evidenz vollständig in den konstant verfügbaren Kubernetes-Daten, weshalb B1–B4 alle eindeutig diagnostizierbar sind. Die Logs liefern hierbei nur zusätzliche Folgeinformationen, während das Runbook für die Ursachenbestimmung nicht erforderlich ist. Diese erwartete weitgehende Unabhängigkeit von den variierten Informationsquellen zeigt sich grundsätzlich auch in den Ergebnissen. Die Root Cause wird in allen Bedingungen überwiegend korrekt erkannt und in B4 sogar in allen drei Durchläufen. Gleichzeitig bleibt die präzise Lokalisierung mit nur drei korrekten Zuordnungen über alle zwölf Ausgaben auffällig schwach, wodurch der Gesamterfolg bei lediglich 25 % liegt. Hierbei wurde häufig `Service/postgres` als eigenes Objekt erkannt, aber im Service-Feld der Ausgabe als `order-service` angegeben. Zusätzlich sind die Objektbezeichnungen teilweise unpräzise. Besonders auffällig ist hier B3/Run2. Dort werden Selector und fehlendes Backend erkannt, aber verfügbare Pod-/Deployment-Labels nicht korrekt mit einbezogen oder sogar als nicht verfügbar dargestellt, woraus eine falsche oder unvollständige Diagnose und Handlungsempfehlung folgen. Die überwiegend korrekte Erkennung der Root Cause entspricht der erwarteten Diagnostizierbarkeit und zwischen den Bedingungen mit und ohne Logs bzw. Runbook zeigt sich kein konsistenter Vorteil zusätzlicher Informationsquellen. Deshalb lassen sich die verbleibenden Fehler nicht primär durch eingeschränkte Informationsverfügbarkeit erklären. Besonders B3/Run2 zeigt vielmehr, dass relevante Evidenz zwar vorhanden ist, aber nicht vollständig untersucht bzw. teilweise falsch als fehlend eingeschätzt wurde. Dadurch liegt wieder eine Trennung zwischen Evidenzverfügbarkeit und Evidenzverarbeitung vor. Bei I3 entstehen die beobachteten Fehler daher eher bei der Auswertung, Verknüpfung und präzisen Lokalisierung verfügbarer Evidenz als durch fehlende Informationen. I3 zeigt damit besonders deutlich, dass eine eindeutige Evidenzlage zwar die Diagnose ermöglicht, aber keine korrekte Verarbeitung und präzise Lokalisierung garantiert.

Für Incident 4 wurde ein falscher Readiness-Pfad im `Deployment/api-service` als Root Cause gewählt. Auch hier liegt die entscheidende Evidenz bereits in den konstanten Kubernetes-/HTTP-Daten. Laut Ground Truth sind B1–B4 alle eindeutig diagnostizierbar, während Logs und Runbook die Diagnose zusätzlich bestätigen. B1–B3 nennen in ihren jeweils drei Durchläufen immer die korrekte Root Cause, lediglich B4 hat einen Lauf mit einer nicht vollständig korrekten Root Cause. Insgesamt erreicht I4 eine Gesamterfolgsrate von 91,7 % und besitzt damit den höchsten Wert aller Incidents. Im fehlerhaften zweiten Durchlauf von B4 wurde statt des korrekten Sollpfades des api-service der order-service als mögliches Ziel der Readiness-Probe angenommen. Hier fällt auch auf, dass auch mehrere inhaltlich erfolgreiche Ausgaben fehlerhafte Zusatzannahmen über die Wirkung oder das Ziel der Readiness-Probe enthalten, was sich auf den Evidenzbezug oder die Handlungs-

empfehlung auswirkt. Trotzdem entspricht das tatsächliche Diagnoseverhalten weitgehend der Ground Truth, da die Ursache auch ohne Logs und/oder Runbook fast durchgehend bestimmt wird. Durch die relativ kurze Verbindung von Konfigurationsabweichung zu beobachtetem Verhalten besitzen Logs und Runbook für die eigentliche Root-Cause-Bestimmung bei I4 nur begrenzten zusätzlichen diagnostischen Wert. Auch dass B4 noch 2/3 vollständige Diagnosen erreicht, passt zu dieser Evidenzlage. Der fehlerhafte Lauf 2 lässt sich deshalb nicht primär durch fehlende Evidenz erklären, da die relevante Deployment-Konfiguration vorhanden war, aber nicht vollständig ausgewertet wurde. Stattdessen wurde eine falsche Servicebeziehung konstruiert. Dadurch zeigt sich, dass auch bei I4 verbleibende Fehler eher auf Verarbeitung und korrekte kausale Verknüpfung vorhandener Evidenz zurückgehen als auf fehlende diagnostische Informationen. Dementsprechend decken sich die erwartete Diagnostizierbarkeit und die tatsächliche Diagnoseleistung weitgehend und die verfügbare Kubernetes-/HTTP-Evidenz reicht in fast allen Läufen zur korrekten Ursache und Lokalisierung aus. Gleichzeitig zeigt I4, dass korrekte Root Cause und Lokalisierung nicht automatisch eine vollständig belastbare RCA bedeuten.

In der incidentübergreifenden Betrachtung des 2×2-Designs zeigt sich, dass sich der Einfluss der Informationsverfügbarkeit nicht gleichmäßig über die Szenarien verteilt. In den beiden Log-Vergleichen beträgt der Unterschied der Root-Cause-Korrektheit 25 Prozentpunkte, wobei dieser rechnerisch vollständig von I1 getragen wird. Bei I2–I4 gleichen sich positive und negative Einzelunterschiede insgesamt aus. Deswegen darf der Log-Effekt nicht als gleichmäßiger Vorteil von Logs über alle Incident-Typen interpretiert werden. Der diagnostische Wert von Logs hängt eher davon ab, welche Rolle sie im Evidenzprofil des konkreten Incidents spielen. Bei I1 liefern Logs die direkte ursachennahe Evidenz zur fehlenden Pflichtvariable, wodurch sich eine starke Abhängigkeit von der Informationsbedingung zeigt, gleichzeitig zeigt B2, dass eine grundsätzlich ausreichende Evidenz noch keine korrekte Verarbeitung garantiert. Bei I2 bleibt der Authentifizierungsfehler auch ohne Logs über HTTP sichtbar, wodurch eine geringere Abhängigkeit von der Log-Verfügbarkeit entsteht. Die zusätzlichen Quellen erhöhen eher die Präzision und Absicherung als die grundsätzliche Diagnostizierbarkeit. I3 enthält die entscheidende Ursache vollständig in seinen Kubernetes-Daten, während Logs und Runbook für die Bestimmung der Root Cause großteils redundante Informationen liefern. Die verbleibenden Fehler betreffen vor allem die Verarbeitung und die Lokalisierung. Auch I4 enthält die entscheidende Evidenz direkt in den Kubernetes-Daten, was als Ursache für die gute Qualität gewertet werden kann. Zusätzliche Quellen besitzen für die reine Root-Cause-Bestimmung nur begrenzten Mehrwert. Bei der incidentspezifischen Betrachtung des Runbook-/Domänenwissens ergibt sich ebenfalls kein einheitliches Muster. Das Runbook kann vorhandene Konfigurationen durch einen Sollzustand ergänzen, aber sein diagnostischer Mehrwert ist stark davon abhängig, ob überhaupt eine Soll-Ist-Lücke geschlossen werden muss, die Ursache bereits direkt aus den technischen Basisdaten hervorgeht oder andere Quellen denselben Informationsgehalt bereits bereitstellen. Besonders I1/B2 zeigt, dass das Runbook diagnostisch relevante Informationen enthält, diese aber vom System nicht automatisch erfolgreich genutzt werden.

Daher darf ein geringer beobachteter Runbook-Effekt nicht mit grundsätzlich geringer fachlicher Relevanz des Runbooks gleichgesetzt werden. Die Ergebnisse sprechen gegen die vereinfachende Annahme, dass eine höhere Anzahl verfügbarer Informationsquellen zwangsläufig zu einer besseren RCA führt. Für die Interpretation der beobachteten Unterschiede lassen sich drei Aspekte der Evidenz unterscheiden:

- **Ursachennähe:** Beschreibt, ob eine Informationsquelle den konkreten Fehlmechanismus direkt erkennen lässt oder lediglich Symptome bzw. Folgeerscheinungen liefert.
- **Redundanz:** Beschreibt, ob dieselbe diagnostisch relevante Information bereits durch eine andere verfügbare Quelle bereitgestellt wird.
- **Evidenzverarbeitung:** Beschreibt, ob das RCA-System die vorhandenen Hinweise korrekt auswertet, gewichtet, miteinander verknüpft und zur eigentlichen Root Cause zurückführt.

Durch diese Eigenschaften lassen sich für die beobachteten Fehldiagnosen zwei unterschiedliche Erklärungsebenen unterscheiden:

- **Evidenzbegrenzung:** Entscheidende diagnostische Informationen fehlen tatsächlich, wodurch die konkrete Root Cause nicht eindeutig oder nur eingeschränkt begründbar ist.
- **Evidenzverarbeitungsproblem:** Ausreichende diagnostische Informationen sind vorhanden, werden vom RCA-System jedoch nicht korrekt ausgewertet, gewichtet oder miteinander verknüpft.

Zusammenfassend beeinflusst die Informationsverfügbarkeit die RCA-Qualität im untersuchten System incidentabhängig. Eine zusätzliche Informationsquelle besitzt vor allem dann einen diagnostisch hohen Wert, wenn sie neue und ursachennahe Evidenz liefert. Ist dieser Informationsgehalt bereits in anderen Daten verfügbar, fällt der zusätzliche Nutzen geringer aus. Gleichzeitig zeigen mehrere Testfälle, dass selbst eine eindeutige Evidenzlage keine korrekte Diagnose garantiert, wenn die vorhandenen Hinweise nicht zuverlässig verarbeitet werden.

6.2 Qualität und Verlässlichkeit der RCA-Ausgaben

Die Ergebnisse aus Abschnitt 6.1 zeigen, dass neben der Verfügbarkeit der Informationen auch Probleme der Evidenzverarbeitung zur beobachteten Ausgabequalität beitragen. Hierbei ist entscheidend, ob aus der verfügbaren Evidenz eine präzise, evidenzgestützte, nachvollziehbare und operativ brauchbare Diagnose entsteht. Die Gesamtergebnisse zeigen bereits eine deutliche Abstufung. Die vollständige Root Cause wurde in 75,0 % der Ausgaben

vollständig korrekt benannt, die Lokalisierung in 62,5 %, der Gesamterfolg wurde nur noch bei 54,2 % erreicht und der evidenzgestützte Erfolg nur von 29,2 %. Damit nimmt die Erfolgsquote ab, sobald mehr Qualitätsanforderungen als die reine Ursachenbenennung gestellt werden. Daraus ergeben sich drei übergreifende Probleme, welche eingeordnet werden müssen. Erstens ist der Qualitätsverlust entlang der RCA-Kette zu betrachten, bei dem die korrekte Ursache nicht immer präzise lokalisiert oder in eine vollständige Handlungsempfehlung überführt wird. Des Weiteren sind die Qualität der Begründungen und der Umgang mit Unsicherheit zu betrachten. Auch eine richtige Diagnose kann von falschen oder unzureichend belegten Begründungen begleitet werden. Schließlich müssen Stabilität und Korrektheit getrennt betrachtet werden, da stabile Ausgaben nicht automatisch qualitativ korrekt oder evidenzgestützt sind. Halluzinationen und Stabilität werden damit nicht isoliert betrachtet, sondern in die übergeordnete Beurteilung der Qualität und Verlässlichkeit der RCA-Ausgaben einbezogen. Ziel ist es, herauszuarbeiten, an welchen Stellen zwischen richtiger Ursachenhypothese und belastbarer RCA die Qualität verloren geht und welche dieser Probleme eher auf Evidenzverarbeitung als auf fehlende Information zurückzuführen sind.

Die erste Verluststufe zeigt sich zwischen der korrekten Bestimmung der Root Cause und der präzisen Lokalisierung des verantwortlichen Objekts. Von den 36 RCA-Ausgaben mit vollständig korrekter Root Cause erreichen zehn keine vollständig korrekte Lokalisierung. Dies deutet darauf hin, dass das System den Fehlermechanismus teilweise richtig erkennt, die vollständigen Lokalisierungsanforderungen jedoch nicht durchgängig erfüllt. I3 dient hierfür als bereits in Abschnitt 6.1 diskutierter Repräsentant, bei dem die beobachteten Abweichungen von unpräzisen Objektbezeichnungen bis hin zur Zuordnung eines falschen Objekts reichen. Das spricht dafür, dass die abstrakte Fehlerklasse bzw. der Fehlmechanismus leichter erfasst wird als dessen präzise operative Verankerung. Auch bei der zweiten Verluststufe, der präzisen Diagnose zur operativen Ableitung, lässt sich ein wiederkehrendes Muster erkennen. Die vorgeschlagenen Änderungen adressieren häufig den erkannten Fehlermechanismus, beinhalten aber nicht immer eine Funktionsverifikation oder Prüfung des resultierenden Systemzustands. In manchen Fällen werden sogar falsche Änderungsziele trotz korrekter Diagnose genannt. Der Qualitätsverlust entsteht damit nicht nur bei der Ursachenfindung selbst, sondern vor allem in den nachgelagerten Schritten der präzisen Lokalisierung und operativen Ableitung. Das System erkennt den Fehlmechanismus häufig korrekt, kann ihn aber nicht konsistent einem verantwortlichen Objekt zuordnen und eine überprüfbare und nachvollziehbare Handlungsempfehlung erstellen.

Neben der präzisen Lokalisierung und operativen Ableitung entscheidet auch die Begründung der Diagnose darüber, wie belastbar eine RCA-Ausgabe ist. Eine inhaltlich richtige Root Cause kann dennoch auf unvollständig ausgewerteter Evidenz, unbelegten Schlussfolgerungen oder einer unangemessenen Einschätzung bestehender Unsicherheiten beruhen. Deshalb ist zusätzlich zu betrachten, ob das System zuverlässig zwischen direkt belegten Informationen, plausiblen Ableitungen und tatsächlich verbleibenden Informationslücken unterscheidet. In den 48 RCA-Ausgaben lässt sich erkennen, dass das System grundsätzlich

dazu in der Lage ist, Evidenzgrenzen zu erkennen. So wird bei I2 der nicht sichtbare Secret-Wert mehrfach ausdrücklich als Evidenzgrenze behandelt. Des Weiteren werden auch bei I1/B4 fehlende Logs teilweise korrekt als Grenze der Ursachenbestimmung eingeordnet. Trotzdem kommt es auch zu einer unvollständigen Evidenzverarbeitung oder falsch angenommenen Evidenzlücken. In einzelnen Ausgaben wurden vorhandene Informationen nicht vollständig ausgewertet und anschließend fälschlich als Informationslücke behandelt. Dies zeigt sich beispielsweise bei I2/B2/Run 1 anhand der vorhandenen Authentifizierungsevidenz und bei I4/B4/Run 2 anhand der verfügbaren Deployment-Konfiguration. Das Problem liegt hierbei nicht in fehlender Information, sondern in einer fehlerhaften Einschätzung der vorhandenen Evidenz, wodurch eigentlich tragfähige Diagnosen unnötig unpräzise werden. Neben falsch angenommenen Evidenzlücken zeigt sich auch das umgekehrte Problem, dass aus vorhandenen Beobachtungen Aussagen abgeleitet werden, die durch die Daten nicht ausreichend gestützt sind. Dabei werden korrekte Beobachtungen um zusätzliche Aussagen oder Kausalzusammenhänge ergänzt, die durch die Daten nicht gedeckt sind. Besonders aussagekräftig ist hierbei I2/B2/Run2, bei dem die Root Cause und Lokalisierung korrekt sind, aber trotzdem ein nicht vorhandener CrashLoopBackOff in die Kausalkette eingebunden wird. Dadurch scheitert die Ausgabe trotz erfolgreicher Diagnose am evidenzgestützten Erfolg. Ähnlich ist es bei I4/B4/Run1, bei dem der falsche Readiness-Pfad und `Deployment/api-service` korrekt erkannt werden, gleichzeitig aber die lokale Readiness-Probe fälschlicherweise mit dem `order-service` verknüpft wird. Halluzinationen stellen eine besonders deutliche Ausprägung dieser Überschreitung der Evidenzgrenze dar. Hierbei werden Schlussfolgerungen oder Tatsachenbehauptungen formuliert, die über die Aussagekraft der verfügbaren Evidenz hinausgehen. Zusätzlich treten Halluzinationen auch bei korrekter Root Cause und Lokalisierung auf. Zusammenfassend kann das System in vielen Fällen den diagnostischen Kern korrekt erfassen, ist aber weniger zuverlässig darin, den Geltungsbereich seiner Aussagen an die tatsächliche Evidenz anzupassen. Dabei zeigen sich zwei zentrale Fehlformen: Vorhandene Evidenz wird teilweise nicht vollständig verarbeitet und fälschlich als Informationslücke behandelt, während in anderen Fällen die Aussagekraft der vorhandenen Evidenz überschritten wird. Beide Muster tragen dazu bei, dass der evidenzgestützte Erfolg deutlich unter dem Gesamterfolg liegt.

Um die Stabilität des Systems zu beobachten und zu bewerten, wurde jeder der 16 Testfälle mit den gleichen Eingabedaten und experimentellen Rahmenbedingungen dreimal ausgeführt. Bewertet wird dabei ausschließlich die Übereinstimmung der zentralen Root-Cause-Diagnose zwischen den Wiederholungen. Die Ergebnisse zeigen mit 14 von 16 hoch stabilen Testfällen eine hohe Übereinstimmung der zentralen Root-Cause-Diagnosen zwischen den Wiederholungen. Entscheidend ist hierbei die Trennung zwischen Stabilität und Korrektheit. I3/B3 zeigt deutlich, dass eine hohe Stabilität vorliegt. Alle drei Läufe nennen dieselbe zentrale Root-Cause-Diagnose eines PostgreSQL-Service-Selector-Mismatches, aber keiner der drei Läufe erreicht den Gesamterfolg. Die Gesamtwerte schwanken deutlich zwischen 8, 4 und 9 Punkten. Dies zeigt, dass dieselbe diagnostische Richtung reproduzierbar auftreten kann, während Lokalisierung, Evidenzbezug, Handlungsempfehlung

oder Halluzinationen zwischen den Läufen weiterhin qualitativ problematisch bleiben. Interessant hierbei sind auch die zwei Testfälle ohne hohe Stabilität. Beide treten bei I1 ohne Anwendungslogs auf, wobei I1/B2 eine niedrige und I1/B4 eine mittlere Stabilität erreicht, während I1/B1 und I1/B3 hoch stabil sind. Die Ergebnisse von I1 legen nahe, dass die direkte Logevidenz neben der Root-Cause-Korrektheit auch die Stabilität der zentralen Diagnosen begünstigt haben könnte. Allerdings bezieht sich diese Aussage lediglich auf I1 und ist keine allgemeine Aussage über den Einfluss von Logs auf Stabilität.

Führt man diese drei Teilbereiche der Qualität nun zusammen, lässt sich erkennen, dass das System den diagnostischen Kern häufiger korrekt erfasst als eine vollständig belastbare RCA erzeugt. Die Ergebnisse zeigen, dass sich drei voneinander unterscheidbare Anforderungen ergeben. Zum einen müssen Root Cause und verantwortliches Objekt korrekt bestimmt werden. Des Weiteren müssen Aussagen zur tatsächlichen Evidenz passen, Unsicherheiten korrekt eingeordnet und geeignete Handlungsschritte abgeleitet werden. Zum Schluss müssen wiederholte Ausführungen zu einer konsistenten zentralen diagnostischen Richtung führen. Da diese Anforderungen unabhängig voneinander erfüllt oder verletzt werden können, reicht eine einzelne Kennzahl nicht aus, um die Verlässlichkeit der RCA-Ausgaben vollständig zu beurteilen. Die beobachteten Qualitätsprobleme treten einerseits bei tatsächlich begrenzter Evidenz auf, zeigen sich andererseits aber auch dann, wenn vorhandene Evidenz nicht korrekt ausgewertet, gewichtet oder in Schlussfolgerungen überführt wird. Die Verlässlichkeit des Systems hängt nicht nur davon ab, ob die richtige Ursache erkannt wird, sondern auch davon, wie präzise, evidenztreu, operativ nachvollziehbar und reproduzierbar diese Diagnose ausgearbeitet wird. Die Ergebnisse der beiden vorangegangenen Betrachtungen zeigen damit, dass die Qualität der RCA-Ausgaben sowohl von der incidentabhängigen Verfügbarkeit diagnostisch relevanter Informationen als auch von deren Verarbeitung innerhalb des Systems geprägt wird. Auf dieser Grundlage kann im folgenden Abschnitt die Forschungsfrage zusammenfassend beantwortet werden.

6.3 Beantwortung der Forschungsfrage

Die Forschungsfrage zielt darauf ab, wie sich die Verfügbarkeit von Anwendungslogs und Runbook-/Domänenwissen auf die Qualität der RCA auswirkt. Die bisherigen Ergebnisse zeigen, dass zusätzliche Informationsquellen die RCA-Qualität verbessern können, wobei der Umfang dieses Vorteils vom jeweiligen Incident und dessen Evidenzprofil abhängt. Entscheidend ist vielmehr, welchen diagnostischen Beitrag Anwendungslogs und Runbook-/Domänenwissen im konkreten Evidenzprofil leisten und wie zuverlässig die verfügbaren Informationen vom System verarbeitet werden.

Die Ergebnisse der 48 RCA-Ausgaben zeigen insgesamt Vorteile bei vorhandenen Anwendungslogs. Diese fallen jedoch nicht bei allen Incidents gleichermaßen aus, sondern werden insbesondere durch I1 geprägt. Ihr diagnostischer Nutzen hängt vielmehr davon ab,

welchen zusätzlichen Informationsgehalt sie gegenüber den übrigen verfügbaren Quellen liefern. Besonders relevant sind Logs, wenn sie ursachennahe Evidenz bereitstellen, die nicht bereits redundant über andere Quellen verfügbar ist. Ist der Fehlermechanismus bereits über andere Quellen erkennbar, fällt ihr zusätzlicher Nutzen geringer aus. Zudem garantiert die bloße Verfügbarkeit von Logs keine höhere RCA-Qualität, da die vorhandene Evidenz vom System auch korrekt verarbeitet werden muss.

Auch bei verfügbarem Runbook-/Domänenwissen zeigen sich Vorteile in einzelnen Qualitätskennzahlen, jedoch kein durchgängiger Vorteil über alle Incidents und Bewertungskriterien hinweg. Sein diagnostischer Nutzen hängt davon ab, ob das zusätzliche Wissen einen relevanten Sollzustand oder fachlichen Kontext bereitstellt, der aus den technischen Basisdaten allein nicht unmittelbar hervorgeht. Ist der für die Diagnose notwendige Informationsgehalt bereits in anderen verfügbaren Quellen enthalten, fällt der zusätzliche Nutzen des Runbooks geringer aus. Gleichzeitig führt verfügbares Runbookwissen nicht zwangsläufig zu einer höheren RCA-Qualität, da die enthaltenen Hinweise vom System korrekt erkannt, gewichtet und mit der übrigen Evidenz verknüpft werden müssen.

Die Betrachtung der Anwendungslogs und des Runbook-/Domänenwissens zeigt damit, dass die Informationsverfügbarkeit nur einen Teil der beobachteten RCA-Qualität erklärt. Gleichzeitig zeigen die Ergebnisse aus Abschnitt 6.2, dass eine ausreichende Evidenzlage allein keine vollständig belastbare RCA garantiert. Die Qualität hängt zusätzlich davon ab, ob das System die vorhandenen Informationen korrekt auswertet, miteinander verknüpft und in eine präzise Lokalisierung, evidenztreue Begründung und geeignete Handlungsempfehlung überführt. Auch Halluzinationen und Unterschiede zwischen den wiederholten RCA-Ausgaben verdeutlichen, dass Informationsverfügbarkeit, Korrektheit und Stabilität der RCA nicht gleichgesetzt werden können.

Zusammenfassend lässt sich die Forschungsfrage wie folgt beantworten: Die Verfügbarkeit von Anwendungslogs und Runbook-/Domänenwissen beeinflusst die Qualität der Root Cause Analysis des untersuchten lokal ausgeführten LLM-basierten Multi-Agenten-Systems abhängig vom Evidenzprofil des jeweiligen Incidents. In der Gesamtbetrachtung zeigen sich Vorteile bei höherer Informationsverfügbarkeit, vor allem beim mittleren Gesamtwert, der Root-Cause-Korrektheit und dem evidenzgestützten Erfolg. Beide Informationsarten können einen zusätzlichen diagnostischen Nutzen bieten, wenn sie relevante Informationen bereitstellen, die aus den konstant verfügbaren Kubernetes-/HTTP-Daten nicht ausreichend hervorgehen. Sind die entscheidenden Informationen aber bereits über andere Quellen verfügbar, fällt der zusätzliche Nutzen geringer aus. Die erzielte RCA-Qualität wird jedoch nicht allein durch die Verfügbarkeit der Informationsquellen bestimmt, sondern ebenso durch die Fähigkeit des Systems, die vorhandene Evidenz korrekt zu verarbeiten und in eine belastbare RCA-Ausgabe zu überführen. Insgesamt zeigen die Ergebnisse somit, dass zusätzliche Informationsquellen die RCA-Qualität verbessern können, wobei dieser Vorteil je nach Incident und diagnostischer Relevanz der jeweiligen Quelle unterschiedlich stark ausfällt.

6.4 Einordnung in den Forschungsstand

Die Ergebnisse von Riddell et al. (2026) zeigen, dass verfügbare Evidenz von LLM-basierten RCA-Systemen nicht zwangsläufig korrekt verarbeitet wird. Eine vergleichbare Problematik zeigt sich in dieser Untersuchung bei I3, bei dem die Root Cause häufig erkannt, aber nicht zuverlässig lokalisiert wurde. Auch Cornacchia et al. (2025) verdeutlichen die Grenzen der Zuverlässigkeit agentenbasierter RCA. Die vorliegenden Ergebnisse ergänzen diese Erkenntnisse, indem sie zeigen, dass auch unter kontrollierten Bedingungen Unterschiede zwischen Ursachenbestimmung, Lokalisierung und Evidenzabsicherung bestehen bleiben.

Pei et al. (2025) untersuchen den Einsatz von SOP-Wissen in einem LLM-basierten Multi-Agenten-System zur RCA. Im Unterschied dazu wird hier die Verfügbarkeit von Runbook-/Domänenwissen unabhängig von der Verfügbarkeit der Anwendungslogs variiert. Die Ergebnisse zeigen, dass zusätzliches Runbookwissen nicht in allen Incidents zu einer höheren RCA-Qualität führt. Sein beobachteter Nutzen hängt vielmehr davon ab, ob es zusätzliche diagnostische Informationen liefert und vom MAS korrekt verarbeitet wird.

Die Untersuchung grenzt sich zugleich von Arbeiten ab, die neue Agentenarchitekturen entwickeln oder verschiedene Modelle und Architekturen miteinander vergleichen. Ihr Beitrag liegt stattdessen in der kontrollierten Untersuchung zweier Informationsfaktoren innerhalb eines festgelegten RCA-Systems. Die Ergebnisse sind daher als incidentspezifische, deskriptive Befunde und nicht als allgemeiner Leistungsvergleich LLM-basierter RCA-Systeme einzuordnen.

6.5 Limitationen und Gefährdungen der Validität

Da die Ergebnisse aus einem kontrollierten Versuchsdesign stammen, ist eine statistische Verallgemeinerung auf andere Kubernetes-Systeme, Incidenttypen oder LLM-basierte RCA-Systeme nicht gegeben. Die starke Kontrolle des Versuchsaufbaus erhöht die Nachvollziehbarkeit innerhalb der Untersuchung, beseitigt aber nicht automatisch alle Validitätsrisiken. Die wichtigste Grenze liegt darin, dass die RCA-Qualität mit einer für diese Arbeit entwickelten und manuell angewendeten Bewertungsmatrix beurteilt wird. Bei der Lokalisierungsbewertung wurde zudem die Übereinstimmung des angegebenen Services und des verantwortlichen Kubernetes-Objekts verlangt, obwohl das RCA-Ausgabeformat im Servicefeld auch lediglich betroffene Komponenten zulässt. Dies führte insbesondere bei I3 dazu, dass Ausgaben mit korrekt identifiziertem Ursprungsobjekt teilweise nur einen Lokalisierungspunkt erhielten. Trotz festgelegter Bewertungsanker sind subjektive Ermessensentscheidungen bei der manuellen Zuordnung einzelner RCA-Aussagen zu den Bewertungsstufen nicht vollständig auszuschließen. Da keine unabhängige Zweitbewertung durchgeführt wurde, konnte die Übereinstimmung zwischen unterschiedlichen Beurteilenden nicht überprüft werden. Die Ergebnisse bleiben innerhalb des festgelegten

Bewertungsverfahrens nachvollziehbar, lassen sich aber nicht ohne Weiteres auf andere Bewertungsmaßstäbe übertragen. Zusätzlich bildet die Stabilitätsbewertung nur die Übereinstimmung der zentralen Root Cause und nicht die vollständige Konsistenz der RCA-Ausgaben ab. Des Weiteren wird durch das Versuchsdesign die Zusammensetzung der Pipeline geändert, indem bei unterschiedlichen Bedingungen manche Agenten gar nicht erst ausgeführt werden. Deswegen beschreiben die Ergebnisse den Einfluss der Informationsbedingungen innerhalb der konkret implementierten Multi-Agenten-Pipeline und keinen architekturunabhängigen Effekt der einzelnen Informationsquellen. Eine weitere Einschränkung betrifft die Ausgestaltung des verwendeten Runbooks. Dieses enthält neben System- und Baseline-Informationen überwiegend allgemeine Diagnoseprinzipien, jedoch keinen umfassenden Katalog fehlerbildspezifischer Diagnose- und Behebungsanweisungen. Da das Runbook im Versuchsdesign als vollständige Informationsquelle bereitgestellt oder vorenthalten wird, lässt sich zudem der Einfluss des enthaltenen Systemwissens nicht unabhängig vom Einfluss der allgemeinen Diagnosehinweise bestimmen. Die Ergebnisse beschreiben daher den Nutzen des konkret verwendeten Runbooks und sind nicht ohne Weiteres auf anders strukturierte oder stärker handlungsorientierte Runbooks übertragbar. Auch die Übertragbarkeit der Ergebnisse ist durch die kleine lokale Testumgebung, die vier kontrollierten Incident-Szenarien und die Verwendung eines einzelnen festgelegten Multi-Agenten-Systems mit Qwen3:8b begrenzt. Die Ergebnisse beschreiben daher das Verhalten dieses Untersuchungssystems unter den gewählten Kubernetes- und Informationsbedingungen und lassen sich nicht direkt auf andere Incidenttypen, Produktionsumgebungen, Modelle oder Agentenarchitekturen verallgemeinern. Die technischen Ausgangsbedingungen der RCA-Läufe sind durch eingefrorene Eingaben, dokumentierte Konfigurationen und Hashprüfungen gut nachvollziehbar und reproduzierbar. Die Modellantworten selbst bleiben jedoch stochastisch, weshalb identische Ausgaben nicht erwartet werden. Die drei RCA-Durchläufe pro Testfall machen diese Variation sichtbar, erlauben aufgrund ihrer geringen Anzahl jedoch nur eine begrenzte Aussage über die langfristige Stabilität des Systems. Die kontrollierte Versuchsdurchführung und die dokumentierten Bewertungs- und Reproduzierbarkeitsmaßnahmen schaffen eine belastbare Grundlage für den Vergleich der Informationsbedingungen innerhalb des untersuchten Systems. Die Aussagekraft bleibt jedoch auf die vier kontrollierten Incident-Szenarien, die verwendete Multi-Agenten-Pipeline und das festgelegte Bewertungsverfahren begrenzt. Die Ergebnisse sind daher als deskriptive Befunde dieses Versuchsaufbaus und nicht als allgemeingültige Aussagen über LLM-basierte Kubernetes-RCA zu verstehen.

7 Fazit und Ausblick

Insgesamt zeigt die Arbeit, dass die Verfügbarkeit von Anwendungslogs und Runbook-/Domänenwissen die RCA beeinflusst, aber nicht unabhängig vom Incident. Entscheidend hierbei ist der diagnostische Zusatznutzen gegenüber den konstant verfügbaren Kubernetes-/HTTP-Daten. Die Untersuchung verdeutlicht, dass zusätzliche Informationen nicht zwangsläufig die Diagnose verbessern und dass neben der Informationsverfügbarkeit auch die korrekte Verarbeitung der vorhandenen Evidenz eine wesentliche Voraussetzung für eine belastbare Diagnose darstellt. Des Weiteren reicht die reine Root-Cause-Korrektheit nicht für die Beurteilung der Qualität der RCA aus. Vielmehr ist entscheidend, ob die Fehlerursache präzise lokalisiert wird, die Diagnose durch die verfügbare Evidenz gestützt ist und die RCA-Ausgabe keine Halluzinationen enthält. Daraus folgt, dass die RCA mehrere Qualitätsdimensionen besitzt, welche differenziert bewertet werden müssen. Aufgrund des begrenzten Untersuchungsumfangs und der kontrollierten Kubernetes-Testumgebung lassen sich die Ergebnisse nicht ohne Weiteres auf andere LLM-basierte RCA-Systeme oder produktive Betriebsumgebungen übertragen.

Zukünftige Arbeiten könnten daher den Versuchsaufbau erweitern, um die Übertragbarkeit und Robustheit der gewonnenen Erkenntnisse zu überprüfen. Hierfür bieten sich zusätzliche Incident-Szenarien, größere und produktionsnähere Kubernetes-Umgebungen, weitere Informationsquellen wie Metriken und Traces sowie eine höhere Anzahl an Wiederholungen an. Zusätzlich könnten der Vergleich unterschiedlicher lokaler Sprachmodelle und Agentenarchitekturen sowie gezielte Verbesserungen der Evidenzauswahl, Evidenzzusammenführung und Unsicherheitskommunikation weitere Erkenntnisse über die Zuverlässigkeit LLM-basierter RCA-Systeme liefern. Im Mittelpunkt steht dabei die Frage, wie verfügbare Informationen nicht nur zur Identifikation einer Ursache, sondern auch für eine nachvollziehbare und belastbare Root Cause Analysis genutzt werden können.

Literaturverzeichnis

Cornacchia, A., Alabdulaal, I., Saghier, I., Mirdad, A., Fayoumi, O., & Canini, M. (2025). Between promise and pain: The reality of automating failure analysis in microservices with LLMs. *Proceedings of the 16th ACM SIGOPS Asia-Pacific Workshop on Systems*, 155–167. https://doi.org/10.1145/3725783.3764388

Di, P., Chen, F., Bai, X., Yang, H., Li, Q., Wei, G., Mou, J., Shi, F., Chen, K., Tang, P., Shen, Z., Li, Z., Shi, W., Guo, J., & Yu, H. (2026). OpenDerisk: An industrial framework for AI-driven SRE, with design, implementation, and case studies. *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering: Software Engineering in Practice*, 186–199. https://doi.org/10.1145/3786583.3786864

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, *55*(12), 248:1–248:38. https://doi.org/10.1145/3571730

Kubernetes. (2025a, 22. Juni). *EndpointSlices*. Verfügbar 20. August 2026 unter https://kubernetes.io/docs/concepts/services-networking/endpoint-slices/

Kubernetes. (2025b, 24. November). *Objects in Kubernetes*. Verfügbar 18. August 2026 unter https://kubernetes.io/docs/concepts/overview/working-with-objects/

Kubernetes. (2026a, 13. Mai). *Event*. Verfügbar 18. August 2026 unter https://kubernetes.io/docs/reference/kubernetes-api/core/event-v1/

Kubernetes. (2026b, 15. März). *Labels and selectors*. Verfügbar 18. August 2026 unter https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Kubernetes. (2026c, 17. April). *Liveness, readiness, and startup probes*. Verfügbar 18. August 2026 unter https://kubernetes.io/docs/concepts/workloads/pods/probes/

Kubernetes. (2026d, 22. Mai). *Pod lifecycle*. Verfügbar 18. August 2026 unter https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/

Kubernetes. (2026e, 18. Juli). *Pods*. Verfügbar 18. August 2026 unter https://kubernetes.io/docs/concepts/workloads/pods/

Kubernetes. (2026f, 23. Juli). *Secrets*. Verfügbar 18. August 2026 unter https://kubernetes.io/docs/concepts/configuration/secret/

Kubernetes. (2026g, 3. Juni). *Service*. Verfügbar 18. August 2026 unter https://kubernetes.io/docs/concepts/services-networking/service/

Las-Casas, P., Kumbhare, A. G., Fonseca, R., & Agarwal, S. (2024). LLexus: An AI agent system for incident management. *ACM SIGOPS Operating Systems Review*, *58*(1), 23–36. https://doi.org/10.1145/3689051.3689056

Ollama. (2025, 28. Mai). *Streaming responses with tool calling*. Verfügbar 18. Oktober 2026 unter https://ollama.com/blog/streaming-tool

OpenTelemetry. (2026a, 7. August). *Logs*. Verfügbar 18. August 2026 unter https://opentelemetry.io/docs/concepts/signals/logs/

OpenTelemetry. (2026b, 10. März). *Signals*. Verfügbar 18. August 2026 unter https://opentelemetry.io/docs/concepts/signals/

Pei, C., Wang, Z., Liu, F., Li, Z., Liu, Y., He, X., Kang, R., Zhang, T., Chen, J., Li, J., Xie, G., & Pei, D. (2025). Flow-of-Action: SOP enhanced LLM-based multi-agent system for root cause analysis. *Companion Proceedings of the ACM on Web Conference 2025*, 422–431. https://doi.org/10.1145/3701716.3715225

Qwen. (2025, 21. Mai). *Qwen3-8B model card*. Verfügbar 1. September 2026 unter https://huggingface.co/Qwen/Qwen3-8B

Riddell, E., Riddell, J., Sun, G., Antkiewicz, M., & Czarnecki, K. (2026). Stalled, biased, and confused: Uncovering reasoning failures in LLMs for cloud-based root cause analysis. *Proceedings of the 2026 IEEE/ACM Third International Conference on AI Foundation Models and Software Engineering*, 172–183. https://doi.org/10.1145/3793655.3793732

Roy, D., Zhang, X., Bhave, R., Bansal, C., Las-Casas, P., Fonseca, R., & Rajmohan, S. (2024). Exploring LLM-based agents for root cause analysis. *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 208–219. https://doi.org/10.1145/3663529.3663841

Sun, W., Wang, T., Tian, X., Lan, W., Feng, X., Li, H., & Wang, F. (2026). MetaKube: An experience-aware LLM framework for Kubernetes failure diagnosis. *Proceedings of the ACM Web Conference 2026*, 5612–5623. https://dl.acm.org/doi/10.1145/3774904.3792631

Tundo, A., Mobilio, M., Riganelli, O., & Mariani, L. (2023). Automated probe life-cycle management for monitoring-as-a-service. *IEEE Transactions on Services Computing*, *16*(2), 969–982. https://doi.org/10.1109/TSC.2022.3180500

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, *18*(6), 186345. https://doi.org/10.1007/s11704-024-40231-1

Wang, Z., Liu, Z., Zhang, Y., Zhong, A., Wang, J., Yin, F., Fan, L., Wu, L., & Wen, Q. (2024). RCAgent: Cloud root cause analysis by autonomous agents with tool-augmented large language models. *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 4966–4974. https://doi.org/10.1145/3627673.3680016

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. *The Eleventh International Conference on Learning Representations*. Verfügbar 23. Juli 2026 unter https://openreview.net/forum?id=WE_vluYUL-X

A Runbook: Betrieb und Diagnose der BA-RCA-Testanwendung

Dokumentationsgrundlage: Das nachfolgende Runbook entspricht inhaltlich der im Experiment verwendeten Datei `docs/runbooks/system-runbook.md` aus dem eingefrorenen Repository-Stand `bd10aa54320caeeda10cb794e10dc4afbd9a7368`.

A.1 Zweck und Abgrenzung

Dieses Runbook beschreibt den erwarteten Aufbau, den fehlerfreien Baseline-Zustand und die allgemeine Diagnosemethodik der BA-RCA-Testanwendung.

Es dient als System- und Domänenwissen für die Root Cause Analysis. Es enthält keine aktuellen Zustandsdaten, keine Informationen über experimentelle Incident-Szenarien und keine Ground Truth.

Für die Diagnose gelten folgende Regeln:

- Das Runbook beschreibt, wie das System im fehlerfreien Zustand aufgebaut sein und funktionieren soll.
- Ob eine Abweichung tatsächlich vorliegt, muss anhand der bereitgestellten Kubernetes-Daten, HTTP-Probes und Logs belegt werden.
- Ein Runbook-Hinweis ist keine Evidenz für eine konkrete Ursache.
- Bei Widersprüchen haben die beobachteten Zustandsdaten Vorrang.
- Fehlende Informationen und verbleibende Unsicherheit müssen ausdrücklich benannt werden.

A.2 Systemübersicht

Die Anwendung simuliert eine einfache Bestellverarbeitung und wird als containerisierte Microservice-Anwendung in einem lokalen Kubernetes-Cluster betrieben.

Tabelle A.1: Systemübersicht der BA-RCA-Testanwendung

Eigenschaft	Erwarteter Wert
Kubernetes-Namespace	`ba-rca`
Anwendungskomponenten	`frontend`, `api-service`, `order-service`, `worker-service`, `postgres`
Interne Service-Bereitstellung	`ClusterIP`
Extern bereitgestellter Service	`frontend` als `NodePort`
Datenbankbetrieb	PostgreSQL als `Deployment`

PostgreSQL wird in dieser Testumgebung zur kontrollierten Reproduzierbarkeit als `Deployment` betrieben. Ein produktionsnaher Datenbankbetrieb mit `StatefulSet` und persistentem Storage ist nicht Gegenstand dieses Systems.

A.2.1 Abhängigkeitsmodell

```
frontend -> api-service -> order-service -> postgres
worker-service -> order-service -> postgres
```

Ein Fehler an einem Einstiegspunkt kann daher durch eine nachgelagerte Abhängigkeit verursacht werden. Meldungen in vorgelagerten Services können Folgefehler sein und belegen nicht automatisch, dass dort die direkte Ursache liegt.

A.3 Service- und Abhängigkeitskatalog

Tabelle A.2: Komponenten und Abhängigkeiten der Testanwendung

Komponente	Aufgabe	Port	Relevante Endpunkte	Direkte Abhängigkeit
`frontend`	Simuliert den Checkout	8002	`GET /health`, `GET /ready`, `POST /checkout`	`api-service:8000`
`api-service`	Stellt die Order-API bereit und leitet Anfragen weiter	8000	`GET /health`, `GET /ready`, `POST /api/orders`	`order-service:8001`
`order-service`	Erstellt und liest Orders	8001	`GET /health`, `GET /ready`, `GET /orders`, `POST /orders`	`postgres:5432`

Komponente	Aufgabe	Port	Relevante Endpunkte	Direkte Abhängigkeit
`worker-service`	Simuliert die Verarbeitung eines Hintergrundjobs	8003	`GET /health`, `GET /ready`, `POST /process-job`	`order-service:8001`
`postgres`	Speichert die Order-Daten	5432	PostgreSQL-Protokoll	keine weitere Anwendungsabhängigkeit

A.3.1 Bedeutung der Anwendungsendpunkte

`/health` zeigt an, dass der jeweilige Anwendungsprozess erreichbar ist und der HTTP-Endpunkt antwortet. Ein erfolgreicher Healthcheck belegt nicht, dass alle Abhängigkeiten erreichbar sind oder die gesamte Servicekette funktioniert.

`/ready` zeigt den durch den jeweiligen Service implementierten Betriebsbereitschaftszustand an. Beim `order-service` umfasst diese Prüfung eine aktive Datenbankverbindung. Bei `frontend`, `api-service` und `worker-service` bestätigt der Endpunkt nur die lokale Betriebsbereitschaft und prüft nicht die gesamte nachgelagerte Servicekette.

Die fachlichen Endpunkte prüfen einen größeren Teil der Abhängigkeitskette:

Tabelle A.3: Fachliche Endpunkte und ihre Abhängigkeitsketten

Fachlicher Endpunkt	Einbezogene Kette
`POST frontend:8002/checkout`	`frontend -> api-service -> order-service -> postgres`
`POST api-service:8000/api/orders`	`api-service -> order-service -> postgres`
`POST worker-service:8003/process-job`	`worker-service -> order-service -> postgres`
`GET order-service:8001/orders`	`order-service -> postgres`

A.4 Verifizierter Baseline-Zustand

Im fehlerfreien Zustand gelten die folgenden Erwartungen:

- Alle fünf Deployments besitzen jeweils eine gewünschte und eine verfügbare Replik.
- Alle fünf Pods laufen und sind `Ready`.
- Die Services `frontend`, `api-service`, `order-service`, `worker-service` und `postgres` sind vorhanden.

- Jeder Service besitzt mindestens einen zugeordneten, bereiten Backend-Endpunkt.
- Alle `/health`- und `/ready`-Probes liefern einen erfolgreichen HTTP-Status.
- Checkout, Order-Erstellung, Order-Abfrage und Job-Verarbeitung funktionieren.
- Die beiden fachlichen Aufrufketten erreichen den `order-service` und die PostgreSQL-Datenbank.

A.4.1 Baseline-Konfiguration

Die folgenden Werte beschreiben die erwarteten Service-Abhängigkeiten:

Tabelle A.4: Erwartete Baseline-Konfiguration der Anwendungskomponenten

Komponente	Konfiguration	Erwarteter Wert
`frontend`	`API_SERVICE_HOST`	`api-service`
`frontend`	`API_SERVICE_PORT`	`8000`
`api-service`	`ORDER_SERVICE_HOST`	`order-service`
`api-service`	`ORDER_SERVICE_PORT`	`8001`
`worker-service`	`ORDER_SERVICE_HOST`	`order-service`
`worker-service`	`ORDER_SERVICE_PORT`	`8001`
`order-service`	`DB_HOST`	`postgres`
`order-service`	`DB_PORT`	`5432`
`order-service`	`DB_NAME`	`ba_orders`
`order-service`	`DB_USER`	`ba_user`

Der `order-service` benötigt zusätzlich Datenbankzugangsdaten. PostgreSQL und `order-service` müssen für die Authentifizierung konsistente Zugangsdaten verwenden. Geheime Werte selbst gehören nicht zum Diagnoseinput und dürfen nicht aus Secret-Metadaten abgeleitet werden.

Kubernetes-Services ordnen ihre Backends über Labels und Selektoren zu. In der Baseline muss diese Zuordnung für jeden Service mindestens einen bereiten Backend-Endpunkt ergeben.

Für konfigurierte Kubernetes-Readiness-Probes müssen Ziel, Port und Prüfmethode mit dem tatsächlich bereitgestellten Endpunkt übereinstimmen. Eine erfolgreiche Probe und der Anwendungsendpunkt `/ready` sind getrennte Evidenzquellen: Die Kubernetes-Probe beeinflusst den Ready-Zustand und die Traffic-Zuordnung, während die aktive HTTP-Probe die konkrete Antwort der Anwendung dokumentiert.

A.5 Bedeutung der Evidenzquellen

Keine einzelne Datenquelle belegt in jedem Fall bereits eine Root Cause. Relevante Beobachtungen müssen über mehrere Quellen korreliert werden.

Tabelle A.5: Aussagekraft und Grenzen der verfügbaren Evidenzquellen

Evidenzquelle	Was sie unmittelbar zeigt	Was sie allein nicht beweist
Pod- und Deployment-Zustand	Phase, Ready-Zustand, Restarts, gewünschte und verfügbare Replikate	konkrete Ursache einer Abweichung
Pod- und Deployment-Beschreibung	konfigurierte Container, Umgebungsreferenzen, Probes, Status und zugehörige Events	erfolgreiche fachliche Funktion
Deployment- und Pod-YAML	tatsächlich ausgerollte Konfiguration und aktueller Kubernetes-Status	ob eine Abhängigkeit im Beobachtungszeitpunkt erreichbar war
Service-YAML und Service-Beschreibung	Service-Typ, Ports, Selektoren und zugeordnete Backends	warum ein Backend fehlt oder eine Anfrage fehlschlägt
EndpointSlices	aktuell einem Service zugeordnete Backends und deren Bereitschaft	alleinige Ursache einer fehlenden oder unbereiten Zuordnung
Klassische Endpoints	kompakte ergänzende Sicht auf zugeordnete Adressen und Ports	vollständige moderne Bereitschaftssemantik
Kubernetes-Events	von Kubernetes gemeldete Vorgänge und Fehler im Beobachtungszeitraum	vollständige Anwendungshistorie oder fachliche Ursache
Aktuelle Containerlogs	Meldungen des aktuell laufenden Containers	Ereignisse vor einem Neustart oder Kubernetes-Konfiguration
Vorherige Containerlogs	Meldungen der vorherigen Containerinstanz	Zustand anderer Komponenten
Secret-Metadaten	Existenz eines Secrets sowie vorhandene Schlüsselbezeichnungen	geheime Werte oder deren inhaltliche Übereinstimmung
`GET /health`	Erreichbarkeit des Prozesses und des Health-Endpunkts	Betriebsbereitschaft aller Abhängigkeiten
`GET /ready`	Antwort der implementierten Readiness-Logik	Funktionsfähigkeit der vollständigen Servicekette
Fachlicher Endpunkt	Ergebnis einer konkreten Funktion über die jeweils beteiligte Abhängigkeitskette	genaue Position der Ursache ohne weitere Evidenz

EndpointSlices sind die primäre Quelle für die Zuordnung und Bereitschaft von Service-Backends. Die klassische Endpoint-Ausgabe wird nur ergänzend betrachtet.

Eine nicht vorhandene oder leere Datei ist nicht gleichbedeutend mit einem fehlerfreien Zustand. Zuerst muss geprüft werden, ob die betreffende Informationsquelle überhaupt

bereitgestellt und erfolgreich erfasst wurde.

A.6 Allgemeiner Diagnoseablauf

A.6.1 Sichtbares Symptom bestimmen

Zuerst ist festzuhalten, welcher beobachtete Einstiegspunkt abweicht: ein Kubernetes-Objekt, ein Health- oder Readiness-Ergebnis oder eine fachliche Funktion.

A.6.2 Betroffene Abhängigkeitskette verfolgen

Ausgehend vom sichtbaren Symptom wird die zugehörige Servicekette in Richtung der nachgelagerten Abhängigkeiten geprüft. Ein weitergereichter Fehler ist von dem frühesten belegbaren Fehler in der Kette zu unterscheiden.

A.6.3 Ist- und Sollzustand vergleichen

Der beobachtete Zustand wird mit dem Baseline-Zustand verglichen. Dabei sind konkrete Abweichungen in Status, Konfiguration, Routing, Probes und HTTP-Ergebnissen mit ihren Dateipfaden festzuhalten.

A.6.4 Diagnoseebenen getrennt betrachten

Für jede betroffene Komponente werden vier Ebenen getrennt bewertet:

1. Läuft der Prozess beziehungsweise Container?
2. Ist der Pod aus Kubernetes-Sicht bereit?
3. Ist ein Backend über den zugehörigen Service für Traffic ausgewählt?
4. Funktioniert die fachliche Operation einschließlich ihrer Abhängigkeiten?

Ein positives Ergebnis auf einer Ebene impliziert kein positives Ergebnis auf den nachfolgenden Ebenen.

A.6.5 Evidenz korrelieren

Kubernetes-Zustände und -Konfigurationen, Events, HTTP-Ergebnisse sowie verfügbare aktuelle und vorherige Logs werden miteinander abgeglichen. Eine Hypothese wird stärker, wenn unabhängige Beobachtungen zu derselben Kausalkette passen, und schwächer, wenn sie beobachteten Daten widerspricht.

A.6.6 Ursache, Symptom und Folgefehler trennen

Die Root Cause ist die früheste hinreichend belegte Abweichung, welche die beobachteten Symptome erklärt. Fehlermeldungen in abhängigen oder vorgelagerten Services können Auswirkungen dieser Abweichung sein.

A.6.7 Alternative Erklärungen prüfen

Mindestens eine ernsthaft plausible Alternative ist gegen die vorhandene Evidenz zu prüfen. Eine Alternative darf erst ausgeschlossen werden, wenn eine Beobachtung ihr widerspricht oder eine andere Erklärung deutlich stärker belegt ist.

A.6.8 Unsicherheit dokumentieren

Wenn die Daten mehrere Erklärungen zulassen, darf keine spezifischere Ursache behauptet werden, als die Evidenz trägt. Fehlende Logs, Konfigurationswerte, Zustandsinformationen oder zeitliche Zusammenhänge sind ausdrücklich zu benennen.

A.7 Verifikation und Eskalation

Eine Handlungsempfehlung muss direkt zur diagnostizierten Abweichung passen. Sie darf nicht als bereits ausgeführt oder erfolgreich dargestellt werden.

Nach einer Korrektur müsste der Baseline-Zustand erneut vollständig geprüft werden:

- Deployment- und Pod-Verfügbarkeit,
- Ready-Zustände,
- Service-Backends in den EndpointSlices,
- Health- und Readiness-Endpunkte,

- Checkout und Order-Erstellung,
- Order-Abfrage und Job-Verarbeitung.

Eine eindeutige Root Cause ist nicht festzustellen, wenn zentrale Daten fehlen, mehrere Hypothesen gleich gut zu den Beobachtungen passen oder sich die Evidenzquellen wesentlich widersprechen. In diesem Fall muss die Diagnose auf die engste belegbare Aussage begrenzt und der zusätzlich benötigte Informationsbedarf angegeben werden.

Eine Eskalation sollte das sichtbare Symptom, die relevanten Beobachtungen mit Dateipfaden, die geprüften Hypothesen, bestehende Widersprüche und die fehlenden Informationen enthalten.

A.8 Experimentelle Abgrenzung

Dieses Runbook wird für alle Versuchsfälle unverändert verwendet. Es enthält keine Kennungen oder Beschreibungen der eingesetzten Incidents, Observability-Bedingungen oder Wiederholungsläufe.

Das Runbook stellt ausschließlich allgemeines System- und Baselinewissen bereit. Die konkrete Root Cause muss aus den im jeweiligen Eingabebündel verfügbaren Observability-Daten abgeleitet werden.

B Ground Truth der Incident-Szenarien

Dieser Anhang dokumentiert die für die vier Incident-Szenarien vorab festgelegte Ground Truth.

Die zentralen Referenzpunkte der vier Incidents sind in Tabelle B.1 zusammengefasst.

Tabelle B.1: Kurzüberblick über die Ground Truth der vier Incident-Szenarien

ID	Kanonische Root Cause	Verantwortliches Objekt
I1	`DB_HOST` fehlt im Container des `order-service`.	`Deployment/order-service`; Container-Umgebungsvariable `DB_HOST`.
I2	Das vom `order-service` verwendete PostgreSQL-Passwort stimmt nicht mit dem von PostgreSQL erwarteten Passwort überein.	`Secret/order-service-db-secret`; Schlüssel `DB_PASSWORD`.
I3	Der Selector von `Service/postgres` stimmt nicht mit dem Label des PostgreSQL-Pods überein.	`Service/postgres`; Feld `spec.selector`.
I4	Die Readiness-Probe des `api-service` verwendet `/invalid-ready` anstelle von `/ready`.	`Deployment/api-service`; Feld `readinessProbe.httpGet.path`.

Zur einheitlichen Beschreibung der Evidenzlage wird die Diagnostizierbarkeit gemäß Tabelle B.2 in drei Stufen eingeordnet. Die Einstufung beschreibt ausschließlich die vorab festgelegte Evidenzlage eines Incidents unter der jeweiligen Informationsbedingung und legt keine spätere Punktbewertung einer RCA-Ausgabe vorweg.

Tabelle B.2: Stufen zur Beschreibung der Diagnostizierbarkeit

Einstufung	Bedeutung
Eindeutig	Die konkrete Ursache und der verantwortliche Ursprungsort sind in den verfügbaren Daten unmittelbar ablesbar oder durch unabhängige Signale geschlossen belegt.
Stark begründbar	Die Diagnose ist die mit Abstand plausibelste Erklärung, enthält jedoch mindestens einen nicht direkt beobachtbaren Schritt.
Eingeschränkt	Nur Symptom, Fehlerklasse oder betroffener Bereich sind sicher erkennbar; die konkrete Ursache erfordert eine nicht hinreichend belegte Annahme.

B.1 Incident I1 – Fehlendes `DB_HOST` im `order-service`

Die incidentspezifische Referenz für I1 ist in Tabelle B.3 zusammengefasst.

Tabelle B.3: Ground Truth und Evidenzgrenzen für Incident I1

Merkmal	Festlegung
Fehlerklasse	Anwendungskonfiguration / Startfehler.
Kanonische Root Cause	Die erforderliche Umgebungsvariable `DB_HOST` fehlt im Container des `order-service`. Der Anwendungscode liest die Variable beim Start als Pflichtvariable ein, sodass der Prozess bei fehlendem Wert nicht erfolgreich startet.
Verantwortliches Objekt	`Deployment/order-service`; Container-Umgebungsvariable `DB_HOST`.
Handlungs-empfehlung	`DB_HOST=postgres` wiederherstellen, Deployment anwenden, Rollout und Pod-Stabilität prüfen und anschließend die betroffenen HTTP-Pfade testen.
Erwartbare Symptome / Folgefehler	Wiederholte Containerstarts bis hin zu `CrashLoopBackOff`, steigende Restarts, keine dauerhaft verfügbare Replik des `order-service`, fehlgeschlagene direkte und vorgelagerte fachliche HTTP-Aufrufe. Verfügbare Logs benennen die fehlende Variable.
B1	**Eindeutig.** Deployment ohne `DB_HOST`, Laufzeitfehler in den Logs, Restarts und HTTP-Ausfälle sowie Runbook-Sollwert `DB_HOST=postgres`.
B2	**Eindeutig.** Deployment ohne `DB_HOST`, Restarts und HTTP-Ausfälle sowie Runbook-Sollwert `DB_HOST=postgres`.
B3	**Eindeutig.** Deployment ohne `DB_HOST`, Restarts und HTTP-Ausfälle; die Logs benennen die fehlende Pflichtvariable.
B4	**Eingeschränkt bis stark begründbar.** Das Fehlen von `DB_HOST`, die übrigen vorhandenen Datenbankvariablen sowie Restarts und HTTP-Ausfälle sind sichtbar. Die Pflichtwirkung der Variable wird ohne Logs oder Runbook jedoch nicht unmittelbar belegt.
Bewertungsgrenze	`CrashLoopBackOff`, fehlende Service-Backends und fehlgeschlagene Bestellpfade sind Symptome beziehungsweise Folgefehler. Die Root Cause ist die fehlende Pflichtvariable `DB_HOST`.

B.2 Incident I2 – Falsches Datenbankpasswort im `order-service`

Die incidentspezifische Referenz für I2 ist in Tabelle B.4 zusammengefasst.

Tabelle B.4: Ground Truth und Evidenzgrenzen für Incident I2

Merkmal	Festlegung
Fehlerklasse	Zugangskonfiguration / Authentifizierungsfehler.
Kanonische Root Cause	Das vom `order-service` verwendete PostgreSQL-Passwort stimmt nicht mit dem von PostgreSQL erwarteten Passwort überein. Die Netzwerkverbindung erreicht PostgreSQL, die Authentifizierung wird jedoch abgelehnt.
Verantwortliches Objekt	`Secret/order-service-db-secret`; Schlüssel `DB_PASSWORD`.
Handlungs-empfehlung	Einen mit PostgreSQL konsistenten Wert für `DB_PASSWORD` im `order-service-db-secret` herstellen, den `order-service` neu ausrollen und anschließend `/ready`, die Order-Endpunkte und die fachlichen Aufrufpfade prüfen.
Erwartbare Symptome / Folgefehler	Der `order-service` läuft grundsätzlich weiter und kann lokal gesund erscheinen. Datenbankabhängige Readiness- und Order-Aufrufe schlagen jedoch mit einem PostgreSQL-Authentifizierungsfehler fehl. Vorgelagerte fachliche Aufrufketten können dadurch ebenfalls fehlschlagen. PostgreSQL-Pod und Service-Backend bleiben grundsätzlich funktionsfähig.
B1	**Stark begründbar.** Logs und HTTP-Probes zeigen den Authentifizierungsfehler; Kubernetes-Konfiguration und Runbook stützen einen Credential-Mismatch. Die Secret-Werte selbst bleiben verborgen.
B2	**Stark begründbar.** Direkte HTTP-Antworten zeigen den Authentifizierungsfehler, Kubernetes-Zustand und Backends bleiben weitgehend intakt und das Runbook fordert konsistente Credentials.
B3	**Stark begründbar.** Logs und HTTP-Probes zeigen den Authentifizierungsfehler; Secret-Referenzen sind sichtbar, die konkreten Secret-Werte jedoch nicht.
B4	**Stark begründbar.** HTTP-Probes zeigen den Authentifizierungsfehler; PostgreSQL, Service-Backends und Secret-Referenzen sind sichtbar. Der konkrete Secret-Inhalt und die tatsächlich veränderte Seite des Credential-Mismatchs sind nicht direkt beobachtbar.
Bewertungsgrenze	Die konkreten Secret-Werte werden nicht als RCA-Eingang erhoben. Eine korrekte RCA darf daher den Credential-Mismatch benennen und zugleich Restunsicherheit über den verborgenen Secret-Inhalt ausdrücken. Konkrete Passwortliterale dürfen nicht als beobachtete Evidenz dargestellt werden.

B.3 Incident I3 – Selector-Mismatch beim `Service/postgres`

Die incidentspezifische Referenz für I3 ist in Tabelle B.5 zusammengefasst.

Tabelle B.5: Ground Truth und Evidenzgrenzen für Incident I3

Merkmal	Festlegung
Fehlerklasse	Kubernetes-Service-Konfiguration / Backend-Zuordnung.
Kanonische Root Cause	`Service/postgres` verwendet den Selector `app=postgres-broken`, während der PostgreSQL-Pod das Label `app=postgres` besitzt. Dadurch wird dem Service kein passendes PostgreSQL-Backend zugeordnet.
Verantwortliches Objekt	`Service/postgres`; Feld `spec.selector`.
Handlungs-empfehlung	Selector auf `app=postgres` korrigieren und anschließend Endpoint-/EndpointSlice-Zuordnung sowie die Datenbankpfade des `order-service` prüfen.
Erwartbare Symptome / Folgefehler	Der PostgreSQL-Pod bleibt `Running` und `Ready`, während `Service/postgres` kein passendes Backend besitzt. Der `order-service` erreicht PostgreSQL nicht; daraus können Fehler in den vorgelagerten fachlichen Aufrufketten entstehen. Anwendungslogs können Verbindungsfehler zeigen.
B1	**Eindeutig.** Falscher Service-Selector, abweichendes Pod-Label, fehlende Backend-Zuordnung sowie HTTP- und Logfolgefehler sind sichtbar; zusätzlich steht das Runbook zur Verfügung.
B2	**Eindeutig.** Selector, Pod-Labels, EndpointSlices beziehungsweise Endpoints, PostgreSQL-Pod-Zustand und HTTP-Folgefehler belegen die Ursache; Logs sind nicht erforderlich.
B3	**Eindeutig.** Selector, Pod-Labels und Backend-Zuordnung belegen den Mismatch unmittelbar; Logs liefern lediglich zusätzliche Folgeinformationen.
B4	**Eindeutig.** Kubernetes- und HTTP-Basisdaten reichen zur direkten Gegenüberstellung von Service-Selector, Pod-Label und fehlender Backend-Zuordnung aus.
Bewertungsgrenze	Leere Endpoints beziehungsweise EndpointSlices sind das unmittelbare Routing-Symptom. Die Root Cause ist der falsche Selector `app=postgres-broken` in `Service/postgres`, nicht ein Ausfall des PostgreSQL-Pods.

B.4 Incident I4 – Falscher Readiness-Pfad im `api-service`

Die incidentspezifische Referenz für I4 ist in Tabelle B.6 zusammengefasst.

Tabelle B.6: Ground Truth und Evidenzgrenzen für Incident I4

Merkmal	Festlegung
Fehlerklasse	Kubernetes-Betriebs- / Readiness-Konfiguration.
Kanonische Root Cause	Die Readiness-Probe des `api-service` prüft `/invalid-ready` anstelle des tatsächlich implementierten Endpunkts `/ready`. Die Probe erhält dadurch HTTP 404 und der Pod wird aus Kubernetes-Sicht nicht bereit.
Verantwortliches Objekt	`Deployment/api-service`; `readinessProbe.httpGet.path`.
Handlungsempfehlung	Probe-Pfad auf `/ready` zurücksetzen, Port 8000 beibehalten und anschließend Rollout, Pod-Readiness, EndpointSlice und Checkout prüfen.
Erwartbare Symptome / Folgefehler	Der Prozess des `api-service` läuft, der Pod bleibt jedoch nicht `Ready`. Kubernetes meldet fehlgeschlagene Readiness-Probes mit HTTP 404 und dem Service fehlt ein regulär bereites Backend. Der Checkout kann deshalb scheitern. Ein Containerneustart wird durch diesen Readiness-Fehler nicht verursacht.
B1	**Eindeutig.** Probe-Konfiguration `/invalid-ready`, nicht bereiter Pod, 404-Probe-Events, fehlendes Service-Backend, Logs und Runbook-Sollpfad `/ready` stimmen überein.
B2	**Eindeutig.** Probe-Konfiguration, Pod-Zustand, 404-Probe-Events, EndpointSlice-/HTTP-Auswirkungen und Runbook reichen ohne Anwendungslogs aus.
B3	**Eindeutig.** Probe-Konfiguration, Pod-Zustand, 404-Probe-Events und HTTP-Auswirkungen belegen die Ursache; Logs liefern zusätzliche Bestätigung.
B4	**Eindeutig.** Probe-Konfiguration `/invalid-ready`, Pod-Zustand, 404-Probe-Events, fehlendes Backend und HTTP-Auswirkungen sind bereits in den Kubernetes-/HTTP-Basisdaten enthalten.
Bewertungsgrenze	Das Fehlschlagen einer HTTP-Anfrage über `Service/api-service` beweist für sich allein keinen defekten `/ready`-Anwendungsendpunkt, da dem Service wegen des nicht bereiten Pods bereits ein reguläres Backend fehlt. Die Probe-Konfiguration und die 404-Readiness-Events lokalisieren die Ursache.

C Bewertungsmatrizen und Entscheidungsregeln

Der Anhang dokumentiert die vollständigen Bewertungsanker, Ableitungsregeln und Erfassungsfelder des in Abschnitt 4.4 beschriebenen Bewertungsverfahrens.

C.1 Bewertung einzelner RCA-Ausgaben

Die dimensionsspezifischen Bewertungsanker für die direkte Bewertung einer RCA-Ausgabe sind in Tabelle C.1 festgelegt.

Tabelle C.1: Bewertungsmatrix für die direkte Bewertung einer RCA-Ausgabe

Kriterium	0 Punkte	1 Punkt	2 Punkte
Root-Cause-Korrektheit	Ursache falsch, widersprüchlich oder nicht genannt.	Richtige Fehlerklasse oder Diagnoserichtung erkannt, aber die entscheidende konkrete Ursache fehlt oder ist teilweise falsch.	Kanonische Root Cause gemäß Ground Truth korrekt, konkret und ohne sachlichen Widerspruch benannt.
Service-/Objekt-Lokalisierung	Verantwortlicher Service beziehungsweise das relevante Kubernetes-Objekt falsch oder nicht genannt.	Richtiger Systembereich oder Service erkannt, aber verantwortliches Objekt beziehungsweise Ursprung nicht präzise lokalisiert.	Verantwortlicher Service und das für den Incident relevante Kubernetes- beziehungsweise Konfigurationsobjekt gemäß Ground Truth korrekt lokalisiert.
Evidenzbezug und Nachvollziehbarkeit	Keine relevante Evidenz, falsche Evidenz oder eine Begründung, die den bereitgestellten Daten widerspricht.	Relevante Signale werden genannt, aber nur teilweise ausgewertet; zwischen Evidenz und Diagnose bleibt eine wesentliche Begründungslücke.	Die Diagnose wird mit konkreter, korrekt interpretierter Evidenz aus den verfügbaren Daten begründet und als nachvollziehbare Kausalkette dargestellt.
Umgang mit unvollständigen Informationen	Fehlende oder nicht aussagekräftige Daten werden ignoriert; die Ausgabe urteilt unbegründet sicher.	Einschränkungen oder Unsicherheit werden allgemein erwähnt, aber nicht auf die tatsächlich fehlenden beziehungsweise nicht eindeutigen Informationen bezogen.	Aussagekraft und Grenzen der verfügbaren Daten werden korrekt eingeschätzt; fehlende Quellen werden, sofern relevant, konkret benannt und die Sicherheit der Diagnose angemessen kalibriert.
Handlungs-empfehlung	Empfehlung fehlt, adressiert die Root Cause nicht, ist sachlich falsch oder potenziell schädlich.	Empfehlung ist grundsätzlich plausibel, bleibt aber allgemein oder behandelt überwiegend das Symptom.	Empfehlung adressiert die konkrete Root Cause, ist technisch passend und operativ nachvollziehbar.

Die binären Erfolgskennzahlen werden gemäß Tabelle C.2 aus der direkten Bewertung abgeleitet.

Tabelle C.2: Abgeleitete primäre Erfolgskennzahlen

Kennzahl	Berechnungsregel	Bedeutung
Root Cause korrekt	1, wenn *Root-Cause-Korrektheit* = 2; sonst 0.	Die kanonische Ursache wurde vollständig getroffen.
Lokalisierung korrekt	1, wenn *Service-/Objekt-Lokalisierung* = 2; sonst 0.	Verantwortlicher Service und relevantes Objekt wurden vollständig lokalisiert.
Gesamterfolg	1, wenn *Root Cause korrekt* = 1 und *Lokalisierung korrekt* = 1; sonst 0.	Ursache und Ursprungsort stimmen mit der Ground Truth überein.
Evidenzgestützter Erfolg	1, wenn *Gesamterfolg* = 1, *Evidenzbezug und Nachvollziehbarkeit* = 2 und Halluzination = *keine*; sonst 0.	Die korrekte Diagnose ist durch die verfügbaren Eingangsdaten getragen und enthält keine erfundenen oder den verfügbaren Daten widersprechenden Tatsachen.

Halluzinationen werden anhand des Schweregrads und des Halluzinationstyps gemäß den folgenden Entscheidungsregeln erfasst.

Tabelle C.3: Schweregrade der Halluzinationsbewertung

Schweregrad	Bewertungsregel
Keine	Tatsachenbehauptungen sind durch die bereitgestellten Daten gedeckt oder ausdrücklich als Hypothese beziehungsweise Unsicherheit gekennzeichnet.
Leicht	Einzelne unbelegte oder übermäßig spezifische Annahmen liegen vor, verändern jedoch die zentrale Root Cause, die Lokalisierung oder die Handlungsempfehlung nicht wesentlich.
Schwer	Eine schwere Halluzination liegt insbesondere dann vor, wenn Logs, Events, Objekte oder Systemverhalten erfunden oder den Eingangsdaten widersprechend dargestellt werden, unbelegt sichere Kausalketten behauptet oder nicht existierende Komponenten genannt werden sowie wenn darauf falsche oder potenziell schädliche Handlungsempfehlungen beruhen.

Zusätzlich zum Schweregrad wird der Halluzinationstyp gemäß Tabelle C.4 dokumentiert.

Tabelle C.4: Typisierung von Halluzinationen

Typ	Bedeutung
Widersprechend	Die Aussage steht im Widerspruch zu den tatsächlich bereitgestellten Eingangsdaten.
Nicht verifizierbar	Die Aussage ist anhand der im jeweiligen Lauf verfügbaren Eingangsdaten nicht überprüfbar oder nicht ausreichend belegt.
Entfällt	Es liegt keine Halluzination vor.

C.2 Bewertung und Aggregation der Wiederholungen

Die Stabilität der drei Wiederholungen eines Testfalls wird nach den in Tabelle C.5 festgelegten Regeln bestimmt.

Tabelle C.5: Regeln zur Stabilitätsbewertung eines Testfalls

Stabilitätsstufe	Regel
Hoch	Alle drei Läufe nennen inhaltlich dieselbe zentrale Root Cause.
Mittel	Genau zwei der drei Läufe nennen inhaltlich dieselbe zentrale Root Cause.
Niedrig	Keine zwei Läufe stimmen in der zentralen Root-Cause-Diagnose überein.

Ergänzend werden die in Tabelle C.6 aufgeführten Merkmale dokumentiert.

Tabelle C.6: Ergänzende Merkmale zur Einordnung der drei Wiederholungen

Zusatzmerkmal	Erfassung
Anzahl erfolgreicher Läufe	Anzahl der Läufe mit Gesamterfolg innerhalb des Testfalls; Wertebereich 0 bis 3.
Spannweite des Gesamtwerts	Differenz zwischen dem höchsten und niedrigsten 0–10-Punkte-Gesamtwert der drei Wiederholungen.
Halluzinationsschwere	Dokumentation, ob sich die Halluzinationsschwere zwischen den drei Wiederholungen unterscheidet.

Die für die Zusammenfassung der Einzelbewertungen verwendeten Berechnungsregeln sind in Tabelle C.7 festgelegt.

Tabelle C.7: Aggregationsregeln für die Bewertungskennzahlen

Auswertungsebene / Kennzahl	Berechnungsregel	Wertebereich
Einzelne RCA-Ausgabe	Summe der fünf direkten Bewertungskriterien.	0–10 Punkte
Testfall	Mittelwert der Gesamtwerte der drei Wiederholungen desselben Testfalls.	0–10 Punkte
Informations-bedingung	Mittelwert der Gesamtwerte der zwölf zugehörigen RCA-Ausgaben.	0–10 Punkte
Erfolgsrate	Anzahl der Ausgaben mit *Gesamterfolg* = 1, geteilt durch 12 und multipliziert mit 100.	0–100 %
Evidenzgestützte Erfolgsrate	Anzahl der Ausgaben mit *evidenzgestütztem Erfolg* = 1, geteilt durch 12 und multipliziert mit 100.	0–100 %
Halluzinationsrate	Anzahl der Ausgaben mit leichter oder schwerer Halluzination, geteilt durch 12 und multipliziert mit 100.	0–100 %
Schwere Halluzinationsrate	Anzahl der Ausgaben mit schwerer Halluzination, geteilt durch 12 und multipliziert mit 100.	0–100 %

C.3 Bewertungsbogen pro RCA-Ausgabe

Die bei der Bewertung jeder einzelnen RCA-Ausgabe zu erfassenden Felder sind in Tabelle C.8 zusammengefasst.

Tabelle C.8: Bewertungsbogen für eine einzelne RCA-Ausgabe

Feld	Eintrag
Incident / Bedingung / Lauf	I1–I4 / B1–B4 / Run 1–3
Root-Cause-Korrektheit	0 / 1 / 2
Service-/Objekt-Lokalisierung	0 / 1 / 2
Evidenzbezug und Nachvollziehbarkeit	0 / 1 / 2
Umgang mit unvollständigen Informationen	0 / 1 / 2
Handlungs-empfehlung	0 / 1 / 2
Gesamtwert	0–10
Root Cause korrekt	0 / 1, automatisch abgeleitet
Lokalisierung korrekt	0 / 1, automatisch abgeleitet
Gesamterfolg	0 / 1, automatisch abgeleitet
Evidenzgestützter Erfolg	0 / 1, automatisch abgeleitet
Halluzination	keine / leicht / schwer
Halluzinationstyp	widersprechend / nicht verifizierbar / entfällt
Kurzbegründung	Bezug auf Ground Truth und konkrete Eingangsdaten
Bewertungsnotiz / Grenzfallentscheidung	Kurze Begründung bei nicht eindeutiger Zuordnung; die festgelegte Auslegungsregel wird anschließend auf vergleichbare Ausgaben angewendet.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel

Prototypische Evaluation eines LLM-basierten Multi-Agenten-Systems zur Root Cause Analysis unter eingeschränkten Observability-Bedingungen in einer Kubernetes-Testumgebung

selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Bei der Erstellung dieser Bachelorarbeit wurde ChatGPT (OpenAI) als unterstützendes KI-Werkzeug eingesetzt. Die Unterstützung umfasste die Strukturierung des Runbooks und der Ground Truth, die Unterstützung bei der Implementierung des LLM-basierten Multi-Agenten-Systems sowie die sprachliche Überarbeitung der Bachelorarbeit. Die Verantwortung für die fachlichen Inhalte, die Ergebnisse und die endgültige Fassung der Arbeit liegt beim Verfasser.

Hamburg, 21.10.2026

Ort, Datum	Unterschrift