Mälardalen University
Department of Computer Science and Engineering
Västerås, Sweden

Thesis for the Degree of Master of Science (120 credits) in Computer Science with Specialization in Software Engineering

LLM-ASSISTED EVALUATION GENERATION FROM TRACES AND SYSTEM PROMPTS

Ishara Madhavi
ige25001@student.mdu.se

Examiner: Eduard Paul Enoiu
Mälardalen University, Västerås, Sweden

Supervisors: Wasif Afzal
Mälardalen University, Västerås, Sweden

Company supervisor: Sasa Moilanen, Tor Sandén
Zoined Oy, Finland

October 7, 2026

This thesis was conducted within the Erasmus Mundus Joint Master Degree Programme on the Engineering of Data-intensive Intelligent Software Systems (EDISS).

Acknowledgements

First and foremost, I'm deeply grateful to Zoined Oy for the opportunity to conduct this thesis in an industrial setting. I would especially like to thank the CEO, Sasa Moilanen, for his invaluable guidance and practical insights throughout this research. I am also sincerely thankful to Tor Sandén for his technical expertise, thoughtful suggestions, and for helping me bridge knowledge gaps and improve the quality of this work.

I want to express my heartfelt appreciation to my academic supervisor, Professor Wasif Afzal, for his unwavering guidance from the very beginning of this research. His insightful discussions, constructive feedback, and continuous support have been instrumental in shaping this thesis.

I'm also grateful to my examiner, Associate Professor Eduard Paul Enoiu, for his valuable feedback, thoughtful suggestions, and for helping validate and refine the research direction during the early stages of this work.

Furthermore, I would like to thank the Erasmus Mundus Joint Master's Program in Engineering of Data-intensive Intelligent Software Systems (EDISS), Mälardalen University, and Åbo Akademi University for providing an excellent academic environment and the opportunity to pursue this master's degree.

Finally, I would like to express my deepest gratitude to my family and friends for their unwavering love, patience, and encouragement throughout this journey. The support of those close to me has made this achievement possible.

Ishara Madhavi
Helsinki, August 2026

Abstract

Large Language Model (LLM)-based agentic chatbots are increasingly being deployed in real-world applications to assist users with a wide range of tasks through multi-step conversations. As these systems become more capable and autonomous, ensuring that they behave reliably across the diverse interactions they encounter in practice has become an important challenge.

Existing evaluation approaches typically rely on manually designing test cases to assess chatbot behaviour, making them labour-intensive, difficult to maintain as systems evolve, and often limited in coverage. This thesis investigates how LLMs can be used to automate the creation of scalable, high-quality evaluation suites for agentic AI chatbots.

The proposed framework generates structured evaluations from conversation traces collected during development and production use, while incorporating expert feedback to improve the quality and consistency of the evaluation process. To address behaviours that are not represented in previous interactions, the framework analyses the system's behavioural specifications and automatically generates additional evaluation scenarios to improve coverage.

The framework was empirically evaluated as a case study at Zoined Oy[1] *using ZED*[2]*, a retail analytics chatbot. Results demonstrate that expert-feedback refinement improved the evaluation acceptance rate from 59% to between 88% and 94% across independent runs, while specification-driven eval generation raised behavioural coverage from 30% to between 73% and 80%. Overall, this work contributes a reusable, application-agnostic evaluation framework and provides empirical evidence that LLMs can support systematic evaluation of agentic AI chatbots.*

[1] https://zoined.com/fi
[2] https://zoined.com/meet-zed

Table of Contents

List of Abbreviations **1**

1 Introduction **2**

2 Background **5**
2.1 Agentic AI Systems . . . 5
2.2 Test Cases, Evals, and Criteria . . . 5
2.3 LLM-as-judge evaluation . . . 6
2.4 Tracing Agentic AI Applications . . . 7
2.5 Evaluation Challenges for Agentic AI Systems . . . 7
2.6 Test Oracles . . . 8
2.7 Role of Human Oversight . . . 8

3 Related Work **10**
3.1 Observability in Agentic AI Systems . . . 10
3.2 LLM-Assisted Testing and Test Generation . . . 11
3.2.1 LLM-Assisted Unit Test Generation . . . 12
3.2.2 High-Level and End-to-End Test Generation . . . 12
3.2.3 Test Oracle Generation . . . 13
3.2.4 Test Augmentation and Mutation-Based Improvement . . . 14
3.3 Evaluation Frameworks for Conversational Agentic Systems . . . 15
3.4 Summary and Positioning of the Proposed Work . . . 16

4 Problem Formulation **18**
4.1 Purpose . . . 18
4.2 Research Questions . . . 18
4.3 Motivation . . . 18
4.4 Goals . . . 19

5 Method **21**
5.1 Overall Research Approach . . . 21
5.2 Case Study Context . . . 21
5.3 Method for Goal 1 – Trace-Driven Eval Construction . . . 22
5.4 Method for Goal 2 – Human-in-the-Loop Refinement . . . 23
5.5 Method for Goal 3 – Specification-Coverage-Driven Eval Generation . 25

6 Ethical and Societal Considerations **27**
6.1 Research Ethics . . . 27
6.2 Societal Considerations . . . 28

7 Design and Implementation **30**
7.1 Architecture Overview . . . 30
7.2 Phase 1 – Trace-Driven Eval Construction . . . 30
7.2.1 Trace Collection . . . 30
7.2.2 Trace Filtering and Deduplication . . . 32
7.2.3 Session Grouping . . . 32
7.2.4 Conversation Replay . . . 32

7.2.5 Rubric-Based Eval Construction . . . 33
7.3 Phase 2 – Human-in-the-Loop Refinement . . . 36
7.3.1 Expert Review . . . 36
7.3.2 Feedback Classification via Trace Validation . . . 37
7.4 Phase 3: Specification-Coverage-Driven Augmented Question Generation . . . 37
7.4.1 Behavioural Specification Extraction . . . 39
7.4.2 Specification Coverage Analysis . . . 39
7.4.3 Augmented Conversation Generation . . . 40

8 Results **42**
8.1 Evaluation Measures and Terminology . . . 43
8.2 Evaluating RQ2 . . . 43
8.3 Evaluating RQ3 . . . 47

9 Discussion **50**
9.1 Achievement of the Research Goals . . . 50
9.2 Interpreting the Expert-Feedback Refinement Results . . . 51
9.3 Interpreting the Specification-Driven Coverage Results . . . 52
9.4 Limitations of the Work . . . 53
9.5 Connections to Related Work . . . 55
9.6 Generalisability and Broader Implications . . . 56

10 Conclusions and Future Work **58**
10.1 Conclusions . . . 58
10.2 Future Work . . . 60

11 Declaration of Generative AI and AI-Assisted Tools **62**

References **67**

Appendix A Literature Search Strategy **68**

Appendix B Eval Generation Prompt **69**

List of Abbreviations

AI	Artificial Intelligence
API	Application Programing Interface
DSR	Design Science Research
GDPR	General Data Protection Regulation
HITL	Human-in-the-loop
JSON	JavaScript Object Notation
LLM	Large Language Model
RCA	Root Cause Analysis
SDK	Software Development Kit

1 Introduction

Recent advances in Large Language Models (LLMs) have enabled the rapid emergence of agentic AI systems capable of performing complex reasoning and context-aware decision-making [1]. Unlike traditional rule-based or deterministic applications, agentic conversational systems can autonomously interact with APIs, retrieve external knowledge, and adapt their behaviour based on dynamic user inputs [2]. For example, when a user asks for the current weather, an agent may first identify that external information is required, select an appropriate weather service, invoke its API, process the returned data, and then formulate a response. Throughout this process, the agent performs multiple intermediate reasoning and tool-use steps before producing the final answer. In subsequent turns, it can reuse the conversational context to answer follow-up questions, such as whether rain is expected later in the day, without requiring the user to repeat previously provided information. As a result, such systems are increasingly deployed in analytical, conversational, and operational settings where reliability and correctness are critical [3].

Evaluating agentic AI systems, however, remains a significant challenge [3, 4, 5]. In traditional software testing, program behaviour is typically verified by comparing the actual output against a predefined expected output or test oracle, resulting in a deterministic pass or fail outcome. This approach is well suited to conventional applications where the correct behaviour can be explicitly specified in advance. In contrast, the probabilistic nature of large language models means that multiple responses may be valid at the same time, making it impractical to define a single correct output for many tasks [6]. For example, a weather chatbot may answer the same question in multiple equally valid ways, differing in wording or level of detail. Therefore, the response should be evaluated based on criteria such as factual correctness, relevance, completeness, and clarity rather than an exact output match. Consequently, AI applications are evaluated using *evals*, which assess whether the behaviour of the application satisfies predefined evaluation criteria rather than matching a single expected answer [7, 8]. The challenge becomes even greater for agentic systems, whose behaviour is influenced not only by the initial prompt but also by intermediate reasoning steps, tool interactions, external observations, and contextual constraints, making it difficult to define what a correct response or criteria looks like in advance [3]. Although human judgement is considered the gold standard for evaluation, manually creating and maintaining test cases that comprehensively capture the diverse behaviours of agentic systems is time-consuming, inconsistent, and difficult to scale as these systems evolve [4, 9]. Thus, automated and robust evaluation strategies have become essential for ensuring the reliability and correctness of agentic AI applications [10, 5].

Recent work has explored LLM-assisted test generation, where a language model produces eval cases based on descriptions or templates [10, 5, 11]. Although these approaches have demonstrated encouraging results, current research remains fragmented, with most studies addressing specific testing tasks or application domains rather than providing a comprehensive solution [5]. As a result, important challenges such as hallucinated or unrealistic test cases, limited behavioural coverage, and weak alignment with real-world user interactions remain largely unresolved [12, 5]. Although grounding evaluations in real user behavior is crucial, relatively little research has explored this direction [5, 13]. A rich and largely underutilised source of real

user behaviour in agentic applications is execution traces – records of the inputs, outputs, tool calls, and intermediate steps produced during actual interactions with the system. These traces can serve as a foundation for grounding evaluation in real application behaviour. Existing observability platforms support the collection of such traces, but they do not address the challenge of automatically generating and refining evaluation criteria from them, nor do they provide mechanisms for systematically expanding coverage beyond what has already been observed in production [10].

In this context, this thesis presents an automated, LLM-assisted evaluation framework organized as a three-phase pipeline consisting of: (i) generating structured, rubric-based evaluations from conversation traces collected during real user and developer interactions; (ii) iteratively improving evaluation generation through a lightweight expert feedback loop; and (iii) expanding behavioural coverage by extracting specifications from the system's agent prompts and generating augmented question sessions that target previously untested behaviours. Conversation traces are collected from the agentic application under test, using Langfuse[3], an LLM observability platform, and processed by an LLM to generate evaluation criteria that are executed against the live application using Promptfoo[4], with a second LLM acting as a judge. The LLM-generated eval cases undergo a lightweight human review to identify hallucinated or low-quality evaluations before being incorporated into the evaluation suite. Evaluations that fail this review are used as direct feedback to automatically refine the eval-generation prompt, improving the quality of subsequent evaluation runs without requiring repeated manual review. Furthermore, behavioural specifications are automatically extracted from the application's agent prompts, mapped against the existing eval suite to identify coverage gaps, and used to guide an LLM in generating augmented multi-turn chat sessions that are fed back through the same pipeline to generate new evals – ensuring that latent, underrepresented behaviours are brought within the scope of the evaluation suite.

Empirical evaluation of the framework as a case study at Zoined Oy, using ZED, a retail analytics chatbot built on an agentic LLM architecture, demonstrates the practical effectiveness of the approach. Results show that incorporating expert feedback increased the evaluation acceptance rate from 59% to between 88% and 94% across independent runs, while specification-driven augmentation increased behavioural coverage from 30% to between 73% and 80%. These results suggest that combining trace-based generation, expert refinement, and specification-driven augmentation provides a scalable and practical methodology for evaluating agentic AI systems in industrial settings.

The main contributions of this work are as follows.

1. An automated, LLM-assisted evaluation framework that generates evaluation cases (evals) directly from real-world execution traces of an agentic application, grounding evaluation in actual system behavior.

2. A lightweight human-in-the-loop mechanism that incorporates expert feedback to iteratively refine the eval generation process, improving the accuracy of generated evals over time.

3. A method for systematically expanding evaluation coverage by generating new

[3] https://langfuse.com
[4] https://www.promptfoo.dev

evals based on behavioural specifications, targeting requirements that are underrepresented or absent from existing production traces.

4. Empirical evidence, through a case study using real-world data from Zoined Oy, demonstrating the framework's effectiveness in producing reliable and high-coverage evaluations for an agentic AI application.

To the best of the author's knowledge, this work is the first to present an end-to-end evaluation pipeline for agentic AI chat applications that is novel across multiple dimensions – grounding evaluation in real conversation traces, improving reliability through expert-feedback-driven prompt refinement, and expanding behavioural coverage through specification-driven augmentation – each representing a distinct contribution within a single cohesive framework.

The remainder of this report is organized as follows. Section 2 introduces background concepts on agentic AI systems, execution tracing, and evaluation fundamentals. Section 3 reviews related work on observability, LLM-assisted testing, and evaluation frameworks for conversational and agentic systems, and positions the proposed approach relative to the state of the art. Section 4 formulates the problem addressed by this thesis, including the research questions, motivation, goals, and limitations. Section 5 describes the research method, mapping each research question to a corresponding design goal and sub-method. Section 6 discusses the ethical and societal considerations relevant to this work. Section 7 presents the design and implementation of the three-phase evaluation pipeline. Section 8 reports the experimental results for the research questions. Section 9 discusses the implications, limitations, and broader significance of the findings. Finally, Section 10 concludes the thesis and outlines directions for future work.

2 Background

This section introduces the core concepts and terminology required to understand the problem addressed in this thesis and the proposed solution. This includes the foundations of agentic AI systems and execution tracing, followed by the key challenges associated with evaluating these systems. It then presents the concepts of test oracles and concludes by discussing the role of human oversight in AI evaluation.

2.1 Agentic AI Systems

Recent advances in generative AI have given rise to a new generation of intelligent systems known as agentic AI systems. These systems are capable of decomposing complex requests into intermediate tasks, interacting with external tools and information sources [14, 15, 16]. In this context, tools refer to external capabilities that extend the functionality of the language model beyond text generation, such as APIs, databases, search engines, calculators, or other software services that enable the agent to retrieve information or perform actions in the external environment [17]. While conventional autonomous systems may operate without continuous human input, they typically follow fixed rules and lack goal-oriented reasoning or adaptation. In contrast, agentic systems can interpret objectives, decide on actions, and adjust their behaviour based on changing contexts [16].

Agentic AI often adopts a multi-agent architecture similar to Figure 1, in which multiple specialized agents collaborate to accomplish complex tasks. Rather than relying on a single agent, an orchestrator may decompose a user's request into subtasks and delegate them to agents with different responsibilities [1]. For example, in a sales analytics assistant, one agent may retrieve sales data from external sources, another may analyse trends and identify possible causes of declining sales, while a third synthesizes the findings into a natural language response. By coordinating their reasoning and tool use, these agents collectively solve problems that would be difficult for a single agent to perform efficiently. This capability has led to the adoption of agentic AI systems across a wide range of domains, including healthcare, life sciences, industry, and agriculture, where they support tasks such as clinical decision-making, data analysis, knowledge discovery, and process automation by combining autonomous reasoning with external tools and domain-specific knowledge. [16].

While this adaptability allows agentic AI systems to solve a broader range of problems than traditional AI applications, it also introduces greater uncertainty in how they behave and where the limits of their capabilities lie [15]. Unlike traditional software components with deterministic control flow, agentic systems exhibit nondeterministic behavior due to probabilistic model outputs, evolving prompts, and dynamic tool selection [18]. As a result, correctness is no longer defined solely by final outputs but also by whether intermediate decisions, such as tool choice, parameter selection, or reasoning steps, align with user intent and system design.

2.2 Test Cases, Evals, and Criteria

This thesis draws on terminology from both software testing and evaluation of LLM-based agentic systems, where terms such as test, evaluation, oracle, criterion, and

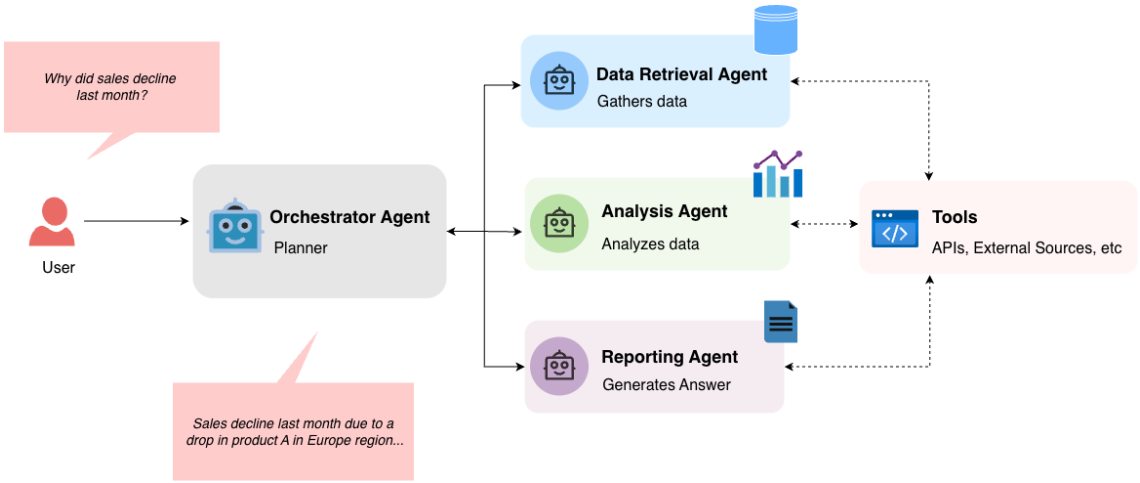

Figure 1: Example workflow of a multi-agent sales analytics assistant. An orchestrator agent decomposes a user's analytical query into subtasks, which are executed by specialized agents for data retrieval, trend analysis, and report generation. The agents coordinate their reasoning and tool use to produce a comprehensive response.

judge may be used differently. To avoid ambiguity, these terms are defined here and used consistently throughout the thesis.

A test case is the classical unit of software testing: an input, the conditions under which it is executed, and an oracle that decides whether the observed output is acceptable [24]. In conventional testing, the oracle is an exact-match assertion or a predicate over a return value.

An eval is a test case for a system whose output is natural language, in which the exact-match oracle is replaced by a set of natural-language criteria applied by an LLM judge. An eval in this work consists of (i) a user query together with its conversational context, (ii) one or more criteria, each stating a property the response must satisfy, and (iii) a judge that returns a pass or fail verdict with a justification for each criterion. An eval is therefore a test case, not a different kind of artifact.

Two finer terms are used consistently. A criterion is a single natural-language assertion inside an eval. A rubric is the complete set of criteria attached to one eval. One eval in this work carries between three and seven criteria.

2.3 LLM-as-judge evaluation

A conventional software test compares a system's output to a deterministic expected value. For natural-language outputs from AI systems, such exact matching is inappropriate: the same correct answer can be expressed in many different ways, and different aspects of correctness (factual accuracy, completeness, format) need to be assessed independently. LLM-as-judge is a technique in which a separate LLM instance is used to assess the quality of a target model's output against a set of natural-language criteria. Each criterion is expressed as an assertion, for example, "*the response correctly identifies the metric selected by the user*," and the judge LLM returns a pass/fail verdict with reasoning. This approach scales to large eval sets and can handle multi-dimensional assessments that would be impractical to specify as exact-match rules.

One such open-source framework is Promptfoo, which supports multiple LLM-

as-judge providers for evaluating LLM applications. It defines a standard YAML schema for evals and executes eval suites programmatically. Each eval case in the Promptfoo format consists of a *vars* block (the input variables, including the user query and any context), an *assert* block (one or more evaluation criteria with their types and expected values), and optional *metadata*.

2.4 Tracing Agentic AI Applications

An execution trace records the internal behavior of an application during runtime. In the context of agentic AI systems, traces typically capture much richer information than conventional logs. In addition to user inputs and final responses, traces may include intermediate reasoning steps, selected tools or APIs, input parameters, retrieved data, and generated intermediate outputs [19].

According to the example in Figure 1, when a user asks a sales analytics assistant, "*Why did sales decline last month?*", the trace may reveal that an orchestrator agent delegated the task to specialized agents responsible for retrieving sales data, analysing trends, and generating a report. Along the way, the trace records tool invocations, retrieved data, intermediate outputs, and the interactions between agents, providing a transparent view of how the final response was produced.

Such traces offer a structured view of agent behavior that can be analyzed, compared across regressions, and therefore used for evaluation purposes. They form the empirical foundation for the trace-driven evaluation approach explored in this thesis. The open-source tracing framework Langfuse is used to capture and organize these traces in the proposed methodology.

2.5 Evaluation Challenges for Agentic AI Systems

Evaluating agentic AI systems introduces challenges beyond those encountered in traditional software systems or static machine learning models [20]. Unlike deterministic systems, LLM-based agents can produce different outcomes for the same input due to their stochastic nature [18]. Moreover, failures may occur at different stages of execution, such as planning, information retrieval, tool selection, or decision-making, even when the final response appears reasonable. Identifying such failures is particularly challenging because agent behaviour involves complex, multi-step interactions that are costly and time-consuming to evaluate manually. Human-based evaluation can also introduce inconsistency and does not scale effectively for continuous assessment of deployed systems [4, 9].

Existing evaluation approaches often focus on final response correctness. However, a correct final answer does not necessarily indicate reliable agent behaviour. An agent may reach an acceptable outcome while using restricted tools, violating constraints, or skipping required intermediate steps. Therefore, evaluating agent reliability requires analysing the underlying execution process, including planning, context retrieval, tool usage, parameter selection, reasoning steps, and adherence to safety mechanisms [21].

Another challenge is ensuring that evaluation scenarios accurately reflect the diversity and complexity of real-world deployments. Many existing approaches rely on limited datasets or simulated environments, which may not capture unexpected user behaviour, noisy inputs, evolving requirements, or operational constraints encountered in practice [22, 23]. Furthermore, final answers may become outdated

as external knowledge, available tools, or task requirements change over time, and multiple responses may be considered valid depending on the context. This makes certain evaluations less suitable for long-term monitoring and regression testing [22].

Finally, the lack of standardized evaluation frameworks remains a barrier to systematic progress. Many studies rely on proprietary, in-house, or simulated datasets with different evaluation criteria and experimental settings, making it difficult to compare results across approaches [23]. In addition, privacy restrictions in sensitive domains limit access to real-world data, reducing reproducibility. Although synthetic environments enable controlled evaluation, they may not fully capture the complexity and uncertainty of practical agent interactions [23].

2.6 Test Oracles

A test oracle defines the expected behaviour of a system for a given input and is used to determine whether the system behaves correctly. However, manually specifying test oracles for agentic AI systems is often impractical because their behaviour depends on context, external knowledge, tool interactions, and multi-step reasoning. In many cases, complete and up-to-date specifications are unavailable, making manually defined or specification-based oracles difficult to maintain. As an alternative, *derived* test oracles can be constructed from existing artifacts such as system documentation or execution traces [24]. In particular, execution traces capture the expected sequence of agent actions and intermediate decisions, providing a practical basis for deriving evaluation criteria or assisting human evaluators. Although trace-derived oracles cannot guarantee the detection of every future defect, they offer a scalable foundation for automated evaluation, especially when complemented by test variation and expert validation.

The trace-derived oracles used in this thesis distinguish observed behaviour from intended behaviour. A criterion generated from a replayed trace describes what the system actually did, but this does not necessarily represent what the system should have done. The observed behaviour therefore serves as evidence for oracle generation, rather than as the source of expected behaviour. When the observed behaviour conflicts with the intended behaviour, the intended behaviour is established through human expert judgement. The framework consequently uses expert feedback as the additional source for validating and correcting trace-derived criteria, preventing incorrect system behaviour from being treated as the expected behaviour.

2.7 Role of Human Oversight

Although agentic AI systems are capable of solving complex tasks autonomously, they remain susceptible to a wide range of failures. Errors may arise in countless ways, including misinterpreting user intent, incorrect tool usage, or failure to recover from mistakes. In software engineering applications, such failures can have significant consequences, ranging from unintended modifications to source code to severe production incidents [25]. These risks make effective evaluation and human oversight essential. In the context of this thesis, Large Language Models (LLMs) are used to transform execution traces into structured eval cases and generate evaluation criteria. However, because LLM-generated artifacts may contain inaccuracies or hallucinations, expert validation remains necessary to ensure that the generated

evaluations accurately reflect the intended system behaviour, particularly in domains requiring business or domain-specific knowledge.

Building on these concepts, this thesis develops a trace-driven evaluation framework that transforms execution traces into structured eval cases, incorporates expert feedback to iteratively refine eval generation, and systematically expands eval coverage by generating new evals within defined specification boundaries. This combination is intended to ground evaluation in real system behaviour while substantially reducing the manual effort required to construct and maintain eval suites for evolving agentic AI systems.

3 Related Work

This section situates the proposed research within existing literature on observability, testing, and evaluation of conversational agentic AI systems. It reviews established approaches, highlights their strengths and limitations, and explains how the proposed trace-driven evaluation methodology extends the state of the art. The discussion is structured around three themes that are central to this thesis: observability of agentic systems, LLM-assisted testing, and evaluation frameworks for conversational agents.

To support transparency and reproducibility of the literature selection, the literature sources searched, the search queries, and the inclusion criteria applied are documented in Appendix A.

3.1 Observability in Agentic AI Systems

As AI systems increasingly adopt agentic designs where large language models autonomously plan actions, invoke tools, and interact with external services, understanding their internal behaviour becomes substantially more complex. Unlike traditional software systems with predictable execution paths, agentic AI applications operate in dynamic and often non-deterministic ways [16]. Their behaviour may also evolve over time as prompts, tools, or available context change. Since the correctness of an agent depends not only on its final response but also on the sequence of intermediate decisions, reasoning steps, and tool invocations that lead to that response, these internal behaviours must also be observable and evaluated [21]. Consequently, observability becomes a fundamental requirement for debugging, safety assurance, accountability, and reliable evaluation rather than merely an optional monitoring feature.

Recent research has addressed these challenges by extending classical DevOps concepts to agentic AI systems. The notion of *AgentOps* has been proposed as a specialized operational paradigm that emphasizes visibility into agent-specific execution artifacts [18]. Rather than focusing solely on input-output logging, this work highlights the importance of tracing higher-level constructs such as agent objectives, plans, reasoning steps, and tool usage. By conducting a systematic analysis of existing tooling, the authors identify gaps in current observability solutions and introduce an artifact relationship model that organizes how these elements evolve throughout an agent's lifecycle. This perspective frames observability as a structured and continuous process that is essential for managing the safety and reliability of autonomous agents.

Building on the need to connect high-level agent behavior with low-level system execution, *AgentSight* proposes a system-oriented observability approach that links semantic intent to concrete system actions [26]. Using instrumentation-free techniques, the framework captures signals from both the language-model layer and the underlying operating system. This enables real-time correlation between prompts, tool calls, and actual system effects. By maintaining this causal linkage, AgentSight demonstrates how inefficiencies, security vulnerabilities, or coordination failures can be identified as they occur, particularly in environments where agents interact with external tools or services.

Observability becomes even more critical in multi-agent systems, where multiple

autonomous entities collaborate to solve complex tasks [1]. In such settings, failures often emerge from interactions between agents rather than from isolated decisions. LumiMAS addresses this challenge by introducing an observability framework designed specifically for multi-agent workflows [27]. The framework combines continuous monitoring and logging with automated anomaly detection and explanation mechanisms, including classification and root cause analysis. Through evaluation across several multi-agent applications, LumiMAS shows that trace-based observability can effectively surface system-level failures, including those caused by hallucinated reasoning or biased decision-making. This highlights the importance of capturing execution context across agents to understand emergent behavior.

Recent research in [28] has also recognized that observability plays a broader role than debugging or system monitoring. Structured execution traces that capture an agent's reasoning, tool interactions, contextual information, and operational decisions provide a transparent record of how a task was performed, improving interpretability, reproducibility, and accountability. Such trace representations enable developers to diagnose failures and better understand agent behaviour. Furthermore, the authors identify agent evaluation as an important future application of structured traces, suggesting that they can support the development of richer evaluation metrics and systematic comparison of agent behaviours beyond final task outcomes. This reinforces the idea that observability is not only essential for operating agentic systems but also forms a foundation for agentic application evaluation.

Beyond technical diagnostics, broader discussions on visibility in AI systems emphasize observability as a key mechanism for governance and oversight [29]. The work in [29] examines practical approaches such as agent identifiers, runtime monitoring, and activity logs. Importantly, it also considers trade-offs related to privacy, system intrusiveness, and control, underscoring that increased transparency must be balanced with ethical and operational constraints.

Together, these studies establish observability as a crucial foundation for understanding, monitoring, and governing agentic AI systems. However, despite the rich execution data produced by these approaches, traces are still primarily used for inspection, debugging, or anomaly detection. They are rarely leveraged in research as structured inputs for systematic evaluation. Existing observability frameworks do not provide direct mechanisms for converting real execution traces into reusable test cases or formal evaluation criteria. As a result, the potential of traces to support scalable, repeatable, and evolving evaluation remains largely untapped. This gap directly motivates the approach taken in this thesis, which builds on observability research by transforming execution traces into structured evaluation artifacts that can be automatically extended, validated, and reused.

3.2 LLM-Assisted Testing and Test Generation

The growing complexity of modern software systems, particularly those incorporating large language models, has led to increased interest in using LLMs themselves to support testing activities. Recent research explores how language models can assist in generating test cases, improving coverage, and reducing the manual effort traditionally required to maintain test suites [30, 31, 32, 5]. These efforts span multiple levels of abstraction, ranging from low-level unit tests to high-level, end-to-end conversational evaluations.

3.2.1 LLM-Assisted Unit Test Generation

Recent research has increasingly explored the use of large language models to support the automatic creation and refinement of unit tests, motivated by the high cost and effort of manual test development [33]. One representative approach is CasModaTest [33], which reframes unit test generation as a two-stage process rather than a single monolithic task. It first focuses on constructing the structural skeleton of a test case and subsequently derives appropriate assertions to validate program behavior. The generated tests are then validated through compilation and execution, with additional automated correction steps applied when failures are encountered. A limitation of CasModaTest is that its effectiveness has been demonstrated primarily on a specific set of unit test datasets, which raises concerns about its generalizability to other programming languages.

Other approaches aim to improve the semantic correctness of generated tests. A3Test [34] enhances automated test generation by incorporating explicit knowledge about assertions, while also checking consistency in naming conventions and test signatures to ensure alignment with expected coding practices. In contrast, AgoneTest2 [35] adopts a system-oriented perspective by generating and evaluating entire test suites for real-world Java projects. It extracts project-specific configuration details, such as dependencies, Java versions, and testing frameworks, from build files and uses this information to construct context-aware prompts. The resulting tests are assessed using multiple quality indicators, including code coverage, mutation testing, and the presence of test smells, enabling a structured comparison between human-written and LLM-generated tests.

ChatUniTest [36] further advances this line of work by dynamically selecting relevant contextual information to include in prompts and by employing an iterative workflow that alternates between generation, validation, and repair. This design reduces the number of syntactic and semantic errors in generated tests and reflects a broader shift toward integrated pipelines that combine LLM generation with automated feedback mechanisms.

While these techniques demonstrate that LLMs can produce syntactically correct and high-coverage unit tests, they largely depend on execution-based validation and predefined project configurations, limiting their ability to generalize to complex, evolving systems and higher-level behaviors.

3.2.2 High-Level and End-to-End Test Generation

To address limitations of unit-level testing, several works explore automated generation of higher-level, end-to-end test scenarios for conversational agents. One line of research proposes simulating realistic user interactions through user models combined with Domain-Specific Languages (DSLs) for defining correctness rules [37]. These frameworks aim to detect functional failures, conversational breakdowns, and unmet goals by executing full dialogue flows rather than isolated responses. While effective in reducing manual testing effort, such approaches typically rely on manually defined user profiles or interaction guidelines, limiting their adaptability to evolving systems.

Coverage-driven methods further advance this direction by introducing systematic criteria for assessing and generating conversational test scenarios. Coverage-based strategies classify conversational elements, such as intents, parameters, en-

tities, vocabulary variations, and dialogue transitions, and generate tests to exercise these dimensions exhaustively [38]. Given a high-level chatbot specification, these methods can explore multiple conversational paths while controlling testing cost. However, they depend heavily on accurate and complete upfront specifications, which are often unavailable or difficult to maintain in real-world deployments [38]. Furthermore, the Chatbot Test Generator (CTG) [13] addresses the challenge of testing task-oriented chatbots built on the Dialogflow[5] platform by automating the creation of conversational test cases. Rather than generating tests from scratch, CTG builds upon existing Botium[6] test cases, treating them as seed scenarios that are systematically expanded to explore alternative dialogue paths and execution branches. By reusing and augmenting prior tests, CTG increases coverage of possible conversational flows while remaining aligned with the chatbot's predefined intents and transitions. However, this approach still depends heavily on the quality and completeness of the initial seed tests and the underlying conversational design, limiting its ability to capture unexpected user behavior or emergent interactions that arise in real deployments. This highlights the need for evaluation strategies that are less dependent on manually specified conversation structures and more closely grounded in actual usage patterns.

Other approaches focus on leveraging existing test descriptions as inputs to LLM-based execution pipelines. For example, work on executing Gherkin test specifications with collaborating LLM agents demonstrates that language models can interpret *Given–When–Then* scenarios, generate executable test code, and autonomously explore the system under test [39]. While promising results are reported for simple and moderately complex scenarios, the authors note difficulties in handling ambiguous failure cases and underspecified scenarios, which become more pronounced in complex, real-world systems.

Research such as AutoEval-ToD [40] further extends high-level evaluation by simulating user interactions and assessing task-oriented dialogue systems across multiple dimensions, including response quality and internal decision processes. Additional work evaluates the broader role of LLMs in software testing, highlighting their usefulness for test generation, fault localization, and error explanation while also identifying risks such as hallucinated outputs and overconfident assertions [41]. These studies emphasize the importance of structured inputs, cross-checking results, and combining automated generation with human validation to improve reliability. Although these frameworks provide structured evaluation pipelines, they typically rely on predefined scenarios and simulated interactions rather than data derived from actual system usage.

3.2.3 Test Oracle Generation

Automated testing critically depends on the availability of reliable test oracles, which determine whether a system behaves correctly for a given input [24]. However, explicitly specifying expected behaviour can be difficult, particularly for complex or evolving systems where requirements may be incomplete or frequently change. To address this challenge, *derived test oracles* infer expected behaviour from existing artifacts, such as execution traces, system outputs, documentation, or previous versions of an application [24]. For example, EvoSuite generates assertions by execut-

[5] https://docs.cloud.google.com/dialogflow/docs
[6] https://botium-docs.readthedocs.io/en/latest/03_testing/01_testing_conversational_ai.html

ing a system and capturing its observed behaviour, creating oracles that represent the current implementation [42]. Although effective for detecting unexpected behavioural changes, these oracles primarily check consistency with previous observations rather than correctness, meaning that existing faults may be propagated and defects introduced in later versions may remain undetected.

To improve oracle generation beyond observed behaviour, specification mining approaches extract expected properties from software artifacts such as documentation and comments [24]. However, their effectiveness depends on the availability of accurate, complete, and up-to-date specifications, which is often challenging in large and continuously evolving software systems.

3.2.4 Test Augmentation and Mutation-Based Improvement

To address coverage gaps and uncover hidden defects, several works investigate test augmentation and mutation-based strategies. EvalPlus [43], for example, enhances evaluation of LLM-generated code by expanding existing benchmarks with automatically generated test cases using both LLM-driven synthesis and mutation techniques. This approach demonstrates that augmenting test suites can significantly improve fault detection beyond static benchmarks.

In the context of conversational systems, tools such as DialTest [44] expand test coverage by deliberately altering user utterances through linguistic variations such as replacing words with synonyms, translating sentences to another language and back, or inserting additional terms. Such perturbations help uncover inputs that are more likely to trigger system errors. In a similar way, mutation-based frameworks like Charm [45] generate more diverse test inputs by applying realistic sentence-level changes, including spelling errors, reworded expressions, and transformations of numerical values. Together, these techniques aim to stress-test conversational systems by exposing them to a broader range of plausible, yet challenging, user inputs.

Mutation testing has also been adapted specifically for task-oriented chatbots. By injecting faults into chatbot designs, such as altered intents, incorrect entity handling, or modified conversation flows, and measuring mutation scores, these approaches provide quantitative assessments of test suite effectiveness [46]. However, mutation strategies are often applied independently of real execution data, limiting their ability to reflect actual user behavior.

Across unit testing, end-to-end scenario generation, oracle construction, and test augmentation, prior research demonstrates the potential of LLM-assisted testing to reduce human effort and improve coverage. Nevertheless, existing approaches either operate at limited levels of abstraction, depend on synthetic or manually specified scenarios, or struggle to ensure oracle validity in evolving systems. These limitations motivate the trace-driven methodology proposed in this thesis, which grounds test generation in real execution data, extends coverage through controlled mutation, and incorporates human validation to balance automation with reliability. Furthermore, fully automated test generation remains an open challenge. LLM-generated tests may overlook complex edge cases, fail to explore diverse execution paths, or contain syntactic and logical inconsistencies that reduce their reliability [5]. These limitations motivate the use of a lightweight human-in-the-loop validation layer, where experts review and refine generated eval criteria rather than creating

them from scratch. Such an approach preserves the efficiency and scalability of LLM-assisted test generation while improving the reliability, correctness, and practical usability of the resulting evaluation suite.

3.3 Evaluation Frameworks for Conversational Agentic Systems

Evaluating conversational agentic AI systems requires more than verifying individual responses. Such systems typically consist of multiple interdependent components, including natural language understanding, dialogue management, reasoning logic, tool invocation, and response generation. As a result, errors may arise not only from incorrect outputs but also from flawed intermediate decisions or contextual misinterpretations. To address these challenges, several evaluation frameworks and testing tools have been proposed to automate testing and improve coverage of conversational behaviors.

One of the most widely adopted industry tools is *Botium*, a state-of-the-art framework for testing conversational agents. Botium models a test case as a sequence of user–bot exchanges, enabling end-to-end testing of chatbot functionality from a conversational perspective. It supports automated test execution and basic test generation, making it suitable for regression testing across chatbot versions. Due to its practical applicability, Botium has served as a foundation for several follow-up approaches. However, Botium-based testing typically relies on manually crafted or partially templated inputs, which limits its ability to explore the full conversational input space. Furthermore, defining precise correctness conditions for chatbot responses remains challenging, often leading to brittle or overly permissive test oracles.

To address some of these limitations, *Charm* [45] extends Botium by introducing mechanisms for generating more diverse test cases through mutation-based strategies. By perturbing user utterances, Charm aims to increase coverage and expose robustness issues in conversational agents. While this improves input diversity, the approach still struggles to comprehensively cover complex conversational flows and may generate invalid or semantically incorrect test cases, particularly when predicting valid follow-up interactions becomes difficult.

Beyond test generation, evaluation-focused tools such as *BotTester* [47] emphasize assessing chatbot correctness by comparing system responses against ground-truth answers. The authors highlight the inherent difficulty of evaluating conversational agents due to their reliance on contextual understanding and application-specific logic. While BotTester proposes automated comparison strategies and outlines future directions involving similarity metrics to capture partially correct responses, its evaluation paradigm largely centers on final outputs rather than intermediate decision-making processes.

More recently, observability-oriented platforms such as *Langfuse*[7] have gained traction by providing detailed traces of LLM-based application behavior, including prompts, responses, tool calls, and execution metadata. These platforms enable developers to inspect and debug conversational agents in production environments and to monitor system behavior over time. However, despite offering rich execution data, such tools primarily support monitoring and diagnostics. They do not natively provide mechanisms for incorporating detailed human feedback beyond sim-

[7] https://langfuse.com

ple thumbs-up or thumbs-down signals to improve generated evaluations over time or systematically increase evaluation coverage.

A recent research effort that closely aligns with and inspires the approach proposed in this thesis is EvalGen [10], which leverages LLMs to automatically generate eval criteria and executable assertions while incorporating human feedback to refine their quality. Instead of requiring experts to manually define all evaluation criteria, EvalGen allows users to review, modify, and guide generated criteria, reducing the effort required to construct evaluation suites. However, the approach begins primarily from user prompts and model outputs rather than existing execution data from the target system. As a result, additional human effort may be required to iteratively shape the generated criteria to accurately reflect real system behaviour. Indeed, EvalGen's own study reports a phenomenon of "criteria drift", where users' evaluation criteria change as they grade more outputs, making it impossible to determine criteria completely before human judging [10]. Furthermore, while human feedback improves individual evaluators, the approach does not explicitly address how evaluation suites can be systematically expanded over time to increase behavioural coverage. These limitations motivate approaches that leverage real execution traces as a foundation for evaluation generation and incorporate mechanisms for continuous coverage improvement.

Taken together, existing evaluation frameworks contribute valuable tools for automated testing, regression analysis, and system inspection. However, they face recurring challenges related to limited coverage of the conversational input space, imprecise or brittle test oracles, and reliance on static or manually defined test scenarios. Most importantly, current frameworks rarely exploit real execution traces as first-class evaluation artifacts. This limitation motivates the trace-driven evaluation approach proposed in this thesis, which leverages observed system behavior to generate adaptable, structured evaluation datasets that evolve alongside the agentic application.

3.4 Summary and Positioning of the Proposed Work

In summary, prior work across observability, automated testing, and evaluation frameworks provides essential building blocks but falls short of offering a unified solution for evaluating agentic AI systems in real-world settings. Observability research highlights the value of traces but stops at diagnostics; LLM-assisted testing improves automation but risks hallucination and weak grounding; and existing frameworks struggle with iteratively incorporating human feedback to improve the adaptability and reliability of generated eval criteria.

This thesis integrates these strands by proposing a trace-driven, LLM-assisted evaluation methodology with human-in-the-loop validation. By transforming real execution traces into structured evaluation suites, the proposed approach advances the state of practice toward scalable, reliable, and context-aware evaluation of agentic AI systems.

The literature reveals three key shortcomings relevant to this work:

1. **Lack of grounding in real system behavior.** Many evaluation frameworks rely on synthetic tasks or heuristically generated conversations, neglecting real usage patterns found in production traces.

2. **Insufficient human-in-the-loop refinement of evals.** Existing approaches either require experts to manually define evaluation criteria, which is time-consuming and difficult to scale, or rely on fully automated generation that may introduce inaccuracies. A lightweight expert validation step is needed to efficiently review and refine generated eval criteria. Furthermore, feedback from experts should be incorporated iteratively to improve subsequent generations of evals, increasing their accuracy, relevance, and alignment with system requirements.
3. **Weak adaptability to system evolution.** Static benchmarks and handcrafted test suites cannot keep pace with changes in production, leading to outdated evals.

These limitations jointly motivate a trace-driven, LLM-assisted evaluation framework that (i) converts real execution traces into structured eval cases, (ii) iteratively refines the generated evals via incorporating a lightweight expert feedback step, and (iii) generates new eval cases to extend coverage. Unlike synthetic generation, the proposed approach grounds the evaluation in live system behavior, reconciles automation with expert oversight, and adapts to evolving application changes. This positions the work as an advancement toward a context-sensitive, scalable evaluation of agentic AI systems.

4 Problem Formulation

Evaluating agentic AI applications presents a unique set of challenges due to their autonomous reasoning, multi-turn interactions, and dynamic tool usage. Unlike static conversational models, agentic systems evolve continuously: their prompt structures change, tool capabilities expand, and user queries differ over time. As a result, test suites that are manually curated, which require substantial time and expert effort to create, quickly become outdated and no longer reflect how real users interact with the system. This leads to a fundamental mismatch between evaluation criteria and the actual characteristics of deployed systems, which introduces the intertwined issues: (i) the lack of evaluation cases grounded in real-world system behavior, as manually created evals fail to capture the complexity and variability of production interactions; (ii) the absence of systematic mechanisms to refine eval criteria and generation processes based on expert feedback, leading to accurate evals; and (iii) insufficient coverage of system requirements, particularly for behaviours that are underrepresented or absent from development or production traces, due to the inability to continuously expand eval sets based on up-to-date behavioural specifications.

4.1 Purpose

The purpose of this work is to develop a trace-driven evaluation pipeline that transforms real execution traces from agentic AI systems into structured evals. The pipeline should incorporate automated extraction of trace data, construction of evals based on the extracted trace data, iterative refinement of eval generation through a lightweight expert feedback loop, and systematic expansion of the eval set based on behavioural specifications to improve requirements coverage. Achieving this purpose enables continuous, context-aware evaluation that reflects real usage patterns.

4.2 Research Questions

To achieve the stated purpose, this thesis addresses the following primary research questions. These questions capture the core challenges associated with grounding, adapting, and validating evaluation data in agentic AI contexts.

1. **RQ1: How can real-world execution traces be transformed into structured eval cases for agentic AI applications?**
2. **RQ2: How can expert feedback be effectively incorporated into the eval generation process to iteratively refine and improve the quality and reliability of evaluation cases?**
3. **RQ3: How can eval suites be systematically expanded based on behavioural specifications to improve coverage of system requirements that are underrepresented or absent from existing application traces?**

4.3 Motivation

The rapid adoption of agentic AI systems in real-world applications has created an urgent need for robust and reliable evaluation methods. These systems operate

autonomously, engaging in multi-turn interactions, invoking external tools, and producing outputs that are highly dynamic and context-dependent. As a result, their behavior evolves continuously over time, driven by changes in prompts, tool integrations, and user interactions. Traditional evaluation approaches based on synthetic examples, handcrafted prompts, or static benchmarks are inadequate in this setting because they fail to capture the complexity, variability, and real usage patterns observed in production environments.

A key limitation of existing methods is the disconnect between evaluation data and actual system behavior. Manually curated test cases quickly become outdated and do not reflect how users interact with deployed systems. At the same time, relying solely on LLM-based evaluation introduces challenges such as hallucination, bias, and inconsistent judgment. While end-to-end human evaluation provides higher reliability, it is time-consuming, expensive, and impractical to scale, particularly in scenarios involving continuous deployment and rapid system iteration. Furthermore, current approaches lack mechanisms for systematically improving evaluation quality over time. In particular, there is limited support for incorporating expert feedback into an iterative refinement process, allowing generated evaluation criteria and subsequent evaluation cases to progressively improve in quality and accuracy. Similarly, the literature provides no structured approach for systematically expanding evaluation datasets within the boundaries of defined system specifications to improve behavioural coverage. As a result, evaluation pipelines remain largely static, making them less effective at identifying emerging failure modes, newly introduced behaviours, and edge cases as the system evolves.

These limitations highlight the need for a scalable and adaptive evaluation framework that is grounded in real system behavior while remaining flexible and extensible. This work is motivated by the idea of leveraging real execution traces as the foundation for evaluation, enabling the automatic generation of test cases that reflect actual application usage. By combining trace-driven eval generation with a lightweight human-in-the-loop mechanism for iterative refinement, the framework enables continuous improvement in eval quality. Additionally, systematic expansion of eval sets based on the behavioural specifications of the application allows for broader coverage of requirements and improved detection of previously untested user flows. By grounding evaluations in real-world interactions while supporting iterative refinement and coverage expansion, the proposed approach enables scalable, reliable, and context-aware assessment of evolving agentic AI systems.

4.4 Goals

The primary goal of this research is to design, implement, and evaluate an LLM-assisted framework capable of generating structured evals that can evolve with changes in system behaviour and requirements over time. The framework aims to ensure that evaluations remain grounded in real system behavior while improving in quality and coverage over time. The specific objectives are as follows:

1. **Leverage real-world execution traces**: Utilize detailed execution traces from agentic applications to capture authentic system behavior, and transform these traces into structured evaluation cases that reflect real user interactions.

2. **Incorporate expert feedback for iterative refinement**: Introduce a lightweight

human-in-the-loop mechanism to incorporate expert feedback in refining eval generation, improving the accuracy of evals while minimizing manual effort.

3. **Systematically expand evaluation coverage**: Enable the controlled generation of new evals within the defined application specifications to improve coverage of behavioural requirements.

5 Method

This section presents the methodology adopted in this thesis to answer the research questions. It first introduces the chosen research approach, followed by the case study context, providing the necessary background for the study. The section then describes the methods used to address each research question, including trace-based eval construction, expert feedback for eval refinement, and specification-based eval generation to improve behavioural coverage.

5.1 Overall Research Approach

This thesis adopts the Design Science Research (DSR) paradigm, which is widely used in applied computer science and software engineering research [48]. DSR is well suited to this work because it focuses on designing and developing an artifact to address a clearly identified practical problem and systematically evaluating its effectiveness. In this thesis, the identified problem is the lack of a systematic, trace-driven approach for evaluating agentic AI applications. Accordingly, the research follows a design-oriented process in which an evaluation framework is developed, refined, and empirically evaluated to determine how effectively it addresses the identified challenges.

The artifact produced in this work is an evaluation pipeline that transforms real execution traces from a deployed agentic AI system into structured eval sets and systematically expands them based on behavioral coverage criteria. The three research questions map directly onto the three design goals of the artifact, and each is addressed by a distinct sub-method: a trace processing pipeline (RQ1), a human-in-the-loop refinement mechanism (RQ2), and a specification-coverage-driven eval generation mechanism (RQ3).

5.2 Case Study Context

The artifact is designed and evaluated within a single real-world deployment: an agentic AI analytics chatbot that serves business users at client organisations. The chatbot accepts natural-language queries about business metrics and responds by configuring data retrieval, performing root-cause analysis, or generating comparative analyses. Representative query types include:

- Data retrieval and trend queries – requests to retrieve and display metric values over a specified time period, such as "*show me sales figures for Q3 2025.*"
- Root-cause analysis queries – requests to explain an observed change in a metric, such as "*why did sales drop on the 21st of January?*" or "*what caused the spike in visitor numbers last week?*"
- Filtering and segmentation queries – requests to restrict the analysis to a specific dimension or group, such as "*show revenue for the North America region only*" or "*break this down by product category*"
- Comparative analysis queries – requests to compare metric values across dimensions, time periods, or scenarios, such as "*compare sales performance between Asia and Europe in Q2*" or "*how does this month's conversion rate compare to last quarter?*"

- Configuration and follow-up queries – short, context-dependent follow-ups that modify the current view, such as "*group by subcategory*", "*add discount percentage to this*", or "*now show the same chart for last year*"
- Forecasting and recommendation queries – requests to estimate future metric values based on historical data, such as "*what would sales look like next quarter?*" or "*how much revenue are we likely to generate next month?*"

The chatbot is built using a multi-agent architecture with specialised sub-agents, including triage, analysis, RCA, comparison, and recommendation, each responsible for a distinct class of the query types described above. All user interactions are captured by the Langfuse observability platform, which records every conversation as a structured trace containing the full input context, intermediate agent decisions, tool calls, and final response. These traces form the primary data source for the evaluation pipeline developed in this work. Furthermore, Promptfoo was selected as the tool for running the generated evals against the actual application, with support for multi-turn conversations, configurable LLM-as-judge providers, and structured outputs suitable for programmatic analysis.

5.3 Method for Goal 1 – Trace-Driven Eval Construction

Goal: Utilize detailed execution traces from agentic applications to capture authentic system behavior, and transform these traces into structured and reusable evals that reflect real user interactions.

Method: Pipeline implementation with LLM-as-judge rubric assessment.

The chosen method is the design and implementation of a multi-stage data processing pipeline that ingests raw development and production traces and produces structured eval cases. The pipeline consists of five sequential stages: trace collection, filtering, session grouping, conversation replay, and eval construction.

1. Trace collection uses the Langfuse Python SDK to fetch production conversation traces within a configurable time window.
2. Filtering applies declarative rules (defined in a YAML configuration file) to remove sessions that are structurally invalid, duplicated, or otherwise unsuitable as test cases.
3. Session grouping reconstructs multi-turn conversations from flat trace records.
4. Conversation replay re-runs each session against the live system to obtain fresh, detailed input/output traces, ensuring that the rubric is generated against current system behavior rather than potentially stale historical responses.
5. The eval construction stage submits each trace to an LLM to generate the rubric and wraps the result in the Promptfoo eval format.

The core technical challenge in eval construction is generating evaluation criteria, which form the rubric that determines whether a given response is correct. Two approaches were considered: (i) manual authoring of criteria by a domain expert, and (ii) automated generation using a large language model. Manual authoring

does not scale and introduces bottlenecks as the system evolves. LLM-based rubric generation, a technique that has gained significant empirical support in recent work on automated evaluation [4, 10], was selected because it can operate at scale, produce multi-dimensional criteria, and condition its judgements on the traces of actual system responses.

Inspired by recent work on creating evals as reported in [49], the process began by examining interaction traces collected through the observability platform, Langfuse, to identify recurring errors and failure patterns in agent–user interactions. This initial analysis required a one-time manual effort to document and understand what went wrong at different stages of the interactions. The resulting error cases were then grouped into broader behavioural categories using a combination of manual judgement and LLM-assisted analysis. For example, cases where the system retrieved data for the wrong date range and cases where it misapplied a rolling-window calculation were initially labelled separately but were eventually unified under the *Time Periods* category, since both reflect a failure in temporal scope resolution. Similarly, cases where the system ignored a user-specified dimension filter and cases where it applied the wrong filter value were unified under *Filtering*. This exploratory process converged on seven categories that collectively cover the principal dimensions along which the system's responses can be judged correct or incorrect for this application domain:

Category	**What it assesses**
Metric Selection	Whether the system identified the correct business metric from the user's query
Filtering	Whether the correct dimension filters were applied
Grouping	Whether the data was grouped by the correct dimension
Time Periods	Whether the correct time range was selected
Chart Configuration	Whether the appropriate visualisation type was chosen
Numerical Accuracy	Whether any quoted figures or computations are correct
Forecast Validation	Whether the system selects forecasting or recommendation pathway

Table 1: Rubric categories for LLM-as-judge evaluation, derived from qualitative trace analysis.

It is important to note that these seven categories are specific to the analytics chatbot evaluated in this work and are not claimed to be universally applicable. A chatbot in a different domain, such as a code generation assistant or a customer support agent, would exhibit different failure patterns and therefore require a different set of rubric categories.

5.4 Method for Goal 2 – Human-in-the-Loop Refinement

Goal: Introduce a lightweight human-in-the-loop mechanism to incorporate expert feedback in refining eval generation, improving the accuracy of evals while minimizing manual effort.

Method: Human-in-the-loop iterative refinement process.

A purely automated pipeline cannot guarantee that generated evals are correct without some grounding in domain knowledge. LLM-generated rubrics may produce

criteria that are technically plausible but factually wrong, for example, by asserting that the system should select a metric that was not intended by the user or by generating criteria based on a misinterpretation of the conversation context. Expert feedback is therefore necessary to maintain the validity of the generated evaluations. Moreover, evaluation criteria may need to evolve as new system behaviours and failure cases emerge, creating a risk of criteria drift [10] if the evaluation suite is not periodically reviewed and refined. Iteratively incorporating expert feedback provides a mechanism for correcting such deviations and improving the quality and relevance of subsequent evaluation criteria.

The chosen method adopts a human-in-the-loop (HITL) refinement process, wherein a domain expert evaluates the generated evaluation cases through a purpose-built web interface. The expert provides structured feedback in the form of three possible decisions: **accept**, indicating that the evaluation is correct and can be retained; **reject**, indicating that the evaluation is incorrect or misleading; and **review**, indicating that the evaluation requires further inspection, potentially due to underlying issues in the AI chatbot backend logic. This three-way classification is more informative than a binary accept/reject scheme because it distinguishes between clearly wrong evals and uncertain ones, enabling the feedback loop to handle them differently.

The key design decision in this component is to use rejection feedback constructively rather than simply discarding rejected evals. The method introduces a *feedback classification* step to determine whether the rejection is caused by an incorrect generated criterion or an incorrect system response. An LLM-based trace validator assesses whether the expert's feedback is consistent with the system behaviour recorded in the trace. If the trace supports the expert's feedback, the rejection is attributed to an error in the generated criterion, and the feedback is used to refine the eval-generation prompt. If the trace does not support the expert's feedback, the system response is considered incorrect, and the individual assertion is corrected or refined based on the expert's feedback. This distinction matters because conflating the two cases, such as adjusting criteria to reflect the system behaviour, when the system itself is at fault, would silently lower the standard of the eval set and mask genuine system failures behind adjusted expectations. The trace validation step prevents this by asking the LLM to assess, given the full trace content and the expert's comment, whether the comment describes something the system actually did or something it should have done but did not. criteria.

This lightweight human-in-the-loop design is justified by the principle that expert knowledge should correct systematic errors in automated processes rather than being applied to each eval case from scratch. The expert's time is spent on judgment, including accept, reject, or review decisions, rather than writing criteria from scratch. Criteria generation is delegated to the LLM entirely.

5.5 Method for Goal 3 – Specification-Coverage-Driven Eval Generation

Goal: Enable the controlled generation of new evals based on behavioural specifications to improve coverage of requirements.

Method: LLM-based behavioural specification extraction from agent prompts, coverage matrix construction, augmented question generation, and new eval construction.

The trace-driven pipeline in RQ1 is inherently bounded by what appears in development or production: edge cases or newly introduced capabilities could be underrepresented or absent from the eval set if they have not yet been exercised by the developers or actual end users of the application. A systematic mechanism is needed to detect these gaps and generate evals that fill them.

The method proceeds in three steps.

- **Behavioural specification extraction**: The application's agent prompts are treated as the authoritative behavioral specification of the system. An LLM is prompted to extract a set of requirement specifications from these files, with each specification representing a concise, atomic, and testable description of a functional capability the system must demonstrate. This approach is justified because the system prompts are the single most authoritative source of intended behavior: they define what the agents are instructed to do and therefore constitute a natural specification language. Extracting specs automatically rather than writing them by hand removes a manual bottleneck and ensures the spec set stays synchronized with the actual code.
- **Coverage analysis**: A specification is considered covered when at least one conversation session in the eval suite exercises it. A session exercises a specification when the required conversational context and the user action that triggers the specification are present. An LLM then judges whether the augmented session satisfies the intended specification. The unit of coverage is the session, not the individual eval, because many specifications describe behaviour that only arises across turns – a metric switch relative to a preceding turn, a drill-down from an established grouping, a language change mid-conversation. A single-turn eval cannot exercise these, so a per-eval coverage metric would systematically under-report multi-turn behaviours. Coverage is binary per specification, and the denominator is the full extracted specification set. The coverage percentage is the proportion of specifications covered by at least one session. Coverage is independent of correctness. A specification counts as covered when the behaviour it describes was exercised and the agent's response to it was recorded, whether or not that response was correct. A session that triggers a language switch and receives the wrong language still covers the language-switch specification; the eval derived from that session is what determines whether the behaviour was correct. Keeping the two apart is what allows coverage to be read as a measure of the suite's reach rather than of the application's quality.
- **Augmented question generation**: For each uncovered specification, an LLM generates new augmented questions, either as standalone conversation sessions

or as extensions of existing real-world conversation sessions. Generated conversations are then run through the same pipeline stages as real traces (replay, rubric generation, expert review), so their eval criteria are derived from actual system responses rather than being fabricated. Fabricating expected outputs without running the system would risk introducing criteria that do not reflect actual system behavior, which would make the evals misleading. By running the augmented query against the live system and generating criteria from its real response, the pipeline ensures that every eval is grounded in observed system behavior.

6 Ethical and Societal Considerations

6.1 Research Ethics

This work does not involve human subjects research in the conventional sense. There are no interviews, surveys, or controlled experiments with recruited participants. The primary data source is execution traces produced by an AI analytics chatbot that is deployed as a commercial product and used by external client organisations. The evaluation pipeline itself, including trace processing, rubric generation, and expert review, is operated internally within the developing organisation.

The pipeline ingests conversation traces collected through Langfuse, a production observability platform. These traces contain queries submitted by end users and the chatbot's corresponding responses. The chatbot is used by client organisations to query aggregated business metrics, such as sales figures, revenue trends, and regional comparisons. The interactions are therefore analytical and concern organisation-level business data rather than personally identifiable information (PII), such as contact details, health records, or individual-level personal data. Access to these traces is provided with the mutual agreement of the client organisation and is used solely to support the improvement of the chatbot's correctness and reliability.

Although the traces were not expected to contain PII, the user queries are free-text inputs and could potentially include incidental personal information. No systematic anonymisation, PII scanning, or redaction was performed on the traces before they were processed by the pipeline. This is recognised as a limitation of the study, and future implementations should include appropriate privacy screening and redaction mechanisms before processing trace data, particularly when applying the framework to systems operating in more sensitive domains.

No trace data is shared outside the developing organisation, and the eval artifacts produced remain entirely internal. This work assumes that such agreements permit the use of interaction data for the purpose of product quality assurance and system testing, which is standard practice in commercial software development.

The lightweight human-in-the-loop review process involves domain experts within the developing organisation evaluating the quality of generated eval cases. Their judgements, including accept, reject, or review decisions with optional comments, are used solely to improve the evaluation pipeline and are retained as structured metadata within the pipeline's internal artifact files. This activity constitutes standard professional quality assurance work and does not raise research ethics concerns.

This work does not involve research activities that require formal ethics board approval. The trace data are used within standard software quality assurance activities, and all evaluation artifacts remain within the developing organisation. However, the methodology may be applied to systems that process sensitive personal data, such as health or financial information. In such cases, appropriate data protection and privacy assessments, including a Data Protection Impact Assessment (DPIA)[8] where required, should be conducted before production traces are used for evaluation generation.

[8] https://gdpr.eu/data-protection-impact-assessment-template/

6.2 Societal Considerations

The pipeline helps address the economic cost of maintaining eval sets for AI systems. Manual curation of test cases is labour-intensive and does not scale as systems evolve. By automating trace collection, rubric generation, and coverage-driven expansion, the pipeline reduces the ongoing engineering effort required to sustain a meaningful eval set. For organizations that deploy agentic AI systems in production, this translates to a reduced maintenance effort and shorter time to detect regressions, although neither effect was measured in this work.

Agentic AI systems used in business contexts carry the risk of producing incorrect or misleading outputs that influence operational decisions – for example, incorrectly attributing a revenue decline to the wrong cause. Systematic evaluation is a prerequisite for responsible deployment of such systems. By grounding evals in real usage patterns and tying coverage directly to the system's behavioural specification, this work contributes to a more disciplined approach to AI quality assurance. This is aligned with broader societal goals around AI trustworthiness, transparency, and accountability, as articulated in frameworks such as the EU AI Act[9].

The pipeline makes repeated calls to large language models for requirement specification extraction, coverage mapping, eval generation, and rubric construction. LLM inference carries a non-trivial energy cost, and this should be acknowledged. The design of the pipeline includes several mitigations: coverage-driven generation avoids producing redundant evals (only uncovered specs trigger new generation), and the pipeline is not run continuously but on-demand when the eval set needs to be refreshed. Nevertheless, scaling this approach to very large systems or high-frequency refresh cycles would amplify the carbon footprint of the evaluation process, and future work should consider energy-efficient model choices or batching strategies.

A further societal consideration is the potential overreliance on automatically generated evaluations. LLM-generated criteria may incorrectly interpret observed system behaviour or fail to capture the intended behaviour of the application. For this reason, the generated evaluations are not treated as automatically valid. Human expert review is used to assess the generated criteria and provide feedback for refinement. This oversight helps distinguish observed behaviour from intended behaviour and reduces the risk of incorporating incorrect evaluation criteria into the evaluation suite. For higher-risk applications, additional domain-expert review and independent validation would be necessary.

The traces processed by the pipeline are generated within the organisation that owns and operates the chatbot. The eval artifacts produced, which are structured JSON files containing query–response pairs and rubric criteria, contain business metric data that is subject to the organisation's own data governance policies. Provided those policies are respected, no external legal obligations (such as GDPR data subject rights) are triggered, since the data does not pertain to natural persons.

The pipeline is designed to be application-agnostic: adapting it to a different system requires only a configuration change pointing to a different agent directory and Langfuse workspace. This generalisability is a design goal, but it also means the approach could be applied to systems where the ethical considerations are less straightforward, such as AI systems that process personal health or financial data. Practitioners applying this methodology in such contexts must conduct their own

[9] https://artificialintelligenceact.eu/

assessment of data handling requirements before using production traces as the basis for eval generation.

7 Design and Implementation

This section describes the design and implementation of the proposed solution. The overall architecture is organized as a three-phase pipeline that progressively constructs, improves, and expands the evaluation suite. The complete source code, configuration files, and installation instructions are publicly available on GitHub.

7.1 Architecture Overview

The architecture consists of three sequential pipeline phases, shown in Figure 2, each corresponding to a research question. All LLM-based pipeline components, including eval generation, trace validation, rubric construction, specification extraction, and coverage analysis, used the Azure OpenAI GPT-5.4-mini model.

1. **Phase 1 – Trace-driven eval construction** takes raw production traces from Langfuse and transforms them into a structured eval set with LLM-generated rubric criteria. This phase addresses **RQ1**.
2. **Phase 2 – Human-in-the-loop refinement** introduces a domain expert review step that accepts, rejects, or flags evals for correction, and uses the resulting feedback to improve the correctness of both individual eval criteria and the rubric generation prompt. This phase addresses **RQ2**.
3. **Phase 3 – Specification-coverage-driven generation** identifies behavioral gaps in the accepted eval set by mapping existing evals against automatically extracted system specifications. For specifications that remain uncovered, the framework generates augmented question sessions, which are then used to create new evals and expand behavioral coverage. This phase addresses **RQ3**.

Each phase produces a set of persistent artifact files stored in a versioned directory structure (`artifacts/work/vN/`). A canonical folder (`artifacts/canonical/`) holds the current ground-truth dataset. The versioned structure allows any phase to be re-run without overwriting prior results. The JSON example snippets presented in this chapter are actual pipeline outputs from ZED, extracted from these artifact directories, with UUIDs anonymized and output data redacted.

7.2 Phase 1 – Trace-Driven Eval Construction

Manually designed evals can become outdated, while LLM-based evals may suffer from hallucinations, bias, and inconsistent judgments. Phase 1 addresses these issues by using real application traces to construct structured evals grounded in real-world user-agent behaviour.

7.2.1 Trace Collection

As illustrated in Figure 3, the first stage fetches conversation traces from Langfuse using the Langfuse Python SDK. Traces are retrieved within a configurable time window specified as UTC timestamps or based on provided session IDs from Langfuse. The collection operates in QnA mode, which retrieves lightweight trace summaries containing the user query, the final response, session identifiers, and metadata fields such as customer ID, language, and timezone. Full intermediate trace details are

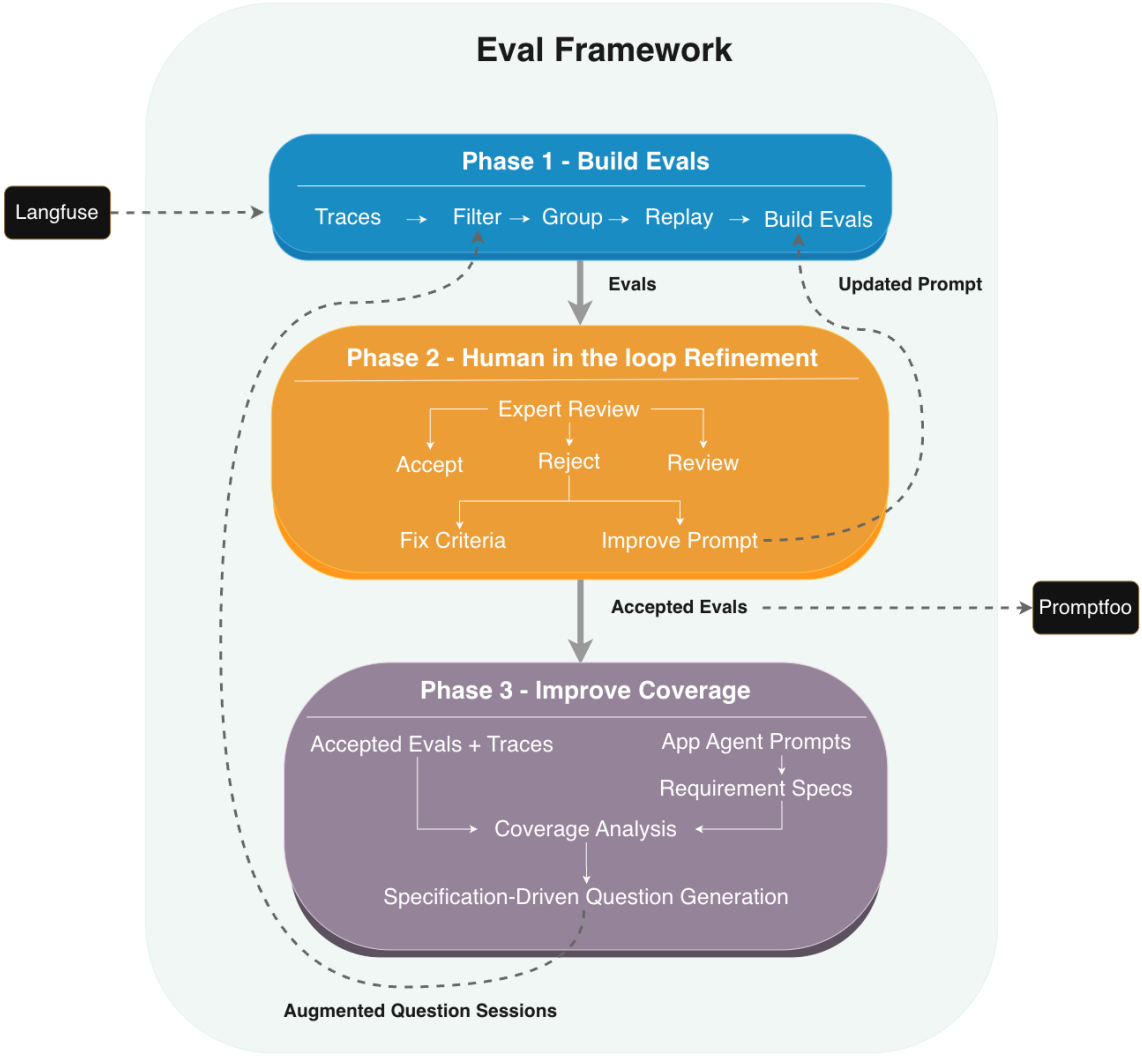

Figure 2: Overview of the evaluation framework.

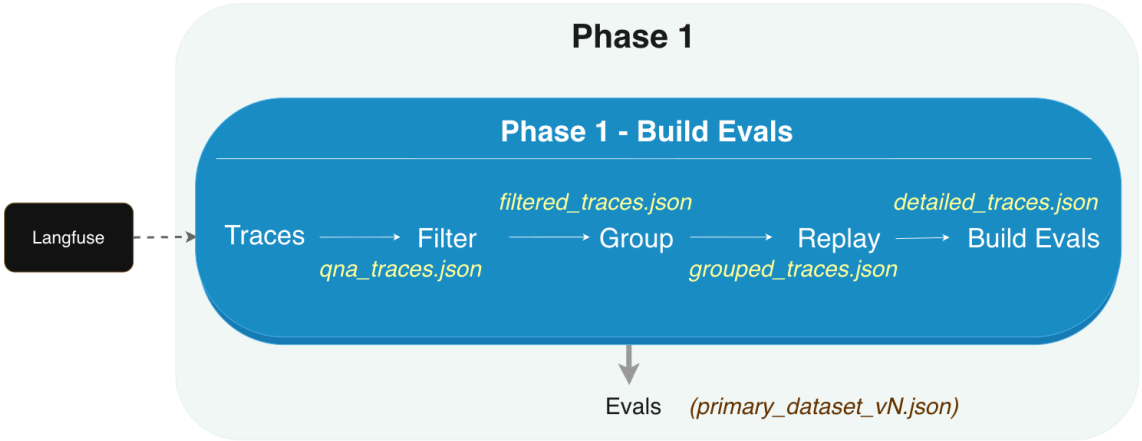

Figure 3: Phase 1 – trace collection, filtering, session grouping, replay, and rubric-based eval construction.

```
[
   {
      "id": "<uuid>",
      "sessionId": "<uuid>",
      "input": {
         "message": "show sales for August 2025",
         "language": "English (US)",
         "time_zone": "Europe/Helsinki",
         "context": {},
         "rca_configuration": "null",
         "recommendations_configuration": "null",
         "currency": "EUR"
      },
      "metadata": {
         "type": "null",
         "customer_id": "<uuid>"
      }
   }
]
```


Figure 4: A sample raw conversation session fetched from Langfuse and recorded in `qna_traces.json`.

not fetched at this stage to minimize API payload size; detailed traces are fetched selectively in Stage 4. The output of the trace collection stage is `qna_traces.json`, a flat list of trace records, as shown in Figure 4.

7.2.2 Trace Filtering and Deduplication

This session-level filtering is important because multi-turn evals require a complete conversation history; a session with a missing or malformed early turn cannot be replayed correctly. Filtering rules include deduplication based on session content, where each ordered sequence of user questions and system responses is hashed to identify duplicate sessions. Sessions are also excluded when the user query is empty, or the corresponding system response is missing. The output of this stage is `filtered_traces.json`, which follows the same format as the traces shown in Figure 4.

7.2.3 Session Grouping

Filtered trace records are reorganized from a flat list into a hierarchical session structure. Traces sharing the same sessionId are grouped together to reconstruct the original conversation sequence. This grouped structure is the input required for the conversation replay stage. The output is `grouped_traces.json`, with a structure similar to that shown in Figure 5.

7.2.4 Conversation Replay

The conversation replay stage re-runs each grouped session against the live chat application by sending each message turn-by-turn through the application's chat API. For each turn, the full system response and intermediate trace data are captured,

```
[
   {
      "sessionId":  "<uuid>",
      "input":  {
         "messages":  [
            {
               "message":  "show sales for August 2025",
               "time_zone":  "Europe/Helsinki",
               "currency":  "EUR",
               "context":  {},
               "recommendations_configuration":  "null",
               "rca_configuration":  "null",
               "language":  "English (US)"
            }
         ],
      },
      "metadata":  {
         "type":  "chat_trace",
         "customer_id":  "<uuid>"
      }
   }
]
```


Figure 5: A sample grouped conversation session recorded in `grouped_traces.json`.

including the agent that handled the turn, the tool calls made, and the final configuration produced. The replay is necessary rather than using the original historical responses because: (i) the system's prompts and agent logic may have changed since the original trace was recorded, and the eval should reflect current system behavior; (ii) the original traces may not contain sufficient intermediate detail (tool call inputs/outputs, agent handoff decisions) for rubric generation. The output of this stage is `detailed_traces.json`, a dictionary keyed by trace ID as shown in Figure 6, where each entry contains the full input/output record for one conversation turn including the complete context object.

7.2.5 Rubric-Based Eval Construction

The eval construction stage generates the eval suite. For each detailed trace in Figure 6, this stage submits the trace content to an LLM with a structured prompt in Appendix B (the `build_dataset_v1` prompt) that instructs it to produce evaluation criteria across seven rubric categories. The rubric categories were not defined a priori but emerged from a preliminary qualitative analysis of the development and production traces, as described in Section 5.3. The results are a set of assertions, where each assertion is a natural-language statement of expected behavior that a Promptfoo LLM-as-judge can evaluate against the system's actual response. This eval construction stage wraps the query, context, and resultant assertions into the format of a Promptfoo eval case and appends the result to the versioned dataset file `primary_dataset_v1.json`. A representative eval, including its query, metadata, and LLM-based rubric assertions, is shown in Figure 7.

```
{
   "id": "<trace_id>",
   "name": "generate_answer_stream",
   "input": {
      "message": "show sales for August 2025",
      "time_zone": "Europe/Helsinki",
      "context": {},
      "language": "English (US)",
      "currency": "EUR",
      "rca_configuration": "null",
      "recommendations_configuration": "null"
   },
   "output": {
      "filterConfiguration": {
         "widgets": {
            "grouping": "day",
            "metrics": "Sales (VAT)"
         },
         "time": {
            "selection": {
               "start_time": "2025-08-01",
               "end_time": "2025-08-31"
            },
            "comparisons": []
         }
      },
      "charts": "column",
      "message": "August sales hit EUR 507,195 and the ...",
      "rca_results": []
   },
   "observations": [
      {
         "type": "SPAN",
         "name": "fetch_data_tool",
         "output": {
            "data": [
               {
                  "name": ["2025-08-01"],
                  "sales": "32151"
               },
               ...
            ]
         }
      },
      {
         "type": "SPAN",
         "name": "update_filter_configuration_tool",
         "output": "Configuration updated: 2025-08-01 to 2025-08-31"
      }
   ]
}
```


Figure 6: A sample conversation trace recorded in `detailed_traces.json`.

```
{
   "vars":  {
      "prompt":  "show sales for August 2025"
   },
   "assert":  [
      {
         "type":  "llm-rubric",
         "value":  "### Metric Selection\n- Sales (VAT) should be selected as the
metric."
      },
      {
         "type":  "llm-rubric",
         "value":  "### Filtering\n- No overall filtering is applied."
      },
      {
         "type":  "llm-rubric",
         "value":  "### Grouping\n- Day should be applied as the grouping at the
global level."
      },
      {
         "type":  "llm-rubric",
         "value":  "### Time Periods\n- The current period should be from 2025-08-01
to 2025-08-31."
      },
      {
         "type":  "llm-rubric",
         "value":  "### Chart Configuration\n- The chart should be a column
chart.\n-"
               "The chart should display Sales (VAT) by Day."
      },
      {
         "type":  "llm-rubric",
         "value":  "### Numerical Accuracy\n- The response should identify August 1
as the peak day for sales.\n-"
               "The observation data should show that August 17 has the lowest sales
value.\n-"
               "The observation data should show that August 1 is the highest sales
day and August 17 is the lowest sales day.\n-"
               "Any claim in the response about the monthly sales total should be
consistent with the observation data."
      }
   ]
   "metadata":  {
      "customer_id":  "<uuid>",
      "is_new_chat":  "1",
      "chat_order":  "1",
      "type":  "chat_trace",
      "language":  "English (US)",
      "currency":  "EUR",
      "time_zone":  "Europe/Helsinki",
      "context":  {},
      "rca_configuration":  "null",
      "recommendations_configuration":  "null"
   }
}
```

Figure 7: A sample eval with LLM-rubric assertions for execution using Promptfoo

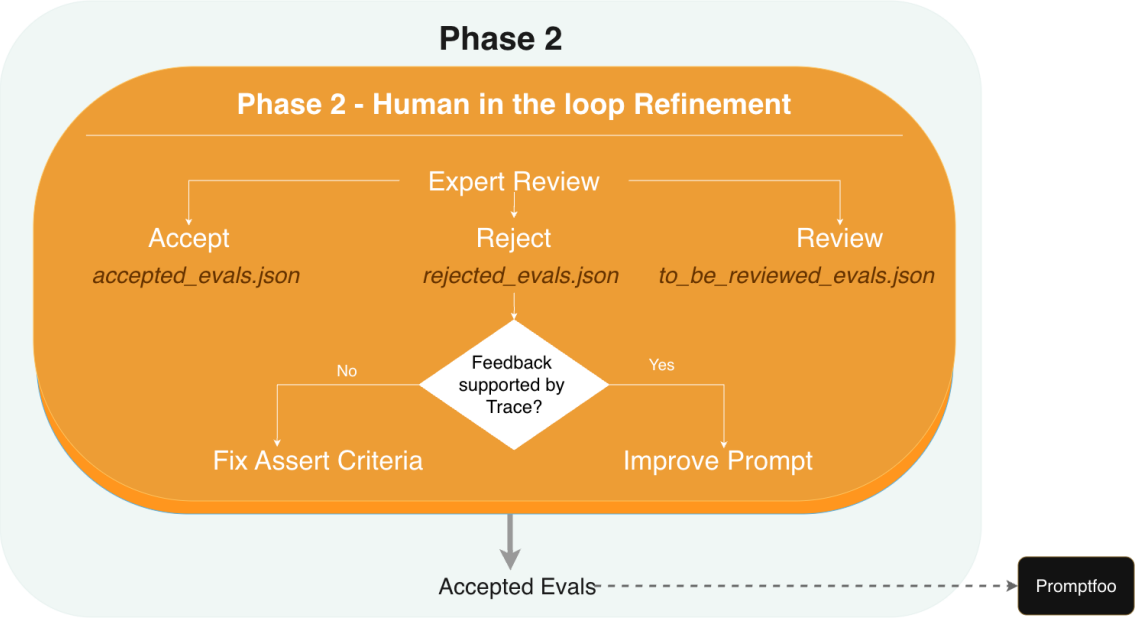

Figure 8: Phase 2 – Generated evals are reviewed by an expert, whose feedback is used to refine evaluation criteria/prompt and improve subsequent eval generation.

7.3 Phase 2 – Human-in-the-Loop Refinement

Since evaluation criteria require an understanding of the application domain, relying entirely on automated generation can result in criteria that appear reasonable but do not accurately reflect the intended system behaviour. To address this limitation, Phase 2 introduces a lightweight human-in-the-loop process in which experts review and refine the generated criteria, improving their accuracy while keeping the required effort low.

7.3.1 Expert Review

As shown in Figure 8, the generated eval set is presented to a human expert. Each eval is displayed with the original user query, the system's response, and the LLM-generated assertions. The expert assigns one of three decisions to each eval:

- **Accept** – the eval appears to be correct and should be retained as a permanent eval case
- **Reject** – the eval is incorrect, and the feedback will be used in Section 7.3.2 to either improve the overall eval generation prompt or to refine the criteria of the corresponding incorrect eval.
- **Review** – the eval is incorrect; potentially due to an underlying bug in the AI chatbot backend logic itself.

As a result, accepted evals are written to `accepted_evals.json` and merged into the canonical folder. Evals marked for review are written to a separate file named `to_be_reviewed_evals.json`, and will be used as a source to identify the bugs in the application. Rejected evals are written to `rejected_evals.json` and held for the feedback processing step described below.

7.3.2 Feedback Classification via Trace Validation

When an expert rejects an eval, the pipeline must determine whether the rejection indicates an error in the generated eval criteria or an incorrect system behaviour. To make this distinction, the expert feedback is validated against the corresponding execution trace. The result of this validation determines whether the pipeline corrects the individual eval or uses the feedback to improve the eval-generation prompt. For each rejected eval, the pipeline submits LLM-summarised trace details and the expert's comment to an LLM trace validator. The validator is prompted to determine whether the expert's comment is consistent with what the system actually produced in the trace. Thus, the pipeline uses trace validation to determine whether expert feedback should improve the individual assertion or the underlying eval-generation process:

- **Improve Prompt:** The system behaved as expected, but the generated criterion was incorrect. The rejected eval is used to refine the generation prompt and improve future criteria.
- **Fix Assertion Criteria:** The system behaved incorrectly, and the generated criterion was also incorrect. The individual assertion is corrected or refined based on the expert's feedback.

If the system's response supports the expert's observation, the trace is considered correct, and the problem is attributed to the eval generation prompt. For example, for the query "*show sales for August 2025*", an expert may comment that "*Sales (VAT), not Revenue, should be selected.*" If the trace shows that the system correctly selected *Sales (VAT)*, the generated assertion expecting *Revenue* is incorrect. The case is therefore added to the prompt-correction set, and similar cases are used to improve the eval generation prompt.

Alternatively, if the trace does not support the expert's feedback, the trace is considered incorrect, and the problem is attributed to the system response. For example, if the same expert comment is provided but the trace shows that the system selected *Revenue* instead of *Sales (VAT)*, the assertion should still specify *Sales (VAT)* as the expected metric, regardless of the system's incorrect response. In this case, the rejected assertion is treated as an *eval correction* and is updated based on the expert feedback, without modifying the generation prompt.

7.4 Phase 3: Specification-Coverage-Driven Augmented Question Generation

Real user conversations often concentrate around common user-agent behavioural workflows, leaving edge cases and complex multi-turn interactions under-represented. To address this limitation, Phase 3 introduces a specification-coverage-driven pipeline that systematically expands the eval suite by generating augmented conversations targeting under-represented behavioural flows.

This phase consists of three stages: (i) extracting behavioural specifications from the agent's system prompts, (ii) analysing coverage of these specifications in the existing eval suite, and (iii) generating augmented conversations for uncovered behavioural flows. Unlike generic synthetic augmentation approaches, the proposed

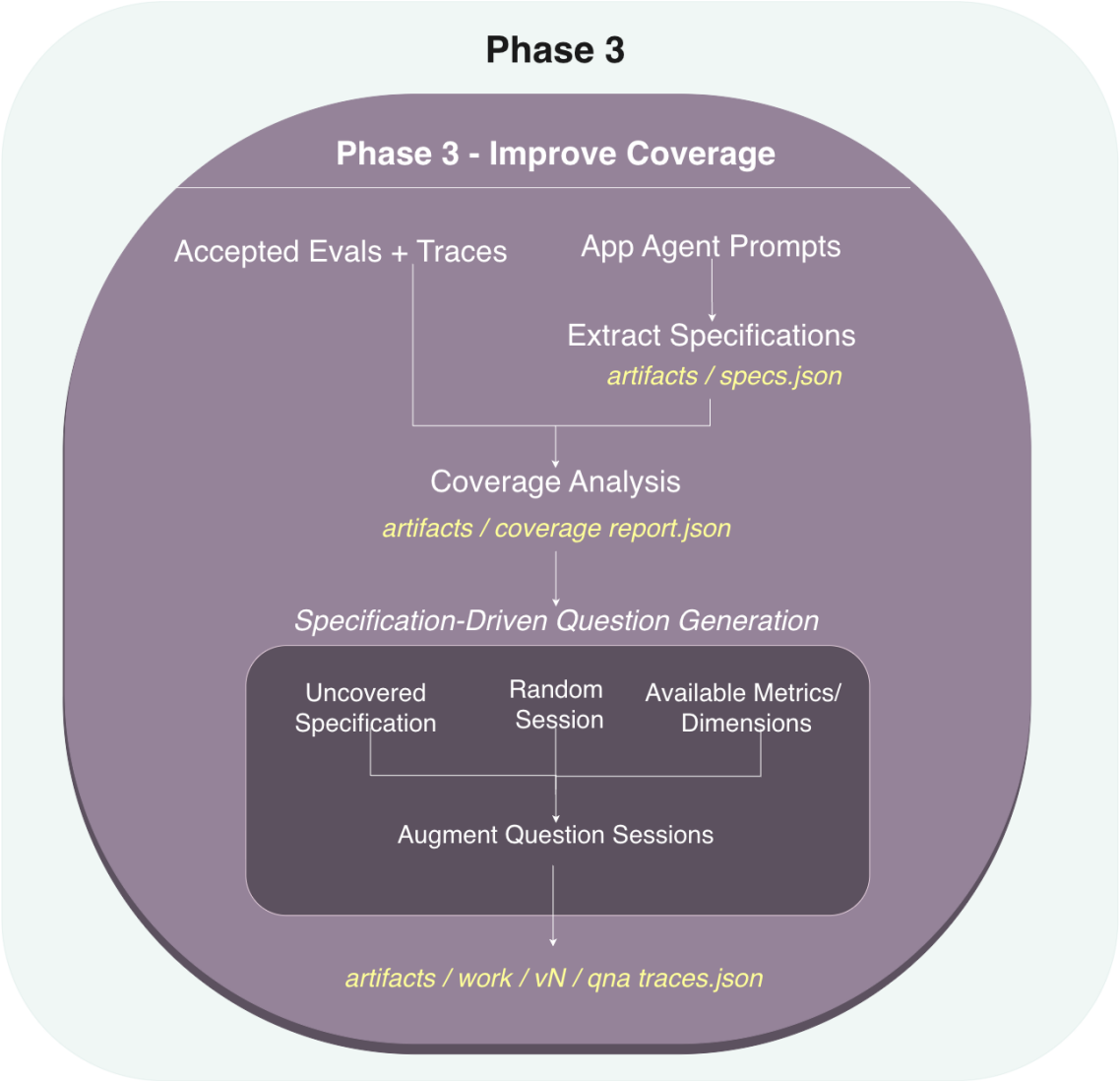

Figure 9: Phase 3 – Behavioural specifications are extracted from agent prompts, mapped to the existing eval suite to identify coverage gaps, and used to generate augmented conversations for uncovered behaviours.

```
{
   "id": "spec_032",
   "source_prompt": "conversational_agent_v2::CONVERSATIONAL_INSTRUCTIONS",
   "category": "edge_case",
   "description": "When the user explicitly asks to use another language (e.g.,
"reply in Spanish" or "can you answer in French?"), the agent must switch to that
requested language instead of staying in the default preferred language.",
   "scenario": "Language switch on request"
}
```

Figure 10: Example evaluation specification entry in specs.json file, representing a language-switch edge case.

method does not generate evals from hypothetical behaviours alone. Instead, it augments existing conversation sessions, executes the augmented sessions against the live application, and uses the resulting execution traces as the basis for generating new evals.

7.4.1 Behavioural Specification Extraction

In the ZED application, which served as the case study for this thesis, no single source of truth or formal requirements specification documentation was available to describe how the agent should behave across different tasks. Therefore, the agent's live system prompts were used as the primary source for defining its intended behaviour. The relevant source files were parsed to identify the prompts used by the application's LLM agents, and an LLM was then used to extract *atomic behavioural specifications* from each prompt section in the following form: *"When the user [action/context], the agent must [observable response]."*.

As shown in Figure 10, specifications describe only user-observable atomic behaviour such as follow-up questions, metric changes, filtering actions, or root-cause explanations, and the results are saved to `artifcats/spec.json` file. Internal implementation details, tool names, and workflow identifiers are removed or translated into natural language to ensure the resulting specifications remain generation-friendly.

7.4.2 Specification Coverage Analysis

Once the specification catalogue is derived, the eval suite is analysed to determine which behaviours are already covered by existing evals. Coverage is evaluated at the *session* level rather than per-turn level because many specifications depend on multi-turn conversational context. Each session is converted into a compact structured representation containing user messages, agent responses, and summaries of the respective traces. An LLM then evaluates whether the session satisfies each specification by verifying the required conversational setup, the triggering user action, and the observable agent response. Coverage results are aggregated into a persistent coverage matrix (`coverage_report.json`) that records which sessions satisfy which specifications as shown in Figure 11. Specifications not covered by existing sessions are forwarded to the synthetic generation stage.

```
"spec_032": {
   "spec": {
      "id": "spec_032",
      "source_prompt": "conversational_agent_v2::CONVERSATIONAL_INSTRUCTIONS",
      "category": "edge_case",
      "description": "When the user explicitly asks to use another language, the
agent must switch to the requested language.",
      "scenario": "Language switch on request"
   },
   "covered_by": [ ],
   "is_covered": "false",
   "reasoning": ""
}
```


Figure 11: A sample behavioural specification with its coverage information.

7.4.3 Augmented Conversation Generation

For each uncovered specification, the system generates targeted augmented conversations as shown in Figure 12. The generation model decides whether to extend an existing session with follow-up questions or create a new standalone session. Specifications benefiting from prior conversational context typically extend existing sessions, while self-contained behaviours generate new sessions.


```
[
   {
      "sessionId": "<generated_uuid>",
      "input": {
         "message": "Can you answer in Spanish?",
         "language": "English (US)",
         "time_zone": "Europe/Helsinki",
         "context": {},
         "rca_configuration": "null",
         "recommendations_configuration": "null",
         "currency": "EUR"
      },
      "metadata": {
         "type": "chat_trace",
         "customer_id": "<uuid>",
         "generated_from_spec": "spec_032",
         "spec_category": "edge_case",
         "spec_scenario": "Language switch on request",
         "chat_order": "2"
      }
   }
]
```


Figure 12: A sample augmented conversation generated for an uncovered behavioural specification (language-switch request).

To improve the realism and diversity of generated conversations, the augmentation process first extracts the metrics and dimensions represented across the existing real-world eval suite and reuses them when generating new conversations. Non-

English languages present in the dataset are also periodically sampled to introduce linguistic diversity. Augmented questions are further constrained to use only entities observed in real sessions, preventing the generation of fabricated store names, regions, or products that are not present in the source data. Before an augmented question is included, an LLM verifies whether it is likely to trigger the intended behavioural specification rather than merely being semantically related to it. Conversations that fail this verification are discarded. This pre-flight check therefore validates the suitability of the generated user message for exercising the target behaviour.

Phase 3 introduces a specification-driven approach to expanding the eval suite. Rather than relying solely on commonly occurring user interactions, the method extracts behavioural specifications directly from the agent's deployed prompts and systematically targets uncovered behaviours through augmented conversations. These conversations are executed against the live application, and the resulting traces are used to generate new evals. This approach expands the eval suite while keeping the generated cases grounded in realistic system behaviour, targeted toward specific behavioural gaps, and measurable in terms of coverage of the agent's intended capabilities.

8 Results

This chapter presents the experimental setup and results obtained from evaluating the proposed automated evaluation framework. The experiments focus on two major aspects of the framework to answer the stated research questions: the effectiveness of automated refinement of eval criteria through lightweight expert feedback (RQ2), and the coverage improvement from specification-driven eval generation (RQ3). Since RQ1 focuses on transforming traces to evals, which has been exercised in building the eval pipeline and is also the foundation for experimenting with the remaining two research questions, only RQ2 and RQ3 are empirically evaluated.

The experiments were conducted using ZED, a retail analytics chatbot built using an agentic LLM architecture. The evals were executed against a locally running version of the chat application using Promptfoo. Since LLM-based systems are inherently non-deterministic, each experiment was repeated across five independent iterations to assess the stability of the observed results. Five iterations were selected as a practical balance between experimental cost and the need to account for stochastic variation. Table 2 summarises which stages of the evaluation pipeline were re-executed in each run or iteration and which stages were performed only once and kept fixed. For RQ2, the same input traces and expert feedback were used across the five runs, while the eval generation and Promptfoo execution were repeated. For a given run, the acceptance rate is the number of evals the domain expert accepted divided by the number of evals that run generated. Although the evaluation generation step is executed in each run, the denominator remains fixed across all runs because the input trace is kept constant. For RQ3, specification extraction and coverage mapping were performed once, while augmented question generation were repeated across iterations to target previously uncovered specifications. For a given iteration, coverage is the number of behavioural specifications marked as covered divided by the total number of specifications extracted. The denominator is fixed because specification extraction is executed once.

Stage	**RQ2 Experiment**	**RQ3 Experiment**
Trace collection	Once	Every run (after augmented session generation)
Filtering and deduplication	Once	Every run (after augmented session generation)
Session grouping	Once	Every run (after augmented session generation)
Conversation replay	Once	Every run (after augmented session generation)
Eval generation	Every run	Every run (after augmented session generation)
Promptfoo execution + judging	Every run	Every run
Expert review	Once, before the 5 runs	Not applicable
Prompt refinement	Once, before the 5 runs	Not applicable
Specification extraction	Not applicable	Once (30 specifications, fixed)
Coverage mapping	Not applicable	Once
Augmented question generation	Not applicable	Every run

Table 2: Stages re-executed across the five RQ2 runs and five RQ3 iterations.

All LLM-based components of the framework used Azure OpenAI GPT-5.4-mini.

Although the same model is used across the LLM-based components, these components perform distinct tasks over different evidence sources. Eval generation uses execution traces to construct evaluation criteria, while trace validation uses the trace together with expert feedback to determine whether the feedback is supported by the observed behaviour. In the subsequent Promptfoo evaluation, the judge LLM does not assess whether the generated criteria themselves are correct. Instead, it applies the generated criteria to the actual application output and determines whether the output satisfies those criteria. Similarly, specification extraction operates on agent prompt files to identify behavioural requirements. Thus, the model is used for separate, task-specific analyses rather than simply being asked to judge its own previously generated output.

8.1 Evaluation Measures and Terminology

Four distinct quantities are relevant to interpreting the experiments and are kept separate because they capture different aspects of the evaluation process. Not all four are empirically measured in this study; the distinction is provided to clarify the interpretation of the reported results.

- **Acceptance rate:** The percentage of generated evals that were accepted as valid by the human expert. This measures the validity of the generated evaluation criteria.
- **Category-wise accuracy (pass rate):** The percentage of evals within an eval category that are successfully passed by ZED's responses when evaluated using Promptfoo with an LLM-as-a-judge. This measures how well ZED satisfies the accepted evaluation criteria for each category. A failure indicates an issue with the application's behaviour, rather than with the validity of the criterion.
- **Judge correctness:** Whether the LLM judge itself makes the correct pass/fail decision when assessing a response against a valid evaluation criterion. This measures the correctness of the judge's assessment, rather than the correctness of the application or the validity of the criterion. This quantity is not empirically measured in this study.
- **Specification coverage:** The proportion of extracted behavioural specifications exercised by at least one session.

8.2 Evaluating RQ2

This experiment evaluates whether a lightweight layer of expert feedback on evals can improve the quality and reliability of automatically generated eval criteria. Initially, 17 evals were automatically generated from randomly selected development and production Langfuse traces. The seed set was chosen to provide a sufficiently diverse representation of real-world interactions, including different metrics, dimensions, and question types, while remaining small enough to examine whether the subsequent specification-driven generation could identify and target previously uncovered behavioural flows. The same 17 traces (questions) were used in every run reported for RQ2. No traces were added, removed, or substituted between runs. Each of the five post-refinement runs re-executed the eval-generation stage over the

identical set of 17 replayed traces used for the baseline run, so that any change in acceptance rate is attributable to the refined generation prompt rather than to a change in input data.

The generated evals were executed against the chat application using Promptfoo. Out of the 17 evals, only 12 passed during the initial experiment. Expert review identified that the failures were caused by incorrect or poorly generalized evaluation criteria. In addition to the failed evals, 2 other evals were also rejected by the expert, as they did not contain eval criteria as per the correct eval category. Therefore, out of the 17 generated evaluations, 10 evaluations were accepted, and 7 evaluations were rejected after expert review. This resulted in an initial acceptance rate of approximately 58.8%.

The rejected evaluations exposed several systematic weaknesses in the automatically generated eval criteria. Table 3 summarizes representative examples of captured incorrect eval criteria and the corresponding expert feedback used to refine the generation prompts. The observed issues revealed several important limitations of the initial LLM-generated eval criteria: criteria were sometimes overly strict and tightly coupled to trace-specific values, derived numerical statements were incorrectly enforced as mandatory outputs, forecast and recommendation questions were misclassified as normal analytical queries, grouping behaviour was inferred from user language instead of actual trace configuration, and temporal validation criteria lacked explicit distinction between current and comparison periods. These observations demonstrated that a light-weight expert feedback layer was necessary not only for correcting failed evaluations, but also for improving the internal consistency and long-term reliability of the evaluation framework itself.

After incorporating expert feedback, the eval generation prompt was regenerated, and the experiment was repeated across five independent runs. Figure 13 shows the acceptance rate before and after iterative expert-feedback refinement. The expert-feedback-driven prompt refinement process substantially improved evaluation quality. Across the five independent runs, accepted evaluations ranged between 15 and 16 out of 17, rejected evaluations were reduced to only 1–2 per run, and acceptance rates improved from 59% to between 88% and 94%. The results indicate that the light-weight expert-feedback loop substantially improved the robustness of eval criteria generation. Importantly, the improvement remained stable across subsequent independent runs despite the non-deterministic behaviour of LLM-based systems.

In addition to the overall acceptance rate, category-wise accuracy was measured. Accuracy for a category was calculated as the percentage of evals within that category that passed successfully. Figure 14 presents the category-wise accuracy results across all iterations, from which the following observations can be made:

- Metric Selection, Filtering, Time Periods, Forecast Validation, and Chart Configuration achieved consistent 100% accuracy after refinement.
- Grouping accuracy improved from 81% initially to between 93% and 100% across later iterations.
- Numerical Accuracy improved from 67% to 88%, but remained the most challenging category.

Nevertheless, the results demonstrate that expert-guided refinement substantially improved evaluation reliability across nearly all behavioural dimensions. Although

User Question	Incorrect Eval Criteria	Expert Feedback Summary
1. *show sales for August 2025*	**Metric Selection** - Validate that the analysis uses sales as the selected metric	Always use the exact human-readable display name of the metric as it appears in the trace rather than internal variable names or database field identifiers.
2. *add discount %*	**Numerical Accuracy** - QUILTED FOOTWEAR should be the top-ranked subcategory by Sales. - QUILTED FOOTWEAR should have a Discount around 8.26%, and it should be above the top-10 average of about 7.71%. - QUILTED JACKET should have the highest Discount among the top 10 subcategories, around 8.86%. - JEANS should have the lowest Discount among the top 10 subcategories, around 7.20%	Do not generate strict evaluation criteria for derived or implicit percentage calculations that are not explicitly required in the response.
3. *show Avg. Purchase Value (VAT) (incl. Returns) of August 2025*	**Numerical Accuracy** - The highest daily value in the period should be identified as August 29, 2025. - The lowest daily value in the period should be identified as August 1, 2025. - The response should reflect that the metric increased toward the end of the month, with the last three days staying above 60 EUR.- The response should preserve the identified peak intraday period for August 29, 2025 as 00–12.	Separate observation-data validation from response-consistency validation. Facts directly observable from traces should be validated independently from generated response claims.
4. *Why is the Avg. Purchase Value (VAT) (incl. Returns) highest in the eCommerce store*	**Time Periods** - Use the custom period from 2025-08-01 to 2025-08-31 as the current period. - Compare it against the previous 125-day window from 2025-03-29 to 2025-07-31.	Explicitly distinguish the current period and comparison period using separate validation criteria instead of vague temporal descriptions.
5. *compare this with July 2025*	**Grouping** - No grouping should be applied	Extract grouping information only from the global-level filter configuration present in the trace output rather than inferring grouping from the user query or response text.
6. *What should I do for next week with Sales w/Tax by Store?* 7. *Can you recommend the best stores to focus on next week?*	Incorrectly added Metric Selection, Filtering, and Grouping criteria instead of the Forecast/Recommendation category.	Questions requesting future guidance, planning, recommendations, or prioritization should be categorized under Forecast/Recommendation evaluation logic rather than standard analytical evaluation categories.

Table 3: Representative examples of incorrect evaluation criteria identified through expert review and the corresponding refinement feedback.

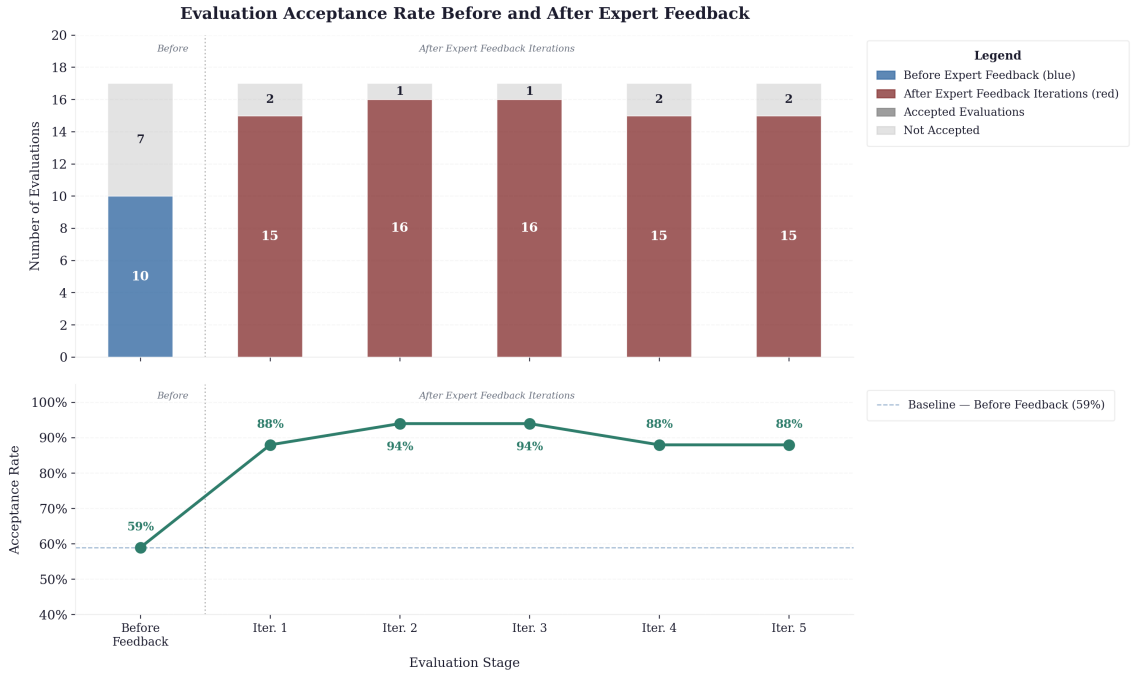

Figure 13: Evaluation acceptance rate before and after expert-feedback refinement

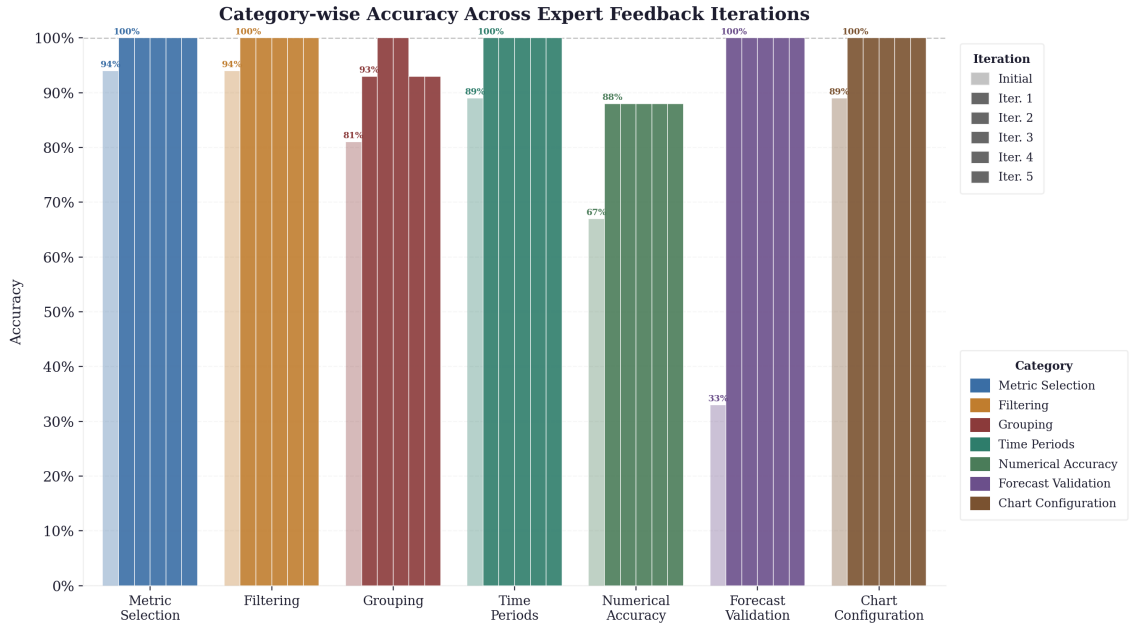

Figure 14: Category-wise evaluation accuracy across expert-feedback iterations

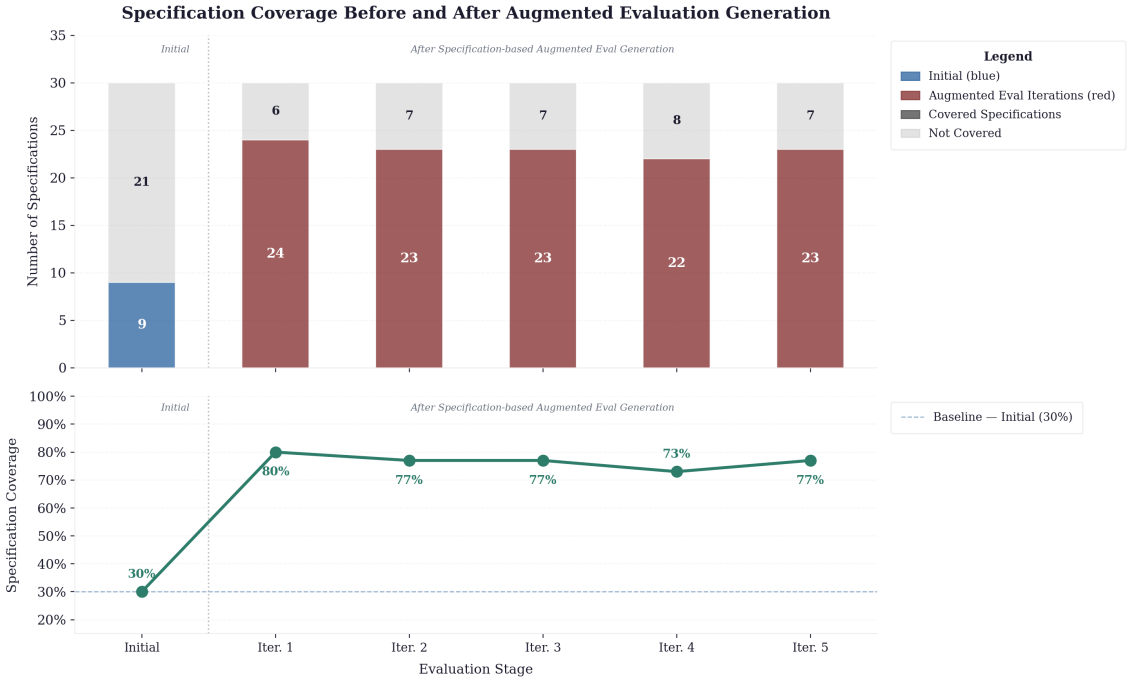

Figure 15: Behavioural specification coverage before and after specification-driven augmentation. The upper panel shows covered and uncovered specifications out of 30; the lower panel shows coverage as a percentage across five independent iterations.

the initial generation process successfully extracted useful assertions from traces, it lacked sufficient abstraction and behavioural understanding to consistently produce reusable evaluations. The expert feedback process improved the framework in several important ways by reducing overly trace-specific assertions, improving semantic categorization of evaluation types, introducing clearer distinctions between observation validation and response validation, improving temporal and grouping consistency checks, and enhancing long-term evaluation stability for future model executions. Overall, the results suggest that combining automated trace mining with a lightweight expert feedback layer creates a practical and scalable method for generating high-quality evaluation suites for agentic AI systems.

8.3 Evaluating RQ3

This experiment evaluates whether specification-driven eval generation can systematically improve coverage of the eval suite. A formal specification set consisting of 30 behavioural specifications was extracted from the system prompts used by the ZED chat application codebase. Since ZED is an agentic chatbot, the selected prompts covered instructions provided for multiple agents: root cause analysis agent, triage agent, analysis agent, conversational agent, time agent, and recommendation agent. These prompts contain behavioural instructions defining how the agent should interpret user intent, perform tool invocation, configure filters and metrics, execute RCA workflows, provide recommendation-oriented responses, and manage conversational context. Following specification extraction, a coverage matrix was constructed to identify which specifications were exercised by the existing evaluation set. The initial set of evals covered only 30% of the extracted requirement specifications. The 30 be-

havioural specifications were extracted once and held fixed across all five iterations. Each iteration started from the same baseline state – the accepted eval set derived from the 17 original traces, covering 9 of the 30 specifications – and independently generated its own augmented sessions. Iterations were therefore not cumulative: the coverage figures reported per iteration in Figure 15 are five independent attempts to close the same initial gap, not a running total.

To improve coverage, the framework generated augmented multi-turn question sessions targeting uncovered specifications. In the RQ3 experiment, coverage results were collected across five independent iterations, where each iteration either created new question sessions or augmented existing sessions by appending user questions that the LLM predicted would trigger coverage of the previously uncovered specifications. Representative examples from the augmented question sessions are shown in Table 4.

As illustrated in Figure 15, the results demonstrate a substantial increase in behavioural coverage, increasing from 30% to between 73% and 80%. The results indicate that specification-driven eval generation successfully complements the initial eval set. The greatest improvements were observed in recommendation-oriented interactions, multi-turn contextual reasoning, time-based comparison handling, RCA trigger conditions, and conversational follow-up flows. The remaining uncovered specifications were typically associated with highly specialized or infrequently triggered behaviors that required specific conversational contexts. For example, the derived specification: *When the user replies with a number after the agent previously showed numbered clarification options with a mapping comment (e.g., the user first saw options for a metric or filter, then replies "2"), the agent must treat the number as a clarified selection and apply the corresponding metric, filter, or grouping choice*, was too abstract for the LLM to reliably generate or augment sessions for. This was primarily because the specification depended on the agent's prior response and did not provide sufficient details about what clarification options would be presented. However, since the specification is no longer latent, a developer can manually execute the scenario during development, add the corresponding session ID to the pipeline, and generate evaluations to cover these types of specialized uncovered scenarios.

Overall, the RQ3 results indicate that specification-driven augmentation can substantially improve evaluation coverage beyond what is achievable using development or production traces alone. The results suggest that combining trace mining, expert-guided evaluation refinement, and specification-driven augmentation provides a scalable methodology for constructing high-coverage evaluation suites for complex agentic AI systems. The achieved coverage levels of 73%–80% indicate that the framework can successfully generalize beyond observed production behaviour and proactively test latent agent capabilities defined within system prompts.

Uncovered Specification	**Newly Generated Question Sessions**	**Reason for New Coverage**
When the user explicitly asks to use another language in the current message, the agent must switch to that requested language.	1. *Can you break down Avg. Purchase Size (excl. Returns) by Store for April 2026?* 2. *Answer in Spanish.*	Turn 2 contains an explicit language-switch request ("Answer in Spanish"), directly invoking the specification while preserving the prior analytical context from Turn 1.
When the user asks what the assistant can do or how to get started, the agent must provide a concise capability overview instead of fetching data.	1. *What can you help me with?*	The user message directly requests assistant capabilities, triggering the capability-overview behavior rather than analytical processing.
When the user sends a greeting or pleasantry, the agent must respond with a friendly conversational greeting in the preferred language.	1. *Hi there!*	The message is a direct greeting and activates the greeting-response behavior defined in the specification.
When the user asks a broad or underspecified question, the agent must ask a clarifying question instead of immediately providing analysis.	1. *Can you analyze this?* 2. *Show me the numbers.*	Both turns are underspecified analytical requests without sufficient context, thereby invoking the clarification requirement.
When the user asks for a specific subrange inside the currently active time selection, the agent must narrow the time filter before analysis.	1. *Show Sales w/Tax by Store for last week* 2. *Can you forecast Sales w/Tax for the stores next week* 3. *Show me Sales w/Tax by Store for this month* 4. *Just show me Tuesday to Thursday*	Turn 3 establishes a monthly time context, while Turn 4 requests a narrower subrange (Tuesday to Thursday) within that active selection.
When the user asks for a different metric than the one established in the immediately preceding turn, the agent must switch to the new metric before analysis.	1. *Show conversion rate in January 2025 compared to February 2025* 2. *Show me Sales (VAT) in January 2025* 3. *Why did Visitor Hit Rate change in the same period?*	Turn 2 changes the active metric from Conversion Rate to Sales (VAT), while Turn 3 switches back again, creating valid metric-transition scenarios.
When the user asks to drill down one level deeper from the current grouping, the agent must use the next drill-down dimension in the hierarchy.	1. *What should I do for next week with Sales w/Tax by Store?* 2. *Can you recommend the best stores to focus on next week?* 3. *Break down Avg. Purchase Value (VAT) (incl. Returns) by Store for August 2025.* 4. *Can you break Wholesale down by Subcategory?*	Earlier turns establish Store as the active grouping context, while Turn 4 explicitly requests a deeper drill-down by Subcategory.

Table 4: Representative examples of augmented multi-turn question sessions generated to improve specification coverage for previously uncovered behaviors.

9 Discussion

This chapter interprets the results reported in the previous chapter and assesses their significance in relation to the research questions, reflects on the limitations of the work, and situates the findings within the broader landscape of automated evaluation for agentic AI systems. The discussion proceeds from a consideration of whether the stated research goals were achieved, through a deeper interpretation of the two empirical experiments, to an examination of what the results suggest, beyond the specific context in which they were obtained.

9.1 Achievement of the Research Goals

The central goal of this thesis was to investigate whether the process of constructing eval suites for agentic LLM-based chatbots could be substantially automated, reducing the manual overhead that currently makes continuous evaluation impractical in industrial settings. Three research questions guided the work, and the results together provide a reasonably complete answer to each of them.

- **RQ1** asked how observable production and development traces could be transformed into reusable evaluations. Rather than being evaluated in isolation, RQ1 underpins the entire pipeline: the trace-to-eval transformation it describes is the mechanism by which all subsequent experiments were made possible. The successful construction and operation of the pipeline – capable of collecting Langfuse traces, filtering and grouping them, replaying conversations, and generating categorised rubric-based evaluation criteria – demonstrates that the transformation is technically feasible. The fact that the generated evals could be executed automatically against a live chat application via Promptfoo, with results that were meaningful enough to distinguish agent failures from evaluation-criteria failures, is itself evidence that RQ1 has been answered affirmatively.
- **RQ2** asked whether a lightweight layer of expert feedback could improve the quality and reliability of automatically generated eval criteria. The initial acceptance rate of approximately 59% rose to between 88% and 94% across five independent runs after a single round of targeted expert feedback was incorporated into the generation prompt. This improvement is substantial, stable, and achieved with a comparatively small investment of expert effort – the feedback addressed only seven rejected evaluations, yet the resulting prompt refinements generalised to produce broadly better criteria. The results therefore indicate that expert input can play an important role in correcting criteria that cannot be reliably inferred from execution traces alone.
- **RQ3** asked whether specification-driven eval generation could systematically improve the coverage of the evaluation suite. The results suggest that the method can substantially improve behavioural coverage. Coverage increased from 30% to between 73% and 80% across the five runs by generating augmented multi-turn question sessions targeting previously uncovered behavioural specifications. This demonstrates that relying exclusively on production traces for evaluation is an inherently incomplete strategy, and that the latent behavioural requirements encoded in system prompts constitute a valuable, largely

untapped source of guidance for evaluation design. The framework successfully operationalised this insight at scale.

Taken together, the three research questions show how different stages of the framework contribute to building and maintaining an automated evaluation suite. Trace-based generation provides a way to derive evaluation criteria from existing application behaviour, reducing the need to manually create evaluation cases from scratch. Expert-guided refinement improves the validity of evaluation criteria and acceptance rate of the generated evals, while specification-driven augmentation increases behavioural coverage beyond what is represented in the available production traces. These results suggest that combining these stages can reduce manual effort in evaluation development and provide a more systematic way to extend evaluation suites as new behaviours and coverage gaps are identified. However, the study does not directly measure cost reduction or scalability, so these benefits should be considered as potential advantages rather than empirically established outcomes.

9.2 Interpreting the Expert-Feedback Refinement Results

The improvement in acceptance rate observed in RQ2 also provides insight into the limitations of LLM-generated evaluation criteria. The seven rejected evaluations revealed a consistent pattern: the initial generation process accurately extracted facts from a trace, such as the metric used, the filters applied, and the top-ranked result, but consistently conflated these facts with test rules. A fact observed in a trace (for example, a specific daily sales figure) is only meaningful in the context of that particular data snapshot; a reusable evaluation criterion must instead express a behavioural rule that should hold regardless of the underlying data at the time of re-execution. The initial LLM generation failed to make this distinction, producing criteria that were tightly coupled to trace-specific values rather than generalisable assertions about agent behaviour. Additional issues included misclassification of forecast and recommendation questions as standard analytical queries, and a failure to distinguish between validating raw observations present in the trace and validating the agent's interpretation of those observations in its final response.

These failures are characteristic of a well-known challenge in LLM-based test generation: a language model can reproduce the content of a trace with high fidelity, but the act of deciding which aspects of that content should be generalised into a stable, rerun-safe criterion requires a form of domain understanding that is difficult to elicit purely from the trace itself. Expert feedback closes this gap not by replacing the LLM but by shaping the conceptual vocabulary it uses to formulate criteria – distinguishing, for instance, between a fact that was true in a specific trace and a rule that should hold across all valid agent responses to a given class of question.

The category-wise accuracy results reinforce this interpretation. The categories that achieved 100% accuracy after refinement, Metric Selection, Filtering, Time Periods, Forecast Validation, and Chart Configuration, are largely *structural*. They focus on whether specific and verifiable configurations are reflected in the agent's response, and the rules for generating these criteria are relatively clear once explicitly defined. Numerical Accuracy, by contrast, remained the most challenging category even after refinement, reaching only 88%. A significant part of this residual gap, however, is attributable not to a weakness in the evaluation framework itself but to a genuine defect discovered in the core ZED application during the course of

the experiments. Consider the evaluation for the question *show sales for August 2025*: the LLM judge correctly identified that the agent's trace observations included all 31 daily sales records and showed August 17 as a lower-value day at 3,999, yet the agent's final response to the user incorrectly stated that August 31 was the weakest day at 5,633. This is a direct inconsistency between the agent's own internal data and the statement it communicated to the user – a reasoning error in the ZED agent's response-generation logic rather than a flaw in the evaluation criterion. The promptfoo judge detected this discrepancy precisely because the criterion required consistency between the trace observations and the final response, and flagged the eval as a failure. This outcome is in fact a demonstration of the framework's diagnostic value: the evaluation pipeline surfaced a latent application bug that would otherwise have been invisible to users and developers alike. From an evaluation standpoint, there is little more that can be done to improve the criterion further; the correct remediation path is to fix the identified reasoning bug in the core ZED application and then re-run the evaluation to confirm the failure is resolved. The persistent 88% ceiling in Numerical Accuracy is therefore partly a measure of application quality rather than evaluation quality, and should be expected to improve as underlying application defects are addressed.

It is also noteworthy that the improvement in acceptance rate was stable across five independent runs despite the inherently non-deterministic nature of LLM generation. This stability suggests that the expert-refined prompt did not merely nudge the model toward slightly better outputs on average; rather, it shifted the generation process onto a more reliably constrained path, one from which the model could not easily deviate even under different random seeds or context variations. This is an important practical finding: it implies that the effort invested in prompt refinement yields durable rather than ephemeral gains, making the approach viable for long-running evaluation maintenance.

9.3 Interpreting the Specification-Driven Coverage Results

Development and production traces constitute samples of observed system behaviour, reflecting the queries submitted by users, the execution paths taken by the agent, and the data states available at the time of each interaction. Consequently, such traces cannot be expected to exhaustively represent the full range of behaviours that the system is designed to support. In particular, infrequently occurring edge cases, multi-turn interaction sequences, and error-handling behaviours may be absent or under-represented in the observed trace corpus.

The results demonstrate that specification-driven augmentation can address these coverage gaps, increasing behavioural coverage to 73–80% by generating conversational sessions targeted at previously uncovered specifications. The augmented sessions presented in Table 4 demonstrate that the generated interactions can be both realistic and behaviourally targeted. The generated turns are naturally phrased, remain contextually consistent with preceding interactions, and are designed to exercise the specific behaviour defined by the target specification rather than merely resemble it semantically. The language-switch example, in which a multi-turn session concludes with a request to "Answer in Spanish", illustrates how specification-driven generation can introduce valid test scenarios that may be unlikely to occur in a limited corpus of naturally observed production interactions.

The 20–27% of specifications that remained uncovered after augmentation are also informative. The representative example discussed in the results section, a specification that depends on the agent having previously shown numbered clarification options, illustrates a class of specifications that is fundamentally harder to synthesise: those where the agent's prior response is itself a precondition for the test scenario. No amount of prompt engineering can reliably generate sessions for these specifications without access to a faithful simulation of the agent's intermediate outputs, which is itself a non-trivial engineering challenge. Rather than treating this residual gap as a failure of the framework, it should be understood as a natural boundary condition that points toward a productive direction for future work: a feedback mechanism that allows developers to manually execute difficult edge cases and inject them back into the pipeline, thereby converting latent specifications into covered evaluations.

The stability of coverage improvements across independent runs is again notable. Coverage did not collapse back toward its initial level when the pipeline was re-run, which could have been a concern given that the augmented sessions are themselves LLM-generated and subject to variability. The consistency of the 73–80% range suggests that the specification matching process, which uses an LLM to determine whether a session tests a given specification, is reliable enough to produce similar results across runs, even though the exact set of covered specifications may vary slightly between runs.

An important boundary of the RQ3 approach, however, is that it targets exclusively *behavioural* specifications, those that describe how the agent should respond to a particular class of user input. The augmentation mechanism works by generating conversational turns that a real user might plausibly send and that are predicted to trigger the desired observable behaviour. This strategy is well-suited to user-facing response behaviours, but it cannot directly address a separate class of requirements: the internal, technical specifications that govern how the agent invokes its tools and constructs its intermediate reasoning steps. For example, a specification that mandates a particular sequence of tool calls, a specific data transformation step, or a precise threshold applied during a root cause analysis workflow cannot be reliably targeted by formulating a user question – there is no guaranteed mapping from a natural language user turn to the execution of a specific internal tool invocation. These internal specifications would require a different form of evaluation, such as trace-level assertions that inspect the agent's tool call logs directly rather than its final response. The current pipeline does not produce such assertions, and the coverage metric therefore reflects only behavioural coverage over user-observable outputs. Extending the framework to cover internal tool-invocation specifications represents a meaningful direction for future work, and would require tighter integration between the evaluation pipeline and the agent's internal execution traces.

9.4 Limitations of the Work

While the results are encouraging, several important limitations of the work must be acknowledged.

- **Single-system evaluation.** All experiments were conducted using a single agentic application: ZED, a retail analytics chatbot. The findings regarding acceptance rates, coverage levels, and the effectiveness of the expert-feedback

mechanism are therefore contingent on the particular characteristics of this system, including its domain, its agent architecture, the nature of its system prompts, and the style of its production traces. It would be premature to assert that the specific numerical results (e.g., 59% initial acceptance rate, 30% initial coverage) are representative of agentic systems in general. What can be more confidently generalised is the directional finding: that automated generation without feedback is unreliable, that expert feedback produces durable improvements, and that production traces alone are an insufficient basis for coverage.

- **Expert feedback as a bottleneck.** The light-weight expert-feedback mechanism demonstrated in RQ2 required a domain expert to review seven rejected evaluations and articulate refinement guidance. While this is modest compared to the cost of writing evaluation suites entirely by hand, it remains a dependency on scarce expertise. In a large organisation with many AI systems, the cumulative demand for such expertise could become a bottleneck. Furthermore, the quality of the resulting evaluations is bounded by the quality of the expert feedback: systematic biases or blind spots in the expert's review will propagate into the refined prompt and be amplified across all future generations.

- **LLM non-determinism and evaluation reliability.** The experiments were repeated across five independent runs to account for LLM non-determinism, and the results were broadly consistent. However, five runs is a relatively small sample for drawing strong statistical conclusions. Edge-case variations in acceptance rate or coverage could emerge over a larger number of runs or when the system is used with different underlying model versions. The results should therefore be interpreted as indicative rather than precisely calibrated estimates of framework performance.

- **Absence of ground-truth oracles.** The evaluation criteria generated by the pipeline are rubric-based: an LLM judge assesses whether the agent's response satisfies each criterion. While this approach is broadly consistent with the LLM-as-judge paradigm that has become standard in the evaluation of open-ended generative systems, it introduces a second source of non-determinism and potential error. A criterion that is perfectly well-formed may still be evaluated inconsistently if the judge model applies different reasoning across runs. The acceptance rate improvements measured in RQ2 are themselves subject to this confound.

- **Scope of the evaluation categories.** The evaluation rubric covers a fixed set of behavioural categories relevant to ZED: metric selection, filtering, grouping, time periods, chart configuration, numerical accuracy, and forecasting. While these categories were designed to be comprehensive for a retail analytics chatbot, they may not transfer without modification to agentic systems with different task structures – for instance, a code-generation assistant, a customer-service agent, or a document-reasoning system would require a different category taxonomy. The category design is therefore a form of domain knowledge embedded in the framework that would need to be revisited in each new application context.

- **Sensitivity to model choice.** All LLM-based pipeline components in this work used Azure OpenAI GPT-5.4-mini. Substituting a different model, whether a larger, more capable model or a smaller, more cost-efficient one, could affect the results in several ways: a more capable model might produce more abstractly generalised rubric criteria from the outset, reducing the number of expert rejections and potentially raising the initial acceptance rate above the 59% baseline observed here; a weaker model might require more rounds of expert feedback to reach comparable quality. Similarly, specification extraction and coverage mapping accuracy in RQ3 may vary with model capability, affecting the coverage levels achieved. The core methodology is model-agnostic, and the directional findings are expected to hold regardless of model choice, but the specific numerical results reported here should be interpreted as dependent on the particular model used rather than as fixed properties of the framework.

9.5 Connections to Related Work

The results of this thesis speak to several themes that have emerged prominently in the recent literature on LLM evaluation and testing.

The challenge of constructing robust evaluation criteria for open-ended generative outputs is widely recognised. The LLM-as-judge paradigm – using a language model to evaluate the outputs of another language model – has been extensively studied and shown to offer a practical alternative to human annotation at scale. The findings of this thesis are consistent with this line of work in demonstrating that LLM judges can produce reliable assessments when the criteria are well-formed, but add an important nuance: the quality of the criteria is at least as important as the quality of the judge. The expert-feedback mechanism described in RQ2 is, in effect, a method for improving the criteria themselves rather than the judge, and the results suggest that this investment yields disproportionate returns.

The specification-driven approach to evaluation coverage explored in RQ3 resonates with the tradition of specification-based testing in classical software engineering, where behavioural requirements are systematically translated into test cases. Prior work in behavioural testing for NLP – most notably the CheckList framework [50] – has similarly argued that evaluation should be structured around a comprehensive taxonomy of capabilities rather than determined solely by the distribution of available data. The contribution of this thesis is to show that this principle can be operationalised in an automated pipeline by extracting specifications directly from the system prompts of an agentic application, rather than relying on manually curated capability taxonomies.

The finding that production traces cover only a minority of specified behaviours is consistent with a common limitation of data-driven testing. Evaluations based only on observed user behaviour may miss less frequent but important scenarios, which can still lead to failures in real-world use. Specification driven augmentation helps address this gap by extending coverage to behaviours that are not well represented in the available traces, while complementing the trace based approach.

The use of multi-turn conversational sessions to trigger specific behaviours is particularly relevant in the context of agentic systems, where the agent's response to any given turn depends heavily on the conversational context established by prior turns. This contextual dependency makes agentic systems significantly harder to evalu-

ate than single-turn models, and the framework's ability to synthesise contextually coherent multi-turn sessions targeting specific latent behaviours represents a meaningful advance over evaluation approaches designed for stateless question-answering systems.

9.6 Generalisability and Broader Implications

A central question for any empirical study is whether its findings can be generalised beyond the specific context in which they were obtained. Several aspects of this work are likely to generalise broadly; others are more context-dependent.

The most confidently generalisable finding is the structural claim: that a three-component approach – trace mining, expert-guided refinement, and specification-driven augmentation constitutes a viable and practical methodology for constructing evaluation suites for agentic AI systems. Each component addresses a different limitation of a fully automated approach, and together they achieve broader results than any individual component alone. This structural insight is independent of the specific domain, model architecture, or application type, and should apply to any agentic system that maintains observable traces and whose intended behaviour is encoded in system prompts or comparable artifacts.

The specific mechanism of extracting specifications from system prompts is also broadly applicable. Most modern LLM-based applications encode their intended behaviours in system prompts, and these prompts are an authoritative, up-to-date source of behavioural requirements that is routinely overlooked as an evaluation resource.

The framework also has natural applicability in continuous integration and deployment contexts. As agentic AI systems evolve with new capabilities added, agent architectures revised, or underlying models swapped, the evaluation suite needs to evolve correspondingly. The pipeline described in this thesis is designed to be re-run: new traces can be collected, new specifications extracted, and coverage re-assessed whenever the system changes. This makes it suitable for integration into a continuous evaluation workflow, where eval coverage becomes a monitored metric alongside functional test coverage and performance metrics.

Beyond the immediate context of retail analytics chatbots, the framework could be applied to any domain in which an LLM-based agent produces structured responses that can be validated against observable properties: customer-service agents, medical triage assistants, financial advisory chatbots, or educational tutoring systems. The specific evaluation categories would need to be adapted to each domain, and the expert-feedback mechanism would require domain-appropriate reviewers, but the pipeline architecture and the underlying methodology are domain-agnostic.

It is also worth considering the implications of this work for the broader question of AI system reliability. A persistent challenge in the deployment of LLM-based systems is the difficulty of providing meaningful assurances about their behaviour – assurances of the kind that traditional software engineering provides through formal verification, unit testing, and integration testing. The framework developed in this thesis does not solve this problem, but it makes a practical contribution by showing that the gap between observed behaviour and specified behaviour can be measured and systematically narrowed using automated tooling. In a field where the absence of rigorous evaluation practices has contributed to high-profile failures and eroded

user trust, the development of scalable, principled evaluation pipelines is not merely a technical contribution but a step toward the kind of engineering discipline that responsible deployment of agentic AI systems requires.

Finally, the role of the human expert in the framework should be considered. The expert feedback in RQ2 was not simply used to correct mistakes that a more capable LLM might avoid. Experts also provided domain knowledge about what should be considered a valid evaluation criterion, which cannot always be inferred from system traces alone. This suggests that human input will remain important even as LLM capabilities improve. Future work should therefore focus on making expert involvement more efficient while preserving its contribution to evaluation quality.

10 Conclusions and Future Work

This chapter presents the principal conclusions drawn from the experimental results, reflects on the extent to which the three research questions have been answered, and discusses the broader significance of the proposed automated evaluation pipeline. It also identifies the limitations of the current work and outlines concrete directions for future investigation, including improvements to the pipeline itself and open questions that the work has raised but not yet resolved.

10.1 Conclusions

This thesis addressed the challenge of constructing eval suites for agentic LLM-based chatbot applications systematically. As these systems grow more complex, orchestrating multiple specialised agents and managing multi-turn conversational context, evaluating whether they respond correctly becomes increasingly difficult to do manually. Traditional software testing methods do not transfer readily to systems whose outputs are open-ended, context-dependent, and non-deterministic, yet the absence of systematic evaluation creates unacceptable risks for organisations that deploy these systems in production. The central question motivating this work was therefore: to what extent can the construction of evals for agentic AI chat applications be automated, and what mechanisms are necessary to make automated generation reliable and comprehensive?

To investigate this question, an end-to-end automated evaluation pipeline was designed and implemented, using ZED – a retail analytics chatbot built on an agentic LLM architecture, as the experimental platform. The pipeline proceeds through several stages: collecting observable traces from a production and development observability platform, filtering and grouping them by conversational session, replaying sessions to generate a structured dataset, and finally using an LLM to generate rubric-based evaluation criteria categorised by behavioural dimension. The generated evaluations are executed automatically against a running instance of the application using Promptfoo, with an LLM judge assessing each criterion. Two mechanisms for improving the baseline pipeline were then proposed and empirically evaluated: a light-weight expert-feedback loop for refining the evaluation generation process, and a specification-driven question augmentation mechanism for improving behavioural coverage.

The experimental results provide clear and consistent answers to the three research questions that guided the work.

- **RQ1 – trace-to-eval transformation** was validated through the construction and operation of the pipeline itself. The successful extraction of evaluation criteria from real production and development traces, and their reliable execution via an automated evaluation harness, demonstrates that the transformation is both technically feasible and practically useful. The pipeline serves as the foundation upon which the remaining two experiments were built.
- **RQ2 – expert-feedback refinement** produced the most directly measurable improvement. Without any expert input, the initial pipeline accepted approximately 59% of generated evaluations as correct and reusable. After a single round of targeted expert review covering seven rejected evaluations,

the acceptance rate rose to between 88% and 94% across five independent reruns, and this improvement remained stable across all subsequent iterations. Category-wise analysis revealed that most behavioural dimensions – including metric selection, filtering, time periods, forecast validation, and chart configuration – reached 100% accuracy after refinement. The Numerical Accuracy category remained the most difficult, reaching 88%, though a portion of the residual failures was attributable to a genuine reasoning defect discovered in the ZED application itself: the agent was found to report an incorrect weakest-day value to the user despite its own internal trace observations containing the correct figure. The evaluation framework successfully detected this inconsistency, demonstrating that a well-constructed evaluation suite has diagnostic value beyond regression testing. The core finding of RQ2 is that automated generation alone is insufficient, but that lightweight human feedback can elevate the accuracy of the generated evals to levels that are practical for continuous use.

- **RQ3 – Specification-driven coverage improvement** examined whether the evals generated from production traces provided sufficient coverage of the system's behaviour. The initial trace-derived suite covered only 30% of the 30 behavioural specifications extracted from the ZED system prompts – a finding that demonstrates that real usage data, however rich, systematically underrepresents the full range of behaviours a system is designed to support. By generating augmented multi-turn conversational sessions specifically targeting uncovered specifications, the framework raised coverage to between 73% and 80%, a more than twofold improvement that remained stable across independent runs. The strongest gains were observed in recommendation-oriented interactions, multi-turn contextual reasoning, time-based comparison handling, and conversational follow-up flows – precisely the categories of behaviour that are least likely to appear frequently in ordinary production traces but most likely to fail silently if left untested.

Taken together, the results show that combining production trace mining, expert-guided criteria refinement, and specification-driven augmentation is a practical way to build an evaluation suite for the system studied. Each component addresses a distinct limitation of the others: trace mining grounds the evaluation in real user behaviour but may lack sufficient coverage; expert feedback corrects cases where the LLM generates criteria that describe observed behaviour rather than define the expected behaviour; and specification-driven augmentation extends the suite to cover behavioural scenarios that may not appear in production data. No single component is sufficient on its own, but their combination produces results that would be difficult and expensive to achieve through entirely manual means.

The evidence presented in this thesis is derived from a single application within a single domain, using one underlying model, 17 seed traces, and 30 extracted specifications. This design supports claims about the observed outcomes in this case study and the mechanisms underlying them. However, the results do not support claims about the frequency with which these effects occur across agentic systems in general. Where the discussion extends beyond the scope of this case study, such statements are therefore presented as hypotheses for future investigation rather than as empirical findings.

10.2 Future Work

Several directions for future work emerge naturally from the findings and limitations of this thesis.

- **Resolving identified application defects and rebuilding affected evaluations.** The evaluation framework revealed a concrete reasoning inconsistency in the ZED agent's response generation – a case where the agent's final user-facing message was inconsistent with the observations present in its own trace. The immediate next step is to identify and fix the root cause of this behaviour in the core application, after which the affected evaluations should be rebuilt from fresh traces and re-executed to confirm the defect has been resolved. This process illustrates a broader workflow that should be institutionalised: treating evaluation failures not only as signals about eval quality but as a triage mechanism for surfacing application bugs.
- **Extending coverage to internal tool-invocation specifications.** The current specification-driven coverage mechanism operates exclusively over user-observable behavioural specifications – those that can be triggered by crafting a natural-language user turn. A meaningful class of requirements is left outside its scope: internal specifications governing how the agent invokes its tools, constructs intermediate reasoning steps, or applies specific thresholds during processes such as root cause analysis. These cannot be reliably reached by generating user questions, since there is no guaranteed mapping from a conversational turn to a particular internal tool call sequence. Extending the framework to cover this class of requirements would require trace-level assertions that inspect the agent's tool call logs directly, and would constitute a substantial enhancement of the pipeline's coverage model.
- **Adopting a more nuanced coverage model.** The binary coverage metric used in RQ3 – whether a specification is exercised by at least one session – captures breadth but not depth. A specification may be formally covered by a single session that tests only the most straightforward instance of the specified behaviour, leaving corner cases and edge conditions untested. Future work could explore a graduated coverage model that distinguishes between specifications that are minimally triggered, robustly tested across multiple contexts, and exhaustively validated across a representative range of input variations. Such a model would provide a more actionable picture of evaluation quality and guide targeted augmentation toward the highest-value gaps.
- **Evaluating generalisability across systems and domains.** All experiments in this thesis were conducted on a single application in a single domain. While the methodology and pipeline architecture are designed to be domain-agnostic, the specific numerical results, including acceptance rates, coverage levels, and category wise accuracy, may not be representative of other agentic systems. Applying the framework to systems in different domains, with different agent architectures and system prompt styles, would provide important evidence about how much of the observed performance is attributable to the methodology itself and how much to characteristics specific to ZED. Such a study would also surface any modifications to the evaluation category taxon-

omy or the specification extraction process that are necessary for the framework to transfer cleanly to new contexts.

Overall, the automated evaluation of agentic AI systems remains an open and consequential challenge, and this thesis has taken one concrete step toward meeting it. The results demonstrate that reliable, high-coverage evaluation suites can be constructed with a modest investment of human effort when production traces, expert-guided refinement, and specification-driven augmentation are used in combination. Perhaps as importantly, the work has shown that a well-constructed evaluation pipeline is not merely a regression-testing tool but a diagnostic instrument capable of surfacing genuine application defects that would otherwise go undetected in production. As agentic AI systems grow more capable and more widely deployed, the engineering discipline surrounding their evaluation will need to grow alongside them. The framework developed in this thesis offers one practical foundation for that effort.

11 Declaration of Generative AI and AI-Assisted Tools

During the preparation of this thesis, the author used language support tools, including ChatGPT and Grammarly, to improve wording, readability, punctuation, and LaTeX formatting, as well as to help troubleshoot formatting issues. These tools were used only as writing and editing aids and not to produce experimental results or scientific claims. All suggestions were reviewed and adapted by the author, who is responsible for the final content of the thesis.

References

[1] M. Abou Ali, F. Dornaika, and J. Charafeddine, "Agentic ai: a comprehensive survey of architectures, applications, and future directions," *Artificial Intelligence Review*, vol. 59, no. 1, p. 11, Nov 2025. [Online]. Available: https://doi.org/10.1007/s10462-025-11422-4

[2] S. S. Chowa, R. Alvi, S. S. Rahman, M. A. Rahman, M. A. K. Raiaan, M. R. Islam, M. Hussain, and S. Azam, "From language to action: a review of large language models as autonomous agents and tool users," *Artificial Intelligence Review*, vol. 59, no. 2, p. 71, Jan 2026. [Online]. Available: https://doi.org/10.1007/s10462-025-11471-9

[3] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 3, Mar. 2024. [Online]. Available: https://doi.org/10.1145/3641289

[4] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging llm-as-a-judge with mt-bench and chatbot arena," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2023.

[5] M. Tasarsu, A. V. Tokmak, and C. Catal, "Test case generation using large language models: a systematic literature review," *Cluster Computing*, vol. 29, no. 4, p. 227, Mar 2026. [Online]. Available: https://doi.org/10.1007/s10586-026-06021-z

[6] D. Moshkovich and S. Zeltyn, "Taming uncertainty via automation: Observing, analyzing, and optimizing agentic ai systems." IEEE Press, 2025, p. 3840–3844. [Online]. Available: https://doi.org/10.1109/ASE63991.2025.00327

[7] Anthropic. (2026, Jan.) Demystifying evals for ai agents. Anthropic Engineering Blog, accessed 2026-03-24. [Online]. Available: https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents

[8] A. E. Filali and I. Bedar, "Towards more standardized ai evaluation: From models to agents," 2026. [Online]. Available: https://arxiv.org/abs/2602.18029

[9] K. Wei, S. Guth, G. Wu, and P. Paskov, "Methodological challenges in agentic evaluations of AI systems," in *ICML Workshop on Technical AI Governance (TAIG)*, 2025. [Online]. Available: https://openreview.net/forum?id=ZhSKG8IslC

[10] S. Shankar, J. Zamfirescu-Pereira, B. Hartmann, A. Parameswaran, and I. Arawjo, "Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences," in *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3654777.3676450

[11] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," vol. 33, no. 8, Dec. 2024. [Online]. Available: https://doi.org/10.1145/3695988

[12] Q. Li, L. Cui, L. Kong, and W. Bi, "Exploring the reliability of large language models as customized evaluators for diverse NLP tasks," in *Proceedings of the 31st International Conference on Computational Linguistics*, O. Rambow, L. Wanner, M. Apidianaki, H. Al-Khalifa, B. D. Eugenio, and S. Schockaert, Eds. Abu Dhabi, UAE: Association for Computational Linguistics, Jan. 2025, pp. 10 325–10 344. [Online]. Available: https://aclanthology.org/2025.coling-main.688/

[13] R. G. Rapisarda, D. Ginelli, D. Clerissi, and L. Mariani, "Test case generation for dialogflow task-based chatbots," in *2025 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2025, pp. 93–102.

[14] A. Casella and W. Wang, "Performant llm agentic framework for conversational ai," 2025. [Online]. Available: https://arxiv.org/abs/2503.06410

[15] H. Candello and M. Brachman, "Designing conversational agent(ic) systems," in *Proceedings of the 7th ACM Conference on Conversational User Interfaces*, ser. CUI '25. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3719160.3728630

[16] C. Moralles, L. A. L. F. D. Costa, S. J. Rigo, R. Kunst, V. C. D. Souza, P. Silva, G. L. E. Prado, T. D. C. Schardosim, and A. Roehrs, "A systematic literature review of agentic ai: Definitions, architectures, and challenges," *IEEE Access*, vol. 14, pp. 36 176–36 190, 2026.

[17] C. Qu, S. Dai, X. Wei, H. Cai, S. Wang, D. Yin, J. Xu, and J.-R. Wen, "Tool learning with large language models: A survey," *Frontiers of Computer Science*, vol. 19, no. 8, p. 198343, 2025.

[18] L. Dong, Q. Lu, and L. Zhu, "Agentops: Enabling observability of llm agents," 2024. [Online]. Available: https://arxiv.org/abs/2411.05285

[19] T. Sirqueira, M. Viana, E. Cirilo, P. Alencar, D. Cowan, and C. Lucena, "Accountable autonomous systems: A provenance-aware multiagent approach," *IEEE Access*, vol. 14, pp. 94 334–94 356, 2026.

[20] J. Cabot, L. Burgueño, R. Clarisó, G. Daniel, J. Perianez-Pascual, and R. Rodriguez-Echeverria, "Testing challenges for nlp-intensive bots," in *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, 2021, pp. 31–34.

[21] S. Akshathala, B. Adnan, M. Ramesh, K. Vaidhyanathan, B. Muhammed, and K. Parthasarathy, "Beyond task completion: An assessment framework for evaluating agentic ai systems," in *Proceedings of the 2026 International Workshop on Agentic Engineering*, ser. AGENT '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 9–17. [Online]. Available: https://doi.org/10.1145/3786167.3788414

[22] M. Gritta, D. Paul, X. Li, L. Shang, J. Wang, and G. Lampouras, "Process evaluation for agentic systems," in *Findings of the Association for Computational Linguistics: EACL 2026*, V. Demberg, K. Inui, and L. Marquez, Eds. Rabat, Morocco: Association for Computational Linguistics, Mar. 2026, pp. 2678–2692. [Online]. Available: https://aclanthology.org/2026.findings-eacl.140/

[23] O. Khalid, A. U. H. Farooqi, and M. Bilal, "Agentic ai: a review, applications and open research challenges," *Computing*, vol. 108, no. 6, p. 80, 2026.

[24] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, p. 507–525, May 2015. [Online]. Available: https://doi.org/10.1109/TSE.2014.2372785

[25] S. Dhanorkar, S. Passi, and M. Vorvoreanu, "Human oversight of agentic systems in practice: Examining the oversight work, challenges, and heuristics of developers using software agents," in *The 2026 ACM Conference on Fairness, Accountability, and Transparency*, 2026, pp. 6438–6465.

[26] Y. Zheng, Y. Hu, T. Yu, and A. Quinn, "Agentsight: System-level observability for ai agents using ebpf," in *Proceedings of the 4th Workshop on Practical Adoption Challenges of ML for Systems*, ser. PACMI '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 110–115. [Online]. Available: https://doi.org/10.1145/3766882.3767169

[27] R. Solomon, Y. Yerushalmi Levi, L. Waknin, E. Aizikovich, A. Baras, E. Ohana, A. Giloni, S. Bose, C. Picardi, Y. Elovici, and A. Shabtai, "Lumimas: A comprehensive framework for real-time monitoring and enhanced observability in multi-agent systems," in *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '26. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2026, p. 3749–3757. [Online]. Available: https://doi.org/10.65109/PNWL9707

[28] A. AlSayyad, K. Y. Huang, and R. Pal, "Agenttrace: A structured logging framework for agent system observability," 2026. [Online]. Available: https://arxiv.org/abs/2602.10133

[29] A. Chan, C. Ezell, M. Kaufmann, K. Wei, L. Hammond, H. Bradley, E. Bluemke, N. Rajkumar, D. Krueger, N. Kolt, L. Heim, and M. Anderljung, "Visibility into ai agents," ser. FAccT '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 958–973. [Online]. Available: https://doi.org/10.1145/3630106.3658948

[30] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," *IEEE Transactions on Software Engineering*, vol. 50, no. 4, pp. 911–936, 2024.

[31] N. Alshahwan, J. Chheda, A. Finogenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang, "Automated unit test improvement using large language models at meta," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 2024, pp. 185–196.

[32] V. Bayrı and E. Demirel, "Ai-powered software testing: The impact of large language models on testing methodologies," in *2023 4th International informatics and software engineering conference (IISEC)*. IEEE, 2023, pp. 1–4.

[33] C. Ni, X. Wang, L. Chen, D. Zhao, Z. Cai, S. Wang, and X. Yang, "Casmodatest: A cascaded and model-agnostic self-directed framework for unit test generation," 2024. [Online]. Available: https://arxiv.org/abs/2406.15743

[34] S. Alagarsamy, C. Tantithamthavorn, and A. Aleti, "A3test: Assertion-augmented automated test case generation," *Inf. Softw. Technol.*, vol. 176, no. C, Dec. 2024. [Online]. Available: https://doi.org/10.1016/j.infsof.2024.107565

[35] A. Lops, F. Narducci, A. Ragone, and M. Trizio, "Agonetest: Automated creation and assessment of unit tests leveraging large language models," ser. ASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 2440–2441. [Online]. Available: https://doi.org/10.1145/3691620.3695318

[36] Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, and J. Yin, "Chatunitest: A framework for llm-based test generation," 2024. [Online]. Available: https://arxiv.org/abs/2305.04764

[37] J. de Lara, A. del Pozzo, E. Guerra, and J. Sánchez Cuadrado, "Automated end-to-end testing for conversational agents," *J. Syst. Softw.*, vol. 233, no. C, Jun. 2026. [Online]. Available: https://doi.org/10.1016/j.jss.2025.112685

[38] P. C. Canizares, D. Ávila, S. Perez-Soler, E. Guerra, and J. De Lara, "Coverage-based strategies for the automated synthesis of test scenarios for conversational agents," ser. AST '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 23–33. [Online]. Available: https://doi.org/10.1145/3644032.3644456

[39] S. Bergsmann, A. Schmidt, S. Fischer, and R. Ramler, "First experiments on automated execution of gherkin test specifications with collaborating llm agents," in *Proceedings of the 15th ACM International Workshop on Automating Test Case Design, Selection and Evaluation*, ser. A-TEST 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 12–15. [Online]. Available: https://doi.org/10.1145/3678719.3685692

[40] A. Jain, P. Aggarwal, R. Sahay, C. Dong, and A. Saladi, "AutoEval-ToD: Automated evaluation of task-oriented dialog systems," in *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, L. Chiruzzo, A. Ritter, and L. Wang, Eds. Albuquerque, New Mexico: Association for Computational Linguistics, Apr. 2025, pp. 10 133–10 148. [Online]. Available: https://aclanthology.org/2025.naacl-long.508/

[41] Y. Li, P. Liu, H. Wang, J. Chu, and W. E. Wong, "Evaluating large language models for software testing," *Computer Standards Interfaces*, vol. 93, p. 103942, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920548924001119

[42] G. Fraser and A. Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 416–419. [Online]. Available: https://doi.org/10.1145/2025113.2025179

[43] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," ser. NIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2023.

[44] Z. Liu, Y. Feng, and Z. Chen, "Dialtest: automated testing for recurrent-neural-network-driven dialogue systems," ser. ISSTA 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 115–126. [Online]. Available: https://doi.org/10.1145/3460319.3464829

[45] S. Bravo-Santos, E. Guerra, and J. de Lara, "Testing chatbots with charm," in *Quality of Information and Communications Technology*, M. Shepperd, F. Brito e Abreu, A. Rodrigues da Silva, and R. Pérez-Castillo, Eds. Cham: Springer International Publishing, 2020, pp. 426–438.

[46] P. Gómez-Abajo, S. Pérez-Soler, P. C. Cañizares, E. Guerra, and J. de Lara, "Mutation testing for task-oriented chatbots," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 232–241. [Online]. Available: https://doi.org/10.1145/3661167.3661220

[47] M. Vasconcelos, H. Candello, C. Pinhanez, and T. dos Santos, "Bottester: Testing conversational systems with simulated users," in *Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems*, ser. IHC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3160504.3160584

[48] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q.*, vol. 28, no. 1, p. 75–105, Mar. 2004.

[49] H. Husain and S. Shankar. (2025, Sep.) Building eval systems that improve your ai product. [Online]. Available: https://www.lennysnewsletter.com/p/building-eval-systems-that-improve

[50] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond accuracy: Behavioral testing of NLP models with CheckList," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds. Online: Association for Computational Linguistics, Jul. 2020, pp. 4902–4912. [Online]. Available: https://aclanthology.org/2020.acl-main.442/

A Literature Search Strategy

This appendix lists the search queries and keyword combinations used to identify relevant literature for the thesis. The literature search was conducted using major scholarly databases in computing and software engineering, including ACM Digital Library[10], IEEE Xplore[11], Elsevier ScienceDirect[12], and SpringerLink[13]. In addition, arXiv[14] was used when relevant literature was limited in the other reputable sources.

- "LLM agent" OR "autonomous agent" AND (observability OR tracing)
- "LLM agent tracing"
- "monitoring and observability in multi-agent systems"
- "AI agents visibility"
- "automated test case generation"
- "LLM unit test generation"
- "chatbot eval generation"
- "AI evals"
- ("test oracle" OR "oracle generation") AND (LLM OR "large language model")
- (chatbot OR "conversational AI") AND (evaluation OR "test oracle" OR "evaluation criteria")
- "LLM" AND ("software testing" OR "automated testing") AND (generation OR evaluation)

[10] https://dl.acm.org/
[11] https://ieeexplore.ieee.org/
[12] https://www.sciencedirect.com/
[13] https://link.springer.com/
[14] https://arxiv.org/

B Eval Generation Prompt

The following listing presents the final version of the prompt used for LLM-assisted eval generation. The prompt was initially developed as a basic instruction for generating evaluation criteria from traces and was subsequently refined programmatically over multiple iterations. Expert feedback on rejected evals was used to identify weaknesses in the generated criteria and improve the prompt accordingly. This iterative process required only lightweight human intervention, with experts primarily validating generated evals rather than manually authoring the evaluation criteria.

The prioritization rules in the prompt were deliberate design choices for ZED. The evaluation suite runs weekly through the CI/CD pipeline. Forecast and recommendation questions were given highest priority and evaluated only using Forecast Validation, as the objective was to validate the correctness of the forecasting or recommendation pattern rather than impose strict numerical or time-based criteria. These responses are difficult to evaluate using strict numerical, time-period, or format-specific criteria because the expected answer can depend on when the evaluation is executed. For example, a forecast or recommendation requested today may produce a different answer when the same question is evaluated the following week, as the underlying data, relevant time period, and resulting forecast may have changed. Therefore, forecast and recommendation questions are evaluated only using the Forecast Validation category. Similarly, for relative-time questions, time-period checks are excluded because expressions such as “last week” or “next week” refer to different calendar periods across weekly CI/CD runs, making hardcoded time-period criteria unsuitable. All other generic questions are evaluated against all applicable categories.

```
model: gpt-5.4-mini
prompt_template: |
   You are an evaluator that generates rubric criteria for tests,
   grouped by category. The trace details are provided as the input
   file_content.
   The JSON content is:
   {{ file_content | tojson(indent=2) }}
   Your task:
   Generate rubric criteria grouped by relevant categories.
   The goal is to identify what should be evaluated in test cases
   based on the provided JSON.
   Only use information explicitly present in the JSON fields ---
   do not infer or fabricate information.
   Provide each category only if applicable.
   Do not mention the source field names in the criteria.
   Always generate the criteria in English.
    Step 1: Apply Category rules:
   - Metric Selection:
      Identify the metric and use its exact human-readable display name.
   - Filtering:
      Identify and extract the applied filters. If none are applied,
      state that no filters should be applied.
   - Grouping:
      Identify groupings from the global-level filter configuration.
      If no grouping is present, state that no global grouping is applied.
   - Time Periods:
      Derive criteria from fields specifying time ranges. For comparisons,
      specify the current and comparison periods separately.
   - Chart Configuration:
      Include chart-level filters and groupings.
   - Numerical Accuracy:
      Include criteria reflecting quantitative relationships or trends.
      Prefer checks implied by the query over exact totals.
      Observation criteria should verify facts from fetched data.
      Response claims should be consistent with observation data.
      Avoid criteria requiring derived percentage calculations.
   - Forecast/Recommendation Validation:
      Apply only when the question asks for forecasts, recommendations,
      advice, or predicted values.
   Omit any category for which no valid criteria exist.
   " Step 2: Return the final JSON"
   Return only valid JSON with the applicable categories.
   Do not use markdown, code fences, or backticks.
   "Important --- category priority rules:"
   "1. Forecast/Recommendation rule:"
      If the question asks for a forecast, prediction, recommendation,
      advice, or future guidance, return only Forecast Validation.
   "2. Relative time period rule:"
      For references such as last week, last month, or previous period,
      return only Metric Selection, Filtering, and Grouping.
   "3. Normal rule:"
      Otherwise, generate all applicable categories normally.
```

Figure 16: Final version of the LLM prompt used for evaluation-criteria generation.