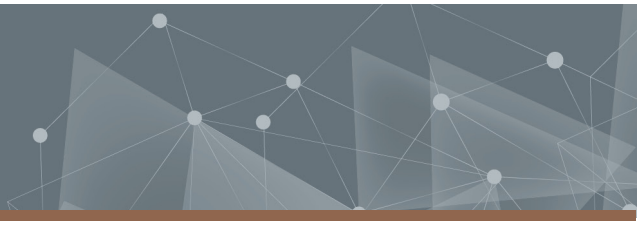
CHALMERS
UNIVERSITY OF TECHNOLOGY



An AI Agent for Exploratory Testing Guidance Using Historical Fault Data

A Case Study on Ericsson MINI-LINK

Master's thesis in Complex Adaptive Systems
Master's thesis in Computer Science - Algorithms, Languages and Logic

Min Yan and Yushu Hu

DEPARTMENT OF MATHEMATICAL SCIENCES

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

Degree project report 2026

An AI Agent for Exploratory Testing Guidance Using Historical Fault Data

A Case Study on Ericsson MINI-LINK

MIN YAN, YUSHU HU

CHALMERS
UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF MATHEMATICAL SCIENCES
Chalmers University of Technology
Gothenburg, Sweden 2026

An AI Agent for Exploratory Testing Guidance Using Historical Fault Data
A Case Study on Ericsson MINI-LINK
Min Yan, Yushu Hu

Industrial Supervisor: Kent Persson, Ericsson
Examiner: Johan Jonasson, Department of Mathematical Sciences

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

Typeset in LaTeX
Gothenburg, Sweden 2026

An AI Agent for Exploratory Testing Guidance Using Historical Fault Data
A Case Study on Ericsson MINI-LINK
Min Yan, Yushu Hu
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Exploratory testing is effective at finding defects that scripted tests miss, but it depends heavily on domain knowledge that is hard to transfer between testers. In telecommunications, where systems have thousands of configurable parameters, much of this knowledge stays locked in bug trackers and individual experience.

This thesis explores how AI can support exploratory testing by making historical fault data searchable and actionable. The result is a conversational assistant, the ETAgent, developed and evaluated at Ericsson for the MINI-LINK microwave product family.

The system builds a knowledge base from over 4,000 trouble reports. Unsupervised topic modelling discovers recurring fault patterns from verified reports, which are summarised into actionable guides. A hybrid retrieval pipeline combining sparse and dense search with cross-encoder reranking handles diverse query types, achieving an MRR of 0.815. A LangGraph-based agentic architecture lets the model choose retrieval strategies and synthesise guidance across multi-turn conversations.

In a blind evaluation, eleven domain experts preferred the ETAgent over both a foundation model and an enterprise baseline in 72% of comparisons. The advantage was clearest on diagnosis and knowledge queries, where retrieval-grounded answers provide concrete ticket references that neither baseline can offer. The system's value lies in surfacing specific historical evidence rather than general test planning advice, where the raw foundation model remains competitive.

Keywords: agentic retrieval-augmented generation, exploratory testing, large language models, bug report mining, topic modelling, telecommunications.

Acknowledgements

We would like to express our gratitude towards our supervisor Kent Persson from Ericsson for his support and guidance. His deep expertise and thoughtful mentorship taught us a lot during the entire master's thesis project. We would also like to thank our team manager Alexander Hjertén and colleagues from the Microwave Test team as well as the CI team for their support in providing expert experience, knowledge, and resources such as AWS and the kubernetes environment, as well as supporting our final evaluation and testing the prototype. We also thank our examiner Johan Jonasson at Chalmers University of Technology for his continuous academic guidance and insights.

Min Yan, Yushu Hu, Gothenburg, May 2026

List of Acronyms

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

ACE	Agentic Context Engineering
AI	Artificial Intelligence
AWS	Amazon Web Services
BERT	Bidirectional Encoder Representations from Transformers
BGE	BAAI General Embedding
BGP	Border Gateway Protocol
BLEU	Bilingual Evaluation Understudy
BM25	Best Matching 25
CLI	Command Line Interface
CPC	Context-aware Prompt Compression
CPI	Customer Product Information
c-TF-IDF	Class-based Term Frequency-Inverse Document Frequency
CTM	Combined Topic Model
DCN	Data Communication Network
DSR	Design Science Research
ELEX	Ericsson Library Explorer
EQP	Equipment Protection
ET	Exploratory Testing
GPT	Generative Pre-trained Transformer
GRPO	Group Relative Policy Optimization
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
HLAN	Hybrid LAN
hRLB	Hierarchical Radio Link Bonding
IR	Information Retrieval
JSONL	JSON Lines
KPI	Key Performance Indicator
LAN	Local Area Network
LDA	Latent Dirichlet Allocation
LLM	Large Language Model
MBB	Multi Bound Booster
MCP	Model Context Protocol

MDS	Multi-Document Summarisation
MRR	Mean Reciprocal Rank
NETCONF	Network Configuration Protocol
NIV	Node Integration Verification
NLP	Natural Language Processing
NPMI	Normalised Pointwise Mutual Information
NPU	Network Processing Unit
O&M	Operation and Maintenance
PCA	Principal Component Analysis
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
PTP	Precision Time Protocol
QA	Question-Answer
RAG	Retrieval-Augmented Generation
RAGAS	Retrieval Augmented Generation Assessment
RAPTOR	Recursive Abstractive Processing for Tree-Organized Retrieval
RAU	Radio Access Unit
RDS	Radio Deep Sleep
ReAct	Reasoning and Acting
RL	Reinforcement Learning
RLP	Radio Link Protection
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
RRF	Reciprocal Rank Fusion
SWAT	System-Wide Automation Test
TAF	Test Automation Framework
TDM	Time-Division Multiplexing
TF-IDF	Term Frequency-Inverse Document Frequency
TR	Trouble Report
UMAP	Uniform Manifold Approximation and Projection
VLAN	Virtual Local Area Network
WAN	Wide Area Network
XPIC	Cross-Polarization Interference Cancellation

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i, j	Indices for words, queries, or documents
t	Step index (test step or agent reasoning step)
r	Index over ranked lists

Sets

$\mathcal{Q}$	Test space ($\mathcal{C} \times \mathcal{X} \times \mathcal{O}$)
$\mathcal{C}$	Configuration space (hardware variants and feature flags)
$\mathcal{X}$	Stimulus space (input sequences)
$\mathcal{O}$	Observation schema space
$\mathcal{Y}$	Response space (observable outputs)
$\mathcal{S}$	Conversation state space
$\mathcal{A}$	Action space (natural-language replies or tool calls)
$\mathcal{A}_{\text{tool}}$	Subset of actions that are tool calls
$\mathcal{M}^*$	Set of message sequences
$\mathcal{K}_t$	Tester's knowledge at step t
$\mathcal{K}_t^{\text{sys}}$	System knowledge component
$\mathcal{K}_t^{\text{faults}}$	Fault hypotheses component
$\Delta(\mathcal{A})$	Probability simplex over the action space
R	Set of ranked lists from heterogeneous retrievers
Q	Set of evaluation queries
C_{resp}	Set of atomic claims extracted from a response

$\mathcal{R}$	Retrieved context (set of passages)
$G = (V, E)$	Directed graph with nodes V and edges E

Parameters

N	Test budget (number of tests) or top-N keywords
K	Number of latent topics
k	Top-k retrieval count or RRF smoothing constant
k_1	BM25 term-frequency saturation parameter
b	BM25 length normalisation parameter
avgdl	Average document length in the corpus
B_{full}	Full zone character budget (100 000)
B_{comp}	Compressed zone character budget (200 000)
T	Overflow threshold (50 000)
α	Hybrid retrieval interpolation weight

Variables and Functions

$q = (c, x, o)$	A test triple (configuration, stimulus, observation)
f	System under test mapping
$\text{ok}(c, x, y)$	Correctness predicate ($\in \{0, 1\}$)
$s_t = (M_t, h_t)$	Conversation state at step t
M_t	Ordered message history at step t
$\tilde{M}_t$	Context-window view of the history
h_t	Auxiliary bookkeeping (tool-call counter, metadata)
π	LLM stochastic policy
ρ_v	Router function at node v
ϕ	Deterministic tool execution map
a_t	Action sampled at step t
$\text{Obs}(s_t)$	Observation function (context construction)
d	A document
D	A document (in BM25 formulation)
$tf(q_i, D)$	Term frequency of term q_i in document D
$\text{IDF}(q_i)$	Inverse document frequency of term q_i

$P(w_i)$	Marginal probability of word w_i
$P(w_i, w_j)$	Joint probability of words w_i and w_j
$\text{rank}_r(d)$	Position of document d in ranked list r

Metrics

$\cos(\mathbf{a}, \mathbf{b})$	Cosine similarity between vectors $\mathbf{a}$ and $\mathbf{b}$
$\text{BM25}(D, Q)$	BM25 relevance score of document D for query Q
$\text{NPMI}(w_i, w_j)$	Normalised Pointwise Mutual Information
$\text{RRF}(d)$	Reciprocal Rank Fusion score for document d
Hit@k	Proportion of queries with a relevant result in top k
MRR	Mean Reciprocal Rank
Faithfulness	Fraction of response claims supported by retrieved context

Contents

List of Acronyms **ix**

Nomenclature **xii**

List of Figures **xxi**

List of Tables **xxiii**

1 Introduction **1**
1.1 Background . . . 1
1.2 Purpose and Goals . . . 2
1.3 Methodology Overview . . . 3
1.4 Limitations . . . 3
1.5 Thesis Outline . . . 4
1.6 Use of AI Tools . . . 4

2 Theory **5**
2.1 Exploratory Testing . . . 5
2.1.1 A Formal View . . . 5
2.2 Bug Report Mining and Pattern Discovery . . . 7
2.2.1 Bug Report Retrieval . . . 7
2.2.2 Topic Modelling . . . 9
2.2.3 Cluster Summarisation . . . 9
2.3 Retrieval-Augmented Generation . . . 10
2.3.1 Standard RAG Pipeline . . . 10
2.3.2 Advanced RAG Architectures . . . 11
2.3.3 Agentic RAG . . . 11
2.4 LLM Agent Architectures . . . 12
2.4.1 Agent Frameworks . . . 12
2.4.2 Tool Use and Integration . . . 12
2.4.3 Context Engineering . . . 13
2.4.4 A Formal Model of LangGraph ReAct Agents . . . 13
2.5 Evaluation . . . 14
2.5.1 Topic Coherence Metrics . . . 15
2.5.2 Information Retrieval Metrics . . . 15
2.5.3 Reference-Free Generation Metrics . . . 15
2.5.4 LLM-as-Judge . . . 16

2.5.5 Expert Evaluation 16

3 Methods **19**
3.1 Data Preparation 19
3.1.1 Dataset 19
3.1.2 Domain Glossary Extraction 20
3.1.3 Multimodal Enrichment and Text Normalisation 21
3.1.4 Structured Attribute Extraction 22
3.2 Fault Pattern Discovery 24
3.2.1 Embedding 24
3.2.2 Dimensionality Reduction 25
3.2.3 Clustering 26
3.2.4 Cluster Summarisation 27
3.3 Retrieval System Design 28
3.3.1 Document Indexing Strategy 28
3.3.2 Hybrid Retrieval with Cross-Encoder Reranking 29
3.3.3 External Documentation Search 31
3.4 Conversational Agent Design 31
3.4.1 Agent Architecture 32
3.4.2 Tool Integration 32
3.4.3 Conversation History Management 33
3.4.4 Prompt Design 34
3.4.5 Serving and User Interface 34
3.5 Evaluation Design 35
3.5.1 Retrieval Pipeline Comparison 35
3.5.2 Agent Quality Evaluation 35
3.5.3 Expert Blind Evaluation 36
3.6 Implementation Summary 37

4 Results **39**
4.1 Fault Pattern Discovery Quality 39
4.2 Retrieval Pipeline Comparison 41
4.3 Agent Quality Evaluation 42
4.3.1 Golden Dataset Evaluation 42
4.3.2 Real-Session Triplet Evaluation 43
4.4 Expert Blind Evaluation 44
4.4.1 Domain Enrichment Impact 44
4.4.2 Baseline Comparison 45

5 Discussion **47**
5.1 Answering the Research Question 47
5.2 Retrieval Architecture 48
5.3 Grounding and Hallucination 49
5.4 Domain Enrichment: Pattern Summaries 49
5.5 Expert Evaluation Insights 50
5.6 Design Trade-offs 50
5.7 Threats to Validity 52

6 Conclusion **53**
6.1 Contributions 53
6.2 Future Work 53

Bibliography **55**

List of Figures

1.1 The MINI-LINK 6000 product family, comprising packet microwave nodes and millimetre-wave radio units for mobile backhaul. 1

2.1 The exploratory testing cycle with AI augmentation. The ETAgent augments the tester's knowledge $\mathcal{K}_t$ by externalising fault hypotheses from historical bug reports and surfacing official product documentation. 7
2.2 The BERTopic pipeline [18]. Each step is modular; the default configuration shown here uses a transformer encoder, UMAP, HDBSCAN, and class-based TF-IDF. 9
2.3 Standard RAG pipeline. Top row: offline indexing. Bottom row: online query processing with retrieval from the vector store. 11
2.4 Minimal ReAct agent graph. The router ρ decides whether to invoke tools (continuing the loop) or terminate. Edge labels show the policy π sampling an action and the deterministic tool-execution map ϕ. . . 14

3.1 End-to-end system architecture. Data flows from raw sources through enrichment into the knowledge base, consumed by the agentic reasoning layer and exposed via a streaming web interface. 19
3.2 Four-stage enrichment pipeline. Each raw Jira issue is processed sequentially: image attachments are described via LLM vision capabilities, the description is cleaned of markup, image content is merged into the text, and comment threads are summarised. All stages except regex preprocessing are performed by the same LLM. 21
3.3 Structured attribute extraction per TR. If a configuration file is attached, rule-based parsing extracts both features and hardware; otherwise, an LLM extracts both from the TR title, description, and comments. Results are merged into the enriched records. 23
3.4 Fault pattern discovery pipeline. Each stage feeds into the next, producing structured fault pattern summaries from raw enriched records. 24
3.5 Hybrid retrieval pipeline. Dense and sparse retrievers each produce 15 candidates, fused via RRF and reranked by a cross-encoder to yield the final top-5 documents. 30
3.6 ETAgent state graph. The router ρ inspects the last AI message: tool calls with rounds below 8 continue to TOOLS; at the cap limit, synthetic results force a final answer; otherwise the agent terminates. 32

3.7 Three-tier history management. Recent turns are kept in full, older turns are shortened, and the oldest are summarised by Claude Haiku. 33

4.1 UMAP projection of issue embeddings coloured by cluster assignment. Embeddings were reduced via PCA(300) followed by UMAP ($n\_components$=8, $n\_neighbors$=20, min_dist=0.0), then projected to 2D for visualisation. 40
4.2 Retrieval performance across four pipeline configurations. 41

List of Tables

3.1 Agent tools and their execution strategies. 32
3.2 Key implementation parameters. 37

4.1 Clustering evaluation across representative configurations. Coh. and Interp. are mean LLM-as-judge scores (1–5 scale). 39
4.2 Ten largest fault pattern clusters discovered by the pipeline. 40
4.3 Overall retrieval performance across 180 queries. *The full pipeline combines Hybrid + RRF + Rerank. 41
4.4 MRR by query type and retrieval strategy. 42
4.5 MRR by document type and retrieval strategy. 42
4.6 Golden dataset results by difficulty level. 43
4.7 Real-session triplet evaluation scores, 1–5 scale, 83 turns. 43
4.8 Failure cause frequency across 83 evaluated turns. 44
4.9 Expert preference: with vs without pattern summaries (22 queries, 44 votes). 44
4.10 Expert preference across three systems, 175 votes from 11 reviewers. . 45
4.11 Expert preference by query category. 45

1

Introduction

1.1 Background

Ericsson's MINI-LINK product family provides microwave radio links for mobile backhaul, connecting cell towers and network sites to the core network. The product line spans multiple node families and millimetre-wave units, each differentiated by processing hardware, radio capabilities, and networking features. A single software release must be verified across this entire matrix of node types, hardware variants, radio configurations, and protocol features.

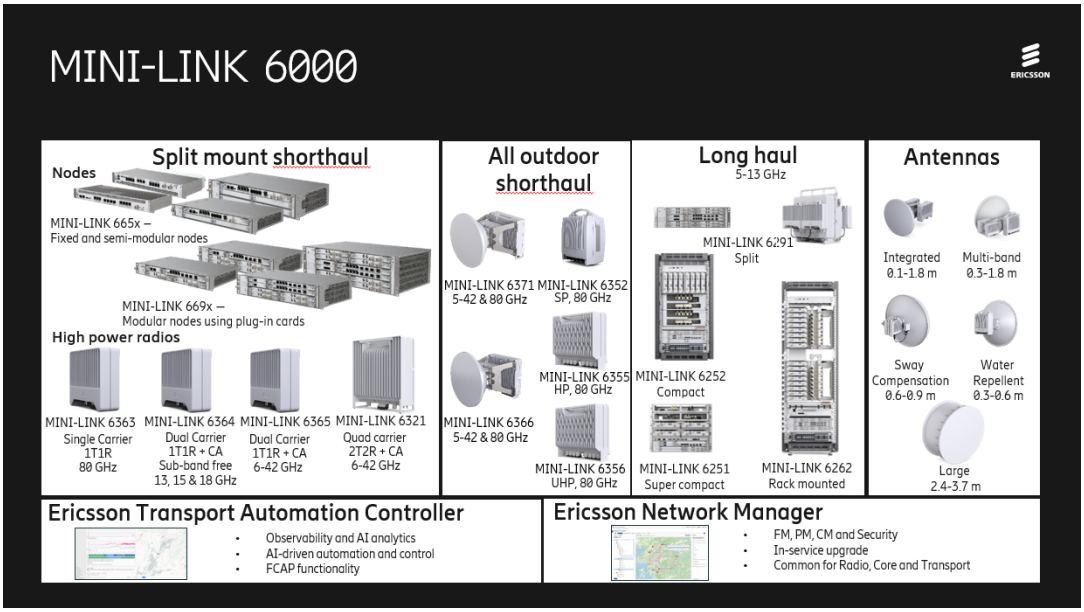


Figure 1.1: The MINI-LINK 6000 product family, comprising packet microwave nodes and millimetre-wave radio units for mobile backhaul.

The Node Integration Verification (NIV) team is responsible for this system-level verification. The team operates over fifteen physical test benches ranging from simple dual-node setups to multi-hop topologies with mixed node types and attenuators. Their daily workflow combines automated regression testing—thousands of scripted Java tests executed nightly via the TAF/SWAT framework—with manual exploratory testing sessions that target areas the automation cannot reach. When tests fail or anomalies appear, engineers investigate, reproduce, and file trouble reports (TRs) in the team's Jira tracker.

Scripted tests are effective at catching regressions against known requirements, but they cannot cover the combinatorial space of configurations and interactions that characterise real deployments [48]. Exploratory testing (ET) fills this gap: testers simultaneously design, execute, and evaluate tests guided by domain knowledge and real-time observations [3]. ET is particularly effective at uncovering defects that scripted tests miss [19], but its effectiveness depends heavily on the tester's experience and familiarity with historical failure modes [20].

In practice, the NIV team faces two challenges that limit ET effectiveness. First, much of the fault knowledge accumulated over years of testing resides in individual experience, informal discussions, and thousands of historical trouble reports [4]. When experienced testers leave or new members join, this knowledge is difficult to transfer. Second, the volume of historical data (over 4 000 TRs spanning six years) makes it impractical for any individual to maintain a comprehensive mental model of where faults tend to concentrate across the product's configuration space.

Recent advances in Large Language Models (LLMs) offer new possibilities. LLMs can reason about technical documentation and generate contextually relevant responses [7]. Retrieval-Augmented Generation (RAG) grounds these responses in domain-specific knowledge, reducing hallucination [26]. In software engineering, RAG-based systems have been applied to code generation, bug localisation, and documentation search [16], but their application to exploratory testing support remains largely unexplored. General-purpose LLMs also lack access to Ericsson's internal fault history and domain terminology—a gap that motivates a domain-specific approach.

Unsupervised techniques such as topic modelling can discover recurring fault patterns across large bug corpora [18], providing testers with a higher-level view of where failures concentrate. Combining these pattern-level insights with fine-grained retrieval over individual reports creates an opportunity to externalise the fault knowledge that experienced testers carry, making it accessible to the entire team through a conversational interface.

1.2 Purpose and Goals

The purpose of this thesis is to investigate how AI can augment the NIV team's exploratory testing workflow by leveraging historical bug data. The research addresses the following question:

> *RQ: How can AI augment exploratory testing of complex telecommunications equipment by leveraging historical bug data to guide test prioritisation and fault discovery?*

The thesis has the following goals:

1. **Build a domain-specific knowledge base** from historical trouble reports by enriching, normalising, and structuring issue data from the NIV team's Jira tracker.
2. **Discover recurring fault patterns** by applying unsupervised topic modelling to the enriched corpus, producing higher-level fault categories that reveal systematic areas of risk.

3. **Design and implement a retrieval system** that combines dense semantic search, sparse lexical search, and cross-encoder reranking to make the knowledge base accessible for diverse query types.
4. **Develop a conversational agent** that enables testers to interact with the system in natural language, receiving context-aware guidance on what to test, which areas carry the highest risk, and how past faults relate to their current work.
5. **Evaluate the system** to assess retrieval quality, fault pattern coherence, and the usefulness of the agent's guidance for exploratory testing.

1.3 Methodology Overview

The work is grounded in the following hypothesis:

> *By mapping expert knowledge into external retrieval tools (RAG over historical TRs and product documentation) and restricting the LLM's reasoning flow via a directed state graph, we can optimise its testing policy—enabling it to suggest targeted, historically-informed exploratory tests rather than generic advice.*

To test this hypothesis, the thesis follows a five-phase methodology:

1. **Data preparation:** Extract, enrich, and normalise 4 218 trouble reports and official product documentation into a structured knowledge base.
2. **Fault pattern discovery:** Apply BERTopic-style clustering to uncover macro-level fault categories from the corpus.
3. **Retrieval system:** Implement a hybrid dense+sparse retrieval pipeline with cross-encoder reranking.
4. **Agent implementation:** Build a LangGraph-based ReAct agent that autonomously selects retrieval tools, reasons over results, and generates test guidance.
5. **Evaluation:** Assess the system through retrieval metrics, pattern coherence scores, and blind expert evaluation against historical data.

The directed graph architecture (LangGraph) constrains the agent's reasoning to a well-defined loop of observe–reason–act, preventing unconstrained generation and ensuring that every response is grounded in retrieved evidence. This design directly addresses the hallucination and relevance problems observed with general-purpose LLMs applied to domain-specific testing tasks.

1.4 Limitations

The following limitations apply to this work:

- The system is developed and evaluated in the context of a single product family (MINI-LINK) within a single organisation. Generalisability to other products or domains is not assessed.

- The knowledge base is limited to historical trouble reports and official product documentation. Other sources such as design specifications, test plans, or operational logs are not included.
- The system provides advisory support only. It does not execute tests, interact with the equipment under test, or replace the tester's judgement.
- Evaluation is conducted within the scope of this thesis and does not include long-term deployment or longitudinal studies.
- The system relies on a cloud-hosted LLM (AWS Bedrock), introducing dependencies on external infrastructure and data sensitivity considerations.

1.5 Thesis Outline

The remainder of this report is structured as follows:

Chapter 2 presents the theoretical foundations: exploratory testing, bug report mining, retrieval-augmented generation, and LLM agent architectures.

Chapter 3 describes the system design and implementation, covering data preparation, fault pattern discovery, the hybrid retrieval pipeline, and the conversational agent.

Chapter 4 reports evaluation results across five dimensions: pattern quality, retrieval performance, domain enrichment impact, routing accuracy, and expert assessment.

Chapter 5 discusses findings, limitations, and implications for practice.

Chapter 6 summarises contributions and identifies directions for future work.

1.6 Use of AI Tools

AI-assisted tools were used during this thesis in the following capacities:

- **Implementation:** Claude and Kiro-cli assisted with code development for the data pipeline, retrieval system, and agent
- **Writing support:** Claude was used for proofreading and rephrasing individual sentences. All content was structured, reviewed, and validated by the authors.

The research design, methodology decisions, analysis, interpretation of results, and conclusions are entirely the authors' own work.

2

Theory

This chapter presents the theoretical foundations and related work underlying the AI-augmented exploratory testing assistant. It covers exploratory testing and its challenges, bug report mining, retrieval-augmented generation, and LLM agent architectures.

2.1 Exploratory Testing

Exploratory testing is a software testing approach in which test design, execution, and learning happen simultaneously [3]. Unlike scripted testing, ET relies on the tester's domain knowledge and real-time observations to guide the process.
Whittaker [48] describes ET through the analogy of software tours, where testers explore a system much like a tourist explores a city, following different strategies depending on the goal. These tours provide structured yet flexible approaches to coverage: focusing on features, boundaries, or error-prone areas.
Empirical studies show that ET is effective at finding defects that scripted tests miss, but its effectiveness varies significantly between testers [19]. This variation is largely explained by domain knowledge. Itkonen et al. [20] identify four knowledge categories that influence ET performance: system knowledge, domain knowledge, knowledge of common fault types, and knowledge of testing techniques. Experienced testers use this knowledge to form hypotheses about where faults are likely to occur.
This dependency on individual expertise creates two practical challenges. First, ET effectiveness is unevenly distributed across a team. Second, much of the relevant fault knowledge resides in individual experience and historical bug reports rather than in documented procedures [4]. When experienced testers leave, this knowledge is difficult to transfer. These challenges motivate using AI to capture and make accessible the domain knowledge that experienced testers rely on.

2.1.1 A Formal View

A compact notation helps state precisely what the ETAgent system aims to provide. The formalisation below names only the objects that the rest of the thesis refers to; it does not model the cognitive process of testing.

Test space. A *test* is a triple of a configuration, a stimulus, and an observation schema:

$$q \;=\; (c,\, x,\, o) \;\in\; \mathcal{C} \times \mathcal{X} \times \mathcal{O} \;\triangleq\; \mathcal{Q}. \tag{2.1}$$

Here c selects the hardware variant and feature flags, x is the input sequence applied to the system, and o specifies which responses are recorded (alarms, counters, logs). In practice, the valid stimuli and observations depend on the configuration because not every hardware variant supports every feature, so the effective test space is a strict subset of the full Cartesian product. Even so, $|\mathcal{Q}|$ remains effectively unbounded [48].

System under test. The system under test is modeled as a function f that describes how the equipment responds to a given configuration and stimulus:

$$f\colon \mathcal{C} \times \mathcal{X} \longrightarrow \mathcal{Y}, \tag{2.2}$$

That is, given a hardware setup and feature configuration c together with an input action x, the system produces an observable response $y = f(c, x) \in \mathcal{Y}$. For example, a set of alarms raised, counter values, or measured traffic throughput.
Whether a particular response constitutes correct behaviour is determined by a separate correctness criterion:

$$\mathrm{ok}\colon \mathcal{C} \times \mathcal{X} \times \mathcal{Y} \longrightarrow \{0, 1\}. \tag{2.3}$$

Here $\mathrm{ok}(c, x, y) = 1$ means the response y is acceptable for that configuration and stimulus, while $\mathrm{ok}(c, x, y) = 0$ indicates a fault. A test $q = (c, x, o)$ reveals a fault whenever $\mathrm{ok}\big(c,\ x,\ f(c, x)\big) = 0$.
Crucially, neither f nor ok is fully known to the tester. The system's behaviour is understood partially through documentation and prior testing; the correctness specification is often incomplete or ambiguous [20]. Exploratory testing exists precisely because of this partial knowledge.

ET objective. Given a budget of N tests, exploratory testing seeks to maximise the number of faults discovered:

$$\max_{q_1,\ldots,q_N} \sum_{t=1}^{N} \mathbf{1}\Big[\,\mathrm{ok}\big(c_t, x_t, f(c_t, x_t)\big) = 0\,\Big]. \tag{2.4}$$

Each q_t depends on all prior observations, making this a sequential decision problem whose quality is bounded by the tester's knowledge.

Knowledge gap. Following Itkonen et al. [20], the tester's knowledge at step t can be written as $\mathcal{K}_t = (\mathcal{K}_t^{\mathrm{sys}}, \mathcal{K}_t^{\mathrm{faults}})$. This groups system and domain knowledge into an approximate model of f, and knowledge of common fault types into *fault hypotheses* identifying sub-regions of $\mathcal{Q}$ believed to be fault-prone. The fourth of Itkonen's categories, testing techniques, governs how tests are selected rather than what is known.
In practice, both components are difficult to access. $\mathcal{K}_t^{\mathrm{faults}}$ is the scarce resource: experienced engineers carry rich hypotheses while junior engineers typically do not [4], and much of this knowledge is implicitly encoded in the organisation's bug-tracking system rather than in documented procedures. $\mathcal{K}_t^{\mathrm{sys}}$ is fragmented across hundreds

of interlinked internal documents that pretrained language models have no access to. The ETAgent addresses both gaps through retrieval-augmented generation: historical trouble reports and fault pattern summaries augment $\mathcal{K}_t^{\text{faults}}$, while retrieved product documentation augments $\mathcal{K}_t^{\text{sys}}$. Figure 2.1 illustrates how the agent fits into in the ET cycle.

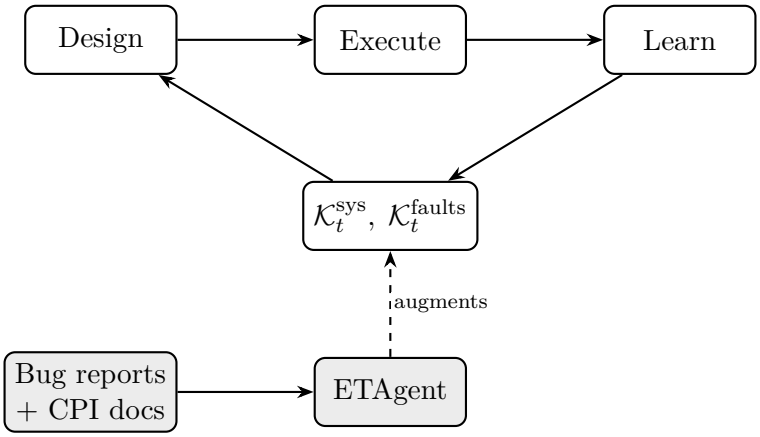


Figure 2.1: The exploratory testing cycle with AI augmentation. The ETAgent augments the tester's knowledge $\mathcal{K}_t$ by externalising fault hypotheses from historical bug reports and surfacing official product documentation.

The rest of this chapter covers the techniques behind this approach: extracting patterns from bug reports, retrieving relevant knowledge, and building an LLM agent that delivers it to testers.

2.2 Bug Report Mining and Pattern Discovery

Bug tracking systems accumulate thousands of reports over a project's lifetime, encoding latent knowledge about recurring failures, affected components, and resolution patterns. Mining this knowledge is a prerequisite for any system that aims to assist testers at scale. This section covers three threads: similarity-based retrieval, topic modelling, and cluster summarisation.

2.2.1 Bug Report Retrieval

Retrieving relevant bug reports from a large repository is the foundation of the RAG system developed in this thesis, since the agent must find historical faults that match a tester's current question. Early retrieval approaches treated bug reports as multi-field documents. The REP function [47] extended BM25F weighting to structured fields of a bug report and learned field-specific weights, establishing that incorporating metadata alongside free text substantially improves retrieval. Word embeddings were later combined with IR methods to capture semantic similarity beyond exact word matching [49].
A persistent problem is that semantically equivalent reports are often textually dissimilar. Enriching bug reports with natural-language explanations of domain

terms produces significant retrieval gains, with the largest improvements on textually dissimilar pairs [37]. This motivates preprocessing corpora before indexing rather than relying solely on the retrieval model.
The transition to transformer-based encoders marked a step change. A hybrid system using sentence-BERT as first-stage retriever followed by a RoBERTa reranker outperforms either approach alone [35]. Comparative studies confirm that BERT-based encoders consistently outperform TF-IDF and Word2Vec for semantic retrieval over bug corpora [41]. The GitBugs dataset [40] provides a large-scale benchmark confirming that retrieval over bug repositories remains non-trivial even with strong encoders.
The two core similarity measures used in retrieval are cosine similarity for dense vectors:

$$\cos(\mathbf{a}, \mathbf{b}) \;=\; \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}, \tag{2.5}$$

which measures semantic closeness independently of vector magnitude, and BM25 for sparse lexical matching, which scores a document D against a query $Q = \{q_1, \ldots, q_n\}$ by summing weighted term contributions:

$$\mathrm{BM25}(D, Q) \;=\; \sum_{i=1}^{n} \mathrm{IDF}(q_i) \;\cdot\; \frac{tf(q_i, D)\;(k_1 + 1)}{tf(q_i, D) + k_1 \left(1 - b + b\,\dfrac{|D|}{\mathrm{avgdl}}\right)}, \tag{2.6}$$

where $\mathrm{IDF}(q_i)$ is the inverse document frequency of term q_i (rarer terms receive higher weight), $tf(q_i, D)$ is the term frequency of q_i in D, $|D|$ is the document length, avgdl is the average document length, k_1 controls term-frequency saturation, and b controls length normalisation. BM25 excels at matching exact identifiers and error codes that dense embeddings may not distinguish.
When both dense and sparse retrieval are used in parallel, their ranked lists must be combined. Reciprocal Rank Fusion (RRF) merges on rank position alone:

$$\mathrm{RRF}(d) \;=\; \sum_{r \in R} \frac{1}{k + \mathrm{rank}_r(d)}, \tag{2.7}$$

where R is the set of ranked lists, $\mathrm{rank}_r(d)$ is the position of document d in list r, and k is a smoothing constant. Because RRF depends only on ordinal positions, it requires no score calibration between retrievers.
The merged candidates can then be refined by a cross-encoder reranker. Unlike bi-encoders, which encode query and document independently, a cross-encoder processes the query–document pair jointly through full transformer attention:

$$\mathrm{score}(q, d) \;=\; \sigma\Big(\mathbf{w}^{\top} \, \mathrm{Enc}(\texttt{[CLS]}\ q\ \texttt{[SEP]}\ d)_{\texttt{[CLS]}}\Big), \tag{2.8}$$

where σ is the sigmoid function and $\mathrm{Enc}(\cdot)_{\texttt{[CLS]}}$ is the encoder's output at the classification token. This joint attention provides finer-grained relevance estimation at the cost of higher latency, since each candidate requires a separate forward pass. In practice, cross-encoder reranking is applied only to a small candidate set (e.g. 15 documents) produced by the faster first-stage retrievers.

2.2.2 Topic Modelling

Retrieval finds individual reports matching a query, but cannot reveal the recurring fault patterns shared across many reports. Topic modelling discovers this macro-level structure, producing cluster-level knowledge that complements point-to-point retrieval. Latent Dirichlet Allocation (LDA) [6] remains the standard baseline, representing each document as a mixture over K latent topics. The DBTM system [38] augmented retrieval with LDA-derived topic features, improving duplicate detection by 5–7% points.
BERTopic [18] represents the dominant neural approach: embed documents with a transformer, cluster with HDBSCAN, then extract representative keywords via class-based TF-IDF (Figure 2.2). Top2Vec [1] follows a similar cluster-then-label strategy using Doc2Vec embeddings. The Combined TM (CTM) [5] integrates contextual embeddings into a neural variational framework. Of these, BERTopic is most suited to this work because it allows any pre-trained embedding model, handles variable cluster counts through HDBSCAN, and explicitly labels outliers as noise rather than forcing every document into a topic.

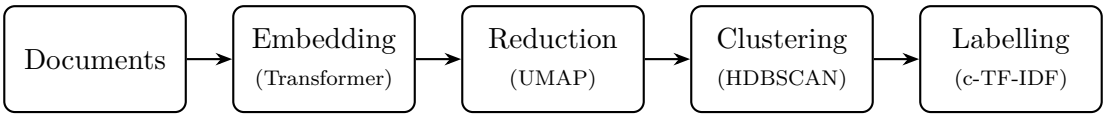


Figure 2.2: The BERTopic pipeline [18]. Each step is modular; the default configuration shown here uses a transformer encoder, UMAP, HDBSCAN, and class-based TF-IDF.

Topic coherence is measured by Normalised Pointwise Mutual Information (NPMI) over the top-N word pairs of a topic:

$$\mathrm{NPMI}(w_i, w_j) \;=\; \frac{\log \dfrac{P(w_i, w_j)}{P(w_i)\,P(w_j)}}{-\log P(w_i, w_j)}. \tag{2.9}$$

where $P(w_i)$ is the probability of word w_i appearing in a document and $P(w_i, w_j)$ is the probability of both words co-occurring in the same document. A topic's coherence is the mean NPMI across all $\binom{N}{2}$ pairs of its top-N keywords. Higher values indicate that the topic's words co-occur more than chance would predict, signalling a semantically coherent cluster. The numerator is the standard pointwise mutual information, measuring whether two words co-occur more than expected under independence. The denominator $-\log P(w_i, w_j)$ normalises the score to the range $[-1, +1]$, ensuring comparability across word pairs with different marginal frequencies: $+1$ indicates perfect co-occurrence, 0 indicates independence, and -1 indicates that the two words never co-occur.

2.2.3 Cluster Summarisation

Once reports are grouped by topic, the resulting clusters must be condensed into concise summaries that capture the shared pattern without losing critical detail.

This is a multi-document summarisation task complicated by noisy cluster boundaries and domain-specific terminology. Over-compression risks omitting relevant information, while insufficient abstraction leaves the output unwieldy.

Two broad strategies exist. Extractive methods select representative passages from the cluster, preserving original phrasing but often producing disjointed output. Abstractive methods generate new text that synthesises across documents, offering fluency at the cost of potential hallucination. Recent LLM-based approaches combine both: representative documents are selected from the cluster, then an LLM generates a coherent summary from them[55].

LLMLogAnalyzer [8] is particularly relevant to this work. It clusters system log events, then uses an LLM to summarise each cluster into patterns, root causes, and anomaly descriptions. By passing only representative cluster content rather than entire log files, it addresses context-window limitations while preserving semantic coverage, achieving 39–68% improvements over baselines that process logs without clustering. A topic-guided reinforcement learning approach [28] further shows that conditioning summarisation on explicit topic labels improves content selection and reduces drift.

No prior work applies cluster summarisation specifically to bug report corpora for the purpose of generating exploratory testing guidance. The approach adopted in this thesis (Chapter 3) follows the cluster-then-summarise paradigm, using domain glossary injection and product documentation as intermediate context to constrain the LLM's generation.

2.3 Retrieval-Augmented Generation

RAG addresses a fundamental limitation of LLMs: parametric knowledge is fixed at training time. By coupling a generator with a searchable knowledge base, RAG enables grounding outputs in retrieved evidence, reducing hallucinations and supporting knowledge-intensive tasks over private corpora. For the ETAgent, RAG is essential because the Ericsson bug reports and product documentation are proprietary and not part of any LLM's training data.

2.3.1 Standard RAG Pipeline

The foundational RAG framework [27] combines a language model with a dense vector index accessed through a neural retriever. Given a query, the retriever returns the top-k passages, which are concatenated with the query and passed to the model for answer generation. RAG has been widely adopted in software engineering [17], where combining LLMs with external retrieval consistently outperforms using the model's built-in knowledge alone.

The standard pipeline has three stages: chunk and embed documents offline, retrieve top-k neighbours at query time, then generate a response conditioned on retrieved context (Figure 2.3). Known failure modes include inability to handle queries requiring global corpus understanding and sensitivity to noisy context.

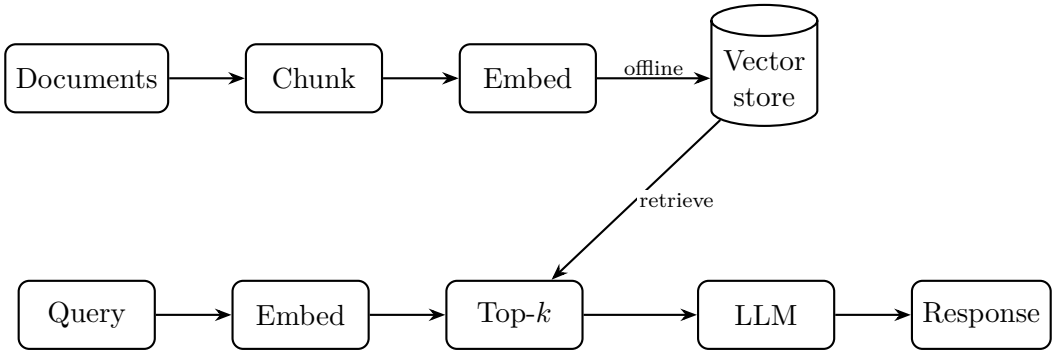


Figure 2.3: Standard RAG pipeline. Top row: offline indexing. Bottom row: online query processing with retrieval from the vector store.

2.3.2 Advanced RAG Architectures

RAPTOR [44] recursively clusters and summarises chunks into a tree, enabling retrieval at multiple abstraction levels. On multi-step reasoning benchmarks, RAPTOR with GPT-4 improved accuracy by 20% points over flat retrieval. Meta Knowledge RAG [36] generates synthetic question-answer pairs and cluster-level summaries offline, demonstrating that offline corpus preparation is a key lever for RAG quality. GraphRAG [13] extracts entity-relation graphs and precomputes community summaries, substantially improving comprehensiveness on global queries.
The shared insight is that enriching the index offline, through hierarchical summaries, synthetic QA pairs, or graph structure, addresses limitations that retrieval alone cannot solve. This principle motivates two design choices in this thesis: precomputing cluster-level fault pattern summaries rather than relying solely on individual report retrieval, and enriching the bug corpus with domain terminology and visual descriptions before indexing.

2.3.3 Agentic RAG

Standard RAG fixes the retrieval strategy in advance: every query triggers the same pipeline regardless of intent. Agentic RAG returns the retrieval decision to the model itself, allowing it to choose which sources to query, issue multiple retrieval calls, and decide when sufficient evidence has been gathered. A survey [46] identifies four core patterns: reflection, planning, tool use, and multi-agent collaboration.
A-RAG [12] demonstrates this concretely by exposing multiple retrieval interfaces (keyword search, semantic search, and chunk read) directly to the model as callable tools. The agent decides which tool to invoke at each step based on the query and intermediate results, outperforming static RAG while retrieving comparable or fewer tokens. The key argument is that neither single-shot retrieval nor predefined multi-step workflows qualify as truly agentic, since neither allows the model to adapt its strategy to the specific query.
This paradigm directly motivates the ETAgent design. A tester's question may require a pattern-level overview, a specific bug report lookup, or an official documentation check, and the appropriate retrieval strategy differs for each. Rather than routing all queries through a single fixed pipeline, the agent selects among

retrieval tools (semantic search, keyword lookup, pattern search, documentation search) based on its reasoning about query intent, and may call multiple tools in sequence when a single retrieval is insufficient.

Research gap. Existing work applies RAG to software engineering tasks such as code generation and bug localisation [17], and agentic architectures have been demonstrated on open-domain QA [12] and document analysis [8]. However, no prior system combines agentic RAG with unsupervised fault pattern discovery to support exploratory testing. The specific challenge addressed by this thesis — externalising implicit fault hypotheses from a historical bug corpus and delivering them as actionable test guidance through multi-turn conversation — has not been explored in the literature. The closest related systems either retrieve individual bug reports without macro-level pattern synthesis, or perform topic modelling without integrating the results into a conversational agent. This thesis bridges that gap.

2.4 LLM Agent Architectures

An LLM agent extends a language model beyond passive generation by equipping it with planning, tool calling, and memory. This section covers frameworks, tool integration, context management, the formal agent model, and evaluation.

2.4.1 Agent Frameworks

A survey of agent architectures [31] characterises the design space along two axes: single-agent patterns (ReAct, Reflexion) and multi-agent patterns where specialised agents collaborate and divide labour.
ReAct [50] is the single-agent pattern adopted in this thesis. It interleaves reasoning traces with tool calls in a loop: the model reasons about what information it needs, calls a tool to obtain it, observes the result, and decides whether to continue or answer. This think-act-observe cycle repeats until the model judges it has sufficient evidence to respond.
LangGraph [24] provides the execution substrate for implementing such patterns. It models agent workflows as directed graphs with cycles, conditional branches, and persistent state, allowing developers to define exactly when the agent should loop, branch, or terminate. A survey of workflow systems [53] finds broad convergence on Python-based execution chains but persistent fragmentation in tool schemas and message formats, motivating standardised protocols such as MCP (Section 2.4.2).

2.4.2 Tool Use and Integration

Toolformer [45] demonstrated that language models can learn to decide which APIs to call in a self-supervised manner, achieving performance competitive with much larger models. The Model Context Protocol (MCP) [2] provides a universal open standard for connecting agents to external tools, replacing fragmented custom integrations. Context-Aware MCP [21] extends this with a shared context store, en-

abling servers to coordinate without repeated LLM round-trips and reducing response failures on planning benchmarks.

2.4.3 Context Engineering

As agent systems grow in complexity, managing what information the model sees, and when, becomes a first-class engineering concern. A comprehensive survey [33] formalises context engineering as a discipline spanning retrieval, generation, processing, and management. ACE [56] treats contexts as evolving playbooks that prevent brevity bias and context collapse. Mem0 [11] provides a persistent memory layer for long-term fact extraction across sessions. Acon [22] maintains structured, versioned context stores that agents query across task boundaries. These systems address a shared problem: as multi-turn conversations grow, naive truncation loses critical context while full retention exceeds the model's effective window. The ETAgent addresses this with a three-tier scheme (Section 3.4.3): recent messages are kept in full, older messages are shortened to just the questions and answers, and the oldest are summarised into a brief paragraph by a smaller model.

2.4.4 A Formal Model of LangGraph ReAct Agents

The preceding sections described three concerns in tool-augmented LLM agents: orchestration, tool invocation, and information management. This subsection provides a compact mathematical model of a LangGraph ReAct agent that names the objects the evaluation metrics ultimately act on. Figure 2.4 illustrates the minimal graph structure.

State. The conversation state at step t is

$$s_t \;=\; (M_t,\; h_t), \tag{2.10}$$

where M_t is the ordered message history and h_t is session metadata (e.g. tool-call counter, thread identifier, compression state).

Graph and router. A LangGraph agent models execution as a directed graph $G = (V, E)$ where each node is a processing step (e.g. running the LLM, executing a tool) and edges define the possible transitions between steps. At branching points, a router selects which edge to follow based on the current state:

$$\rho_v \colon \mathcal{S} \;\longrightarrow\; \{\, u : (v, u) \in E \,\}. \tag{2.11}$$

In the ReAct pattern, the key router sits after the agent node and decides: continue to the tools node (if the LLM requested a tool call) or terminate (if the LLM produced a final answer).

Policy. The language model acts as a stochastic policy over an action space $\mathcal{A}$ (natural-language replies or tool calls):

$$\pi(\,\cdot \mid \tilde{M}_t\,) \;\in\; \Delta(\mathcal{A}), \tag{2.12}$$

where $\tilde{M}_t$ is the context-window view of the history (possibly truncated or summarised from M_t).

Tool map. Tool execution is deterministic: given a tool call, it returns one or more result messages $\mathcal{M}^*$ that are appended to the conversation history.

$$\phi \colon \mathcal{A}_{\text{tool}} \longrightarrow \mathcal{M}^*. \tag{2.13}$$

All stochasticity in a ReAct trajectory enters through π; ϕ contributes only deterministic feedback from the retrieval system.

ReAct step. One iteration of the agent loop works as follows: the context-window view $\tilde{M}_t = \text{Obs}(s_t)$ is constructed from the full history (applying compression if needed), the language model samples an action a_t based on what it sees, and the action together with any tool results are appended to the message history. The router then decides whether to loop (if the action was a tool call) or terminate (if the action was a final answer) [50, 24]. This cycle repeats until the agent answers or the safety cap forces termination.

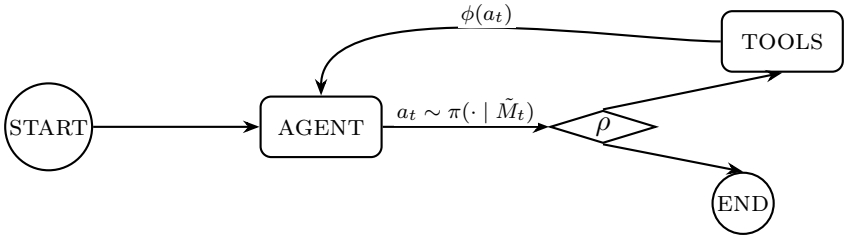


Figure 2.4: Minimal ReAct agent graph. The router ρ decides whether to invoke tools (continuing the loop) or terminate. Edge labels show the policy π sampling an action and the deterministic tool-execution map ϕ.

2.5 Evaluation

Evaluating LLM-based systems differs fundamentally from traditional software testing. Outputs are stochastic and open-ended, meaning there is rarely a single correct answer against which to compute exact-match accuracy. Moreover, for generative tasks such as summarisation or question answering, multiple valid surface forms may express the same correct content. Yu et al. [54] survey evaluation approaches for RAG systems and conclude that reference-based metrics like BLEU or ROUGE are theoretically insufficient for this setting, as they penalise valid paraphrases and reward surface-level overlap regardless of factual correctness.

A key architectural concern is whether to evaluate the system end-to-end or decompose evaluation by component. End-to-end metrics capture overall output quality but cannot isolate whether degradation originates in retrieval, context selection, or generation. Salemi and Zamani [43] demonstrate that component-level evaluation, measuring each stage against its own criteria, is essential for diagnosing failures in

RAG pipelines. This decomposition is particularly important for agentic RAG systems, where retrieval failures, routing errors, and generation hallucinations require fundamentally different remediation strategies.

2.5.1 Topic Coherence Metrics

As introduced in Section 2.2.2, NPMI (Equation 2.9) quantifies whether a topic's top words co-occur beyond chance. Lau et al. [25] conduct a large-scale comparison of automated coherence measures against human topic quality judgements and find that NPMI achieves the highest correlation, outperforming alternatives such as UCI and UMass coherence. This makes NPMI the standard automated proxy for cluster quality in topic modelling pipelines.

2.5.2 Information Retrieval Metrics

Retrieval system evaluation relies on rank-aware metrics that measure both whether relevant documents are retrieved and where they appear in the ranked list. Manning et al. [30] formalise the standard evaluation framework for information retrieval, defining Hit@k as a binary indicator of whether the ground-truth document appears within the top k results:

$$\text{Hit@}k = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \mathbf{1}[\text{rank}_i \leq k]. \tag{2.14}$$

Mean Reciprocal Rank (MRR) rewards systems that place the relevant document higher:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i} \tag{2.15}$$

where rank_i is the position of the first relevant document for query i. Together, Hit@k captures recall while MRR captures ranking precision.
Ablation studies isolate the contribution of individual pipeline components by systematically removing or adding them and measuring the resulting performance change. Meister and Cotterell [34] review ablation methodology in NLP and argue that proper ablation requires controlling for confounding variables and reporting statistical significance, not merely comparing raw scores. For hybrid retrieval architectures, ablation quantifies whether each component (semantic embeddings, BM25, cross-encoder reranking) provides additive value or introduces redundancy.

2.5.3 Reference-Free Generation Metrics

Evaluating generated text without human-annotated ground truth requires metrics that assess factual grounding from retrieved context alone. Es et al. [15] introduce RAGAS, a reference-free framework that decomposes RAG evaluation into four independent dimensions: faithfulness, answer relevance, context precision, and context recall. Each dimension targets a different failure mode, enabling practitioners to pinpoint whether quality issues originate in retrieval or generation. The faithfulness

metric decomposes the generated answer into individual claims and verifies each against the retrieved context:

$$\text{Faithfulness} = \frac{|\{c \in C_{\text{resp}} : c \text{ is supported by } \mathcal{R}\}|}{|C_{\text{resp}}|} \tag{2.16}$$

where C_{resp} is the set of atomic claims extracted from the response and $\mathcal{R}$ is the retrieved context. For domain-specific applications where hallucination of technical terminology poses safety risks, this claim-level decomposition enables precise diagnosis of generation errors.
When combined with controlled ablation, such as comparing outputs with and without injected domain knowledge, faithfulness scoring isolates the causal effect of specific enrichment components on hallucination rates.

2.5.4 LLM-as-Judge

Criteria-based evaluation using LLM-as-judge has emerged as a scalable alternative to human annotation for assessing open-ended generative outputs. Li et al. [29] provide a comprehensive survey of LLM-based evaluation methods and categorise them into pointwise scoring, pairwise comparison, and listwise ranking. Rather than assigning a single holistic score, decomposed criteria-based evaluation assesses responses along predefined quality dimensions such as factual accuracy, source citation, and actionability.
However, LLM judges exhibit systematic biases. Ye et al. [51] identify position bias (favouring responses presented first), verbosity preference (rating longer answers higher regardless of quality), and self-preference (favouring outputs stylistically similar to the judge's own generation). Using the same model as both generator and judge amplifies this problem; Panickssery et al. [39] show that models consistently rate their own outputs higher than those from other models of comparable quality, even when evaluators cannot explicitly identify authorship.
Prompt design also significantly affects judge reliability. Kim et al. [23] demonstrate that fine-grained, dimension-specific rubrics substantially improve correlation with human ratings compared to generic evaluation instructions. Their Prometheus model, trained explicitly on rubric-based evaluation, achieves near-human agreement when provided with detailed scoring criteria, but degrades to near-random performance with vague prompts such as "rate the quality of this response."
For tool-calling agents, routing accuracy measures whether the agent selects the correct tool for a given query intent. Patil et al. [42] formalise tool selection as a function-call classification problem, constructing a benchmark of over 1,600 API calls and demonstrating that first-turn accuracy is a reliable proxy for downstream task success in multi-tool agents. Confusion matrices reveal whether misrouting errors are systematic or random, enabling targeted improvements to the agent's tool selection logic.

2.5.5 Expert Evaluation

While automated metrics assess system properties in isolation, the ultimate validation of an AI-assisted tool requires human judgement of its practical utility in

context. Yehudai et al. [52] survey evaluation methods for LLM-based agents and identify human evaluation as irreplaceable for assessing real-world utility, particularly when automated metrics fail to capture domain-specific correctness and actionability.

Blind preference evaluation presents experts with outputs from different system configurations without revealing which configuration produced each output. Zheng et al. [57] demonstrate that pairwise blind comparison produces more reliable and reproducible rankings than absolute scoring, as it eliminates individual differences in scale interpretation and anchoring effects. Experts select the preferred response based on predefined quality criteria, controlling for expectation bias that may inflate or deflate ratings of AI-generated content.

3
Methods

This chapter presents the methodology for designing an AI-augmented exploratory testing assistant for MINI-LINK microwave network equipment. The system is developed as a five-phase pipeline: data preparation, fault pattern discovery, retrieval system design, conversational agent design, and evaluation. The core research question is:

> *RQ: How can AI augment exploratory testing of complex telecommunications equipment by leveraging historical bug data to guide test prioritisation and fault discovery?*

Figure 3.1 provides an end-to-end overview of the resulting system.

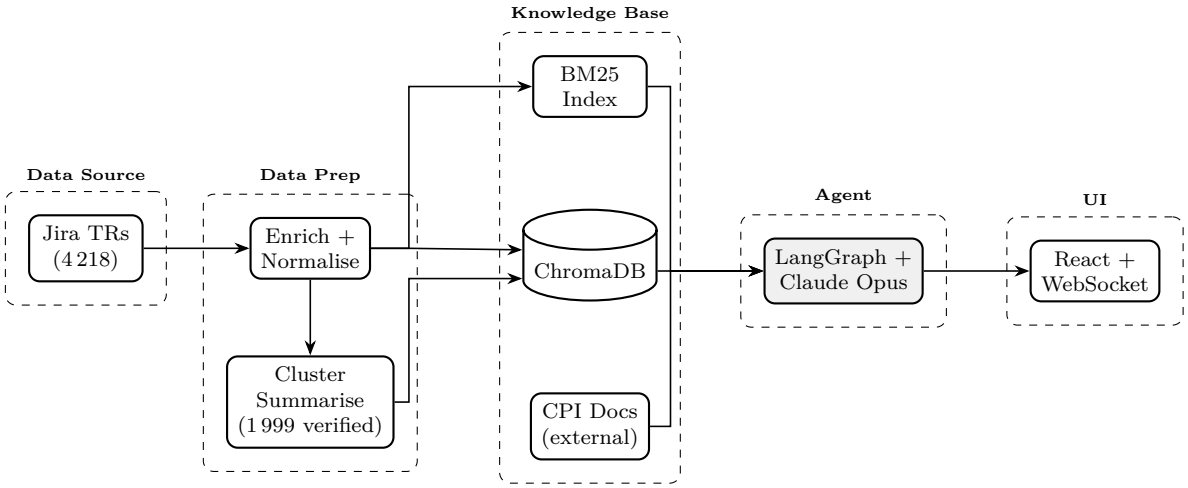


Figure 3.1: End-to-end system architecture. Data flows from raw sources through enrichment into the knowledge base, consumed by the agentic reasoning layer and exposed via a streaming web interface.

3.1 Data Preparation

3.1.1 Dataset

The system draws on two internal Ericsson data sources that together cover both historical fault evidence and authoritative product knowledge. Trouble reports document what bugs have occurred and how to reproduce them, while CPI documentation provides the foundational technical knowledge for the product, covering system

architecture, configuration procedures, and expected behaviour. This combination enables the agent to ground its exploratory testing guidance in real failure history while referencing correct product specifications.

- **Trouble Reports (TRs):** Exported from a Jira-based issue tracker for the MINI-LINK product family. A total of 4 218 TRs associated with the ML66/AOD Node Integration Verification (NIV) test dashboard, created between 2020-06 and 2026-04, were extracted. All 4 218 issues are indexed for retrieval. For the fault pattern discovery pipeline (Section 3.2), only the 1 999 issues resolved as *Verified OK* are used, ensuring that only confirmed, real bugs enter the clustering process and excluding duplicates, invalid reports, and issues that could not be reproduced. Each record includes structured metadata (issue key, creation and update timestamps, severity, closed_as) and unstructured fields (title, description, comments, image attachments).
    1. **Retrieval knowledge base:** All 4 218 TRs are indexed into the vector store for individual issue retrieval. Even reports resolved as *Not a bug* or *Not reproducible* contain valuable investigation context (symptoms observed, components involved, and troubleshooting steps attempted) that can inform exploratory testing decisions.
    2. **Fault pattern discovery:** Only the subset of 1 999 TRs resolved as *Verified OK* is used for topic modelling and cluster summarisation. This filtering ensures that the macro-level fault patterns reflect confirmed defects rather than noise from misreported or unreproducible issues.
- **Customer Product Information (CPI):** Official technical documentation from the MINI-LINK product family, specifically the latest release (M26.Q1, 2026-03-20). These documents describe high-level product descriptions (e.g., system nodes, hardware components, and software features) as well as detailed operation and maintenance (O&M) manuals [14]. They cover critical areas such as planning, installation, configuration activities, alarm handling, fault management, and troubleshooting. The documentation provides essential contextual background for understanding the product's structure and operational boundaries. Notably, the documents in CPI are systematically organized and extensively interlinked, forming connected information flows that are particularly valuable for guiding complex troubleshooting use cases. The files are available in HTML and PDF formats, containing text, figures, and tables, and everything can be accessed centrally through the Ericsson Library Explorer (ELEX).

3.1.2 Domain Glossary Extraction

Within Ericsson, the telecommunications environment utilizes a vast number of domain-specific, company-specific, and department-specific acronyms. When processing these texts with standard pre-trained LLMs, the models frequently generate hallucinated explanations. For example, the common Ericsson term **hRLB** (Hierarchical Radio Link Bonding) is often incorrectly expanded by the LLM as "Hitless Radio Link Bound", and **MBB** (Multi-Band Booster) is misinterpreted as "Make

before Break". To address this, a domain glossary was constructed in three steps:

1. **Official extraction:** The Ericsson Glossary of Terms and Acronyms PDF was parsed using PyMuPDF, using font formatting (bold text) to separate terms from definitions. This yielded 719 unique terms.
2. **TR-based expansion:** A Python script scanned titles, descriptions, comments, and root cause fields across all 4 218 TRs (before filtering) using targeted regular expressions for telecom abbreviation formats (uppercase, camelCase prefixes, versioned suffixes). After noise filtering, 139 new acronym candidates not present in the official glossary were identified.
3. **Expert validation:** The combined list was reviewed with Ericsson engineers and cross-checked against CPI documents. After deduplication and filtering, the final glossary contains 692 terms stored in JSON format, where each entry contains the acronym, its full expansion, and its technical definition.

3.1.3 Multimodal Enrichment and Text Normalisation

Raw Jira issues contain a mix of markup, template boilerplate, image references, and conversational comment threads. Before indexing, each record passes through a four-stage enrichment pipeline, using Claude Sonnet 4.6 (via AWS Bedrock) for all generative steps.

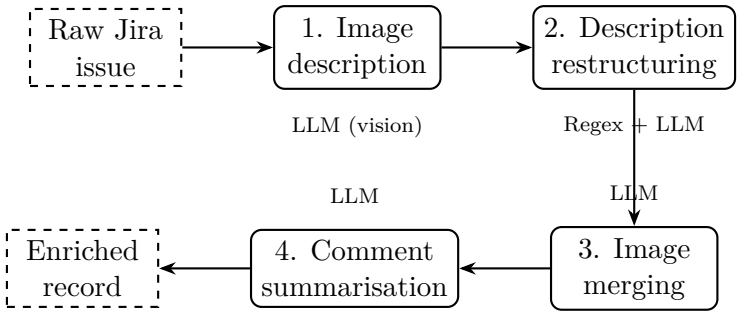


Figure 3.2: Four-stage enrichment pipeline. Each raw Jira issue is processed sequentially: image attachments are described via LLM vision capabilities, the description is cleaned of markup, image content is merged into the text, and comment threads are summarised. All stages except regex preprocessing are performed by the same LLM.

Stage 1: Image description

Bug reports frequently include visual evidence, screenshots of alarm panels, CLI terminal output, traffic graphs, and network topology diagrams, that is invisible to text-based retrieval. For each TR, the pipeline scans the downloaded attachment directory for image files (PNG, JPEG, GIF, WebP). Each image is base64-encoded (resized if above 3.75 MB), paired with the parent issue's description as context, and sent to LLM (with image input) with the instruction:

> *"Translate this image into plain text that captures ALL visible information: values, timestamps, alarm names, interface names, IP addresses, status indicators, graph trends."*

Results are cached to avoid reprocessing. Across the corpus, this step converted several hundred screenshots into searchable text.

Stage 2: Description restructuring

Jira descriptions arrive with wiki markup, HTML fragments, `noformat` tags, and empty template headings (e.g. "SHORT DESCRIPTION", "HOW TO REPRODUCE THE FAULT"). A regex-based pass strips these template headings and boilerplate instructions, then the LLM removes remaining markup while preserving all technical content unchanged. The output is a clean, plain-text description.

Stage 3: Image merging

For issues that contain both a description and image descriptions, the LLM replaces inline image references (e.g. `!screenshot.png!`) with a concise version of the corresponding image content. Images not referenced inline are appended only if they contain fault-relevant information. The result is stored as `full_description`—a self-contained narrative combining text and visual evidence.

Stage 4: Comment summarisation

Jira comment threads often span dozens of messages mixing investigation findings with conversational noise (greetings, status requests, reassignments). The LLM summarises each thread into a concise technical update covering: investigation findings, reproduction steps, relevant logs and error messages, workarounds applied, and resolution status. Threads with no technical content are discarded.
The final enriched record for each TR contains: the original title, a cleaned description, a full description with merged image content, a summarised comment thread, and a dictionary of per-image text descriptions. These fields form the text that is later embedded and indexed for retrieval.

3.1.4 Structured Attribute Extraction

Beyond free text, each TR is annotated with structured metadata extracted from the device configuration files (`.cfg`) attached to many issues. These files contain the complete CLI configuration of the node under test at the time of failure. Two dimensions are extracted:

Configuration features (rule-based)

A rule-based Python script parses the CLI blocks and detects enabled features by matching specific line patterns. The parser first splits the configuration into top-level blocks (lines at column 0) and their indented sub-lines, then applies pattern-matching rules for each feature category.

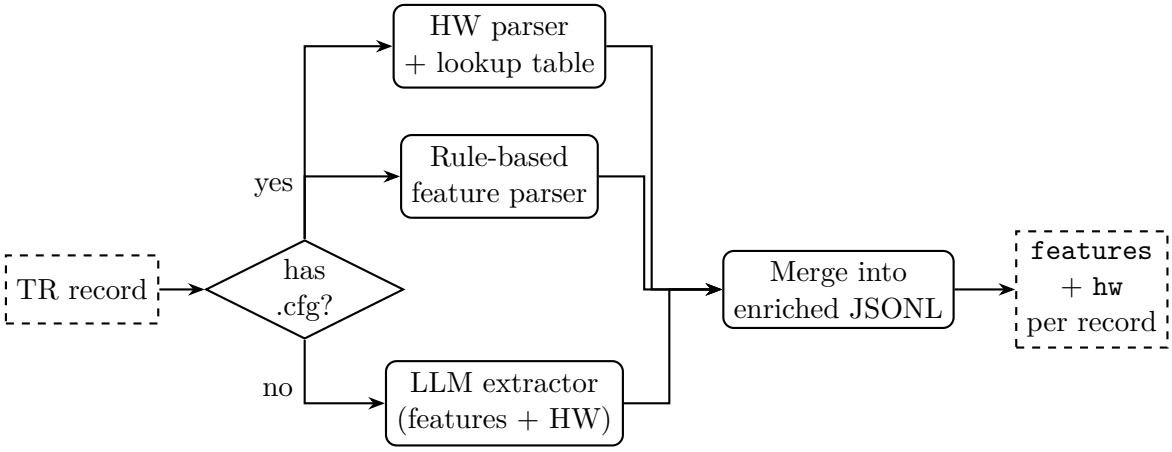


Figure 3.3: Structured attribute extraction per TR. If a configuration file is attached, rule-based parsing extracts both features and hardware; otherwise, an LLM extracts both from the TR title, description, and comments. Results are merged into the enriched records.

LLM-assisted extraction was intentionally avoided for this step. The strict, deterministic CLI syntax is better served by exact pattern matching—generative models tend to hallucinate features from superficially similar keywords (e.g. interpreting "protection switching" as NPU-PROTECTION).

For TRs without attached configuration files, a complementary LLM-based extractor identifies features mentioned in the title, description, and comments using a detailed feature catalog prompt with synonym mappings.

Hardware identification

Node types and board variants are extracted from configuration files by another Python script. The parser identifies three hardware categories: chassis (node type), boards and RAUs (Radio Access Units).

For TRs without configuration files, an LLM fallback extracts hardware mentions from the issue text using a structured prompt with the full hardware catalogue as reference.

Merge step

The extracted features and hardware are merged back into the enriched JSONL records as per-node dictionaries. Each record's `features` field maps node identifiers to their detected feature sets, and the `hw` field maps nodes to their board and RAU inventory. These structured fields are stored alongside each document and serve two purposes: as contextual metadata displayed in retrieval results (enabling the agent to reference specific hardware and feature configurations in its responses) and as categorical inputs during topic modelling. They are kept separate from the main text to avoid diluting the semantic signal used for embedding and retrieval.

3.2 Fault Pattern Discovery

A fundamental limitation of naive RAG is that it retrieves information point-to-point: given a query, it returns the top-k most similar documents. This works well for specific questions but fails for broader ones like "what are the most common failure modes after a software upgrade?" Answering such questions requires macro-level knowledge that spans many individual reports. To extract this knowledge, an unsupervised topic modelling pipeline was applied to the enriched dataset, producing structured fault pattern summaries that the agent can retrieve alongside individual bug reports.

The pipeline has four stages: embedding, dimensionality reduction, density-based clustering, and LLM-driven cluster summarisation. Each stage is described below.

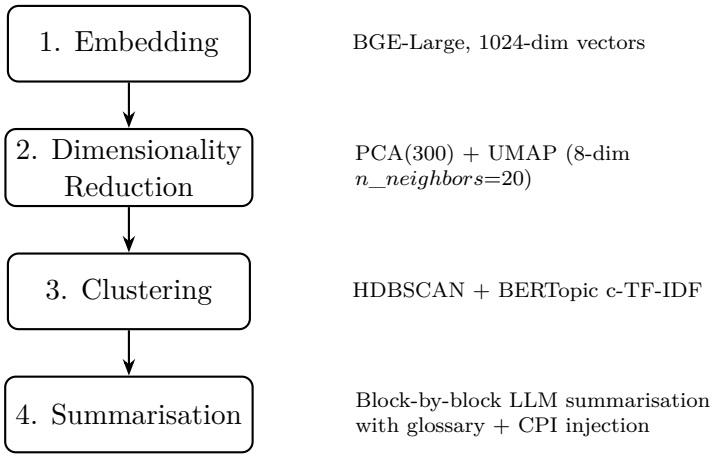


Figure 3.4: Fault pattern discovery pipeline. Each stage feeds into the next, producing structured fault pattern summaries from raw enriched records.

3.2.1 Embedding

The first step translates each issue's text into a dense vector that captures its semantic content. For each enriched record, the input string is formed by concatenating the cleaned title and the LLM-normalised description (as produced by the enrichment pipeline in Section 3.1):

```
text = title + "\n" + description
```

This concatenated string is then encoded using **BGE-Large** (`BAAI/bge-large-en-v1.5`) [10], a bi-encoder model that produces 1024-dimensional dense vectors. BGE-Large was selected for its strong performance on semantic textual similarity benchmarks and its balance between embedding quality and inference speed. The model is accessed through the FlagEmbedding library, wrapped to match the SentenceTransformer interface expected by BERTopic. All embeddings are cached as NumPy arrays to avoid redundant computation during the hyperparameter grid search.

Before proceeding to clustering, the embedding space was characterised using three diagnostics:

1. **PCA explained variance**: A PCA decomposition was fitted to determine the intrinsic dimensionality of the embedding space. The cumulative variance curve indicates how many dimensions are needed to capture 90% and 95% of the total variance, providing a lower bound on the UMAP target dimensionality.
2. **Hopkins statistic**: This measures clustering tendency by comparing nearest-neighbour distances in the actual data against distances from uniformly random points. A value above 0.7 indicates moderate clustering tendency; above 0.85 indicates strong natural clusters. This confirms whether density-based clustering is appropriate for the data.
3. **k-NN distance distributions**: The distribution of distances to the k-th nearest neighbour (for $k \in \{5, 10, 15, 20, 30, 50\}$) reveals whether the data has uniform or highly variable local density. A high spread ratio (Q75/Q25 > 3) indicates that HDBSCAN, which adapts to varying density, is preferable over fixed-radius methods like k-means.

These diagnostics confirmed that the bug report embeddings exhibit strong clustering tendency and non-uniform density, validating the choice of UMAP followed by HDBSCAN.

3.2.2 Dimensionality Reduction

Density-based clustering degrades in high-dimensional spaces because distances become increasingly uniform as dimensionality grows (the "curse of dimensionality"). The 1024-dimensional BGE-Large embeddings must therefore be projected into a lower-dimensional manifold before clustering.

Three reduction strategies were evaluated:

- **PCA alone**: A linear projection that preserves global variance but cannot capture non-linear manifold structure.
- **UMAP alone**: A non-linear projection [32] that preserves both local neighbourhood relationships and approximate global structure, using a cosine distance metric.
- **PCA $\rightarrow$ UMAP**: A two-stage approach where PCA first reduces to certain dimensions, then UMAP projects to the final target dimensionality. The linear pre-compression discards low-variance dimensions and speeds up UMAP's nearest-neighbour computation.

The hyperparameters swept for each strategy were:

- **PCA alone**: target dimensions $\in \{6, 8, 10, 15\}$.
- **UMAP alone**: `n_components` $\in \{8, 10, 15, 20\}$, `n_neighbors` $\in \{15, 20\}$, `min_dist` $\in \{0.0, 0.05, 0.1, 0.15\}$, cosine distance.
- **PCA $\rightarrow$ UMAP**: PCA $\in \{200, 300\}$; then UMAP with the same ranges as above.

All UMAP configurations used a fixed random seed for reproducibility.

3.2.3 Clustering

The reduced embeddings were clustered using HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) [9]. HDBSCAN was chosen for two properties that make it well-suited to bug report data:

1. It does not require a pre-specified number of clusters. The number of fault patterns in a bug corpus is unknown a priori and varies with the product's history.
2. It explicitly labels low-density points as noise (cluster -1), preventing unrelated or ambiguous issues from contaminating coherent fault patterns.

The clustering was integrated into a BERTopic [18] pipeline, which adds a topic representation step on top of the clustering. After HDBSCAN assigns cluster labels, BERTopic applies a class-based TF-IDF (c-TF-IDF) procedure to extract the most representative keywords for each cluster. Unlike standard TF-IDF which operates on individual documents, c-TF-IDF treats all documents within a cluster as a single concatenated document, then computes term importance relative to all other clusters. This surfaces words that are frequent within a cluster but rare across the corpus, effectively identifying the distinguishing vocabulary of each fault pattern. The tokenisation step was configured with:

- N-gram range $(1, 3)$ to capture multi-word telecom terms (e.g., "software upgrade failure", "NPU protection switchover").
- A custom token pattern `r"(?u)\b[\w+/.:]+[\w]+\b"` that preserves domain-specific tokens containing dots, colons, and slashes (e.g., protocol names, file paths, error codes).
- English stop-word removal, with minimum document frequency of 5 and maximum of 0.95.

Hyperparameter optimisation

A grid search was conducted over the joint dimensionality reduction and clustering parameter space. For HDBSCAN, the swept parameters were:

- `min_cluster_size` $\in \{5, 8, 10\}$ — the minimum number of issues required to form a cluster. Smaller values produce more fine-grained patterns but risk creating clusters around noise.
- `cluster_selection_epsilon` $\in \{0, 0.3\}$ — merges clusters that are separated by less than this distance, preventing over-fragmentation of closely related fault types.

Each configuration was evaluated on three criteria:

1. **NPMI coherence** (Eq. 2.9): Measures whether the top-10 keywords of each topic co-occur in the corpus more than expected by chance. Higher NPMI indicates that the cluster's keyword representation is internally consistent.
2. **Topic diversity**: The ratio of unique words to total words across all topics' top-10 keyword lists. A value near 1.0 means topics are distinct; a low value means many topics share the same keywords (redundancy).

3. **LLM-as-judge**: Claude Sonnet (via Amazon Bedrock) rated each topic on two dimensions — coherence and interpretability — using a 1–5 scale. The judge received the top-10 keywords and 5 randomly sampled documents per topic, then assessed whether the keywords match the documents and whether the topic can be given a clear, actionable name. Topics were evaluated in batches of 20 to stay within context limits.

3.2.4 Cluster Summarisation

The final stage transforms each cluster from a set of raw issue records into a structured, actionable fault pattern summary. Pre-computing these summaries avoids the need to retrieve and reason over all individual reports within a cluster at query time, allowing the agent to reference a single concise pattern description instead.

Block-by-block strategy

Clusters vary in size from 5 to over 50 issues. Since feeding all issues from a large cluster into a single LLM call would exceed the context window, a Map-Reduce strategy is used:

1. **Small clusters** (≤ 15 issues): Summarised in a single LLM call.
2. **Large clusters** (> 15 issues): Split into blocks of 15 TRs. Each block is summarised independently (the "map" step), producing a sub-summary. All sub-summaries are then merged into one final summary (the "reduce" step).

Before summarisation, the issues within large clusters are shuffled (with a fixed seed per cluster) to avoid ordering bias.

Prompt design

The summarisation prompt instructs Claude Sonnet to act as a senior NIV architect and produce a structured JSON output containing:

- **Pattern name**: A concise label for the fault category.
- **Core mechanism**: A 2–3 sentence explanation of the underlying root cause.
- **Symptom signatures**: Recurring alarms, log keywords, and KPI drops.
- **Trigger conditions**: Specific configurations or action sequences that provoke the issue.
- **Exploratory test heuristics**: Concrete test steps specifying which interface to use (CLI/GUI/NETCONF), what to do, and what to observe.
- **Outlier TRs**: Issues that do not fit the main pattern, explicitly excluded to avoid contaminating the heuristics.

Two forms of context enrichment are injected into the prompt:

1. **Domain glossary**: The pipeline scans all text fields in the cluster for acronyms matching the curated glossary (Section 3.1). Matched terms and their definitions are prepended to the prompt, preventing the LLM from hallucinating incorrect expansions.

2. **CPI context**: The cluster's topic name and keywords are used to query the official product documentation (via the same CPI search tool available to the agent). Up to 3 relevant documentation excerpts are included, giving the LLM access to official specifications when formulating root cause hypotheses.

The merge prompt instructs the LLM to deduplicate findings across sub-summaries, resolve contradictions by majority view, and produce a single consolidated pattern. The final summaries are stored as a JSON file that serves as the macro-level knowledge layer in the retrieval system (Section 3.3).

3.3 Retrieval System Design

The retrieval system bridges the historical knowledge base and the language model. Its job is to find the right bug reports or fault patterns for a given query, so the LLM can ground its answers in real evidence rather than generating from parametric memory alone. Because queries in exploratory testing range from vague ("what goes wrong after upgrades?") to precise ("show me MLP-6774"), the system must support both semantic similarity and exact lexical matching.

3.3.1 Document Indexing Strategy

All documents are stored in a single ChromaDB collection, partitioned by a `type` metadata field into two categories:

Issue documents

Each enriched bug report is converted to a text string by concatenating its title and full (LLM-normalised) description. This string is then split into chunks of 1 500 characters with 200-character overlap using a recursive character text splitter. Chunking is necessary because some reports exceed 5 000 characters after enrichment, and embedding models produce better representations for shorter, focused passages.

The chunk size of 1 500 characters was selected based on the length distribution of the enriched issue corpus. The median issue length is 2 061 characters (mean 2 355), with 32.6% of issues fitting within a single chunk and 42.2% requiring exactly two chunks. This configuration ensures that most issues are represented by 1–2 semantically focused chunks rather than being fragmented into many small pieces that lose context. The 200-character overlap prevents sentence-level information from being split across chunk boundaries, which is particularly important for bug reports where root cause descriptions often span paragraph breaks. A larger chunk size would reduce splitting but dilute the embedding's semantic specificity; a smaller size would improve specificity but fragment the causal narratives that make bug reports useful for test guidance.

Each chunk carries metadata that enables downstream filtering and display:

- `key` — the Jira ticket ID (e.g., `MLP-6774`)
- `title`, `severity`, `updated`, `closed_as`

- `hw` — hardware identifiers (JSON-serialised)
- `features` — enabled configuration features (JSON-serialised)
- `answer` — the root cause or resolution text, when available

Pattern documents

The cluster summaries from Section 3.2 are indexed differently. Rather than storing each summary as a single monolithic document, the system splits each summary into its individual sections: core mechanism, symptom signatures (alarms, log keywords, traffic impact), trigger conditions, and exploratory test heuristics. Each section becomes a separate document, prefixed with the pattern name for context.
When a tester asks about "alarms after cold restart", the retriever should match the symptom signatures section of the relevant pattern, not a 2 000-character summary where the alarm names are buried in the middle. Sectioning produces shorter, more focused documents that align better with specific query intents.

BM25 index

In parallel with the vector store, a BM25 sparse index is built over the issue documents and serialised to disk as a pickle file. This index enables exact lexical matching for ticket IDs, hexadecimal error codes, and domain-specific identifiers that dense embeddings cannot reliably distinguish (e.g., `MLP-88436` vs. `MLP-88437`).

3.3.2 Hybrid Retrieval with Cross-Encoder Reranking

The retrieval pipeline operates in two stages: a high-recall first stage that casts a wide net, followed by a precision-focused reranking stage that selects the most relevant results for the agent's context window.

First-stage retrieval

Three retrieval strategies are available, selected by the agent based on query intent (Section 3.4.2):

1. **Hybrid search** (for general issue queries): Both the dense retriever and the BM25 retriever return 15 candidates each. These two lists are merged using Reciprocal Rank Fusion (RRF, Eq. 2.7 in Chapter 2) with $k = 60$, which deduplicates and ranks the combined set by fusing rank positions. The top 15 documents from the fused list are retained for reranking. RRF avoids the need to normalise scores across different retrieval methods, since BM25 scores and cosine similarities are on incomparable scales.
2. **Dense-only search** (for pattern queries): Queries the vector store filtered to `type=pattern` documents. Semantic similarity is sufficient here because pattern documents are well-written summaries with consistent vocabulary.
3. **Exact lookup** (for ticket IDs and error codes): First attempts a direct metadata match on the `key` field. If no exact match is found, falls back to BM25 for partial matches.

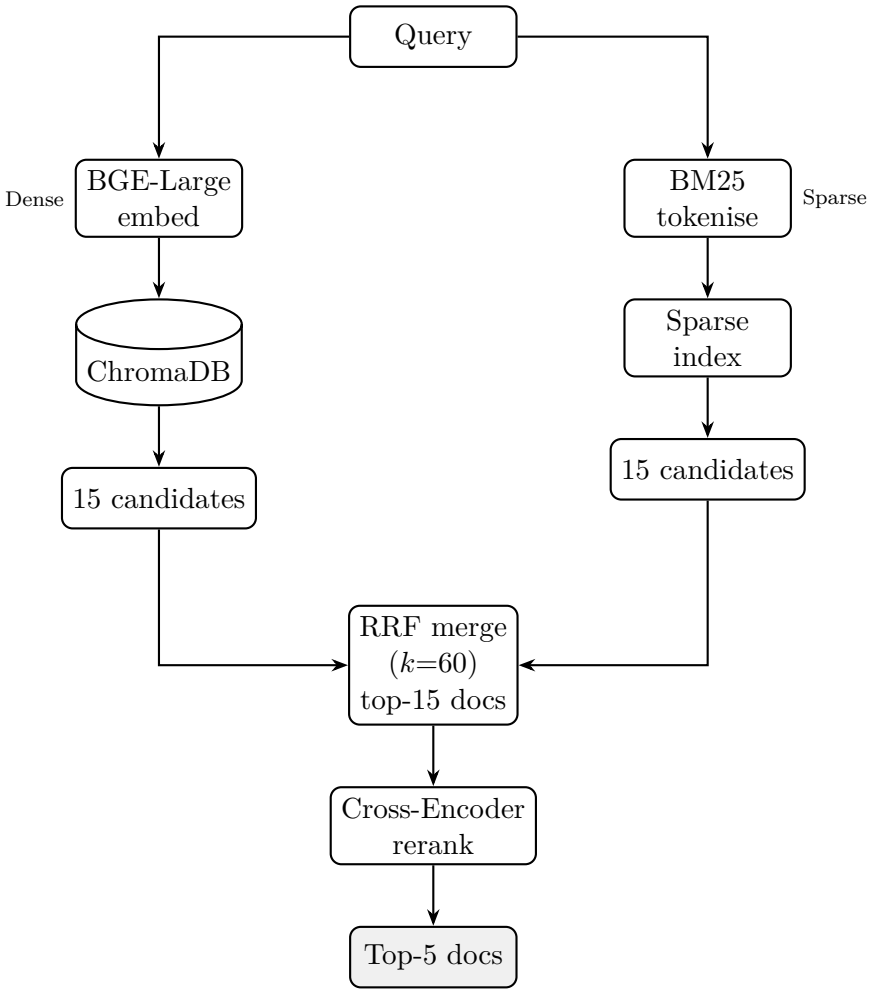


Figure 3.5: Hybrid retrieval pipeline. Dense and sparse retrievers each produce 15 candidates, fused via RRF and reranked by a cross-encoder to yield the final top-5 documents.

To ensure high recall before reranking, each retriever in the hybrid strategy returns 15 candidates, and RRF selects the top 15 from the fused list. For pattern queries, the dense retriever returns 9 candidates directly.

Second-stage reranking

The first-stage retrievers optimise for recall, so a second stage applies a cross-encoder (Section 2.3.1) to re-score each candidate. The cross-encoder takes each query-document pair as joint input and outputs a single relevance score, which is used to reorder the candidate list. The model used is `ms-marco-MiniLM-L-6-v2`, chosen for its balance between accuracy and latency. It is trained on the MS MARCO passage ranking dataset, while its compact size (6 layers, 22M parameters) keeps inference fast enough for interactive use. Since the cross-encoder only processes 15 candidates per query rather than the full corpus, the added latency remains acceptable for a conversational system.

After reranking, the system returns the top-5 documents for issue queries and top-3 for pattern queries. These are formatted with their metadata and injected into the agent's context as tool results.

3.3.3 External Documentation Search

In addition to the local knowledge base, the agent can query Ericsson's official Customer Product Information (CPI) documentation through an external RAG API. This is implemented as a skill (Section 3.4.2) that calls the Ericsson Library Explorer service.

The CPI search:

- Automatically filters to the latest released version of each MINI-LINK product library.
- Uses hybrid retrieval (lexical + semantic, $\alpha = 0.5$) with a dedicated reranker on the server side.
- Returns up to 5 evidence passages with document titles, section headers, and source URLs.

This gives the agent access to official specifications, feature descriptions, and configuration procedures that complement the historical fault knowledge in the local store. While bug reports capture what has gone wrong in the past, CPI provides the ground truth on how the system is designed to work, enabling the agent to answer questions about product capabilities, configuration steps, and expected behaviour.

3.4 Conversational Agent Design

The system follows an agentic RAG architecture in which the language model autonomously decides when and how to retrieve information, rather than following a fixed pipeline.

3.4.1 Agent Architecture

The agent is implemented as a stateful LangGraph state machine powered by Claude Opus 4.6 (via AWS Bedrock) with extended thinking enabled (budget of 2 048 tokens). The execution graph contains three nodes (Figure 3.6):

1. **Agent node:** The LLM receives the conversation history and system prompt, then decides whether to call a tool or produce a final answer.
2. **Tools node:** Executes the selected tool and returns results to the agent node.
3. **Cap node:** Activated when the maximum tool round count (8) is reached, injecting synthetic tool results and forcing a final answer with available information.

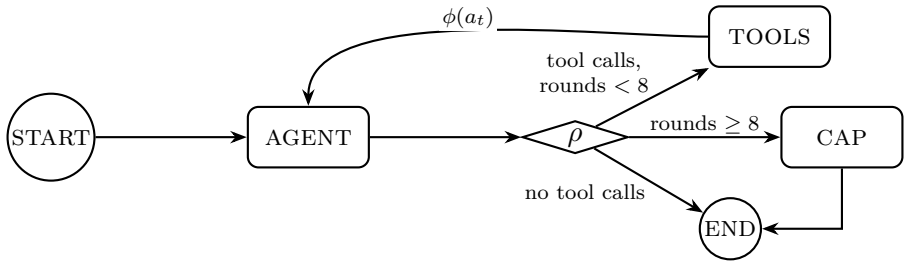


Figure 3.6: ETAgent state graph. The router ρ inspects the last AI message: tool calls with rounds below 8 continue to TOOLS; at the cap limit, synthetic results force a final answer; otherwise the agent terminates.

3.4.2 Tool Integration

The agent selects retrieval strategies via tool calling, allowing the LLM to reason about query intent before choosing a method. Six tools are available (Table 3.1).

Table 3.1: Agent tools and their execution strategies.

Tool	Strategy	Purpose
`search_issues`	Hybrid (Dense+BM25 + RRF+Rerank)	Find bugs combining semantic and exact matches
`search_summaries`	Dense + Rerank	Retrieve macro-level fault patterns
`keyword_search`	Metadata match / BM25	Look up specific ticket IDs or error codes
`load_skill`	Skill loader	Load on-demand capability instructions
`run_skill_script`	Dynamic execution	Execute skill scripts (CPI search, glossary lookup)

Skills are self-contained markdown modules loaded on demand. Two skills are implemented: `cpi-search` (queries Ericsson's CPI documentation via an external RAG

API) and `abbreviation-glossary` (JSON lookup of 692 domain terms to prevent hallucinated expansions).

3.4.3 Conversation History Management

Multi-turn conversations accumulate messages rapidly: each turn adds the user's question, the agent's reasoning, tool calls, and tool results. Without management, the total message history can exceed the LLM's context window within a few turns, causing either API errors or silent loss of early context. To prevent this, the agent applies a three-tier compression scheme that keeps recent context in full while progressively compressing older history (Figure 3.7).
The scheme partitions the conversation history into three zones, each with a character budget:

$$|M_{\text{full}}| \leq B_{\text{full}}, \quad |M_{\text{comp}}| \leq B_{\text{comp}}, \quad |M_{\text{over}}| > T \implies \text{summarise}(M_{\text{over}}), \quad (3.1)$$

with $B_{\text{full}} = 100\,000$ characters, $B_{\text{comp}} = 200\,000$ characters, and overflow threshold $T = 50\,000$ characters. The three zones are:

- **Tier 1 (Full zone, M_{full}):** The most recent turns are kept intact, including tool calls and results, up to B_{full}. This preserves the reasoning traces needed for the current interaction.
- **Tier 2 (Compressed zone, M_{comp}):** Older turns are stripped to human messages and final AI answers only (tool calls and intermediate results are removed), up to B_{comp}.
- **Tier 3 (Summary):** When the content that does not fit in either zone exceeds the threshold T, Claude Haiku generates a dense paragraph summary of the overflow. This summary is persisted in SQLite and injected into the system prompt for all subsequent turns.

The scheme is applied once per new user message. Turns are processed from most recent to oldest: each turn is placed in the full zone if it fits, otherwise compressed and placed in the compressed zone, and any remaining overflow triggers summarisation. The current turn is always kept in full regardless of length. This ensures that the model always sees recent context at full fidelity while retaining the gist of earlier conversation.

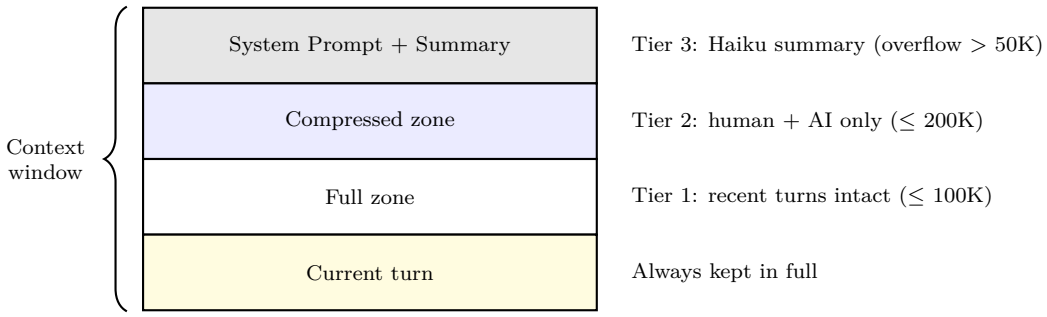


Figure 3.7: Three-tier history management. Recent turns are kept in full, older turns are shortened, and the oldest are summarised by Claude Haiku.

3.4.4 Prompt Design

The system prompt is structured as a modular composition of five sections: role definition, tool routing guidance, validation rules, reasoning strategies, and response formatting. This separation enables independent iteration on each concern without destabilising the others.

- **Role and knowledge framing:** The agent is instructed to combine its general telecommunications knowledge with tool-retrieved information, rather than relying solely on retrieval results. This prevents the failure mode where RAG agents ignore parametric knowledge and produce shallow answers when retrieval returns limited results.
- **Tool routing guidance:** Each available tool is described with its purpose and example use cases. The prompt explicitly maps query types to tools: semantic queries to `search_issues`, exact identifiers to `keyword_search`, macro-level overviews to `search_summaries`. The agent is permitted to call multiple tools in parallel and to answer general questions without retrieval.
- **Validation rules:** A verify-before-respond protocol is enforced for hardware-feature compatibility claims. The agent must call CPI documentation search before asserting any combination claim (hardware-feature, hardware-hardware, or feature-feature compatibility). If CPI provides no evidence, the agent must explicitly state it cannot confirm the claim rather than relying on parametric knowledge. Additionally, a hardware-feature binding rule prevents cross-contamination of metadata: a feature belongs only to the node where it is listed in retrieved results.
- **Outlier reasoning:** When retrieved issues lack a matching fault pattern, the agent applies a structured template: (1) extract trigger conditions, (2) identify boundary variations to explore, (3) suggest observables to monitor (alarms, states, logs, traffic), (4) recommend a mini test sequence, and (5) flag interactions with the tester's current work. This transforms raw retrieval results into actionable exploratory testing guidance.
- **Self-check and response style:** Before responding, the agent executes a self-verification checklist: whether the answer addresses the actual question, whether any claims lack support, whether any ticket IDs or technical specifications are fabricated, and whether the response provides actionable guidance rather than data dumps. The response style is constrained to be conversational and concise, addressing the user's question first and supporting with data second.

3.4.5 Serving and User Interface

The system is deployed as a full-stack web application. The backend (FastAPI) manages user authentication, session persistence (MySQL), and agent orchestration. Communication with the React/TypeScript frontend uses WebSocket for real-time token streaming. The application is containerised with Docker and deployed on Ericsson internal server.

3.5 Evaluation Design

The evaluation is structured across four dimensions, each targeting a different system component. This decomposition ensures that failures can be attributed to specific pipeline stages rather than observed only as end-to-end degradation.

3.5.1 Retrieval Pipeline Comparison

A golden dataset of 180 queries is constructed to evaluate retrieval quality. For each ground-truth document, three queries are generated, one per query type:

- **Symptom:** a natural language description of the fault (e.g., "traffic loss after cold restart on dual-carrier node").
- **Keyword:** an exact identifier like a ticket ID or error code (e.g., "MLP-6774").
- **Diagnostic:** a troubleshooting question that requires understanding the document's content (e.g., "what causes NPU restart during software upgrade?").

We selected 50 issue documents and 10 pattern documents, giving 150 + 30 = 180 queries total (60 per type). Because all three queries for a given document share the same ground truth, we can directly compare how each retrieval method handles different ways of asking about the same fault.
Four retrieval configurations are compared:

1. **BM25 only:** Sparse lexical retrieval.
2. **Dense only:** Semantic vector retrieval using BGE-Large embeddings.
3. **Hybrid RRF:** Dense and BM25 results merged via RRF.
4. **Hybrid RRF + Rerank:** The hybrid set reranked by a cross-encoder.

Metrics are computed per strategy, sliced by query type and document type: MRR, Hit@1, Hit@5, average latency, and RAGAS ContextPrecision and ContextRecall.

3.5.2 Agent Quality Evaluation

Since a general-purpose LLM lacks the domain knowledge to verify telecommunications-specific claims or detect hallucinated technical details, the LLM-as-judge protocol is restricted to domain-independent quality dimensions: tool grounding, response relevance, source citation, and behavioural correctness (e.g., appropriate refusal). Domain correctness is assessed separately through expert evaluation (Section 3.5.3). To mitigate self-enhancement bias, the judge model (Claude Sonnet) differs from the generation model (Claude Opus).
Two complementary datasets are evaluated:

Synthetic golden dataset

A dataset of 51 queries is designed across three difficulty levels: L1 (15 simple, single-tool queries), L2 (25 multi-step queries requiring tool chaining), and L3 (11 complex queries involving ambiguity, edge cases, or refusal). Each query specifies expected tools, expected topics to cover, relevant ticket IDs, and evaluation criteria

(e.g., *actionable*, *cites_sources*, *acknowledges_uncertainty*). Routing accuracy is computed as the proportion of queries where the agent selects the correct tools.

Real-session triplet evaluation

To evaluate the agent on authentic interactions rather than synthetic benchmarks, real user sessions collected during deployment are scored using a triplet-based protocol. Each triplet consists of (user query → agent response → user's next message), where the user's reaction serves as a natural ground-truth signal. The judge scores three dimensions:

- **Tool grounding:** Whether the response references retrieved ticket IDs or pattern names from the tool logs.
- **User reaction:** Whether the user accepted the answer, asked a deepening follow-up, or corrected/rephrased the question.
- **Response relevance:** Whether the agent addressed what the user actually asked.

The judge receives both the conversation triplet and the tool call logs (including retrieved content), enabling verification of whether the agent's claims are grounded in actual retrieval results.

3.5.3 Expert Blind Evaluation

Domain correctness and practical utility are assessed through blind preference evaluation by senior test engineers at Ericsson, who possess the domain knowledge that automated metrics cannot replicate. Two complementary experiments are conducted:

Domain enrichment impact

To isolate the contribution of the fault pattern summaries, 22 representative queries are processed by two agent configurations: one with access to pattern summaries and one without. Both configurations retain access to individual issue retrieval. The resulting response pairs are presented to experts in a blind comparison, where they select the more useful response without knowing which configuration produced it.

Baseline comparison

To validate the system's end-to-end utility against existing alternatives, 60 queries are constructed across four categories: guidance (24 queries covering test planning and configuration), diagnosis (10 queries on troubleshooting and root cause investigation), knowledge (18 queries on feature explanations and specifications), and adversarial (8 queries with wrong premises or hallucination bait). Each query is answered by three systems: the ETAgent, a baseline enterprise AI assistant (EricAI), and raw Claude Opus 4.6 without retrieval augmentation.
Experts evaluate the resulting response sets through blind comparison, selecting the most useful response without knowing which system produced it. This three-way

design measures whether the retrieval augmentation and domain-specific prompt engineering provide value beyond both the enterprise baseline and the raw foundation model.

3.6 Implementation Summary

Table 3.2: Key implementation parameters.

Component	**Configuration**
Reasoning LLM	Claude Opus 4.6 (`us.anthropic.claude-opus-4-6-v1`)
Extended thinking	Enabled, budget 2 048 tokens
Summary LLM	Claude Haiku 4.5
Evaluation judge	Claude Sonnet 4.6
Embedding model	BGE-Large (`BAAI/bge-large-en-v1.5`), 1024-dim
Reranker	`cross-encoder/ms-marco-MiniLM-L-6-v2`
Vector store	ChromaDB (persistent)
Chunk size / overlap	1 500 / 200 characters
Retrieval top-k	5 (issues), 3 (patterns)
Candidate multiplier	3× (15 candidates before reranking)
RRF smoothing k	60
Max tool rounds	8
Full budget	100 000 characters
Compressed budget	200 000 characters
Overflow threshold	50 000 characters
Dimensionality reduction	PCA(300) + UMAP(8, $n\_neighbors$=20, min_dist=0.0)
Clustering	HDBSCAN ($min\_cluster\_size$=5, ϵ=0.3)
Infrastructure	AWS Bedrock, Docker, Kubernetes
Backend	FastAPI + Uvicorn, WebSocket streaming
Database	MySQL (auth/sessions), SQLite (agent memory)
Frontend	React + TypeScript

4
Results

This chapter presents the evaluation results of the proposed ETAgent system. To comprehensively assess its effectiveness as an exploratory testing assistant, the evaluation covers four areas: the quality of the unsupervised fault pattern discovery, the performance of the hybrid retrieval pipeline, the agent's routing accuracy and response quality, and an end-to-end evaluation conducted by domain experts at Ericsson.

4.1 Fault Pattern Discovery Quality

The first step of the evaluation assesses the quality of the macro-level knowledge extracted from historical trouble reports. The clustering pipeline was applied exclusively to the 1 999 issues resolved as *Verified OK*, ensuring that only confirmed, real faults enter the knowledge base. Issues closed as duplicates, invalid, or not-a-bug were excluded to prevent noise from degrading cluster quality. We measured the cluster coherence across different hyperparameter configurations (UMAP and HDBSCAN) using NPMI. Furthermore, an LLM-as-a-judge protocol was employed to rate the interpretability of the generated cluster summaries.

Table 4.1: Clustering evaluation across representative configurations. Coh. and Interp. are mean LLM-as-judge scores (1–5 scale).

Configuration	Topics	Outlier %	NPMI	Coh.	Interp.
K-Means (k=50)	50	0%	−0.009	3.72	3.72
PCA + HDBSCAN	7	92.3%	−0.033	3.71	3.86
UMAP + HDBSCAN	34	17.7%	0.038	4.16	4.11
PCA + UMAP + HDBSCAN (selected)	**34**	**16.5%**	**0.050**	**4.38**	**4.12**

As shown in Table 4.1, K-Means produces negative NPMI scores regardless of configuration, indicating that fixed-partition clustering fails to discover coherent fault patterns in this corpus. PCA-only with HDBSCAN also fails, classifying over 90% of issues as outliers due to insufficient density structure in the linear subspace. UMAP-based configurations achieve positive coherence. The selected configuration (PCA + UMAP + HDBSCAN) achieves the highest NPMI (0.050) with the lowest outlier rate (16.5%), while the variant without PCA pre-reduction scores slightly lower (0.038) at a higher outlier rate (17.7%). Besides, it maintains strong LLM-as-judge scores (mean coherence 4.38, mean interpretability 4.12 on a 1–5 scale, with 29/34 clusters scoring ≥ 4 for coherence).

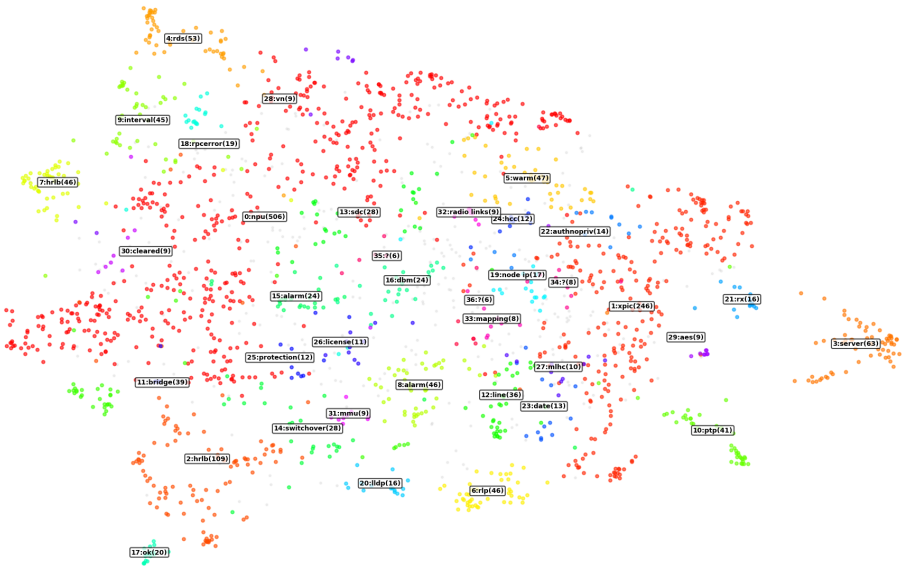


Figure 4.1: UMAP projection of issue embeddings coloured by cluster assignment. Embeddings were reduced via PCA(300) followed by UMAP (*n_components*=8, *n_neighbors*=20, *min_dist*=0.0), then projected to 2D for visualisation.

Figure 4.1 shows the 2D UMAP projection of the clustered issues. The 34 fault patterns form visually distinct groups with minimal overlap, confirming that the embedding and clustering pipeline produces separable topics. Outliers (grey points, 16.5%) are distributed between clusters rather than forming their own dense regions, supporting the HDBSCAN noise classification.

Table 4.2: Ten largest fault pattern clusters discovered by the pipeline.

#	Fault Pattern	Issues
0	NPU Cold Restart & Software Upgrade Failures	276
1	XPIC Radio Link Configuration & Carrier Issues	249
2	hRLB WAN/LAN Port & Network Interface Issues	208
3	Security, Login & Node GUI Usability Issues	131
4	Radio Deep Sleep (RDS) Feature Bugs	63
5	Warm Restart Traffic Impact & TDM Disruptions	56
6	RLP EQP Protection Switching Failures	56
7	hRLB In-Band DCN VLAN & HLAN Configuration Issues	53
8	Alarm Reporting & Configuration Mismatch Issues	49
9	Performance Monitoring (PM) Data & Interval Issues	47

Table 4.2 lists the ten largest clusters, which together account for 1,188 of the 1,669 assigned issues (71%). The cluster names are generated by prompting the LLM with each cluster's top keywords and representative issue titles. The three largest clusters relate to core network equipment functions: NPU restart and upgrade procedures, XPIC radio configuration, and hRLB port management. Smaller clusters capture more specific fault mechanisms such as Radio Deep Sleep behaviour and protection

switching failures.

4.2 Retrieval Pipeline Comparison

Four retrieval configurations were evaluated on the golden dataset of 180 queries (60 symptom, 60 keyword, 60 diagnostic) across two document types (150 issue queries, 30 pattern queries). Table 4.3 reports the overall results. Context precision (CtxPrec) measures the proportion of retrieved documents that are relevant, and context recall (CtxRec) measures whether all needed information is present in the retrieved set [15].

Table 4.3: Overall retrieval performance across 180 queries.
*The full pipeline combines Hybrid + RRF + Rerank.

Strategy	MRR	Hit@1	Hit@5	CtxPrec	CtxRec	Latency
BM25 only	0.728	0.672	0.800	0.828	0.866	32 ms
Dense only	0.564	0.478	0.706	0.701	0.766	106 ms
Hybrid RRF	0.698	0.583	0.856	0.788	0.924	133 ms
Full Pipeline*	**0.815**	**0.761**	**0.883**	**0.869**	**0.951**	**292 ms**

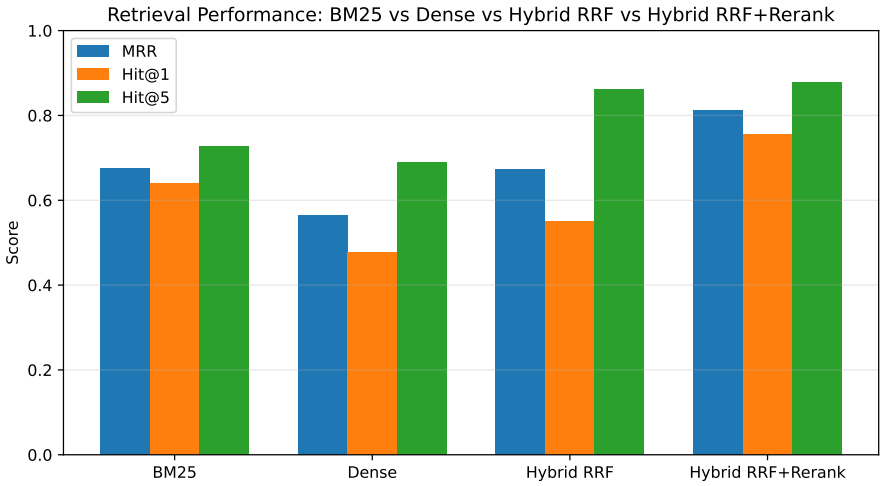


Figure 4.2: Retrieval performance across four pipeline configurations.

The full pipeline (Hybrid + RRF + Rerank) achieves the highest scores across all metrics, with MRR of 0.815 and Hit@5 of 0.883. The cross-encoder reranking stage provides the largest single improvement, lifting MRR from 0.698 (Hybrid RRF) to 0.815 at the cost of increased latency (133 ms to 292 ms).
Table 4.4 breaks down performance by query type. Dense retrieval outperforms BM25 on symptom queries (MRR 0.713 vs 0.701) where semantic similarity matters, but fails on keyword queries (MRR 0.383 vs 0.843) where exact identifier matching is required. The hybrid approach combines the strengths of both, and reranking further improves all categories.

Table 4.4: MRR by query type and retrieval strategy.

Strategy	Symptom	Keyword	Diagnostic
BM25 only	0.701	0.843	0.640
Dense only	0.713	0.383	0.595
Hybrid RRF	0.780	0.632	0.682
Full Pipeline	**0.856**	**0.799**	**0.790**

The results confirm that no single retrieval method is sufficient for the telecommunications domain. BM25 excels at exact identifiers but misses semantically related faults; dense retrieval captures meaning but cannot distinguish similar ticket IDs. The hybrid architecture with reranking addresses both failure modes, justifying the additional latency for the quality improvement.
Table 4.5 shows a further split by document type. All strategies perform substantially better on issue documents than pattern documents. The full pipeline reaches MRR 0.888 on issues but only 0.450 on patterns, suggesting that the pre-computed summaries are harder to match against user queries than the original bug reports. This likely reflects the vocabulary gap: testers describe faults in the language of individual symptoms, while pattern summaries use higher-level abstractions.

Table 4.5: MRR by document type and retrieval strategy.

Strategy	Issue (n=150)	Pattern (n=30)
BM25 only	0.810	0.319
Dense only	0.589	0.436
Hybrid RRF	0.744	0.468
Full Pipeline	**0.888**	**0.450**

4.3 Agent Quality Evaluation

4.3.1 Golden Dataset Evaluation

The 51-query golden dataset was evaluated on two dimensions, routing accuracy and response quality. Routing accuracy measures whether the agent selected the correct tool (or correctly refrained from calling any tool for greetings, meta-questions, and out-of-domain queries). The agent made the correct routing decision in 42 of 51 cases, achieving 82.4% overall accuracy. Table 4.6 reports both routing accuracy and criteria scores per difficulty level.
L1 routing accuracy is 80.0%, with failures on queries where the agent unnecessarily called tools (e.g., answering a general knowledge question via retrieval) or routed configuration questions to the wrong skill. L2 multi-step queries achieved the highest routing accuracy at 92.0%, indicating reliable tool selection even for complex queries. L3 is lowest at 63.6% because adversarial queries designed to provoke refusal instead

Table 4.6: Golden dataset results by difficulty level.

Level	Queries	Routing Acc.	Avg Score	Score ≥ 4
L1, single-tool	15	80.0%	4.41	84%
L2, multi-step	25	92.0%	4.07	73%
L3, complex/adversarial	11	63.6%	4.38	81%
Overall	51	**82.4%**	**4.22**	80%

triggered unnecessary tool calls. Response quality is consistently high across all levels, with an overall average of 4.22 out of 5 and 80% of criteria scores at 4 or above. Topic coverage averages 70.8%, meaning the agent addresses most expected themes but occasionally misses secondary topics.
Of the 51 queries, 12 triggered failure flags. The most significant pattern is adversarial robustness. When given fabricated software versions such as "R99A100", the agent proceeds as if the input is valid rather than questioning its existence, accounting for both hallucination cases. Tool selection errors occur on vague or underspecified queries where the agent calls a lookup tool instead of asking for clarification.

4.3.2 Real-Session Triplet Evaluation

Real user sessions collected during deployment were scored using the triplet-based protocol. A total of 26 sessions comprising 83 interaction turns were evaluated by Claude Sonnet 4.6 as judge. Table 4.7 reports the average scores across the three dimensions.

Table 4.7: Real-session triplet evaluation scores, 1–5 scale, 83 turns.

Dimension	Mean	Interpretation
Response relevance	4.64	Agent addresses the user's actual question
User reaction	3.31	User accepted or deepened in most cases
Tool grounding	3.27	Mixed citation of retrieved evidence

Response relevance is consistently high, with 71 of 83 turns scoring 5, indicating that the agent reliably understands user intent. User reaction scores are concentrated at 3 and 4, covering 68 of 83 turns. A score of 3 indicates the final turn in a session or a neutral follow-up, while 4 indicates a productive deepening question. Only 12 of 83 turns received scores of 1 or 2, meaning the user corrected or rephrased their question.
Tool grounding is the weakest dimension. The agent frequently combines parametric domain knowledge with retrieved evidence rather than strictly citing tool results, which the judge penalises. Table 4.8 shows the distribution of identified failure causes.
The grounding and hallucination failures are partially a consequence of the prompt design, which instructs the agent to combine parametric knowledge with retrieval

Table 4.8: Failure cause frequency across 83 evaluated turns.

Failure Cause	Count
Grounding (claims not tied to retrieved data)	22
Hallucination (fabricated details)	17
Completeness (partial answer)	14
Tool usage (wrong tool or missing call)	4
Coherence	1

results. While this produces more comprehensive answers, it introduces claims that cannot be verified against the tool logs alone.

4.4 Expert Blind Evaluation

4.4.1 Domain Enrichment Impact

Experts compared agent responses with and without access to fault pattern summaries across 22 queries. Two reviewers evaluated all queries, producing 44 preference votes. Table 4.9 reports the preference distribution.

Table 4.9: Expert preference: with vs without pattern summaries (22 queries, 44 votes).

Preferred	Count	%
With summaries	20	45.5%
Without summaries	24	54.5%

The overall preference is roughly split, with a slight edge toward individual report retrieval. Inter-rater agreement is 52.4% (11 of 21 shared queries), indicating high subjectivity in the comparison. When both reviewers agree, they agree on "without summaries" more often (7 queries) than "with summaries" (4 queries).
However, the expert comments reveal a consistent pattern in when each configuration excels. Summaries are preferred for queries involving complex feature interactions and cascade mechanisms, where the pre-computed pattern provides causal chains that individual reports do not surface. For example, on a query about Provider Bridge and XPIC interactions, the reviewer noted that the summary "identifies the critical cascade mechanism — NPU switchover triggers delayed restart, which triggers spurious XPIC recovery, which can mute all carriers" while the version without summaries "treats the bridge mode and XPIC recovery as mostly independent problems."
Conversely, individual retrieval is preferred for queries targeting specific hardware variants or narrow fault types, where summaries add breadth at the cost of precision. On a query about testing hRLB on a new NPU, the reviewer preferred the version without summaries because it "identifies two critical, 100% reproducible NPU1007-specific bugs that are exactly what testing a new NPU should catch first."

4.4.2 Baseline Comparison

Eleven domain experts evaluated 55 of the 60 designed queries answered by three systems in a blind comparison, producing 175 preference votes. Table 4.10 reports the overall preference distribution.

Table 4.10: Expert preference across three systems, 175 votes from 11 reviewers.

System	Preferred	%
ETAgent	126	72.0%
Claude Opus 4.6	26	14.9%
EricAI	23	13.1%

The ETAgent was preferred in 72.0% of comparisons, substantially outperforming both the raw foundation model and the enterprise baseline. Table 4.11 breaks down preferences by query category.

Table 4.11: Expert preference by query category.

System	Guidance (n=81)	Diagnosis (n=28)	Knowledge (n=51)	Adversarial (n=15)
ETAgent	58.0%	89.3%	84.3%	73.3%
Claude Opus 4.6	25.9%	0%	5.9%	13.3%
EricAI	16.0%	10.7%	9.8%	13.3%

The ETAgent's advantage is strongest on diagnosis queries at 89.3% and knowledge queries at 84.3%, where retrieval-augmented answers grounded in historical bug data provide concrete ticket references and fault patterns that neither baseline can offer. The gap narrows on guidance queries to 58.0%, where Claude Opus 4.6 captures 25.9% of preferences by generating plausible general test planning advice from parametric knowledge alone. On adversarial queries designed to provoke hallucination, the ETAgent leads at 73.3% but does not fully dominate, consistent with the hallucination failures identified in the golden dataset evaluation.

5

Discussion

This chapter interprets the evaluation results, discusses the trade-offs inherent in the system design, and identifies threats to validity.

5.1 Answering the Research Question

The research question asks how AI can augment exploratory testing of complex telecommunications equipment by leveraging historical bug data to guide test prioritisation and fault discovery. The evaluation provides converging evidence that the ETAgent achieves this through three mechanisms, each validated by a distinct evaluation dimension.

First, the fault pattern discovery pipeline extracts 34 coherent fault categories from 1 999 verified trouble reports (Section 4.1). In the formal terms of Section 2.1.1, these patterns constitute externalised fault hypotheses $\mathcal{K}_t^{\text{faults}}$ — sub-regions of the test space $\mathcal{Q}$ that have historically yielded faults, now made explicit and queryable rather than residing in individual tester memory. The high coherence scores (mean 4.38/5) confirm that these hypotheses are internally consistent, and the interpretability scores (mean 4.12/5) confirm that testers can act on them.

Second, the hybrid retrieval pipeline makes both individual fault reports and macro-level patterns accessible through natural-language queries, with MRR 0.815 on the full pipeline (Section 4.2). This addresses the access problem identified by Beer and Ramler [4]: the knowledge exists in the organisation's bug tracker, but without effective retrieval it remains practically inaccessible during a testing session.

Third, the agentic architecture autonomously selects retrieval strategies based on query intent, enabling testers to interact with the knowledge base conversationally rather than formulating precise search queries. The agent achieves an overall criteria score of 4.22/5 on the golden dataset (Section 4.3), and domain experts preferred its responses over both baselines in 72.0% of blind comparisons (Section 4.4).

The system does not replace the tester's judgement or execute tests. Its contribution is to reduce the knowledge asymmetry between experienced and junior testers by making historical fault knowledge systematically accessible — effectively lowering the barrier to forming the fault hypotheses that drive effective exploratory testing [20].

5.2 Retrieval Architecture

The retrieval evaluation confirms that no single retrieval method is sufficient for the telecommunications domain. Dense retrieval excels at semantic matching but fails on exact identifiers (MRR 0.383 on keyword queries), while BM25 handles ticket IDs and error codes but misses semantically related faults. The hybrid architecture with cross-encoder reranking achieves the best performance across all query types, with MRR improving from 0.564 for dense-only to 0.815 for the full pipeline.

BM25's overall strength on this corpus (MRR 0.728, outperforming dense retrieval at 0.564) is notable because it contradicts the general trend in the literature, where transformer-based encoders consistently outperform keyword-based methods [41]. The explanation lies in the corpus characteristics. MINI-LINK trouble reports are short (typically 200–400 tokens after normalisation), use consistent internal terminology within a single product family, and are dense with exact identifiers — alarm codes, ticket IDs, hardware part numbers. Within this closed vocabulary, the terminology mismatch that motivates dense retrieval [37] is substantially smaller than in cross-project benchmarks such as GitBugs [40]. Dense retrieval's advantage appears only on symptom queries (MRR 0.713 vs 0.701 for BM25), where paraphrasing and semantic similarity matter.

Reciprocal Rank Fusion proves effective as a fusion strategy, lifting Hit@5 from 0.800 (BM25 alone) and 0.706 (dense alone) to 0.856. This confirms that the two retrieval methods have complementary failure modes: BM25 misses semantically equivalent descriptions while dense retrieval conflates structurally similar identifiers. RRF's rank-based scoring avoids the need to calibrate score distributions between heterogeneous retrievers, making it a practical default for hybrid systems where score scales are not directly comparable.

The cross-encoder reranking stage provides the single largest quality improvement, lifting MRR from 0.698 (Hybrid RRF) to 0.815. This confirms the two-stage retrieve-then-rerank principle documented by Meng et al. [35]: a bi-encoder casts a wide net for recall, and a cross-encoder refines for precision. The latency cost — increasing from 133 ms to 292 ms — is acceptable in this context. Exploratory testing is not a real-time task; testers formulate queries during planning or investigation phases where end-to-end response times are dominated by LLM generation (typically 2–5 s). The 292 ms total retrieval latency is invisible to the user.

One limitation of the current retrieval design is the static candidate multiplier (3×). The system retrieves 15 candidates before reranking down to 5, regardless of query difficulty. For straightforward queries where the top dense result is already correct, this wastes computation. For ambiguous queries where relevant documents are scattered across the embedding space, 15 candidates may be insufficient. An adaptive retrieval strategy — where the agent requests more candidates when initial results score poorly — would address this, but was not implemented in the current version.

A notable gap in retrieval quality exists between document types. The full pipeline achieves MRR 0.888 on individual issue documents but only 0.450 on pattern summaries. This disparity arises because testers naturally describe faults using the same vocabulary found in bug reports (alarm names, ticket IDs, specific symptoms), making issue retrieval a close lexical and semantic match. Pattern summaries, by

contrast, use higher-level abstractions ("cascade mechanism", "combined feature interaction") that are further from how testers phrase queries. This gap is partially mitigated by the agent's ability to call `search_summaries` for pattern-level questions rather than relying solely on dense retrieval, but improving pattern document discoverability remains an area for future work.

5.3 Grounding and Hallucination

The most significant tension in the system design is between comprehensive answers and strict grounding. The prompt instructs the agent to combine its parametric knowledge with retrieved evidence, which produces richer and more contextualised responses than retrieval-only answers. However, this design choice makes it difficult to verify whether every claim is supported by tool results, as reflected in the tool grounding score of 3.27 out of 5 in real-session evaluation.

The golden dataset evaluation reveals two distinct failure patterns. The first is incomplete domain knowledge. When asked about hRLB traffic testing, the agent retrieved relevant bug reports but failed to mention load balancing hashing, a critical concept for designing meaningful test traffic. The retrieved documents did not explicitly discuss hashing, and the agent did not fill this gap from parametric knowledge. This represents a coverage limitation rather than a hallucination.

The second pattern is overconfidence on unsupported claims. When asked to configure BGP on an ML6352, the agent confidently provided configuration commands citing CPI documentation, despite BGP not being supported on that hardware variant. The CPI search returned related networking content, and the agent treated it as confirmation rather than recognising the absence of explicit BGP support. This failure mode is more concerning because the agent presents incorrect information with high confidence and apparent source backing.

Despite the integration of a domain glossary containing 692 terms, the glossary prevents incorrect abbreviation expansions but cannot prevent the model from generating confident explanations on topics where the retrieved evidence is ambiguous or only partially relevant. The validation rules successfully enforce verification for hardware-feature compatibility claims, but do not cover all possible categories of incorrect assertions.

5.4 Domain Enrichment: Pattern Summaries

The ablation results show no dominant preference between the agent with and without fault pattern summaries (45.5% vs 54.5%), with inter-rater agreement near chance at 52.4%. This initially appears to undermine the value of the pattern discovery pipeline. However, the qualitative analysis reveals that the two knowledge layers serve different purposes.

Pattern summaries excel on queries that require understanding how multiple features interact to produce failures. These are queries where individual report retrieval struggles, because the causal chain spans multiple reports and is not visible in any single document. The summaries pre-compute this synthesis, making cascade mech-

anisms and combined failure modes accessible without requiring the agent to reason across dozens of retrieved reports.
Individual retrieval excels on queries about specific hardware, specific bugs, or narrow fault types. Here, the summaries are too broad: they describe general patterns rather than the particular reproducible bug that a tester should prioritise first. The agent without summaries retrieves the exact relevant tickets and presents them directly.
This finding validates the architectural decision to provide both knowledge layers rather than replacing one with the other. The agent's tool selection mechanism enables it to draw on whichever layer is more appropriate for the query at hand.

5.5 Expert Evaluation Insights

The 72.0% overall preference rate masks meaningful variation across query categories. The ETAgent dominates diagnosis and knowledge queries at 89.3% and 84.3% respectively, where retrieval-augmented answers provide concrete ticket references and fault patterns that neither baseline can offer. The advantage narrows on guidance queries to 58.0%, where Claude Opus 4.6 captures 25.9% of preferences by generating plausible general test planning advice from parametric knowledge alone. This pattern suggests that the system's primary value lies in domain-specific knowledge retrieval rather than general reasoning. For queries that require access to historical fault data, the ETAgent is clearly superior. For queries that primarily require general technical reasoning or planning structure, the raw foundation model is sometimes competitive. This distinction has implications for deployment: the system should be positioned as a domain knowledge tool rather than a general-purpose assistant.

5.6 Design Trade-offs

Agentic retrieval vs fixed pipeline

The system uses an agentic RAG architecture where the LLM decides which tools to call, rather than a naive RAG pipeline that retrieves on every query with a fixed strategy. This choice is motivated by the diversity of query types in exploratory testing: a tester might ask for a broad fault pattern overview, a specific ticket lookup, or a CPI manual reference in consecutive turns. A fixed pipeline would either over-retrieve (wasting context window on irrelevant results) or under-retrieve (missing the right source).
The evaluation provides indirect evidence that this design works: tool-usage errors account for only 4 out of 83 evaluated turns (Table 4.8), and the golden dataset criteria scores remain high across all three difficulty levels (Table 4.6). However, the evaluation does not include a direct comparison against a naive RAG baseline with fixed single-retrieval per query. The marginal value of agentic routing over a simpler pipeline therefore remains unquantified. Given that the agent occasionally makes routing errors — for example, calling `search_issues` when `search_summaries`

would yield a more useful overview — a controlled ablation comparing the two architectures would clarify whether the added complexity is justified for all query types or only for multi-step queries.

Knowledge base staleness

The fault pattern summaries are generated from a static snapshot of the bug reports. As new trouble reports are filed, the summaries become outdated and cannot reflect recently discovered faults. Updating requires re-running the full pipeline, which takes several hours. A production deployment would need periodic re-generation or an incremental mechanism that assigns new reports to existing clusters and re-summarises only affected patterns.

Context management across sessions

Multi-turn conversations pose a practical challenge: as the conversation grows, the message history eventually exceeds the model's context window. The system addresses this with a three-tier compression scheme (Section 3.4.3). Recent turns are kept intact (up to 100 000 characters), older turns are stripped to human questions and AI answers only (up to 200 000 characters), and when the compressed zone overflows, a smaller model (Claude Haiku) generates a running summary that is injected into the system prompt.

In practice, the real-session evaluation (26 sessions, 83 turns) suggests that most sessions are short enough that the compression scheme rarely triggers the summarisation tier. The median session length in the deployment was 3–4 turns, well within the full zone budget. This means the summarisation mechanism was not stress-tested during evaluation. For longer sessions — such as a tester systematically working through a feature area over many queries — the quality of the compressed context remains an open question. Degradation in later turns would manifest as the agent forgetting earlier context or repeating itself, but the current evaluation does not measure this.

Fault pattern granularity

The topic modelling pipeline produces 34 fault patterns from 1 999 trouble reports. Whether this is the right granularity depends on the use case. For a tester exploring a specific feature area, 34 patterns is manageable — the agent retrieves the 3 most relevant patterns per query, so the tester never sees all 34 at once. For a test manager seeking a high-level risk overview, 34 may be too fine-grained; a second level of clustering (grouping the 34 patterns into 5–8 macro-categories) could provide a more useful summary.

The 5 clusters that scored below 4 on coherence (Section 4.1) correspond to heterogeneous groupings of lab-specific issues that lack a unifying fault mechanism. These clusters are a known limitation of density-based clustering on noisy data: HDBSCAN correctly identifies them as dense regions in embedding space, but density does not guarantee semantic coherence. In practice, the agent still retrieves indi-

vidual issues from these clusters when relevant; the weak pattern summary simply means the macro-level guidance for those areas is less useful.

5.7 Threats to Validity

Internal validity. The LLM-as-judge evaluation uses Claude Sonnet as judge while the agent uses Claude Opus, mitigating self-enhancement bias. However, both models share the same training lineage, which may introduce subtle preference patterns. The real-session triplet evaluation partially addresses this by using user reactions as ground truth, but the user reaction score of 3 is ambiguous, representing either the final turn in a session or a genuinely neutral response.

External validity. The system was developed and evaluated on a single product family with bug data from one test team. The architecture is domain-agnostic, but the prompt design, glossary, and CPI integration are specific to MINI-LINK. Generalisation to other Ericsson products or other telecommunications vendors would require new domain knowledge injection but not architectural changes.

Construct validity. The expert evaluation measures preference rather than task performance. A preferred answer is not necessarily one that leads to better testing outcomes. A longitudinal study measuring actual fault discovery rates with and without the tool would provide stronger evidence of practical impact.

Evaluation scale. The expert evaluation involved 11 reviewers producing 175 preference votes. While the results are consistent across reviewers, the sample size limits statistical power for detecting small differences between categories. The ablation study is particularly limited, with only 2 full reviewers and inter-rater agreement near chance (52.4%), suggesting that larger-scale evaluation would be needed to draw definitive conclusions about the relative value of pattern summaries.

6
Conclusion

This thesis presented ETAgent, a knowledge-aware conversational assistant for exploratory testing of MINI-LINK microwave network equipment. The system combines unsupervised fault pattern discovery, hybrid retrieval with cross-encoder reranking, and an autonomous LangGraph agent to transform historical bug data into actionable testing guidance. Evaluation across multiple dimensions confirms that the system produces responses that domain experts consistently prefer over both a raw foundation model and an enterprise baseline, with the strongest advantage on queries requiring domain-specific fault knowledge.

6.1 Contributions

The work makes three contributions. First, it demonstrates that an agentic RAG architecture with domain-specific knowledge enrichment can effectively augment exploratory testing, a task traditionally dependent on individual expertise and difficult to automate. Second, it introduces a multi-level evaluation methodology combining automated retrieval metrics, LLM-as-judge scoring on domain-independent criteria, real-session triplet evaluation using user reactions as ground truth, and expert blind preference comparison. Third, it provides empirical evidence that unsupervised fault pattern discovery from historical bug reports produces coherent and actionable macro-level knowledge that improves agent responses.

6.2 Future Work

The current system provides test suggestions but does not execute them. Integrating the agent with a real test bench would enable closed-loop exploratory testing, where the agent observes system responses, adapts its strategy, and verifies whether predicted faults actually manifest. This would shift the system from advisory to autonomous and provide direct measurement of fault discovery rates.

A second direction is incorporating software version information and code change diffs into the retrieval pipeline. By analysing what changed between releases, the agent could prioritise areas of the system most likely to contain regressions, focusing tester attention on components affected by recent modifications rather than treating all historical faults equally.

Third, the current system treats each bug report independently without understanding how nodes are physically connected. Real microwave networks have complex topologies where faults propagate across links and protection groups. Enabling the

agent to reason about network topology, such as which nodes share a protection path or which links form an hRLB group, would allow it to generate test scenarios that target inter-node interactions and cascading failure modes that single-node analysis cannot capture.

Bibliography

[1] Dimo Angelov. Top2vec: Distributed representations of topics. *arXiv preprint arXiv:2008.09470*, 2020.

[2] Anthropic. Introducing the model context protocol. `https://www.anthropic.com/news/model-context-protocol`, 2024. Accessed: 2026.

[3] James Bach. Exploratory testing explained, 2003.

[4] Armin Beer and Rudolf Ramler. The role of experience in software testing practice. In *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, pages 258–265. IEEE, 2008.

[5] Federico Bianchi, Silvia Terragni, and Dirk Hovy. Pre-training is a hot topic: Contextualized document embeddings improve topic coherence. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 2: short papers)*, pages 759–766, 2021.

[6] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] Peng Cai, Reza Ryan, and Nickson M Karie. Llmloganalyzer: A clustering-based log analysis chatbot using large language models. *arXiv preprint arXiv:2510.24031*, 2025.

[9] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer, 2013.

[10] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation, 2025.

[11] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[12] Mingxuan Du, Benfeng Xu, Chiwei Zhu, Shaohan Wang, Pengyu Wang, Xiaorui Wang, and Zhendong Mao. A-rag: Scaling agentic retrieval-augmented generation via hierarchical retrieval interfaces. *arXiv preprint arXiv:2602.03442*, 2026.

[13] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[14] Ericsson. Ericsson manuals and instructions. Accessed: 2026-04-01.

[15] Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. Ragas: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th conference of the european chapter of the association for computational linguistics: system demonstrations*, pages 150–158, 2024.

[16] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, pages 31–53. IEEE, 2023.

[17] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. Large language models for software engineering: Survey and open problems, 2023.

[18] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*, 2022.

[19] Juha Itkonen, Mika V Mantyla, and Casper Lassenius. Defect detection efficiency: Test case based vs. exploratory testing. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 61–70. IEEE, 2007.

[20] Juha Itkonen, Mika V Mäntylä, and Casper Lassenius. The role of the tester's knowledge in exploratory software testing. *IEEE Transactions on Software Engineering*, 39(5):707–724, 2012.

[21] Meenakshi Amulya Jayanti and XY Han. Enhancing model context protocol (mcp) with context-aware server collaboration. *arXiv preprint arXiv:2601.11595*, 2026.

[22] Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents, 2025.

[23] Seungone Kim, Juyoung Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, et al. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2439–2463, 2024.

[24] LangChain AI. LangGraph: Low-level orchestration framework for stateful, long-running agents. `https://docs.langchain.com/oss/python/langgraph/overview`, 2024. Accessed: 2026-04-28.

[25] Jey Han Lau, David Newman, and Timothy Baldwin. Machine reading tea leaves: Automatically evaluating topic coherence and topic model quality. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 530–539, 2014.

[26] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[27] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[28] Chuyuan Li, Austin Xu, Shafiq Joty, and Giuseppe Carenini. Topic-guided reinforcement learning with llms for enhancing multi-document summarization. *arXiv preprint arXiv:2509.09852*, 2025.

[29] Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*, 2024.

[30] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[31] Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey. *arXiv preprint arXiv:2404.11584*, 2024.

[32] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[33] Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, et al. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*, 2025.

[34] Clara Meister and Ryan Cotterell. A systematic review of ablation studies in natural language processing. *Computational Linguistics*, 49(4):1067–1100, 2023.

[35] Qianru Meng, Xiao Zhang, Guus Ramackers, and Visser Joost. Combining retrieval and classification: Balancing efficiency and accuracy in duplicate bug report detection. *arXiv preprint arXiv:2404.14877*, 2024.

[36] Laurent Mombaerts, Terry Ding, Adi Banerjee, Florian Felice, Jonathan Taws, and Tarik Borogovac. Meta knowledge for retrieval augmented large language models. *arXiv preprint arXiv:2408.09017*, 2024.

[37] Usmi Mukherjee and Mohammad Masudur Rahman. Understanding the impact of domain term explanation on duplicate bug report detection. In *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering*, pages 568–579, 2025.

[38] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79, 2012.

[39] Arjun Panickssery, Samuel R. Bowman, and Shi Feng. Llm evaluators recognize and favor their own generations. *arXiv preprint arXiv:2404.13076*, 2024.

[40] Avinash Patil. Gitbugs: Bug reports for duplicate detection, retrieval augmented generation, triage, and more. *arXiv e-prints*, pages arXiv–2504, 2025.

[41] Avinash Patil, Kihwan Han, and Aryan Jadon. A comparative study of text embedding models for semantic text similarity in bug reports. *arXiv preprint arXiv:2308.09193*, 2023.

[42] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*, 2023.

[43] Alireza Salemi and Hamed Zamani. Evaluating retrieval quality in retrieval-augmented generation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2395–2400, 2024.

[44] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*, 2024.

[45] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

[46] Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*, 2025.

[47] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 253–262. IEEE, 2011.

[48] James A Whittaker. *Exploratory software testing: tips, tricks, tours, and techniques to guide test design*. Pearson Education, 2009.

[49] Xinli Yang, David Lo, Xin Xia, Lingfeng Bao, and Jianling Sun. Combining word embedding with information retrieval to recommend similar bug reports. In *2016 IEEE 27Th international symposium on software reliability engineering (ISSRE)*, pages 127–137. IEEE, 2016.

[50] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[51] Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, et al. Justice or prejudice? quantifying biases in llm-as-a-judge. In *International Conference on Learning Representations*, volume 2025, pages 102351–102390, 2025.

[52] Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. Survey on evaluation of LLM-based agents. *arXiv preprint arXiv:2503.16416*, 2025.

[53] Chaojia Yu, Zihan Cheng, Hanwen Cui, Yishuo Gao, Zexu Luo, Yijin Wang, Hangbin Zheng, and Yong Zhao. A survey on agent workflow–status and future. In *2025 8th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 770–781. IEEE, 2025.

[54] Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, and Zhaofeng Liu. Evaluation of retrieval-augmented generation: A survey. In *CCF Conference on Big Data*, pages 102–120. Springer, 2024.

[55] Pavlos Zakkas, Suzan Verberne, and Jakub Zavrel. Sumblogger: Abstractive summarization of large collections of scientific articles. In *European Conference on Information Retrieval*, pages 371–386. Springer, 2024.

[56] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

[57] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems*, 36, 2023.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY