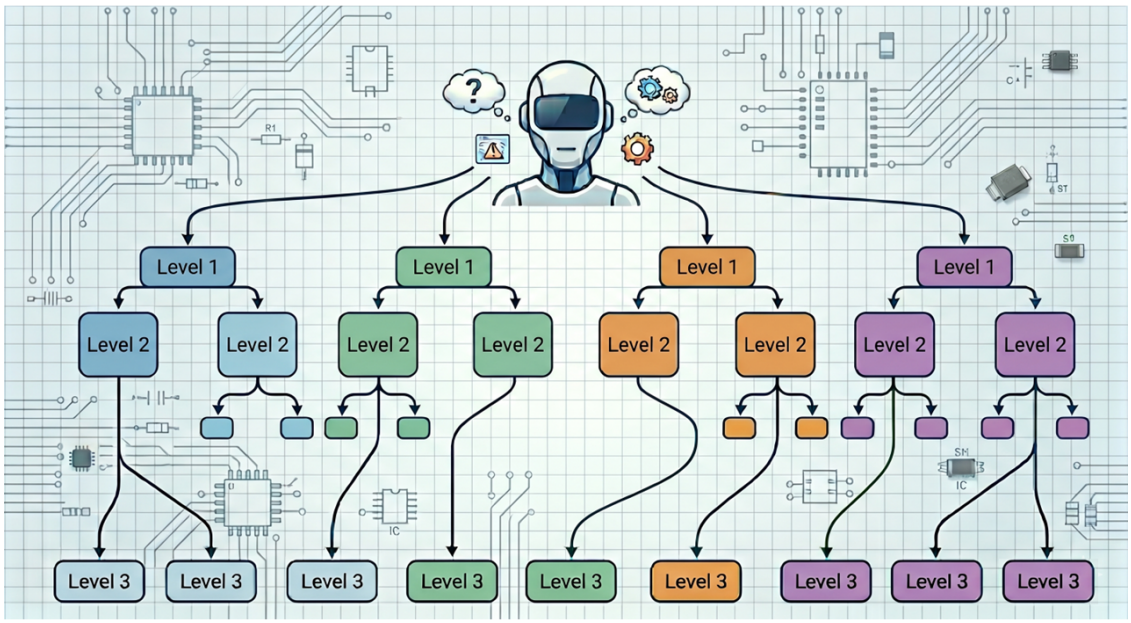
CHALMERS



From Alarms to Root-Cause: Autonomous Hardware Diagnostics through Multi-Agent Systems

A feasibility evaluation and proof-of-concept for LLM based diagnostics

Master's thesis in degree program Data Science & AI

ISAK SIXTEN

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's Thesis 2026

From Alarms to Root-Cause: Autonomous Hardware Diagnostics through Multi-Agent Systems

A feasibility evaluation and proof-of-concept for LLM based diagnostics

Isak Sixten

CHALMERS

Department of Mechanics and Maritime Sciences
Chalmers University of Technology
Göteborg 2026

From Alarms to Root-Cause: Autonomous Hardware Diagnostics through Multi-Agent Systems
A feasibility evaluation and proof-of-concept for LLM based diagnostics
Isak Sixten

Supervisor and Examiner: Marco L. Della Vedova, Department of Mechanics and Maritime Sciences
Advisor: Mathew Goonewardena, Ericsson

Master's Thesis 2026
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover image: A robot thinking through a problem building a tree of thoughts. Generated using Google's Nano Banana Pro.

Typeset in LaTeX
Gothenburg 2026

From Alarms to Root-Cause: Autonomous Hardware Diagnostics through Multi-Agent Systems
A feasibility evaluation and proof-of-concept for LLM based diagnostics
Isak Sixten
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

The rapid adoption of large language models (LLMs) and by extension LLM agents is transforming the way work is done across various industrial sectors. Although systemic performance increases are promised across many white-collar domains, certain areas that could leverage the technology remain largely unexplored. Hardware diagnostics, and by extension root-cause analysis are critical examples within the telecommunications sector, where clients rely on high uptime, network availability, and performance.

Modern telecommunication networks generate large volumes of daily log data that can be leveraged for diagnostics. However, proactive manual human analysis of this data is tedious, time-consuming, and unscalable across large fleets of units. This thesis investigates the feasibility of autonomous LLM agents in this specialized domain, evaluates these inherently non-deterministic workflows, and assesses how custom investigation structure influences investigation behavior.

This thesis proposes Autonomous Diagnostics Agent (ADA), a multi-agent system utilizing a hypothesis-driven investigation structure. ADA employs a dynamic planner that builds a hypothesis tree of different plausible root-causes for a given hardware unit, and autonomously explores and gathers evidence for each branch, while verifying key insights in the logs. To this end, ADA coordinates a fleet of subagents, specialized to tackle knowledge retrieval or log analysis.

ADA has been evaluated through qualitative domain expert feedback and through a small automated test suite with known historical cases. The results show that while ADA can provide utility mostly in-line with domain expert expectations for well-documented issues, ADA also demonstrates multiple failure modes. The investigation structure has an observable impact on the exploration breadth but results in increased token usage, without any major correctness increase for the test cases tried. Despite these challenges, domain-expert feedback suggests that ADA provides a useful diagnostic baseline that helps save engineering triage time.

Ultimately, this work highlights that the overarching bottleneck for enterprise LLM agents in complex engineering domains is evaluation. Since the number of possible answer-solution pairs is vast, evaluation becomes reliant on proving stability in outputs, and faithfulness to ground truth documentation and logs.
Keywords: Agentic AI, LLM, RCA, MAS, Autonomous, Diagnostics, Reporting.

Acknowledgements

Firstly, I would like to extend sincere gratitude to the whole team at Ericsson for the trust, insightful discussions, technical help, availability and feedback received throughout the master thesis project. In particular, I want to thank my manager Adam Pirzada and my industrial supervisor Mathew Goonewardena for trusting me in this endeavor and for providing the opportunity to work in an exciting field.

Moreover, I also want to thank my supervisor & examiner from Chalmers Marco L. Della Vedova, who took on an industrial project with only one thesis student, for the trust, candid and pleasant conversation, encouragement and guidance along the thesis.

Finally, I would like to thank everyone in the wider Ericsson ecosystem who contributed to the progression of the thesis work, whether large or small.

Isak Sixten, Gothenburg, June 2026

Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

CSR	Customer Support Request
HW	Hardware
KB	Knowledge base
LLM	Large-language model
MAS	Multi-Agent system
RAG	Retrieval Augmented Generation
RCA	Root-cause analysis
SME	Subject matter expert
SotA	State of the Art
SW	Software
TR	Trouble Report

Contents

List of Figures

List of Tables

1
Introduction

LLM-based agentic AI has garnered significant attention across various industries, due to its capability of potentially disrupting major sectors, particularly within white collar work roles, such as software engineering, legal and administration. While benchmarks deserve some scrutiny, AI systems have gone from solving 4.4% of problems in SWE-Bench during 2023 to 71.7% during 2024 highlighting the significant capability increase over a short period of time [1]. The rise of LLMs and their exponential growth has changed the day-to-day procedure of certain work roles by a significant amount already, but there still exist plenty of underexplored domains that could leverage the technology.

One such domain is hardware root-cause diagnostics. Software root-cause analysis is inherently a more strong use case for LLM agents, due to the data analyzed being code which most LLMs, have been explicitly trained for. The code itself is also all encompassing for describing the issue of a program unlike for hardware where a mix of factors combined can produce issues. Several papers have discussed potential architectures for diagnostics in a software environment driven by the development of benchmarks for the software scenario, such as OpenRCA [2]. One such architecture is presented by the OpenRCA authors themselves, but other architectures also exists, such as the work done at Microsoft by Roy et al. for enterprise cloud data RCA [3].

Even with the desirable properties of software, RCA systems built with LLMs struggle in their analysis. Kim, et al. explore and set up a taxonomy for failure modes on the OpenRCA dataset. Examples of such failure modes include hallucinations, incomplete exploration, defining a symptom as cause, and limited source usage [4].

For an engineer to accurately diagnose telecom hardware, software or external failures, a tedious and time-consuming multi-step process needs to be conducted. High-level it's a four-step procedure of determining how many units are affected, what caused the error (e.g. a certain manufacturing batch), proposing solutions for said error and finally deciding how the faulty unit should be handled. But in practice, each step is a highly complex workflow with many hidden ambiguities. Different error modes with different solutions conflate in the logs, and certain errors require an actual on-site visit or return to determine the cause. At scale it's highly unfeasible to allocate man-hours for pro-active discovery of such errors unless the discovery of an issue can be boiled down to a diagnostics rule.

The optimal architecture and techniques for agentic AI systems is a new and evolving field of research because the recently introduced tools for creating these systems are still maturing. In specialized domains, the literature is sparse or non-existent proving the need for applied research examining agentic architecture within highly complex environments, since the requirements and constraints are different compared to static benchmark evaluation often used in a research setting.

While agentic AI products often follow the turn-based conversation structure or "natural-language interface" for human-computer interaction popularized by OpenAI in ChatGPT [5], this thesis will explore the limits of autonomous agents for hardware diagnostics, without any explicit guidance by a user during their investigation, effectively conducting a first triage for further investigation by an engineer.

1.1 Background

Issues in telecommunications hardware can arise in an endless number of ways. For instance, it could be related to the manufacturing, the design process, software related issues, hardware degradation, site issues such as power cabling, problems with shielding, weather related incidents and so on. The initial trigger for investigation depends on how the fault was discovered. Proactively collected data allow for discovery prior to fault manifestation, by scanning for issues, such as by looking for specific patterns that act as a precursor to failure in logs. In other cases, the discovery may require more information, which may only be received on unit return or escalation from the user.

At their disposal to provide a verdict based on this log data, engineers have their own domain expertise and also the abundant corpus of unstructured and structured internal information such as investigative reports, previous customer issues, internal documentation and troubleshooting guidelines. Most of this documentation is text-based and as such the work lends itself quite well to analysis and usage by LLMs. Furthermore, an ecosystem of digitalized sources, lends itself well to agents, and their capability to autonomously retrieve information when needed and execute program code to analyze hardware units.

The volume of available data from hardware units is substantial, but most of the data sources relevant for analysis exist through a combination of structured and unstructured data storage systems. Agents are typically leveraged via a chat interface, but ideally no human input would be required for diagnostics, which is what this thesis aims to examine.

A compelling argument for this shift in autonomy is that agents with the right documents can reach satisfying answers with active guidance. If the required knowledge for diagnosing a problem is ingested, a similar investigation procedure could be possible to recreate without human intervention.

The non-deterministic nature of LLM's and by extension agents, poses a credi-

bility threat to any system designed with these models as a backbone, which is something that has been considered during the development process and will be discussed throughout the report.

This thesis will approach this problem in an exploratory and pragmatic way. Rather than focusing on the actual mechanisms of how the LLM functions on a low level in this space, it will evaluate what benefits and limitations exists in solving this specific problem statement using LLM agents.

1.2 Aim

The aim of this thesis is:

- To investigate the practical feasibility of deploying LLM agents for hardware diagnostics in a large-scale enterprise context.

The thesis investigates three research questions to achieve this aim:

R1: To what extent can a multi-agent LLM system autonomously perform proactive / first-triage root-cause analysis for telecommunications hardware?

R2: What aspects should be evaluated in an inherently non-determinstic workflow in this domain?

R3: To what extent does explicit investigation structure scaffolding affect agent behavior in proactive hardware diagnostics?

1.3 Delimitations

Firstly, this thesis does not investigate the trigger mechanisms for proactive hardware diagnostics. It will not focus on designing an architecture for scaling these investigations across the fleet, but the constraints and possible solutions is something that will be discussed.

Secondly, the thesis does not investigate whether the system proposed here would have transferable results to other similar domains.

Finally the thesis does not aim to investigate whether different LLM backbones, fine-tuning or considerably different architectures provide better results in this setting.

1.4 Thesis Organization

This thesis is structured as follows. Chapter 2 introduces the necessary background theory on LLM-based agentic systems. Chapter 3 details the proposed architecture and the research methodology. Chapter 4 presents the experimental results,

incorporating both domain expert feedback and quantitative LLM-as-a-judge evaluations. Chapter 5 analyzes the research findings, addresses the study's limitations, and outlines directions for future work. Finally, Chapter 6 summarizes the main conclusions of the thesis.

Additionally, the specific prompts utilized by the LLM-as-a-judge evaluators are compiled in Appendix A.

2
Background theory

In the following sections, some theory useful for understanding the design choices and implementation of the agentic system is introduced.

2.1 Case-based reasoning (CBR)

Case based reasoning is a problem solving paradigm that can be used in intelligent systems or AI that can be summarized as the use of prior examples of problems and solutions as the main driver of new solutions to unseen or similar problems [6].

An example of such a problem solving procedure is a physician examining a patient and recalling that two weeks ago a patient with similar symptoms walked into the clinic. Based on the diagnosis made previously the physician quickly concludes the diagnostic and treatment can be prescribed for the new patient [6].

The conceptual flow of reusing case knowledge to solve new problems is categorized by Aamodt and Plaza [6] as consisting of the following four steps:

1. Retrieval: Retrieval of the most similar cases to the problem at hand.
2. Reuse: Reuse the knowledge and information in those cases to solve the new problem by adapting the solution.
3. Revise: Evaluate the new solution and revise it if it doesn't fit the new problem.
4. Retain: Store the parts of the process that could be useful in the future into the knowledge base.

The case itself, can be stored both as a monolithic entity or split into parts, distributed across some knowledge source. Cases could be contained to one specific concrete instance, or it may encompass multiple cases that share attributes. The case itself can be used directly to solve a given issue or the difference between the current problem and the retrieved case could in and of itself be informative in providing a solution. Aamodt and Plaza describe one solution as a deep model of general domain knowledge, being able to match cases, adapt solutions and learning from the experience [6].

Aamodt and Plaza also highlight the possibility for a relative amount of autonomous capability in CBR systems, which can both be deployed as completely self-contained

or reliant on user guidance [6].

The advent of LLMs and by extension the agentic possibility by means of structured output has enabled a clear cut realization of the ideas described by Aamodt and Plaza in 1994. Certain research works explore directly how CBR as a methodology can be incorporated into LLM agents. Guo et al. [7] propose DS-Agent, an LLM agent architecture that solves data science tasks, using a rich history of expert knowledge from Kaggle. Similarly Wiratunga et al. [8] propose CBR-RAG for the answer of legal questions using a case-bank of question and answer pairs created from a legal dataset.

2.2 Retrieval Augmented Generation (RAG)

RAG or Retrieval Augmented Generation is a solution to a critical issue with natural language models. The core tension is that while pre-trained transformers learn an impressive amount of concepts and structure from data, there are downsides to the original approach. For instance, if memory needs to change, this necessitates retraining, and there is no straightforward way to cite or refer to the source of the information as the knowledge is embedded latently in the weights of the layers in the neural network. [9]

In an enterprise setting the problem is even more clear. There exists no straightforward way to incorporate internal / confidential knowledge into an LLM by fine-tuning, without creating several security concerns depending on the use case. The only way to enforce deterministically that an LLM doesn't divulge privileged information to someone who shouldn't have access, is by omission [10].

RAG solves this issue while enhancing performance in a clever way. Instead of relearning the parameters of a natural language model, the knowledge can instead be retrieved on demand. In the original paper Lewis et al. propose two distinct methods for the injection of the knowledge: RAG-Sequence and RAG-Token. RAG-Sequence functions by for each document generating a response separately, i.e. the original user query and document. The selected sequence for the final answer is the sequence which maximizes the joint likelihood of retrieval and generating the sequence given the document. For RAG-token each token is generated one at a time for each of the input and document prompts, and for each position in the output sequence the next token chosen is the one which has the highest probability in the weighted probability distribution of all document augmented prompts. [9].

A RAG production system usually functions in a more streamlined manner. A retriever fetches information from a database and directly injects it into the prompt as corroborated by the three main cloud providers [11][12][13].

There are various methodologies for improving both the retrieval itself, as discussed by Gao et al. [14], who declares the above "Naive RAG". Examples of such methods include using pre- and post-retrieval mechanisms such as routing, enhancing data

granularity, reranking, and fusion.

Similarly while RAG is usually associated with the retrieval of text without explicit relational modeling, there exists a subset of RAG associated with graph representations, such as GraphRAG which models text chunks with explicit relationships, to facilitate better retrieval and more context aware responses, due to entities (such as related concepts) having explicit relationships [15].

2.3 LLM Agents

The foundational structure of LLM agents seen today is largely a result of the work by Yao et al. to define the Reasoning + Acting (ReAct) methodology. The idea is simple, instead of only requiring that the agent take an action after seeing an observation, actions are augmented to encompass the space of language. A thought action (effectively output which does not receive external feedback), is the LLM reasoning over current context, to generate new context useful for future reasoning or actions. Yao et al. implement this method using in-context learning, specifically few-shot examples in prompt, with a strong model backbone, and this method yields better performance compared to acting only. Effectively showing reasoning can guide acting. [16]

Another breakthrough paper in the space of modern agentic AI, is the work done by Schick et al. in defining ToolFormer, which is one of the first papers to examine how LLMs, can interact with external tools such as APIs to bypass their innate struggles with for instance, simple arithmetic. By fine-tuning a model to generate output interleaved with API calls (in the form of an explicit token), it was shown that the smaller GPT-J model could outperform GPT-3 on various benchmarks for math, QA and multilingual QA [17].

As described in the large survey paper by Wei et al. agentic reasoning is the central mechanism of intelligent agents, created via in-context orchestration or post-training optimization. Wei et al. describe several different mechanisms that together compose agent reasoning: foundational capabilities, self-evolving adaptation, and collective coordination. Relevant for this thesis in particular, is the planning capability, which Wei et al. divide into subgroups depending on how its realized in agents [18].

One of these subgroups is workflow design, which is characterized by deliberately separating different steps in planning and execution. The workflow governs what can be done and when, while there is also a reactive loop where the agent provides grounding and error recovery. Wei et al. point out that while effective, structures of the workflow type can exacerbate errors over long horizon tasks. [18]

2.4 Test Time Scaling

The behavior of LLM's that is referred to as "test-time scaling" wherein the LLM performs better under larger compute resource allocation during inference time, i.e.

generating more tokens, is a key piece to LLM capability. Snell, et al., explore how test time scaling affects task completion. [19]

The first method tested by Snell et al. uses a verifier or process-based verifier which predicts the best solution or predicts the intermediate solution correctness score respectively. The actual search method is split across three different types:

1. **Best-of-N weighted**: Sample N answers from the LLM and select the best answer according to verifier.
2. **Beam Search**: Sample intermediate step proposals, score with verifier, and continue with the $\frac{N}{M}$ best candidates.
3. **Look-ahead search**: Sample intermediate proposals, perform a k-step roll out from each sample, score and continue with best options. Beam search is look-ahead search with k = 0.

The second method Snell et al. employ, is through fine-tuning the language model to coerce it into revise its solutions in an iterative manner.

The main takeaway is that for lower difficulty questions, within a model's capabilities test-time compute can make up for missing pre-training. For more challenging questions pretraining is more effective. For questions, where a smaller base model can do reasonably well, it can outperform larger models using a test-time scaling approach. [19]. Effectively meaning that there is an upper boundary on what the LLM can reasonably process given its size.

Muennighoff et al. instead describe a method for test-time scaling not explicitly dependent on sampling executions. First Muennighoff et al. fine-tune Qwen2.5-32B-Instruct on a curated dataset of challenging questions. The mechanism they use for enforcing test-time scaling, is by enforcing a minimum and maximum number of thinking tokens. The minimum is enforced by catching the end of thinking token, and instead appending "Wait", from which the model might choose to continue in a manner such as "Wait, let me reconsider the original question". Effectively encouraging the model to reflect in its current generation. For the maximum, the end-of thinking token is instead appended forcing the model to give a final answer based on the current thoughts [20].

The empirical result of Muennighoff testing is that question answering accuracy scales with time spent "thinking" using budget forcing when using the fine-tuned model. Three other baselines were compared to budget forcing, e.g. telling the LLM for how long to think in the prompt, which all had worse performance than budget forcing [20].

In LLM agents this test-time scaling property takes a different form. As Li et al. argue, there are two paradigms for creating test-time scaling in LLM agents, parallel scaling and sequential scaling. Parallel scaling increases breadth of explo-

ration contrary to sequential scaling which increases depth. Parallel scaling works by sampling multiple trajectories for each query, and having the agent assess each trajectory on its own. Sequential scaling instead increases computational depth by extending the interaction horizon by imposing an additional round of environment feedback when the agent attempts to terminate, [21].

Different model providers were evaluated against a custom benchmark, consisting of coding, search, tool-use and reasoning tasks. Coding evaluates how well the agent analyses software issues and how it interactively interacts with the environment to reach a correct final state. Search evaluates how well the agent identifies missing information and deciding on whether additional search steps are needed. Tool-use evaluates agent capability to coordinate tool usage in a large tool suite. Reason evaluates how well the agent can find needle-in-a-haystack type information in a noisy long context [21].

The conclusion reached is that for some domains such as reasoning, sequential scaling does not improve performance, and for some such as search show initial improvement until a certain threshold in the context window is achieved [21].

2.5 LLM Agent Evaluation

Evaluating agents poses unique challenges compared to evaluating other types of traditional machine learning systems or software systems. The agency factor or the capability of an agent to interact with its environment, requires evaluation to go beyond observing only the LLM textual outputs, to assess how well the agent navigates in its environment.

Yehudai et al. [22] categorize several existing benchmarks into a few different dimensions: capability, application specific and generalist evaluations along with specific frameworks designed for evaluation. Yehudai et al. also highlight the trend of designing evaluations which is challenging and therefore realistic and also live benchmarking during execution [22].

Anthropic themselves describe multiple ways of evaluating agents in production systems, such as through automated evals, user feedback and systematic human grading. Notably there are pros and cons with every type of evaluation, such as automated evaluations requiring time up front, and needing regular updates to account for product or context drift. [23]

For evaluating RAG-systems, Es et al. define Retrieval Augmented Generation Assessment (Ragas). It consists of three different evaluation metrics:

1. **Faithfulness** - How grounded is the answer, i.e. how many statements in the final output can be inferred from context?
2. **Answer relevance** - How relevant is the answer to the original user question?

3. **Context relevance** - How relevant is the context received to answer the user question?

The main benefit of these evaluation metrics is that they can assess RAG-systems without being dependent on having ground-truth answers available. For instance, faithfulness is assessed as described by Es et al. by extracting statements from the final output with an LLM, and checking whether the statements are backed by the context used when generating the final report using an LLM. The faithfulness score is then computed as

$$F = \frac{|V|}{|S|}$$

where |V| is the number of verified statements according to the LLM and |S| is the total number of statements. [24]

However there are many challenges associated with the LLM-as-judge methodology. As Gu et al. discuss [25], a primary concern is the reliability of the method itself since, for instance, a typical LLM is very sensitive to changes in the prompt and being biased towards their own responses, if trained with RLHF.

3
Method

The method chapter will outline the problem structure, define the data and intended flow types, the implementation methodology and architecture and finally the evaluation methodology.

3.1 Problem structure

The most notable difficulty with evaluating a chat agent is what the evaluation should cover. Questions can have different granularity, scope and require different resources.

In an effort to simplify the empirical aspect of the thesis, input and output were fixed to emulate two flow types. Input is either a unit serial directly or a diagnostic snapshot attached with some optional additional context and output root-cause of the error. Specifically, the output is primarily a prioritized list of potential candidate root-causes, where the ideal goal is that the agent should be able to pinpoint the cause exactly but also provide good evidence for its conclusions. In this thesis, root-cause refers to an established mechanism of failure for a known issue relevant to a certain group of units. This simplification is motivated, primarily since the goal is to evaluate the feasibility of implementing a proactive diagnostics system for unit hardware failures.

The thesis considers two scenarios where this type of analysis is motivated. Proactive analysis and CSR (Customer support request) prompted analysis. These constitute two separate flows, which require different triggers, analysis and constrain the search space to different data.

3.2 Data

This section will discuss the core entities necessary to understand the diagnostics context and the main sources of data used by the system for analysis of a given unit, split across the two flows, concluding with the common data.

3.2.1 Core entity description

The core entities relevant to understand the following chapters will be described from a high-level perspective here.

- **Alarm** - A fault alarm raised by the radio unit itself.
- **Radio unit** - A radio, can be of different types.
- **Event-driven Log** - A log collection gathered for one unit when an alarm is raised on the radio unit.
- **Service Log** - A collection of logs collected for a specific unit prior to repair which guides repair actions.
- **Diagnostics Snapshot** - A comprehensive log collection containing logs from all hardware components at a single deployment site.
- **Case Document** - A historical issue case, including information about the status, the resolution, and associated information.
- **Root-Cause Document** - A document detailing an in depth investigation for a failure mechanism.

This list is hardly encompassing, but nonetheless provides a brief description of the most relevant concepts used in the remainder of this thesis.

3.2.2 Proactive Analysis

For proactive analysis, the case considered is an alarm being triggered in a unit in a resulting in a event-driven log for the unit. Some test cases additionally relied on service logs. The main reason for some test cases relying on service logs, is that event-driven logs are only collected when specific alarms are fired and the data being more restricted in certain cases.

Conceptually, this type of execution could be triggered in an event-driven manner.

3.2.3 Customer Support Request (CSR)

Compared to the proactive analysis a customer support request investigation typically operates on more information than an event-driven log. The main input in this case is a diagnostics snapshot as well as a general description of the observed issues, which is useful context to start an investigation.

3.2.4 Common across flows

Regardless of diagnostic setting the system also relies on information about the unit itself (e.g. type) is included in both types of investigation.

The system also has access to aggregated information across multiple units through accessible data sources. Along with this there also exists multiple other data sources, containing other information such as design documentation, which in certain cases can be used to provide additional information, should the investigation require it.

3.3 Implementation

This section will briefly walk through the implementation methodology which gives some background to the results as well as the following section proposing the architecture for ADA.

3.3.1 Iterative problem formulation

The original ambition for the thesis was to evaluate the possibility of letting the existing chat agent work in a more "deep-research" like manner such as the capability of OpenAI's deep research [26]. There is some merit to this idea, but in a chat interface, the possible query space is endless, and the tangible value of such an approach is specifically in the cases where this methodology is warranted.

Autonomous diagnostics has a clear value contribution, and, this was formulated as the intended goal of the thesis. The task of identifying potential root causes for observed faults is a structured task with an expected output (in some cases).

The formulation and development has largely been iterative, where additional tools, heuristics and patterns were incorporated based on discussions with domain experts and testing. A factor influencing the project, is that the surrounding technical environment continued evolving during development, in some sense shifting the ground beneath the feet of the project. This could be controlled in the LLM-as-judge evaluation defined in Section 3.4.1. because the evaluation code could be frozen, but it affected the domain expert evaluation defined in Section 3.4.2. since the users were given direct access to the system through a user-facing interface within an experimental deployment environment.

Changes in available functionality were incorporated into the project over time, which all together contributed to the thesis outcomes.

3.3.2 Architecture

The proposed workflow is inspired by the Aime architecture set up by Shi et al. [27]. The main component of interest is the progress management utility which has instead been customized for this domain as a hypothesis tree.

Overall the workflow structure in Figure 3.1 is designed to simulate the workflow of a support engineer. Starting from an observation, gathering an initial context about what has gone wrong, to autonomously generating hypothesis regarding the potential failure mechanism, fetching information from the knowledge base (which would be experience in the human analogy), confirming or ruling out all hypothesis and finally providing a verdict based on the evidence to the concerned party.

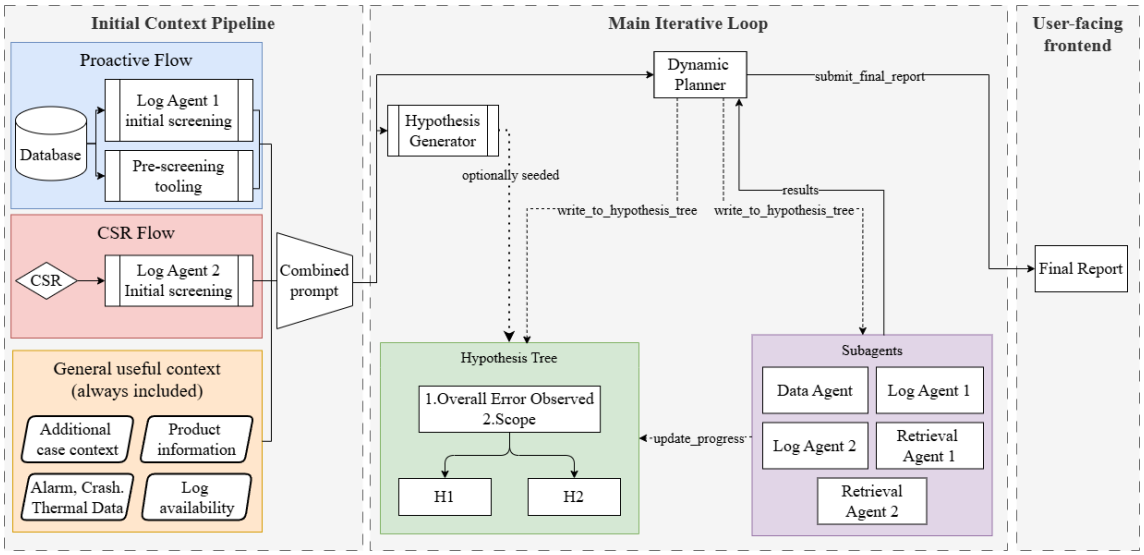


Figure 3.1: The overall architecture of Autonomous Diagnostics Agent (ADA)

Autonomy in the context of ADA does not imply completely unconstrained operational agency. Instead it refers to the agent workflow after initialization with a a failure signature, where the agent autonomously explores possible solutions and constructs its own hypothesis without any human-in-the-loop intervention.

The overall components of the workflow consist of the "initial context pipeline" which is the initial context retrieved about the unit, "generate hypotheses" which is an explicit step for the initial hypothesis tree, the dynamic planner itself (or ADA), the hypothesis tree, the subagents which ADA can invoke, and the final report output.

The iterative progression of evidence gathering and verification functions as an in-context operationalization of Case-Based Reasoning (CBR) without the retention step. Historical documents serve as the case-bank from which the system derives solutions for new issues. The retention step while certainly interesting, would require evaluation over a longer time-frame to see how the system developed and was therefore chosen to not be considered. The retain step in particular will be discussed as a possible future exploration in the discussion.

3.3.2.1 Initial Context Pipeline

For the dynamic planner which doesn't have any ability to directly fetch context on its own to understand the issue under investigation it has to receive some initial context. The initial context given to the agent is determined by the flow type. In the proactive flow, the latest log available is targeted, and an initial health screening is done by a log agent. In parallel with this, existing classification methods to determine the problem category is executed for the targeted log. While usually indicative of subsystem affected, the verdict is typically not conclusive in terms of the actual root-cause, but gives some guidance in terms of where to look.

For the CSR flow, the initial context is a health screening of the node, with a focus on serials with an active alarm, done by by the other log agent. The initial context provided in the CSR along with the log agent report is provided to the dynamic planner.

For both flows, unit specific metadata is retrieved, to help in the subsequent investigation. The content retrieved is combined into a larger context prompt based on the dynamic retrieved context and static information defined in advance, such as headers.

The combined context from all sources is packaged as a set of messages, which is appended to the dynamic planners message state.

3.3.2.2 Hypotheses Generator

After the initial context pipeline has completed an optional step is executed to generate the initial hypothesis in the tree.

The merged initial context prompt is summarized via an LLM invocation and passed to a retrieval agent which creates an initial tree after retrieval, in a free form report. A third LLM call is performed, and the proposed hypothesis tree is fit to the agent state using a tool with a specific input schema, effectively seeding the hypothesis tree, based on retrieved knowledge.

Although the dynamic planner is capable of generating a hypothesis tree of its own, one hypothesized caveat with this, is that the dynamic planner can't do direct knowledge retrieval. Therefore, to set up a tree structure it needs to rely only on the information in the initial context and the system prompt.

R_1 has access to multiple relevant documents for confirming a diagnosis in a radio unit, and as such, generating a hypothesis tree using this subagent directly could result in a more high coverage grounded tree. The generated tree is a base which the dynamic planner can later amend.

The possibility of generating hypotheses up front will be empirically assessed as part of the ablation to answer question ***R3***.

3.3.2.3 Dynamic Planner

The dynamic planner centralizes the investigative state, while delegating defined retrieval or analysis tasks to dedicated subagents. This separation reduces context fragmentation by centralizing high-level context in the orchestrator and limits token accumulation by isolating task-specific reasoning within independent agent-states. The planner maintains the global investigative context, including the evolving hypothesis tree and the artifacts produced by subagents, while each subagent operates within its own self-contained context window tailored to the assigned task.

The orchestrator supports both sequential and parallelized investigations and has access to three tools:

1. write_to_hypothesis_tree - Updates the hypothesis tree structure maintained throughout the investigation.
2. initiate_investigation - Delegates investigation of a specific hypotheses to an appropriate subagent.
3. submit_final_report - Produces the final structured diagnostics report to the user.

Architecturally, the orchestrator loop modifies the foundational ReAct methodology detailed in Section 2.3. Rather than maintaining reasoning traces solely as free-form textual output, the planner documents its progression and hypotheses through the structured planning state represented by a hypothesis tree. This enables persistent tracking of explored, active and eliminated hypotheses across the full scope of the investigation.

The orchestrator is intentionally lightweight and functions primarily as a coordination and decision making layer. Its role is not to perform specialized analysis directly, but to manage the progression of the investigation, with only three defined tools, and its only concern is to act as an orchestrator and final judge for the case based on the evidence provided during the investigation.

The structure of the dynamic planner's action space leads to small context window usage during execution, only containing the system prompt, tool descriptions and tool calls with tool responses, for either a subagent invocation or a write operation to the hypothesis tree.

The system prompt used in the dynamic planner mainly serves three independent goals

1. Define investigative objectives and rules to prevent unwanted behavior.
2. Specify the available actions and the subagent responsibilities.
3. Provide domain context regarding what constitutes a root-cause, available data sources, and relationships between data entities in this domain.

Along with this system prompt, additional behavioral guidance is found in the tool-descriptions for each separate tool. For example, the tool specifications define how the hypothesis tree should be updated and what information must be included in the final structured report.

3.3.2.4 Hypothesis Tree

Inspired by the progress management utility in Shi et al. [27] and the concept of sequential scaling [21] inference-time compute allocation as described in section 2.4

this thesis implements a structure designed to reinforce structured exploration during autonomous root-cause investigations.

The motivation of the tree is the ambiguity inherent to RCA workflows. To a large degree, hardware symptoms admit multiple plausible explanations simultaneously. Superficially convincing evidence can prematurely bias the investigation toward incorrect conclusions. Similar limitations exist in software root-cause analysis structure where agents may suffer from premature termination or incomplete exploration in complex environments [4].

Rather than allocating an additional inference-time compute budget through parallel trajectory sampling, the tree structure reinforces investigative depth by maintaining competing hypotheses in-context throughout the execution. The objective is therefore not only to increase reasoning length, but to prompt the agent to consider multiple plausible explanations in ambiguous investigations.

The tree externalizes the ongoing investigation into a persistent structured state. Each node represents a candidate root-cause at different levels of granularity. Level 1 typically represents the broad fault categories such as HW, SW, configuration or external / site issues. Level 2 typically represents the affected subsystem or software. Lower levels represent increasingly specific failure mechanisms.

The planner interacts with the tree through a structured tool input schema which allows for a specific set of operations, to guarantee that the tree remains cohesive and correctly formatted, while still allowing for revision, expansion or removal of hypotheses as new evidence is gathered.

In a traditional reasoning -> act -> observe workflow, exploration of solutions emerges only implicitly from the latent parameterization of the underlying LLM model. While exploratory behavior can be encouraged in the system prompt alone that would likely require up-front knowledge of the expected findings for a particular case, which is not the case in the diagnostics setting. Therefore, if enforcement is defined in the system prompt, these assumptions should hold generally or otherwise risk introducing undesirable prior assumptions that bias the LLM towards a certain solution regardless.

By externalizing avenues of exploration into a dynamic structure, at least a subset of potential investigation angles remain persistently visible throughout execution, functioning as a form of modular soft-enforcement for exploration. Since the hypothesis tree can be modified freely during execution, it is not considered restrictive in terms of action space, since the agent can at any time choose to add a new or rule out an existing hypothesis.

Several initialization strategies of the hypothesis tree exist. This thesis has explored dynamically creating the structure through the planner or the retrieval agents based on the initial context provided by the pipeline. There is some merit in exploring

whether these trees could be defined up front according to domain-expert best practices in the HW diagnostics workflow, but encoding the domain expert knowledge in a succinct, comprehensive manner with the fragmented expert areas of different SME was deemed unfeasible in the time frame.

Beyond this, the hypothesis tree provides a transparent representation enabling visualization of explored, active and eliminated hypotheses in the GUI.

The benefit of the hypothesis tree in terms of system performance will be assessed as part of the ablation study to answer question ***R3***.

3.3.2.5 Subagents

The subagents themselves has not been the creation or the main component of investigation for this thesis, but are nonetheless used as the main vehicle for the dynamic planner to gather context and evidence used to generate the final report. To avoid disclosure of implementation-specific details only a high level overview of the functionality of the respective agents will be provided here to give some context for how the dynamic planner gathers the required data to provide its prediction:

- **Data Agent (D)** - Provides data querying capability and can also execute code.
- **Log Agent 1 ($\mathbf{L_1}$)** - Provides analysis on log files, in particular event-driven / service logs.
- **Log Agent 2 ($\mathbf{L_2}$)** - Provides analysis on the larger diagnostic snapshots.
- **Retrieval Agent 1 ($\mathbf{R_1}$)** - Provides information grounded in a large knowledge base of internal documentation.
- **Retrieval Agent 2 ($\mathbf{R_2}$)** - Provides information primarily from case reports.

Generally, the above subagents can be divided into two groups. The first consists of the knowledge retrieving agents (R_1, R_2) whose main purpose is to fetch knowledge from the knowledge base and summarize the information to either the user or the agent invoking it, functioning effectively as the enablers of CBR. The second group consists of the analysis agents (L_1, L_2) whose primary purpose is the parsing and analysis of log material. The Data Agent falls somewhere in between these two groups, as it can fetch some knowledge (although technically static fields not rich textual content) but also has the ability to do analysis in the form of calculations and create visualizations.

The main reasons for the existence of the different subagents is in part a matter of use case, e.g. some users only requiring functionality from one of these and in part that different functionality is required for operation. I.e. these agents have different tooling and functionality to handle different tasks. In terms of the overall investigation structure, the multi-agent structure is also is crucial to lessen the burden of token accumulation across turns for any given agent.

The quality of the subagents directly affect the output quality of the full workflow, which necessitates some consideration to be taken. Therefore, some analysis will be done on the respective parts in the results to assess each parts contribution into the success or failure of the final report but the mechanics of how they work will not be considered a part of the thesis work.

3.3.2.6 Final report

Agentic workflows often output their final message in a free-form text, well suited for the underlying LLM model. For the diagnostics task, one consideration was whether the final output should follow a defined structure.

The proposed structure for the final report, iteratively developed via feedback from domain experts, has eight distinct categories:

1. **Investigation Tree** - The final hypothesis tree after the full execution
2. **Summary** - A high-level summary of the investigation
3. **Critical Findings** - The main symptoms found in the unit
4. **Root-cause candidates** - The candidate root-causes according to the investigation
5. **Ruled out hypotheses** - The ruled out hypotheses from the tree, with some further reasoning as to why they were ruled out
6. **Recommended next steps** - Agent defined prioritized list of next steps, based on the investigation
7. **Unexplained symptoms** - Symptoms that do not match the symptom profile of any given item in the candidate root-causes
8. **Investigative constraints** - Any considerations in terms of log completeness, unavailability of subagents or other mechanisms restricting the functionality of the system

Along with the categories described here the agent also provides a verdict which can be one of four different categories:

- **Known root-cause identified** - A known root-cause has been found and applies to this unit
- **Non conclusive** - Several known root-causes have been found but the agent can't ascertain which one it is.
- **Unknown Issue** - No known root-causes have been identified but there is evidently faults present in the unit.
- **No fault found** - There is no evidence suggesting anything is wrong with the unit.

Of these categories, the most directly relevant to assess is the root-cause candidates themselves, while the other categories provide auxiliary information that would be needed in a production setting to make the agent valuable. This means that the main topic of evaluation will be the root-cause candidates, but the domain expert assessment of all report categories is a key result to answer ***R1***.

The structured output format is assessed as part of the ablation study for ***R3***, similar to the hypothesis tree.

The structured report is rendered in a GUI, and the user can then ask follow up questions, potentially causing updates in the final report, but this falls out of scope for answering ***R1, R2, R3***.

3.3.3 Technical details

The full flow was implemented using Python and visualized in a GUI, allowing certain test users to interact with the agent. The framework used for the definition of the agent graph is LangGraph with its associated abstractions.

LLMs used in the agentic setup vary across modules, but are all part of the Anthropic model families. More specifically, the used models in the various parts of the multi-agent system are:

- claude-sonnet-4-6 [28]
- claude-haiku-4-5 [29]

Both general purpose LLMs, which could be used in the enterprise context.

3.4 Evaluation

The four different types of system failures in ADA from a high-level perspective are shown in 3.1.
This taxonomy will constitute the main framework of analysis for the failure modes observed in results, and possible mechanisms for improvement will be covered in the discussion.

While all failure modes are interesting to observe from the perspective of ***R1*** and ***R2***, the most interesting to observe for ***R3***, is procedural failures, assuming all artifacts required for reaching the right conclusion exist in the database.

Due to the very specialized nature of the domain, the evaluation structure was set up to mimic real deployment as closely as possible through two different evaluation methods designed to complement each other which are described in the following chapters.

Table 3.1: Taxonomy of possible diagnostic failures in ADA

Failure Type	Description
Epistemic Failure	A failure of the knowledge base. Occurs when the system does not possess or fails to retrieve the required domain information, documentation, or log data necessary to diagnose the issue.
Procedural Failure	A failure of the procedure. Occurs when the agent can retrieve the correct data but fails to execute the investigation logically, not surfacing the correct evidence.
Reasoning Failure	A failure of the reasoning process. Occurs when the agent possesses the correct data, but fails due to doing incorrect inference using the knowledge extracted.
Communicative Failure	A failure of presentation and transparency. Occurs when the agent fails to surface insights effectively, such as oddly presented facts, omitting the chain of thought / important findings that obscures how the conclusion is reached.

3.4.1 Known ground truth (LLM-evaluated)

Defining a root-cause diagnosis for hardware is not trivial. For a test case to be considered useful for the evaluation of the system, the case was required to satisfy three properties:

1. The unit has an established root-cause, either through a verified repair, a verdict by a specialized rule set for detecting a given issue, or through the means of an investigative report.
2. The unit must have a event driven log prior to its repair, meaning that the symptoms of the unit can reasonably be assessed prior to the root-cause being fixed (and thus verified in the service log).
3. The knowledge required to make the assessment must exist retrievable in the knowledge base.

These three properties considerably narrow the possible evaluation cases. Due to these requirements, the test set is small, with only ten test cases consisting of varying failures. All test cases were constructed based on historical scenarios, either through expert input directly, or previously documented issues.

Due to the evaluation set being so small, there are many considerations that impact how we can reasonably interpret the results. The small sample size means that the LLM-as-judge section will not act as a definitive benchmark of the systems performance. These test cases alone cannot indicate the system effectiveness and generality across different failure modes and types which is up for further discussion in the discussion chapter. However, these test cases provided a baseline to assess

improvements during development of the different modules within the architecture. It also helps assess and characterize system behavior, recurring reasoning failures and whether the core proposed mechanisms influence the investigation outcomes in meaningful ways. For instance by assessing consistency across trials. Trivially, it also constitutes a metric for how good the agentic system performs in assessing these specific root-causes.

A critical consideration for fair evaluation in this setting is data leakage. For instance, consider the case where a serial number with a known root-cause is sourced from the original trouble report. In this case, if the agent can pattern match on the serial number and find the correct trouble report and associated root-cause document, the evaluation only tells us whether the vector search is working as intended. Similarly consider the case where a unit has been sent for repair and the repair action was to replace the root-cause component. If the agent can discover this information, it tells us nothing about how the agent reasons in the proactive case. Therefore two specific mechanisms had to employed to ensure that no data is leaked:

1. When the dynamic planner queries the knowledge retrieval agents (L_1, and L_2) it must not include information such as the serial number which makes the knowledge retrieval trivial.
2. When the dynamic planner queries agents capable of reading log data, all data retrieval queries must be within a predefined time span.

These constraints could not be enforced deterministically without requiring a major refactor of the production environment, and were instead implemented through multiple soft-enforcement prompt changes. Test case runs have also been manually validated through sampling.

The ten test cases consisting of a serial number and cutoff date (input) are described high-level in 3.2.

Table 3.2: Test cases for LLM-as-judge

Test case	**Issue type**	**Fault class**	**Source**
1	HW	HW-01	Scan on logs
2	HW	HW-01	Scan on logs
3	HW	HW-01 (Same serial as above, earlier stage)	Scan on logs
4	HW	HW-01	Scan on logs
5	SW	SW-01	Original case document
6	HW	HW-02	Original root-cause document
7	HW	HW-03	Original case document
8	HW	HW-04	Original root-cause document

Test case	Issue type	Fault class	Source
9	HW	HW-02	Support engineer test case
10	HW	HW-03	Support engineer test case

Each serial number in this test case list has an associated ground truth description, based on its fault class, which has four categories.

1. **Name & Description**: A natural language name of the fault, and a description of the failure mechanism.
2. **Affected group**: A list of unit groups affected by the fault.
3. **Key symptoms**: A description of the key symptoms that surface when the issue is present, such as alarms, fault codes and their relationship.
4. **Known Issue References**: A list of sources that describe this issue.

Each ground truth description was generated by an LLM, which was asked to fill in the template according to the ground-truth document for each respective case. A majority of the generated ground-truth descriptions were directly verified by a support engineer. Test case 5, 7 was not directly verified by a support engineer, primarily since these are old issues, but the information in the ground-truth was compared manually to the ground-truth root-cause document.

The process of defining the issue description introduces a potential source of bias, since the LLM output could be hiding hallucinations, but considering the straightforward task of summarizing the technical details of the root-cause from the ground-truth document into the issue template and the manual check of the output the risk introduced was considered to be negligible.

A notable shortcoming in the test cases is only incorporating one SW issue, zero configuration or external issues and the lack of negative samples (i.e. where no fault should be found). Each of these shortcomings have simple explanations:

1. Lack of varying issue types is in part explained by the prior reasoning of the small sample size, as well as SW issues often requiring source code investigation / which was not available during implementation.
2. True negative samples are in theory technically impossible in the flow. An alarm constitutes an error and an alarm must precede investigation. If there is an alarm a log is generated, and if not there is typically no log available which means nothing to investigate i.e. the agent will most likely state that the unit has no issue based on there existing no evidence.

However, there is an important aspect worth mentioning in regards to negative samples, which is that ADA flagging HW issues where there are none, is a uniquely problematic situation. A unit can alarm without the problem being hardware re-

lated. For example, it could be a software or an external issue. The most problematic scenario is therefore when a HW fault verdict is too conclusive. This was mitigated through adding soft-enforcement against proposing unit return or repair in the system prompt.

The evaluation is structured in a similar way to RAGAS by Es, et al. [24] composed of the evaluation metrics shown in Table 3.3

Table 3.3: Evaluation criteria and description for LLM-as-judge

Evaluation criteria	Description
Correctness	Comparison between final report root-cause candidates and the ground-truth answer • 1.00 - correct root-cause, proposed most likely. • 0.75 - correct root-cause, proposed second most likely candidate. • 0.5 - correct root-cause, proposed as middle-most likely candidate. • 0.25 - correct root-cause, proposed lowest priority candidate. • 0.00 - incorrect root-cause
Faithfulness	Assertions in final report compared to the information given by the subagents.
Efficiency	Count of subagent calls made during the investigation
Input & output tokens*	Tokens used for the investigation

*Note: Not captured for R_2, due to being isolated in a microservice without usage metadata captured.

The reason for the correctness evaluator doing stepwise evaluation is due to the sometimes ambiguous symptom profile and to account for the system proposing multiple root-cause candidates.

A full description of evaluator prompts can be found in Appendix A. The correctness evaluator receives the final output, along with the issue description accompanying the input serial, and an LLM-as-judge evaluates whether the correct root-cause has been proposed and in what order in the prioritized list.

The faithfulness evaluation differs slightly mechanistically from the formulation by Es, et al. [24] due to long outputs. Claims are extracted from the final report using an LLM, fed into a faithfulness style evaluator, with the context preceding the final report as the grounding. Each evaluations is done for at most 10 claims at a time,

in a batched execution. The claim extractor is explicitly told to not extract claims that assign confidence to desired assertions (such as the assertion that a root-cause is the best match), given that agent estimated confidence is by nature a desirable inference by the agent system not grounded in tool outputs.

To account for the probabilistic nature of LLM-generated outputs, the experiment runs were run three times and results will be compared across runs.

Efficiency is captured by counting the tool invocations, and Input & Output tokens by capturing usage metadata from the API provider.

To answer question **R3**, the four LLM-as-judge metrics defined here, will constitute the evaluation to investigate how the different architectural modules included in ADA affect the investigation behavior. Specifically, the four variants described here will be assessed:

1. Full ADA: All functionality enabled
2. Variant A: No hypothesis generator - No seeding of the hypothesis tree.
3. Variant B: No structured output - Free-form text output.
4. Variant C: No hypothesis tree / Generator - Uses direct subagent caller tools. Mostly same system prompt with updated tool descriptions

3.4.2 Domain-expert evaluation

Agent users and SMEs such as support engineers provide the main evaluation of the system quality and utility for the intended use case. The incorporation of domain experts also provides an opportunity to assess and motivate the auxiliary capabilities of the system, such as proposing recommended next actions given a test case or whether the agent output would yield time savings for support engineer examining a case without starting from scratch. Since ground-truth cases are hard to design, letting domain experts assess the system outputs with their own test cases, yields a higher coverage test case set, since the domain-experts know the ground-truth.

The survey was sent out directly to all support engineers through internal communication channels, with an explanatory message for how to access the agent and the intended input. For some domain experts, a test case execution was included in the message. This choice meant that the execution to be sent out for evaluation had to be picked by the author, effectively constituting sampling bias through "cherry-picking". In an effort to mitigate this bias, domain experts were also given access to the system directly, and prompted to pick and run test cases of their own, before making the survey assessment. Some domain experts chose not to answer the survey directly, opting for free-form technical feedback or not responding at all. For these cases, a short-form survey interview call was scheduled and conducted in an effort to gather qualitative domain expert feedback directly.

3.4.2.1 Survey

The survey consists of four total questions. The first of which uses a Likert scale originally defined by Likert in 1932 [30] to assess the main aspects of the final report as shown in Table 3.4.

Table 3.4: Likert scale survey question with criteria

How much do you agree with the following statement about the agent's final report/s?	
Symptom correctness	The report correctly classifies the most important symptoms and observed behavior of the unit.
Root-cause correctness	The report correctly identifies the root-cause or major contributing factors.
Completeness	The report covers all relevant aspects you would investigate.
Actionability	The recommended next steps are practical and likely to resolve the issue.
Faithfulness	The conclusions are supported by the evidence presented.
Time savings	The report would reduce time spent investigating compared to starting from scratch.

A 5-point Likert scale was chosen as it is convenient way to assess sentiment directionality, without overburdening the respondent, which in this case have other duties. Ranging from strongly disagree to strongly agree, with a neutral option. The two first statements concern the correctness of the report, effectively assessing both the output quality in terms of the accurate description of the unit / node state and the root-cause proposals. It was a deliberate decision to separate these two types of correctness, as they are fundamentally different measures. While highly correlated, symptom and root-cause correctness are independent, in the sense that one does not naturally give rise to the other. Symptom correctness not leading to root-cause correctness is intuitive, but the reverse relationship is not immediately clear. One circumstance, that could cause root-cause correctness to be correct, while symptom correctness to not be, is for instance, if a symptom non-indicative of failure is mischaracterized, or exacerbated.

Completeness is concerned with coverage, since the first two statements could be fulfilled, while still missing supporting information that could be relevant for the agent to take into account. For instance, consider the scenario where there is an intermittent failure that could be addressed which is not the direct cause of the main alarm on the unit. Actionability is concerned with whether the respondent feels that the report will help in resolving the issue completely. Since the agent operates in a constrained action space, some actions are not available to it, such as asking

the customer for additional information regarding their site setup. In this case the recommended next steps must be practical and actionable, which is something that a domain expert can assess directly. Time savings is concerned with whether a domain expert actually would benefit from the report being present at the beginning of their investigation which is a crucial utility signal for the agent output.

Along with the Likert scale question, there are three more questions in the survey opening up for more context regarding the assessment.

1. Free form question: Any comments regarding criteria above?
2. Free form question: What is missing or incorrect in this report that you would have caught in your own investigation? (Required)
3. A ternary question: Would you recommend using this agent for future diagnostics? Yes / No / Not sure (Required)

The first free form questions give further description and motivation for the Likert question responses, while the second is probing for information regarding factual inaccuracies. The ternary question is mostly a high-level indicative question for the perceived utility of the system.

3.4.2.2 Interviewer-administrated survey

The interview generally followed a semi-structured approach, consisting of a short introduction regarding the system's capabilities and purpose and a structured walk-through of the survey questions. The domain experts had different prerequisite knowledge about the intended purpose of the thesis and generally different test cases, which is why this method was chosen. The survey administration was done by screen sharing, where the questions could be read by the participant directly, which helped in the case of the Likert question which can be tough to formulate nicely in a simple way through speech. Free-form questions were answered based on a discussion and thoughts during the Likert-scale assessment.

This method has one primary bias, which is social-desirability bias which occurs simply because the researcher is present during the administration of the survey which could impact desirability to present negative or pessimistic sentiment. In an effort to mitigate this bias, it was made clear during the introduction that candid feedback is desired, since the system needs critical feedback to improve. There is also an inherent bias-mitigator due to the domain experts being the intended consumer, and therefore the most concerned stakeholder when it comes to quality issues. The researcher being present directly, also made questions regarding the survey itself, possible to address immediately.

4
Results

The following sections will show the empirical evidence gathered, starting with the domain expert feedback and moving on to the LLM-as-judge cases. As mentioned in the method chapter, the implementation and subsequent gathering of domain expert validation has been iterative, which is something that will be highlighted where necessary.

4.1 Domain-expert evaluation

The domain expert section is split across qualitative feedback received as structured survey responses and unstructured format. Since the data used for inference and the final report both contain proprietary information, the following sections will take more of a narrative approach to describing the investigation, and how that relates to what feedback was received, as well as how it relates to the failure modes defined in the method chapter.

Since the manner of gathering domain expert feedback varied on a case by case basis, Table 4.1 describes the procedure for each domain expert, how many test cases were tried, whether a trial was done independently in the GUI for a test case sampled by the domain expert themselves and some additional context regarding the test cases used.

Table 4.1: Overview of subject matter expert (SME) evaluation procedures

SME Role	Procedure	Cases Tried	Independent Trial	Context & Variations
Senior Product Support Engineer	Detailed unstructured feedback during development + Direct survey answer after send out	2	Yes	Tested already repaired units based on service logs.

Continued on next page

SME Role	Procedure	Cases Tried	Independent Trial	Context & Variations
Senior Product Support Professional 1	30 minute interview & fill out survey together	2	Yes	HW-related issues. Proactive-style executions.
Senior Product Support Professional 2	Detailed unstructured written feedback on survey send out + interview to fill out survey results together	1	Partial*	CSR-style execution, difficult solution with clear failure pattern.
Senior Product Support Professional 3	30 minute interview to explain the agent and what feedback was required, filled in survey after the interview	2	Yes	Two separate CSR style tests.
Technical Solution Engineer	Detailed unstructured feedback + direct survey answer	3	Partial*	Three separate CSR style tests. Two HW-related issues and finally one undiagnosed case
Senior Product Support Professional 4	30 minute interview to explain the agent, how test cases could be chosen and what feedback was required, filled in survey after doing testing on their own after the interview	5	Yes	Five different executions of both multi-serial inputs and single serial inputs, for different proactive cases.
Quality Manager	30 minute call to explain the agent, how test cases could be chosen and what feedback was required, filled in survey after doing testing on their own after the interview	3	Yes	Three separate CSR style executions.

*Note: Suggested test cases independently, which were executed and sent by researcher.

Overall, 18 test cases were tried by domain experts some of which executed multiple times, of which a majority were CSR-type executions with a diagnostics snapshot. The additional context the agent received for the majority of them was general and not the actual case context included in the CSR. A majority of responders conducted their own trial, while some respondents chose their own case, but sent it of to the

researcher for execution. The procedure for all domain experts varied, where some took convincing through an interview, and others answered the survey directly. Five respondents had an interview, and two of those resulted in the survey being filled out during the interview.

4.1.1 Survey responses

Table 4.2 showcases the collected survey results, and that the system mostly performs mostly desirable for the investigations performed by domain experts.

Table 4.2: Survey Responses ($N = 7$)

Statement Evaluated	SD	D	N	A	SA
Symptom correctness	0	1	0	2	4
Root-cause correctness	0	0	2	2	3
Completeness	0	0	2	4	1
Actionability	0	0	1	2	4
Faithfulness	0	0	1	4	2
Time savings	0	1	1	1	4

Looking closer at the results, a majority of domain experts strongly agree that the symptoms are correct, the actionability is high and that it would constitute some form of time-saving compared to starting from scratch for the test cases they examined, although there are a two outliers. This is later clarified in the open-ended questions. For root-cause correctness, which is a central theme in the final report, the distribution is shifted to the left, highlighting issues with misguided verdicts. Since no domain expert answered disagree, it could suggest that either there is a low expectation on the root-cause correctness or that the verdict is not completely wrong, which could be since the agent respects the system prompt boundaries and doesn't present wrong conclusion confidently.

Faithfulness and Completeness are both generally positive, but not strongly positive, which could be interpreted as the agent generally tending to propose high coverage faithful solutions, but that there is still some room for improvement in terms of what is surfaced, and how anchored it is in source material.

Table 4.3 showcases the answers to the first open-ended question. In total four responses were received and three respondents chose not to give any comment.
The first two comments are overall positive, but comment two notes that further testing would be needed and that the judgment was based on a low sample size. Number three is more critical and highlights two epistemic failures. One where the agent can't access commit information to verify software fixes, and one where a relevant case document was missed. Number four is also critical, particularly of the latency of execution. This is highly relevant in the testing scenario, but the high latency is not a result of ADA's proposed architecture.

Table 4.3: Open ended question 1

Any comments regarding the criteria above?
1. *Well structured answers that address key points based on detailed log analysis and good use of relevant root-cause documents, logical flow of ideas and references that demonstrates subject matter comprehension. The response show clear reasoning and application of knowledge base across the different sections.*
2. *Low sample size for testing, but it seems beneficial as a pre-screener. Further testing would make the case stronger.*
3. *Problem was complex, but there is room for improvement in several areas. For instance, the report is generally too long. The agent should be able to access commit information for better understanding of software issues, and if the fix is included in a new release. The agent missed relevant case documents in its investigation.*
4. *The time needed to complete a diagnostics snapshot execution is now becoming a bottleneck, normally a diagnostics snapshot would take (**X amount of time**) or so, the ADA agent is comprehensive, yes, but efficiency will be diluted if there are (**X** number of) diagnostic snapshots sent waiting to be analyzed.*

Note: Three respondents did not provide comments for open-ended question 1.

Similarly Table 4.4 showcases the answers to the second open-ended question. All respondents provided an answer to this question.

Item 1, 2, 4 and 6 do not constitute agent failures. Item 1 and 6, both propose a similar software change, where the citations tab would instead show the direct log lines. This is a possible improvement, where more structured notes of important log lines would be stored. Item 2, is a broader suggestion, which does not constitute a failure under the taxonomy defined in the method. Specifically fleet-level insights are wanted in the final report which the agent can not fit currently, and that the agent should be able to suggest follow ups, such as suggesting new analysis that can be done. While connection issues are noted, it was due to some issues in the production environment at the time of testing. Item 4, is an improvement suggestion, by giving some weight to each finding. The agent assigns confidence to each hypothesis, but not strictly every finding in the logs.

Item 3, is critical feedback, which suggests that the root-cause is wrong. This was surfaced during one of the interview sessions, when filling out the survey together, and was clarified to be not completely unexpected given the difficulty of the problem. This specific case will be walked through in more detail in Section 4.1.2.1. Item 5, describes a problem with the initial trigger and structure of the investigation. While alarms constitute a fault, often times errors are hiding on the event level. In this case, there was no alarm emitted, and the system missed the specific event which constitutes a fault, showcasing a procedural failure.

Table 4.4: Open ended question 2

Missing or incorrect in this report that you would have caught in your own investigation
1. *one suggestion on area could be improved in this agent report: could it be better for sources reference linking abnormal logs(short/simple) directly to verdicts or root cause candidates, for example abnormal ... fault, instead of "show code". as verdicts are drawn from the pattern of abnormal logs. and how to get the normal values for comparison? it would significantly strengthen the report clarity.*
2. *Issues with connecting. Agent is able to identify the fault in this case, but would want visualization to quantify the impact on a fleet level. Let agent suggest follow ups such as offering suggestion for new possible analysis.*
3. *Root-cause is wrong in this case. The fix is SW-related while the agent suggest HW-issue.*
4. *We can fine-tune the results by giving some weighing each finding.*
5. *Analysis is based on alarms mainly, we should target shown events to identify issues.*
6. *What would be useful is to have a link to the raw data would be useful so that we do our due diligence in confirming the report's findings or to do some additional analysis*
7. *Seems the agent cannot based on the target product/product family fetch a "top issue list", but maybe it's related to how the case documents work and there is no design to highlight those important case documents, so the agent is guessing the top issues instead of matching symptoms within a smaller targeted case document or issue list. Perhaps more input is needed, since without proper guidance, case documents can be very 'messy' and trivial.*

Item 7 is not strictly an agent failure but implicitly a critique of the surrounding design. Although the flow tries to mimic the described procedure, ADA does not filter the number of case documents statically to a relevant subset before retrieval, which is highly perceptive feedback.

Figure 4.1 showcases the distribution of the ternary question answer for the seven respondents. A majority of respondents are positive to the tool for future diagnostics, while on respondent remains unsure.

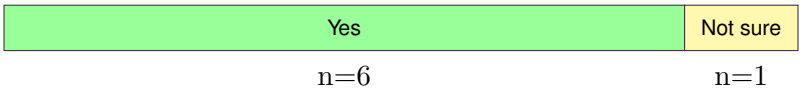


Figure 4.1: Domain expert answer on whether ADA is helpful (ternary question)

The unsure answer was collected during an interview session, and it was clarified that the reason for the answer, is mostly due to the respondent feeling confident that the issue would be handled swiftly, given their expertise. Given that this was

the only test case run for this respondent, the answer chosen was "not sure". This is a negative utility signal, and raises important questions regarding whether LLM agents in this space can actually slow down domain experts.

4.1.2 Free-form feedback

This section provides information regarding some of the feedback received in unstructured format in internal communication channels mostly during development, but also as supplementary information before survey feedback collection.

4.1.2.1 Case A

The feedback in Table 4.5 highlights several key points concerning epistemic, communicative and reasoning failures and some miscellaneous formatting feedback which showcases the high importance in the presentation of the proactive diagnosis.

Table 4.5: Qualitative feedback received (paraphrased)

Title
Senior Product Support Professional

1. *The agent concludes that case document #... is out-of-scope though possibly related, however it keeps referring to it:*
   - *For me who is versed in this product and the many issues that go with it: this is not very useful*
   - *For other people less familiar this could cause some confusion*

   I suggest creating a dedicated section that clearly states: "Here are the problems that I looked at but that are not applicable here" so that this section can be skipped easily. And I don't expect to see any reference to it anywhere else.
2. *"All ... are ...": This is a ... product. Here there are two ... and ... is performed on both directions. This may confuse many people...*
3. *Critical findings should be presented in a better way with ... failure as the main symptom. Then all related consequences should follow, then all other findings. I find the structure "too flat"*
4. *"Root Cause Candidates" -> none of the listed actually applies. Key point: for the second one "Hardware Defect" this type of verdict has to be declared with extreme care by an agent. Actual RCA includes a SW fix for the already installed unit. Also HW replacement in one of the recommended next steps.*
5. *The overall analysis completely missed case document #... that is the root cause of the ... failure.*

Item 1 and 2 are both communicative failures. Item 1 concerns a case where the right conclusion is reached, but the case document itself is referenced in multiple places, as if the LLM is anchoring to this piece of evidence. Item 2 is a communicative failure by the subagent which does a silent multiplication which makes the report imply that there are more physical elements than exist in reality.

Item 3 is not a failure of the agent but the structured output setup, where the symptoms were presented in a bullet list by design, which in part was not easily comprehensible, and the relations between symptoms were neglected. This was later changed.

Item 4 is perhaps the most severe critique, which is actually two separate distinct issues. 1. The root-causes proposed did not apply to the issue. This case is a good case-study for the domain complexity, and will be discussed further in the discussion. 2. The HW replacement suggestion is not warranted in this case, and the verdict is thrown out too hastily, and therefore a suggestion that there needs to be guardrails for replacement suggestions in the first triage. This was later implemented via means of the system prompt.

Item 5 is an epistemic and/or procedural failure, depending on the perspective. A crucial piece of evidence was not retrieved and therefore not used in the assessment. This fact contributes to item 4, since the verdict and root-causes proposed did not take into account this case document's existence. This problem can be one of or potentially multiple of the below failures:

- The retrieval mechanism in R_1/R_2 did not surface the correct case documents, potentially due to misinterpreting or misspecifying the query, or the fidelity of the retrieval mechanism being to low.
- ADA queried the knowledge retrieving agent with too wide instructions to capture this specific case document through semantic search.
- ADA narrowed the search space too early, prompting R_2 to search for unrelated case documents.

In this case specifically, you could argue that the retrieval mechanism should have surfaced the correct case document given the query done by the dynamic planner, but also that there aspects of the dynamic planners query that could have been made more narrow to make sure that the case document was surfaced.

4.1.2.2 Case B

The feedback in Table 4.6 highlights several key points highlighting key epistemic and reasoning failures.

Table 4.6: Qualitative feedback received (paraphrased)

Title
Senior Product Support Engineer

- *From this session, I noticed crash restart logs are targeted, and suggested repair is defined correctly.*
- *HW defect by known repair is sorted out (**implied: correctly**).*
- *But this issue shouldn't be ruled out? (referring to one of the ruled-out hypothesis which corresponds to a possible repair).*
- *Also, recommended next steps, lead to wrong direction? (other fault, later clarified to be okay).*
- *Why inference didn't follow the critical finding: "Persistent ... restarts across two separate repair visits ..." and HW fault defined by ... .*
- *Anyway, I think the agent provides enough good information for this issue based on the service logs.*

The statement regarding a hypothesis being ruled out when it shouldn't have been, showcases an epistemic issue, where the agent could not identify the document referring to a known repair, which was flagged as a possible issue. The reason for this is that these documents are not included in the knowledge base, so the agent could neither confirm nor deny the validity of the hypothesis.

The recommended next steps leading in the wrong direction, were in this case primarily due to the agent being unsure if the affected unit was in a specific time span since date metadata suggested it fell outside this window. While the agent is correct in raising this suspicion it presents it in a way which implies that the issue is low likelihood. At face-value this is reasonable, but constitutes a reasoning failure due to not considering that the error could have been introduced earlier. However, it also points to an epistemic issue, where it does not have access to component details in this instance which would be required to be 100% certain.

4.1.2.3 Case C

Two executions were done for this test case, without any changes in between and the domain expert feedback in Table 4.7 highlights the procedural failure, which arises due to the inconsistent nature of the multi-agent system.

Table 4.7: Qualitative feedback received (paraphrased)

Title
Technical Solution Engineer

First execution

- *I didn't get the main issue which is related to* ***(specific error)***
- *I think the tool should go like this: find out all known-issues first > check-logs > compare and then report*
- *i will use some other logs where we have direct fault (****implied: unlike this one****)*
- *I mean there is no specific alarm for the issues in this*
- *"Quoted log line" this is not a fault it will happen everytime radio is initialized its just telling radio is not supporting this*

Second execution

- *Yes this is good now*

Despite giving no additional context, the multi-agent system succeeded in giving a satisfactory answer on the second execution. Looking closer at the execution itself, it can be seen that the correct error is actually surfaced in the pre-screening in both executions, but the dynamic planner chooses to pursue another direction in the first execution.

The main difference in this case is actually how the pre-screening described the fault chain. In the first execution, the fault chain is deemed to be other fault -> restarts -> specific error. In the second execution it is rightly described as specific error -> other fault -> restarts. Depending on perspective, the dynamic planner should pursue both threads, but if it believes from the pre-screening that the specific errors in this case are a downstream effect of the other fault, it will try to find reasons for the other fault itself highlighting the sensitivity introduced by the LLM-based pre-screening mechanism. Regardless of perspective, it constitutes a clear procedural failure.

4.2 Known ground truth (LLM-as-judge)

This section will walk through the LLM-as-judge test results for the known ground truth cases. The results for the full ADA setup will be showcased and interpreted first and then continue on to the ablation setup results.

4.2.1 Full ADA setup result

Table 4.8 showcases the overall result of the LLM-as-judge test cases for the full ADA setup. The percentage correctness for the test cases is modest at 47.5% while faithfulness rests at a solid 89%. The average run correctness varies from 0.4 to 0.6 while the faithfulness score is highly stable between 0.89 and 0.9.

Table 4.8: ADA performance over three trials per test case (all functionality enabled)

ID	**Issue**	**Correctness** μ [*range*]	**Faithfulness** μ [*range*]
TC01	HW-01	0.33 [0 - 1]	0.87 [0.84 - 0.89]
TC02	HW-01	0.00 [0]	0.85 [0.74 - 0.92]
TC03	HW-01	0.00 [0]	0.87 [0.81 - 0.9]
TC04	HW-01	0.00 [0]	0.90 [0.87 - 0.94]
TC05	SW-01	1.00 [1]	0.91 [0.88 - 0.96]
TC06	HW-02	1.00 [1]	0.92 [0.91 - 0.94]
TC07	HW-03	0.33 [0 - 1]	0.85 [0.8 - 0.88]
TC08	HW-04	1.00 [1]	0.87 [0.85 - 0.91]
TC09	HW-02	1.00 [1]	0.93 [0.84 - 0.98]
TC10	HW-03	0.08 [0 - 0.25]	0.92 [0.88 - 0.94]
Run average	**N/A**	**0.475 [0.40 - 0.60]**	**0.89 [0.89 - 0.9]**

The first thing of note is that the problem type clearly has an impact on the correctness of the agent. In particular, test cases for HW-01 stand out as being incorrect in almost all cases. The reason for this is multi-faceted, and can be traced through the executions, but the general problem is that the alarms raised by the unit for this issue in the logs directly, can be attributed to multiple different failures.

The knowledge required to make the assessment in the HW-01 case, depends on a specific diagnostic. Although there are multiple documents detailing the issue itself from a hardware perspective, the agent at the time of testing did not have access to the document detailing the specific detection method. While problematic from the evaluation standpoint, it surfaces the insight that epistemic failures are a main limiting factor, in terms of what root-causes can reasonably be assigned by the system.

Looking at the other test cases, the probabilistic nature of the agent system is showcased, while also showing that the agent performs relatively stable on certain issues. The faithfulness rests at 89% average overall and is generally quite stable.
The number of subagent invocations per subagent is shown in Table 4.9 and varies across execution. Overall the most popular invocation is R_1 and the least popular D for the test cases tried.
The results showcases interesting behavior both from the perspective of the unstable test cases and the more stable test cases. Notable is that for TC05 which has perfect root-cause correctness across all three trials, still has considerable variance

Table 4.9: Subagent invocation mean over three trials per test case (all functionality enabled)

ID	Log Agent 1	Retrieval Agent 1	Data Agent	Retrieval Agent 2
TC01	2.67 [2 - 3]	2.67 [1 - 4]	1 [1 - 1]	1.5 [1 - 2]
TC02	2.67 [2 - 4]	3.67 [3 - 4]	1.33 [1 - 2]	3.00 [2 - 4]
TC03	1.67 [1 - 2]	2.67 [2 - 3]	1.33 [1 - 2]	2.33 [2 - 3]
TC04	2.67 [2 - 4]	2.33 [1 - 4]	1.5 [1 - 2]	1.00 [1 - 1]
TC05	2.33 [2 - 3]	4.67 [2 - 6]	1.5 [1- 2]	4 [4 - 4]
TC06	2.50 [1 - 4]	2.50 [0 - 3]	0.7 [0 - 1]	2.00 [1 - 3]
TC07	3.67 [1 - 9]	1.67 [1 - 2]	1.00 [1 - 1]	1.67 [1 - 2]
TC08	2.33 [1 - 3]	2.67 [2 - 3]	1.33 [1 - 2]	2.00 [2 - 2]
TC09	2.00 [2 - 2]	3.00 [2 - 4]	1.00 [1 - 1]	3.67 [3 - 5]
TC10	1.67 [1 - 3]	1.67 [1 - 3]	1.00 [1 - 1]	1.33 [1 - 3]
Avg	**2.41**	**2.76**	**1.07**	**1.83**

in the number of subagent invocations. For instance, the number of R_1 invocations vary between a minimum of 2 and a maximum of 6, but consistency in terms of the root-cause is reached, which is the desired behavior. For TC01 which has varying correctness score the variance in subagent invocations is also noticeable, showcasing that the inconsistency in conclusion is also reflected in the number of subagent invocations. The inconsistency in behavior will be investigated further in the discussion, but the consistency in outcome for TC05 is due to the symptoms being somewhat distinct for this root-cause.

The token usage across subagents in Table 4.10 shows that the overall token usage is generally high. The main token input token usage is found in L_1 at 1.913M tokens and D at 777.403k tokens.

Table 4.10: Detailed subagent and orchestrator mean token usage breakdown

	Dynamic Planner		Log Agent 1		Retrieval Agent 1		Data Agent	
ID	In	Out	In	Out	In	Out	In	Out
TC01	289,134	17,081	1,083,108	29,323	169,064	82,618	512,560	11,527
TC02	403,572	21,543	1,785,801	38,088	82,011	33,893	1,919,213	35,983
TC03	248,303	15,780	1,730,524	22,138	115,549	47,860	827,746	22,439
TC04	281,807	15,648	1,577,976	31,338	78,841	41,371	908,999	16,426
TC05	417,428	28,564	3,307,716	48,197	188,916	99,446	488,779	10,583
TC06	721,359	23,640	1,587,022	34,825	245,431	86,431	876,889	19,305
TC07	367,416	17,037	3,941,699	68,219	23,047	15,315	355,163	8,128
TC08	194,029	16,628	1,730,201	41,507	59,712	33,039	343,212	6,592
TC09†	190,533	20,100	1,497,142	25,165	33,329	13,481	1,216,658	17,716
TC10	238,580	14,603	563,413	25,027	16,027	6,527	424,299	10,347
Avg	**292,311**	**18,553**	**1,913,064**	**36,555**	**85,166**	**41,505**	**777,403**	**15,526**

†Note: TC09 includes an extra 19,681 input and 36 output tokens utilized by the L_2, due to the agent hallucinating a call.

The main reason for the high token usage in L_1, is primarily due to context accumulation of raw log data in the multi-turn analysis. L_1 does many sequential steps, which exacerbates the problem with token accumulation. The number of tokens used vary widely across the different test cases, and also the distribution of token

spend across different subagents. For instance, TC02 has the most tokens used in D, compared to L_1.

The orchestrator itself, actually has a quite lean context window percentage use considering the one million token context window of Claude-Sonnet 4.6, although the input token usage is once again exacerbated by the multi-turn workflow. The implications of the token usage and possible mitigation methods will be explored in the discussion.

4.2.2 Ablation study

This section will look at the different ablations and compare them across the test cases to understand more how each piece impacts the final output. The overall results are shown in Table 4.11. Bold values signify the best result in terms of correctness and faithfulness while for efficiency metrics the most efficient. The best performer in terms of correctness, faithfulness and number of invocations is Variant A, without an explicit hypotheses generator. Variant C, with direct caller tools is the most efficient with the least amount of input and output tokens.

Table 4.11: Structured architecture ablations (System-wide LLM-as-judge averages)

Architecture Variant	**Correctness** (μ)	**Faithfulness** (μ)	**Invocations** (μ)	**Mean Input Tokens**	**Mean Output Tokens**
ADA (All Functionality Enabled)	0.475	0.890	7.80	3,070,132	112,146
Variant A: No Hypotheses Generator	**0.55**	**0.916**	**6.86**	3,273,889	85,569
Variant B: No Structured Output Schema	0.40	0.91	7.4	3,197,593	84,151
Variant C: No hypothesis tree / direct caller tools	0.47	0.908	9.67	**1,465,087**	**63,408**

Token counts represent system-wide averages across three trials of the 10 evaluation test cases.

The results of the ablation study show that each variant that includes some aspect of the hypothesis tree generally conducts much larger investigations but also that the correctness improvements are negligible compared to less token usage intense Variant C. Variant C performance suggests that investigation structure through the hypothesis tree primarily affects exploration behavior rather than final correctness for the evaluated samples. This exploration behavior generally amounts to considerably higher token usage, even though the mean number of subagent invocations

is actually the largest in Variant C. This is likely because queries to subagents in Variant C were generally smaller in scope.

The best performing variant for the test cases was the version of ADA without a hypotheses generator, but the difference in terms of correctness score compared to Variant C is not significant. Variant B is the poorest performer in terms of correctness, which is likely due to not being forced to consider multiple candidate root-causes. Interestingly, no hypothesis generator, performs better than the full ADA setup across all metrics except the mean input tokens, suggesting that the up front plan approach is less viable for the test cases tested here. This finding could also explain why the direct caller tools work well in this setting, since it suggest the up-front plan is not necessary for right attribution of failure mechanism. To examine this behavior a bit more closely, the next section will focus on a comparison between tree structures made explicitly or implicitly at the end of execution across variants.

4.2.3 Comparison of final hypothesis tree

The full details of the hypothesis tree can not be shared, since the information is largely proprietary and certain information will be replaced by ellipsis. However the core structure of the hypothesis tree can be shared, showcasing some interesting behavior across the four different scenarios. Specifically, this is the final structure for TC09 for the first run of each ablation in which all variants succeeded in their root-cause assessment.

4.2.3.1 Full ADA & Variant B

The final tree for variant B and the full setup are very similar, and looks roughly like the following having removed the conclusions and investigations marked under each hypothesis:

- **H1**: Hardware Faults
  - **H1.1**: ... subsystem
    - Two L3 Hypothesis
  - **H1.2**: ... / ... subsystem
    - Two L3 Hypothesis
  - **H1.3**: ... / ... / ... Subsystem
    - Two L3 Hypothesis
  - **H1.4**: ... Subsystem
    - One L3 Hypothesis
- **H2**: Software / Firmware Faults
  - **H2.1**: ... / ... Software
    - Two L3 Hypothesis
  - **H2.2**: ... ... Software
    - One L3 Hypothesis
  - **H2.3**: ... / ... Software
    - One L3 Hypothesis
- **H3**: Configuration Faults
  - **H3.1**: ... / ... Config
    - One L3 Hypothesis
  - **H3.2**: ... / ... Configuration
    - One L3 Hypothesis
- **H4**: Site / External Factors
  - **H4.1**: ... / External Factors
    - One L3 Hypothesis
  - **H4.2**: ... Infrastructure
    - One L3 Hypothesis

Notable, is the width and depth of the tree and the amount of failure hypotheses.

4.2.3.2 Variant A

The final tree for variant A is considerably more narrow and looks roughly like the below for the given test case:

- **H1**: Software / Firmware Faults
  - **H1.1**: ... causing repeated crash restarts
- **H2**: Hardware Fault
  - **H2.1**: ... Fault
  - **H2.2**: ... causing crash restarts
- **H3**: External / Site Fault
  - **H3.1**: ... instability causing restarts
  - **H3.2**: ... causing crash restarts

The tree is significantly more shallow, and more symptoms oriented, than in the first case, which focuses more on known issues.

4.2.3.3 Variant C

The most interesting tree, is the one which is created implicitly, when using direct caller tools, through the free-form text "thought" emitted along with the tool-calls themselves:

- **L1-HW**: Hardware fault <- **PRIMARY FOCUS**
- **L2-...**: ... Fault <- **STRONG CANDIDATE** (... crashes)
- **L3**: ...
- **L3**: ...
- **L3**: ...
- **L2-...**: ... <- **WEAK CANDIDATE**
- **L1-EXT**: External/site issue
- **L2-...**: ... instability as a consequence of ... <- likely secondary
- **L2-...**: ... <- **LOW PRIORITY**

This suggests that a standard ReAct workflow can manifest structures resembling the explicit hypothesis tree, purely through prompting, rather than by attaching tools. Naturally, there are no constraints for how this structure looks, which means the LLM might chose not to provide explicit reasoning e.g. for the "primary focus" label, but none-the-less shares some light on how the direct caller tool variant documents the large investigation.

4.2.3.4 Combined interpretation

The results show, that the slimmer hypothesis tree seems to be a winning structure for the test cases tried in the LLM-as-judge portion, and that the direct caller variant is able to construct something similar to the tree, only based on the prompt and its latent parameterization. For the test cases tried, the wide and deep trees negatively impact the output of the agent, which could be because the hypotheses are too detailed, limiting exploration, after generation by the hypotheses generator. With the same reasoning, the more symptom-driven tree in best performing variant A suggests that investigation performance is improved for the test cases evaluated here if guided by the symptoms, rather than case knowledge.

5
Discussion

The following sections will discuss different aspects of the thesis methodology, the system, the results and forward looking statements.

5.1 Analysis of results

Looking back at the failure taxonomy defined in Table 3.1 all failure types described have been displayed throughout testing. Table 5.1 describes their relative prevalence for the evaluated cases and what impact it has on the proposed multi-agent system for autonomous hardware diagnostics and the design implications for MAS in this setting.

Table 5.1: Impact and design implication of diagnostic failures

Failure Type	Relative prevalence	Impact	Design Implication
Epistemic Failure	Most common	Primarily impacts root-cause attribution but also understanding and description of acronyms, issues and by extension user trust.	Epistemic issues is the core underlying bottleneck in highly complex enterprise settings which LLMs cannot reason their way out of. Since retrieval is a coarse instrument mechanisms for ensuring coverage are required.
Procedural Failure	Third most common	Affects what findings are surfaced, often in a non-deterministic way.	All measures to increase determinism in multi-agent setups should be taken, but investigation scaffolding in the form of a hypothesis tree is very dependent on the initial context and cant solve procedural issues on its own.

Continued on next page

Failure Type	Relative prevalence	Impact	Design Implication
Reasoning Failure	Least common	Impacts attribution of root-causes such as in the crash storm case, and the confidence in assertions of root-causes.	Guardrails to limit the amount of reasoning / inference done must be put in place in code to the maximum extent possible, since the underlying LLM model cannot be trusted to not make reasoning failures.
Communicative Failure	Second most common	Affects how users interpret the findings, and their trust in the utility of the tool. For instance, the agent sometimes refers to unrelated case documents while still showcasing that it understands they are unrelated.	The GUI's structure and formulation must conform to user expectation, and the LLM should not be able to govern the exact structure in this case.

A consistent problem during both the implementation and evaluation procedures and when putting an agentic system into production is consistency. Despite the structure proposed in this thesis for the proactive hardware diagnostics case, non-determinism consistently affects the agent output, and therefore its usefulness.

An aspect of this observed both during development and testing is that one of the consistently best indicators of whether the final agent output would be correct is whether the initial report in the "initial context pipeline" is comprehensive in covering the most important faults in the log. Not necessarily that the fault itself is fully detailed, but that it is mentioned as a possible contributor to the alarms present. The structured hypothesis tree helps in creating coverage for possible root-causes to investigate from the initial log analysis, but in general the agent tends to miss some branches completely if it is not present in the initial context. In theory it could recover, but this rarely happened during development and testing. A tempting idea might be to let the dynamic planner receive all log data directly, but this would likely not be particularly effective, given that the token usage would increase catastrophically. The agent rarely uses its ability to formulate new hypothesis during the execution. Admittedly, the capability to do so is not necessarily comprehensively evaluated by the LLM-as-judge test cases, since they are all mostly straightforward one unit test cases, where the relevant failure is immediately clear in contrast to the domain expert evaluated CSRs. For more ambiguous cases, this adaptive capability could possibly be used more frequently.

The original motivation for including the hypothesis tree was not only to structure the investigation but also to try to coerce the LLM into generating a high coverage plan. From the testing during development it seems that the hypothesis tree is mostly a relevant tool for the former. The tree does seem to function well to ensure that the investigation itself proceeds with high coverage given the initial

plan. But it is not relevant in ensuring that the plan itself is high coverage.

These observations indicate that if the initial context is high coverage, the better the plan is and by extension there is a higher likelihood of reaching the desired outcome. Using deterministic methods to the extent possible for the initial context, makes further development easier.

A possible way to enforce a higher coverage initial context, is to for instance, in the CSR case, divide the diagnostics snapshot itself into structural parts each examined by one instance of the L_2 separately. This would in turn likely lead to more consistent outcomes, as the non-deterministic actions on the large log has a higher impact on what findings are surfaced in the LLM summarization step, than on a small subset of the log. A similar argument could be presented for the L_1.

5.2 Sample size and Evaluation bias

The most notable challenge during agent evaluation and assessment is challenges with the data itself as well as the time able to be spent on this matter by domain experts.

5.2.1 Subject matter experts

Subject matter experts in this domain all work in different parts of the chain and have busy schedules, which has impacted the ability to test the system in a more comprehensive way.

Subject matter experts also tend to have a focus area meaning extra time would be required for an SME to evaluate a test case they are not familiar with. This is the main reason why the domain experts were prompted to gather test cases on their own.

In an ideal scenario, the testing by domain experts would have covered more possible failure modes and product variations, but in most cases as described in the result, it was limited to a couple executions due to SME time constraints. Similarly, it would have been preferable for domain expert feedback across ablations. Although the survey results were generally positive, there is risk of bias in results due to how the survey results were gathered, particularly in the interviewer administrated survey.

These constraints mean that the feedback shared by domain experts for their chosen test case should not be taken at face value. A common feedback theme was that the system seems useful, but that there are multiple possible avenues for improvement.

5.2.2 Data challenges

This section will discuss some of the challenges associated with the data, and constructing the flow overall.

5.2.2.1 Domain complexity

The definition of a root-cause can be ambiguous. The reason for this is that radio hardware is very complex, and there are a considerable amount of possible fault mechanisms. There can be multiple failure mechanisms related to one fault, or there can exists multiple independent faults in any given unit.

To understand the difficulty in setup and evaluation it is beneficial with an illustrative example:

In one test case done with a domain expert, the underlying fault mechanism is that certain data that should have been present is missing from the unit. This artifact is present in the logs, and while the root-cause seems straightforward enough in this scenario because there are artifacts in the knowledge base suggesting this issue should result in return. However, in this instance, the fix could be done completely through software. The agent in this case, suggests that it might be an issue that needs unit return to solve. While this is not technically wrong historically, it is the wrong action in this case. The root-cause definition is clearly ambiguous in this case. The agent argues that the data being missing is the root-cause for the symptoms in this case which seems reasonable given that this is what is causing the issues, but a support engineer with knowledge in this case could easily argue that the root-cause actually is a software fix not being implemented [1].

A case such as this showcases one of many ambiguities in the domain, and adds a layer of complexity in terms of user adoption. There is actually no guarantee that the solutions provided in the previous cases were optimal (although naturally, an assumption has to be made that they are), or that the product itself hasn't changed substantially since the documentation was made. This type of concept drift, is prohibitively hard to capture in a structured manner in the knowledge base and presents a challenge for agentic AI employed in this domain.

While this is a concerning fact in the grand scheme of employing agentic LLMs for HW diagnostics, it should be noted, that the actual root-cause candidate, is only a subset of the value provided to an engineer by the final report. The symptoms analysis in this case is mostly correct, and the actual failure mechanism is traced almost all the way to the end. A human engineer could also reasonably fall victim to the same epistemic failure.

5.2.2.2 Knowledge missing

There is data which is currently not captured in ADA which, if incorporated, could provide the system with improved output consistency and analysis correctness. A core missing piece is high level product information. This causes some confusion in the final report of Case A (Section 4.1.2.1), where the agent does an implicit multiplication of the number of physical elements, similar to conflating the number of physical hardware cores and virtual cores in a CPU.

[1] This whole paragraph is a very high-level overview of the actual root-cause mechanism.

Another aspect which is not captured in ADA currently is the component part numbers and what they correspond to. For instance, if you were to give the agent a component part number, it would not be able to reliably answer what that component is. This impacts analysis dependent on component level details.

One possible development specifically for the diagnostics problem is the explicit modeling of document references. A frequently observed issue during both development and testing, is that the agent lacks the ability to traverse across documents that reference each other in a structured manner. The reason for this issue is because of certain knowledge sources not being ingested, but it is also frequently a metadata and ingestion hurdle, since data can be hidden or the links between documents vary in format, for instance as a URL, an attachment or as an explicit string reference to documentation. A possible idea to mitigate this is to build a knowledge base consisting only of document references, that the agent could use for lookup to find relevant relationships between documentation. As an example, certain documents cross-reference others, and sometimes the agent is not able to act up on this relationship, because either its not surfaced by the subagent, the actual reference being hidden or the orchestrator not choosing to pursue the link.

On a more granular level, the agent cannot conclusively confirm certain HW and SW failures, due to certain root-cause or case documents missing in the knowledge base or not containing enough text to be reliably surfaced by semantic similarity. Specifically in regards to software, ADA not accessing implementation-level evidence to check whether certain issues already have been addressed or could be addressed through SW fixes is a major constraint. As noted in the survey results, software commit information was requested, to be able to pinpoint whether software fixes had already been implemented, or are scheduled as part of a new release.

5.2.3 Non-determinism

Even with temperature zero for all LLMs invoked during the process, consistency in outputs is not reached, meaning the final output varies with the same starting point. This is expected behavior for LLMs due to the structure of cloud computing and difficulties associated with parallelization on potentially different hardware. Anthropic even notes in their own glossary that non-determinism is to be expected when using their API even using temperature zero [31].

The inconsistency in LLM outputs is a considerable challenge, since it impacts both the development and the evaluation itself. During development, there is never any guarantee that the implemented change improves the system in any considerable way, as the change in outcome could always be attributed to chance. Some other sources of non-determinism are given below.

The most obvious cause of non-determinism is the read operations from the database and knowledge base. Since data is continuously added and updated, a broad query

might simply capture different content than the previous, leading to the distribution being shifted in the next token prediction. Similarly, any code executed by the agents could have random effects and therefore generate the observed effect. Along with this, errors or unavailability of certain tools and subagents will cause the distribution to change of the next output token.

One of the more interesting sources is that the async operations themselves, inherent to the design, cause non-determinism. When the system calls multiple subagents in parallel with async operations, the ordering can vary from execution to execution which shifts the distribution of the next token. Since the manner in which the agent responds is largely based on integrations with external sources, latency for read and write operations impact the whole chain. However, this specific source of non-determinism could be amended, by ensuring a consistent order based on the tool call ids. But this would not solve the other causes of non-determinism.

Ultimately, non-determinism in LLM agents forces the problem to be framed as an optimization problem where we want to maximize the likelihood of generating a good outcome or answer. The methods used in this thesis and the empirical results indicates that there are structural changes that can be helpful in achieving some relative consistency in outcome, despite the inconsistency in procedure, but it only alleviates the underlying issue, which is exacerbated by the MAS structure.

5.2.4 Test case data

Generally the method for finding relevant test cases, would be through case documents, linking to an root-cause document. Ultimately, this proved discouraging due to issues with the data such as serial numbers not being present in case documents, serial numbers not having event-driven logs collected prior to the repair date, and the amount of documents referenced not available in ADAs knowledge base.

This is the main reason for the test case sample size being small for the LLM-as-judge portion. However, it is prudent to mention, that there is a fundamental constraint in the domain and method (LLMs) such that regardless of the actual test case sample size generalizing capability is hard to prove convincingly. The combinatorial explosion presented by the products with varying components, software, manufacturing and possible failure mechanisms makes exhaustive manual evaluation infeasible. There is also concept drift, i.e. the relation of the features (e.g. products, software) to the goal variable (root-cause) changes over time due to new functionality and design. Due to the probabilistic nature of the system itself, there is also never any guarantee that the next execution is not incorrect (although this is not necessarily likely). Therefore, it is difficult for any empirical evaluation to convincingly demonstrate generality in hardware diagnostics.

The conclusion is therefore that the philosophy for evaluation should be to assess the effectiveness of the system through regular human expert evaluation and automatic assessment of the faithfulness of answers in different scenarios, and the consistency

in generating the same answer.

5.3 System Value in an Industrial Context

Despite the challenges and biases described in the prior discussion chapters, the evaluation indicates that there are benefits from a systems perspective.

5.3.1 Perspective shift

One of the major findings of the report is that valuable execution in the proactive setting is not unfeasible, despite the observed system failures. Domain expert results showcase that there is utility in the output, although it varies between executions (and respondents). If it is provided proactively for problem clusters, it could help reduce initial triage time.

A core finding is also that the framing of results in the LLM case is as important as the output itself. In part, this concerns the actual output structure in the GUI and information included for a given issue. LLM's do hallucinate, and are not consistent naturally in this setting, but the implications of this limitation can be mitigated through the presentation. Two such mitigations are references and the transparent tree structure. However, there are concerns both from the perspective of adoption, trust, and utility, if every datapoint must be fact checked.

Regardless of the current caveats, the autonomous executions showing promise is a shift in use-case that can influence how the system will continue to evolve.

Constraining the workflow itself to only the proactive diagnostics case, increases the possibilities in terms of evaluation, feedback cycles and user engagement than in the open-ended chatbot scenario. Specifically, it becomes natural to evaluate root-cause assessment directly, the possibility for creating general purpose knowledge becomes larger, since it easier to extract from the uniform output, and more users will get engaged since the intended outcome is clear. Similarly, the existence of an end-to-end workflow provides useful information about the actual marginal benefit of each respective subagent and where it might improve.

5.3.2 End-to-end Evaluation paradigm

Although the evaluation methodology presented in this thesis has noticeable caveats, it does provide a realistic baseline for evaluation of the core diagnostics problem compared to typical agentic evaluation.

An illustrative example of a more static evaluation case would be a prompt like: "Create a bar chart comparing data in 2026" While this type of evaluation is crucial for assessing whether there is regression after new code is pushed, and as good software engineering procedure, it does not help in assessing whether the agents are closer to reasoning or achieving the root-cause analysis objective.

The test cases and problem framing defined in this thesis require a much more sophisticated orchestration of tooling to solve. Instead of a one-shot process it usually requires: log retrieval, log parsing to finding the most rich content in the logs, knowledge finding, verification and synthesis.

By forcing the end-to-end diagnostics workflow, the system itself surfaces many of the current bottlenecks for accurate analysis in this domain. For instance, coverage issues in initial screening, epistemic gaps and missing relational modeling. The end-to-end workflow effectively constitutes a testing ground for LLM agent capability against the core business objective, rather than synthetic benchmarks.

5.4 Next steps

This section describes some possible next steps for improving the solution itself, purely from a product perspective, firstly discussing adaptations to the core flow through context compression and presentation layer transparency.

5.4.1 Token usage and Presentation layer

The token usage in results is high, but as noted most of the token usage originates from L_1 and D. One of the findings during development is that the token usage here is mostly artificial due to executing many sequential steps, resulting in a large amount of LLM calls. There are a few alternatives available to decrease the token usage:

- **Context pruning**: Remove low semantic quality information from context (lessening the token accumulation)
- **Context compaction**: Summarize context (reducing context window usage, can have adverse effects)
- **Tool execution parallelization**: Increase parallelization of tool calls (decreases number of total LLM calls during subagent execution)
- **State externalization**: Externalize log state and let agent peek
- **Resource constraints**: Token budget and tool call restrictions.

While typically manageable, the end-to-end ADA workflow puts extra strain on L_1, as it is invoked often. This is something that must be amended for it to be feasible in production. The dynamic planner does not suffer as intensely from token accumulation but in certain situations, it chooses to execute tools sequentially, which impacts total token usage.

While multiple aspects of the final reports were criticized in the domain-expert evaluation, many of the comments made could be interpreted as actually concerning the presentation and transparency rather than the underlying mechanics of analysis.

Epistemic failures that lead to facts being conveyed incorrectly have already been discussed but there are multiple avenues for improvement in terms of the presentation of the result and how it is being conveyed given the constraints of the system. For instance, one could consider cutting the root-cause candidate assessment and just provide the symptoms analysis, which domain experts mostly found to be useful.

Another aspect could be to tackle directly one of the comments made in the survey, by including the raw log details, for expert verification immediately, with optional highlighting of the problematic lines from the agent perspective, this raw grounded information, is highly likely to result in better user trust in the system. An idea that was implemented into the system, surfaced during conversation with the domain experts, was the structured critical findings section, instead of a bullet list of symptoms. While these changes would not affect the model internals or shift the distribution of the next token, the manner in which the results are presented impact user adoption and by extension their willingness to test the system.

5.4.2 Fleet wide diagnostics and Trigger mechanism

During the thesis work, a topic frequently revisited was the actual trigger mechanism for enabling the autonomous investigation. While there are many possible avenues for this, the main bottleneck is latency and cost.

As discussed, the full ADA flow consumes a lot of tokens. The latency of each execution is mostly a infrastructure related topic, but nonetheless a factor in how the information is served and at what time during the day.

This means that running the flow on all events, would be unwise, and at some point a human engineer has to look at these reports, which is highly unfeasible. An alternative structure could be to look at problematic clusters, and running the full workflow for representative samples as a first triage. For instance, if one alarm type is overrepresented in a specific set of units. Examining multiple of these samples and corroborating these insights for a human engineer would typically be a considerable work, while LLM agents in theory present a highly scalable solution to this problem.

Another avenue to support larger systemic investigation could be to tweak the workflow itself. The agent inherently supports multi serial executions, but will act under the assumption that all units suffer from the same issue although it might clarify this later (which was tested in diagnostic snapshot executions). If the tree and structured output format is tweaked, the agent could output root-cause candidates, for instance divided by symptom cluster of a list of serials. There is an inherent ceiling here, due to context window size. But one could consider parallelizing the ADA flow as a whole, and summarizing the insights in a later step.

Although both avenues are interesting as a theoretical exercise, the main thesis findings highlight that there are more pressing factors, such as output consistency, before scaling up the solution.

5.4.3 Encoded domain expert investigation methods

Another important aspect of the system itself is that it tries to mimic the high level flow of a real human engineer working on an issue. However, there are nuances and details in every step of the chain, which impact the final result, which domain experts could drive the development of going forward. For instance, a human engineer might look at the initial symptom profile, and a specific alarm, and immediately recognize that it can't possibly be a configuration issue. An LLM can make similar assumptions, only based on its latent parameterization, which does not translate well into an enterprise context. Similarly, a human engineer might have a set workflow for any given task, such as always analyzing the logs with a specific set of commands in one order, to get a broad overview.

In the agentic system, the creation of piecemeal instructions or "skills" such as the ones defined by Zhang, Lazuka and Murang in an Anthropic blog post [32] could be beneficial. Skills, as described by the authors are dynamically loaded context injections. Even though it is not a revolutionary concept, it has some strengths in this domain / structure. For instance, one skill might define how the hypothesis tree should be structured for a specific hardware alarm, based on domain expert best practice.

5.4.4 Continuous Improvement

Continuous improvement is a common topic in research papers on LLM-based agents. Section 2.1 introduced Case-based reasoning as a general framework guiding the design of the system. CBR includes an explicit retention step, which falls outside the scope of the thesis because of evaluation requirements, but provides insight into possible future improvements.

The two main overarching methods for continuously updating memory in LLM-based agentic systems is parametric and non-parametric solutions [33]. As fine-tuning a SotA LLM is usually prohibitively expensive and faces the same data security challenges that were mentioned in Section 2.2 of the theory, non-parametric improvement is likely a more promising avenue.

While there are multiple possible non-parametric methods for continuous improvement, one of these methods continue on from the concept of "skills". Alzubi et al. [34] explore how a coding agent can iteratively build "skills" on its own, based on the execution trace, agent answer and ground-truth answer which can iteratively make the agent succeed on tasks it has previously struggled with. While ground truth is hard to come by in this domain, it is not entirely unfeasible to implement a similar automatic knowledge storing procedure.

When units are examined proactively by ADA, sooner or later, if these units indeed have an issue, will be returned and repaired or a software fix would be introduced. Since these events are documented and stored, it would be possible to keep track of executions run, and the final outcome, presenting a possible self-evaluation cycle

that could improve the existing knowledge base, with episodic memory based on previous executions and true outcomes.

Originally, this was meant to be explored in the thesis scope, but was ultimately discarded due to time-constraints. The exact mechanism for storing and the structure of these "learnings", "memories" or "skills" would however need to be specified further.

5.5 Future research

While the previous section gives an overview of possible improvements to the design and its value addition, from the academic perspective, there still are many angles to explore from here for solving core problem the thesis tackles. The development and formulation of the architecture has largely been exploratory, and for that reason it is conclusively not optimal.

For instance, one relevant research question that could be explored in this setting is whether different model providers have different capabilities in the setting, and or whether open-source LLM model trained on proprietary data could achieve better performance. Another could be to more systematically examine how the initial prompt affects the final output under the long investigation and multi-turn architecture. The architecture itself has been partly examined as part of the ablation setup to answer ***R3***, but many unexplored perspectives remain in terms of whether for instance, a single agent or multi-agent setup is optimal in this scenario, and what the pros and cons are for each.

Since the evidence for hypothesis tree style investigation is not sufficient to conclude correctness improvements, future work should consider constructing an automated test suite of multi-fault configurations, independent root-causes and highly ambiguous symptom mixtures. This could provide more rigorous empirical evidence for whether agents without explicit investigation structure do fall victim to one path pursuit and premature loop termination systematically.

While the markdown writing harness improves the presentation layer, one effect is exploration increase and by extension token usage increases. Future research could instead investigate how to determine whether the LLM is confident in the initial log analysis, and in this case route it through a lean direct caller pipeline as in Variant C, or if the opposite is true, dynamically let the system spin up the hypothesis tree, to force comprehensive path exploration. One could also consider defining the hypothesis tree as an explicit control mechanism by for instance, enforcing path coverage by injecting messages once the agent ends its execution if some path of the tree is not explored, more similar to the budget forcing mechanism presented by Muennighoff et al. [20]. Similar to suggestions from open-ended question in survey, you could potentially add investigation weight to some parts of a general hypothesis tree, forcing certain areas to be explored first.

Zooming out there doesn't seem to exist any LLM agent benchmarks specific to the HW diagnostics scenario. While this is perfectly reasonable, given that HW is usually associated with a real product under protection of intellectual property rights, it would be very beneficial to gauge how far LLMs have reached in the cross-section between electrical, mechanical, computer and software engineering knowledge. One of the core problems with research benchmarks especially in the agentic setting, is that they typically are a oversimplification of the modern enterprise reality. Data can be and is to a large extent highly unstructured, fragmented and of poor semantic quality.

6
Conclusions

This thesis aimed to explore the practical feasibility of LLM based agents for proactive hardware diagnostics. To this end, the research questions are to what extent a MAS is capable of autonomous diagnostics, how it should be evaluated, and whether explicit investigation scaffolding helps in this endeavor. The results are nuanced, but some conclusions can be drawn regarding each question.

R1: The results in this thesis indicate that the overall system provides outputs mostly in-line with domain expert expectations for a pre-screening mechanism in this domain, but also exhibits all failure modes defined in the taxonomy. Overall, the domain expert feedback shows that presentation of the results matter as much as the analysis itself, and the correctness required from the agent in terms of presentation of technical issues is high. ADA is capable of delivering analysis that can be helpful, but hard to trust at face-value requiring effort from a user to interpret the results. Consistency is a major constraint both to development and utility.

R2: Evaluation proved to be difficult in this setting with LLMs as the underlying solution for multiple reasons, such as inconsistency, domain complexity and concept drift. The conclusion is that while tempting to prove validity across the whole domain, it is largely unfeasible. The findings suggest that evaluation in this domain benefit from focusing on properties such as faithfulness to sources used and consistency in outcome instead of trying to evaluate the whole problem space. At the same time, domain-expert and human feedback is crucial, since domain experts are the only ones who can truly evaluate what is missing from an answer, and thus the improvements that can be made.

R3: The evaluation does not provide sufficient evidence to conclude that the explicit hypothesis tree substantially improves diagnostic correctness for the evaluated benchmark. However, the tree has observable impact on investigative behavior by increasing exploration breadth, and forcing more rationalization than observed without the structure. There is a fundamental tension where efforts to increase consistency by exploring more result in increased token usage to cover the search space in this domain. When knowledge retrieval functions optimally, the scaffolding introduces significant overhead without any major payoff, forcing multi-turn investigation for problems that could be resolved immediately.

Ultimately, the major takeaway and lesson learned is that while LLM agents are a very capable tool, the results are exceedingly hard to evaluate in complex engi-

neering domains, and regardless of the impression the output makes, in a business setting the fine details matter. As stated in the discussion, there are multiple possible avenues for improvement of overall system understanding and by extension consistency in outputs, which can alleviate some of the issues observed.

References

[1] N. Maslej et al., *Artificial intelligence index report 2025*, 2025. arXiv: 2504.07139 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2504.07139.

[2] J. Xu et al., "Openrca: Can large language models locate the root cause of software failures?" In *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=M4qNIzQYpd.

[3] D. Roy et al., *Exploring llm-based agents for root cause analysis*, 2024. arXiv: 2403.04123 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2403.04123.

[4] T. Kim, W. Park, H. Yun, and K. Lee, *Why do ai agents systematically fail at cloud root cause analysis?* 2026. arXiv: 2602.09937 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2602.09937.

[5] J. Liu, "Chatgpt: Perspectives from human–computer interaction and psychology," *Frontiers in Artificial Intelligence*, vol. Volume 7 - 2024, 2024, ISSN: 2624-8212. DOI: 10.3389/frai.2024.1418869. [Online]. Available: https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2024.1418869.

[6] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI Communications*, vol. 7, no. 1, pp. 39–59, 1994. DOI: 10.3233/AIC-1994-7104. eprint: https://journals.sagepub.com/doi/pdf/10.3233/AIC-1994-7104. [Online]. Available: https://journals.sagepub.com/doi/abs/10.3233/AIC-1994-7104.

[7] S. Guo, C. Deng, Y. Wen, H. Chen, Y. Chang, and J. Wang, *Ds-agent: Automated data science by empowering large language models with case-based reasoning*, 2024. arXiv: 2402.17453 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2402.17453.

[8] N. Wiratunga et al., *Cbr-rag: Case-based reasoning for retrieval augmented generation in llms for legal question answering*, 2024. arXiv: 2404.04302 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2404.04302.

[9] P. Lewis et al., *Retrieval-augmented generation for knowledge-intensive nlp tasks*, 2021. arXiv: 2005.11401 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2005.11401.

[10] S. S. Bhatt et al., *Enterprise ai must enforce participant-aware access control*, 2025. arXiv: 2509.14608 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2509.14608.

[11] Amazon, *What is rag? (retrieval-augmented generation)*, Nov. 2023. [Online]. Available: https://aws.amazon.com/what-is/retrieval-augmented-generation/.

[12] M. Azure, *What is rag (retrieval-augmented generation)?* [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-retrieval-augmented-generation-rag.

[13] G. Cloud, *What is retrieval augmented generation (rag)?* [Online]. Available: https://cloud.google.com/use-cases/retrieval-augmented-generation.

[14] Y. Gao et al., *Retrieval-augmented generation for large language models: A survey*, 2024. arXiv: 2312.10997 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2312.10997.

[15] B. Peng et al., "Graph retrieval-augmented generation: A survey," *ACM Trans. Inf. Syst.*, vol. 44, no. 2, Dec. 2025, ISSN: 1046-8188. DOI: 10.1145/3777378. [Online]. Available: https://doi.org/10.1145/3777378.

[16] S. Yao et al., *React: Synergizing reasoning and acting in language models*, 2023. arXiv: 2210.03629 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2210.03629.

[17] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23, New Orleans, LA, USA: Curran Associates Inc., 2023.

[18] T. Wei et al., *Agentic reasoning for large language models*, 2026. arXiv: 2601.12538 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2601.12538.

[19] C. Snell, J. Lee, K. Xu, and A. Kumar, *Scaling llm test-time compute optimally can be more effective than scaling model parameters*, 2024. arXiv: 2408.03314 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2408.03314.

[20] N. Muennighoff et al., *S1: Simple test-time scaling*, 2025. arXiv: 2501.19393 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2501.19393.

[21] X. Li et al., *Benchmark test-time scaling of general llm agents*, 2026. arXiv: 2602.18998 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2602.18998.

[22] A. Yehudai et al., *Survey on evaluation of llm-based agents*, 2026. arXiv: 2503.16416 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2503.16416.

[23] Anthropic, *Demystifying evals for ai agents*, Jan. 2026. [Online]. Available: https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents.

[24] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, *Ragas: Automated evaluation of retrieval augmented generation*, 2025. arXiv: 2309.15217 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2309.15217.

[25] J. Gu et al., "A survey on llm-as-a-judge," *The Innovation*, p. 101 253, 2026, ISSN: 2666-6758. DOI: https://doi.org/10.1016/j.xinn.2025.101253. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666675825004564.

[26] OpenAI, *Introducing deep research*, Feb. 2025. [Online]. Available: https://openai.com/index/introducing-deep-research/.

[27] Y. Shi et al., *Aime: Towards fully-autonomous multi-agent framework*, 2025. arXiv: 2507.11988 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2507.11988.

[28] Anthropic, "Claude sonnet 4.6 system card," Anthropic, System Card, Feb. 2026. [Online]. Available: https://www.anthropic.com/claude-sonnet-4-6-system-card.

[29] Anthropic, "System card: Claude haiku 4.5," Anthropic, Tech. Rep., 2025, Accessed via Anthropic's Transparency Hub. [Online]. Available: https://www.anthropic.com/claude-haiku-4-5-system-card.

[30] R. Likert, "A technique for the measurement of attitudes," *Archives of Psychology*, vol. 22, no. 140, pp. 1–55, 1932.

[31] Anthropic, *Glossary*. [Online]. Available: https://platform.claude.com/docs/en/about-claude/glossary.

[32] B. Zhang, K. Lazuka, and M. Murang, *Equipping agents for the real world with agent skills*, Oct. 2025. [Online]. Available: https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

[33] Y. Hu et al., *Memory in the age of ai agents*, 2026. arXiv: 2512.13564 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2512.13564.

[34] S. Alzubi, N. Provenzano, J. Bingham, W. Chen, and T. Vu, *Evoskill: Automated skill discovery for multi-agent systems*, 2026. arXiv: 2603.02766 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2603.02766.

A

Appendix A

A.1 Evaluator prompts

A.1.1 Correctness

Root-cause correctness evaluator

```
### Root Cause Correctness

You are evaluating an AI agent's root cause analysis.
The agent should propose the correct root-cause as a candidate, but it is
not required to be conclusive due to action constraints.  It should be a
first triage able to help a human engineer narrow the search.

Given the agent's final report and the known ground truth root cause,
determine:
- **correct** (bool):  Did the agent identify the same root cause?  Allow
different wording if meaning matches.
- **score**:  Linear scale based on how prioritized the correct root cause
is among the candidates:
- 1.0 = correct root cause identified as the most likely candidate
- 0.75 = correct root cause is the 2nd most prioritized candidate
- 0.5 = correct root cause appears mid-list among candidates
- 0.25 = correct root cause is the lowest-priority candidate
- 0.0 = correct root cause not proposed as a candidate at all
Interpolate linearly based on position in the candidate list relative to
total candidates.

Be strict on correctness, lenient on phrasing.

Input format:  Agent report + ground truth reasoning
Output schema:  {correct:  bool, score:  float, reason:  str}
```

A.1.2 Faithfulness

A.1.2.1 Claims extraction prompt

Claims extractor

```
### Claim Extraction

Extract all distinct factual claims from this root cause analysis report.

Include claims about:
- The identified root cause or failure mechanism
- Specific symptoms, alarms, or log patterns observed
- Data findings
- References to case documents, or root-cause documents
- Causal relationships between findings

Exclude:
- Procedural statements ("I investigated...", "The following tools were
used...")
- Hedging or uncertainty ("This might be...", "Further investigation
needed")
- Generic statements not specific to this case
- Confidence assessments
- Anything related to the hypothesis tree

Return each claim as a concise, self-contained assertion.

Input format:  Agent report
Output schema:  {claims:  list[str]}
```

A.1.2.2 Faithfulness evaluator prompt

Faithfulness evaluator

```
### Claim Grounding

For each claim below, determine if it is substantively supported by the
tool outputs provided.

A claim is grounded if the tool outputs contain data that confirms or
directly implies it.
A claim is NOT grounded if it goes beyond what the tool outputs show.

Claims:
{claims}

Tool outputs:
{tool_outputs}
```

```
Input format:  Numbered claims + concatenated tool outputs
Output schema:  {results:  list[{claim:  str, grounded:  bool, reason:
str}]}
```

Department of Mechanics and Maritime Sciences
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS