



BA
Documentation

Design and Evaluation of AI Agent Workflows Using Microsoft Foundry and Dify

Semester: Spring 2026

Authors Mario Huber
Jovan Rakic
Edo Husakovic

Project Advisor: Dr. Marco Lehmann
Proofreader: Prof. Stefan Keller
External Expert: Kirusanth Schenkenberg van Mierop
Department: Computer Science

Version: 1.0
Date: 2026-06-11

Abstract

This thesis evaluates how retrieval-augmented enterprise assistants can be implemented on existing Artificial Intelligence (AI) application platforms in the context of managed IT services. The work builds on several previous project phases in which a custom Retrieval-Augmented Generation (RAG)-based platform was designed and extended. For this thesis, the focus shifted from further custom development to the evaluation of two alternative platform approaches: Dify as a self-hosted, source-available platform and Microsoft Foundry as a managed Azure-based platform.

The comparison is grounded in practical implementations rather than feature lists alone. In the context of managed IT services, two representative use cases were implemented using a dataset of approximately 5000 historical support tickets: a technician assistant for conversational access to historical ticket knowledge and an automated ticket enrichment workflow that adds contextual information to newly created tickets. In addition, a smaller Text-to-SQL proof of concept was carried out to examine the feasibility of Large Language Model (LLM)-based access to structured enterprise data. To assess retrieval-augmented quality in a systematic way, the thesis defines an evaluation framework covering retrieval quality, generation quality, and selected system-level characteristics such as latency and cost.

The results show that both Dify and Microsoft Foundry are technically suitable for building retrieval-based enterprise AI workflows, but they differ substantially in their trade-offs. Microsoft Foundry provided the more integrated path for Azure-centric environments, particularly through Azure AI Search, Azure-native identity and access control, and managed service integration. Dify demonstrated that similar workflows can be realized in a self-hosted and platform-independent manner, but required more custom effort in areas such as synchronization, access control, and platform extension. The Text-to-SQL Proof of Concept (PoC) further suggests that LLM-based querying of structured enterprise data is promising, but that the evidence remains limited and practical production viability is not yet established. Overall, the thesis contributes a practice-oriented basis for deciding when a self-hosted platform, a managed cloud platform, or continued custom development is the more appropriate choice for enterprise AI assistants.

Management summary

Objectives and Scope

Companies in managed IT services collect large amounts of operational knowledge over time. Support tickets, troubleshooting notes, customer incidents, and resolution histories contain valuable information for future support cases. In practice, this knowledge is often difficult to reuse. Technicians must manually search old tickets, remember similar incidents, or ask colleagues who have handled comparable problems before.

This thesis investigates how AI assistants can make this existing knowledge easier to access. The focus is on retrieval-augmented generation (RAG), a method where an AI system first searches relevant company data and then generates an answer based on the retrieved information. Instead of relying only on general model knowledge, the assistant can use historical support tickets as evidence.

The work builds on previous project phases in which a custom RAG platform was developed. In this thesis, the direction changed: instead of extending the custom solution further, two existing AI workflow platforms were evaluated as alternatives:

- **Dify**, a self-hosted and platform-independent AI workflow platform
- **Microsoft Foundry**, a managed AI platform integrated into the Azure ecosystem

The goal was not to identify one universal winner. The main objective was to understand under which conditions each platform approach is suitable:

- When does a managed cloud platform make sense?
- When is a self-hosted platform preferable?
- When is continued custom development still justified?
- What implementation effort is required for realistic enterprise workflows?
- How can the quality of RAG-based assistants be evaluated systematically?

The thesis focuses on technical feasibility, architecture, platform trade-offs, and evaluation methods. It does not include a long-term field study with technicians in productive daily operations.

Methodology and Approach

The comparison was based on practical implementation rather than feature lists alone. Two representative use cases from a Managed Service Provider (MSP) environment were implemented using approximately 5 000 historical support tickets.

The first use case was a **Technician Assistant**. This assistant allows technicians to ask questions in natural language, for example about recurring incidents, error messages, or previous solutions. The system searches historical tickets and generates a grounded troubleshooting answer with references to relevant past cases. The basic workflow is shown in Figure 1.

The second use case was **Ticket Enrichment**. In this workflow, a newly created support ticket is automatically analyzed. The system retrieves similar historical tickets and writes an internal note into the ticketing system. This note can include a short summary, related past incidents, and possible first troubleshooting steps. The goal is to give technicians useful context before they begin working on the issue. Figure 2 illustrates this event-driven workflow.

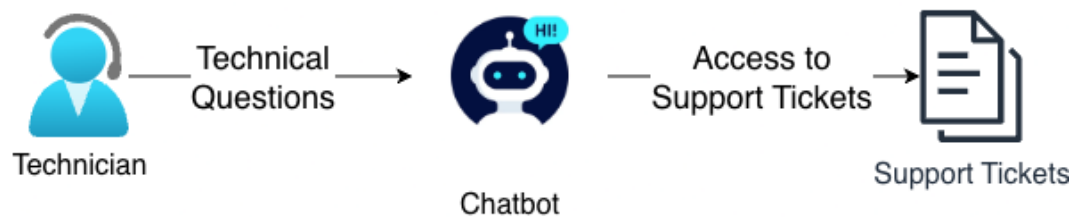


Figure 1: Technician Assistant workflow. Source: own illustration.

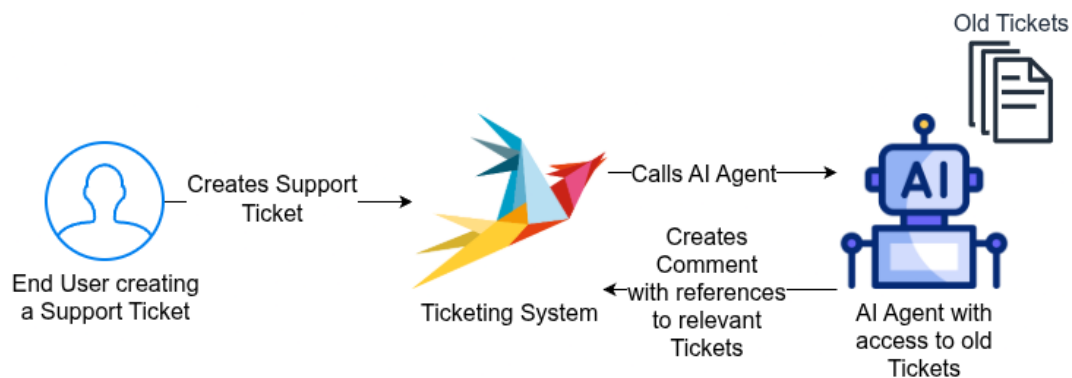


Figure 2: Ticket Enrichment workflow. Source: own illustration.

A smaller **Text-to-SQL PoC** was also implemented. This explored whether large language models can help users query structured enterprise data through natural language. The PoC was treated as a feasibility study, not as a production-ready system.

The platforms were evaluated across several dimensions:

-
- Integration with existing enterprise systems
 - Retrieval quality
 - Generation quality
 - Access control and governance
 - Deployment and hosting model
 - Required implementation effort
 - observability and evaluation support
 - Latency and cost considerations

For systematic assessment, the thesis defined a RAG evaluation framework. This framework separates different failure modes: the system may retrieve the wrong documents, generate a weak answer from good documents, or behave poorly at the system level. The evaluation therefore considered retrieval metrics, generation metrics, LLM-as-a-judge evaluation, latency, cost, and qualitative robustness tests.

Results

Both Dify and Microsoft Foundry proved technically capable of supporting retrieval-based enterprise AI workflows. However, they differ clearly in their strengths, limitations, and operational implications.

Microsoft Foundry was the stronger fit for Azure-centered environments. Its main advantages are managed infrastructure, integration with Azure AI Search, Microsoft Entra ID, Azure-native monitoring, and enterprise access-control patterns. These features reduce the need for custom implementation around identity, governance, synchronization, and operations. For organizations already using Microsoft Azure, Foundry provides a direct and mature path toward production deployment.

The main trade-offs of Microsoft Foundry are vendor dependency, ongoing cloud consumption costs, and the need to build a separate frontend or application layer. Foundry is not an end-user application by itself; it provides the managed backend capabilities for building AI systems.

Dify was effective as a self-hosted and platform-independent alternative. It provides a visual workflow builder, built-in knowledge base functionality, broad model-provider support, and integration with tools such as Langfuse for observability. It is well suited for organizations that require infrastructure control, data sovereignty, or independence from a specific cloud provider.

The main trade-off with Dify is higher implementation and operational responsibility. Synchronization, metadata handling, fine-grained access control, production hardening, and upgrade testing require more custom work. In this thesis, a custom Secure Knowledge Retrieval Plugin was developed to address role-based retrieval filtering. This showed that

Dify can be extended, but also that the responsibility for secure implementation remains with the project team.

Compared with continued custom development, both platforms offered a more mature basis for enterprise AI assistants. Capabilities such as workflow orchestration, retrieval pipelines, prompt management, tracing, and model integration would require significant engineering effort to build and maintain independently. Custom development remains useful only when requirements are very specific or when full control over all system layers is more important than speed and platform maturity.

The implemented use cases showed that the technical platform is only one part of the problem. **Ticket data quality had a direct impact on the usefulness of the AI assistant.** Vague ticket descriptions, incomplete resolution notes, inconsistent terminology, and missing metadata reduced retrieval quality. Better data preparation, structured ticket documentation, and consistent metadata are therefore critical for successful enterprise RAG systems.

The Text-to-SQL proof of concept suggests that LLM-based access to structured enterprise data is promising for targeted factual questions and simple structured queries, but the evidence remains limited. The experiments were intentionally exploratory and do not yet establish practical production viability. At the same time, they confirmed that any production use would require strong safeguards. These include read-only access, permission checks, query validation, result limits, timeout handling, and transparent presentation of generated SQL. Text-to-SQL should therefore be treated as a complementary access pattern, not as a replacement for RAG over documents.

Overall, the thesis provides a practical decision basis:

- **Use Microsoft Foundry** when the organization already relies on Azure and values managed integration, governance, and operational reliability.
- **Use Dify** when self-hosting, model flexibility, cloud independence, or data sovereignty are central requirements.
- **Use custom development** only when neither platform can satisfy the required technical, organizational, or deployment constraints.

Outlook

The next step should be a pilot in a real MSP environment. This thesis evaluated technical feasibility and system behavior, but it did not measure long-term effects on technician productivity. A productive pilot could determine whether the technician assistant reduces search time, improves ticket resolution, supports onboarding, or changes escalation behavior.

Access control should also be implemented and tested in a realistic multi-user environment. This is especially important because support tickets may contain customer-sensitive information. Retrieval must be filtered before the AI model receives context, otherwise sensitive information could appear in generated answers or enrichment notes.

The ingestion process should be extended from a static dataset to continuous synchronization. In productive use, new and updated tickets must be reflected in the knowledge base. The system should support incremental updates, metadata changes, deletion handling, and monitoring of failed ingestion runs.

The evaluation framework should become part of continuous operations. New prompts, retrieval settings, model versions, platform updates, and ingestion changes should be tested against a stable benchmark dataset before release. This would help detect quality regressions early and support controlled improvement over time.

Finally, future work should include a full cost and operations analysis. A production decision should consider not only technical quality, but also licensing, cloud costs, hosting effort, maintenance, monitoring, security reviews, and required engineering capacity. This would complement the technical findings and help organizations choose the most sustainable platform strategy.

Acknowledgements

This thesis was made possible through the support, feedback, and practical assistance of several individuals and partner organizations.

We would like to thank Dr. Marco Lehmann for his valuable feedback throughout the thesis. His guidance helped us refine the direction of the work and improve the quality of the final result. We also thank Kirusanth Schenkenberg van Mierop for his valuable feedback during the interim presentation.

We are grateful to Peter Schmollinger from Avaris/IT AG for the many discussions around the ticket enrichment use case and for providing the ticket data used in this thesis. We also thank the other Avaris/IT AG participants who joined the meetings and contributed their perspectives.

We would also like to thank Simon Joller and the other Raiffeisen participants present in the meetings for providing the Einzelkrediterhebung (EKE) sample queries and for giving us insight into Raiffeisen's RAG architecture.

Finally, we thank Jan Untersander for his support with the Kubernetes infrastructure hosted at Eastern Switzerland University of Applied Sciences (OST) and for helping us with hosting-related questions. His quick and reliable help was essential whenever infrastructure support was needed.

Contents

List of Abbreviations	xx
I introduction	1
1 Introduction	2
1.1 Prior Work and Starting Point	2
2 Objectives	4
2.1 Research Questions and Thesis Goal	4
2.2 Concrete Objectives	5
2.3 Scope and Delimitation	5
II Workflow Platforms	6
3 Platform Selection	7
3.1 Evaluation Criteria	7
3.2 Considered Options	8
3.3 Decision	8
4 Dify	10
4.1 Overview	10
4.1.1 Architecture and Core Features	10
4.1.2 Licensing	13
4.1.3 Deployment Models and User Interaction	13
4.1.4 Governance and Compliance	13
4.2 RAG and Data Ingestion	14
4.2.1 Secure Knowledge Retrieval Plugin	15
4.3 Summary	18
5 Microsoft Foundry	20
5.1 Overview	20
5.1.1 Architecture and Core Features	20

5.1.2	Licensing	22
5.1.3	Deployment Models and User Interaction	22
5.1.4	Governance and Compliance	23
5.2	RAG and Data Ingestion	23
5.3	Summary	26
6	Comparison	27
6.1	Feature Comparison of Evaluated Platforms	27
6.2	Summary	34
III	Use Cases	35
7	MSP Support Automation Use Cases	36
7.1	Introduction	36
8	Technician Assistant	37
8.1	Task Overview	37
8.2	Requirements	37
8.2.1	Functional Requirements	37
8.2.2	Non-Functional Requirements	38
8.2.3	Requirements Coverage	39
8.3	Data Source and Ticket Dataset	40
8.3.1	Support Ticket Structure	40
8.3.2	Ticket Labeling for Evaluation Dataset Creation	40
8.3.3	Data Preparation and Ingestion	41
8.4	Knowledge Base Construction	43
8.4.1	Azure AI Search	43
8.4.2	Dify Knowledge Base	43
8.4.3	Access Control in RAG	44
8.5	System Architecture	46
8.6	Conclusion	55
9	Ticket Enrichment	57
9.1	Task Overview	57
9.2	Requirements	58
9.2.1	Functional Requirements	58
9.2.2	Non-Functional Requirements	59
9.2.3	Requirements Coverage	60
9.3	Data Source and Ticket Dataset	60
9.3.1	Support Ticket Structure	60
9.3.2	Data Preparation and Ingestion	61
9.4	Knowledge Base Construction	61
9.4.1	Access Control in RAG	61

9.5	System Architecture	62
9.5.1	Ticketing System Simulation	62
9.6	Conclusion	67
10	RAG Evaluation	68
10.1	Evaluation Objectives and Scope	68
10.2	What to Evaluate (Evaluation Targets)	68
10.2.1	Retrieval Targets	69
10.2.2	Generation Targets	69
10.2.3	System-Level Targets	70
10.3	Evaluation Metrics	70
10.3.1	Retrieval Metrics	71
10.3.2	Generation Metrics	72
10.3.3	LLM-as-a-judge	74
10.3.4	LLM-as-a-judge Prompts	74
10.4	Selected Evaluation Approach	76
10.5	Evaluation Strategy	77
10.6	Dataset Creation	78
10.6.1	What Makes a Good Evaluation Sample	78
10.6.2	Ticket Selection and Labeling	79
10.6.3	Dataset Creation Workflow	80
10.6.4	Qualitative Robustness Samples	80
10.7	Evaluation Setup	81
10.7.1	Role of Dify and Langfuse	81
10.7.2	Managed Live Evaluators	81
10.7.3	Dataset-Based Evaluators with Ground Truth	82
10.7.4	Custom Metrics	82
10.7.5	RAG Technique Comparison Setup	82
10.8	Results	83
10.8.1	Retrieval and Generation Results	84
10.8.2	Faithfulness and Relevance	85
10.8.3	Qualitative Robustness Results	85
10.9	Conclusion	85
11	Text-to-SQL	87
11.1	EKE Task Overview	87
11.2	Requirements	88
11.2.1	Functional Requirements	88
11.2.2	Non-Functional Requirements	89
11.2.3	Requirements Coverage	90
11.3	Risks and Guardrails	91
11.4	Implementation Context	91
11.5	Methods	92
11.5.1	External Knowledge Integration	92

11.5.2	Feedback-Driven Refinement	93
11.5.3	Schema Linking	93
11.5.4	Voting	93
11.5.5	Column Exploration	94
11.5.6	Reasoning	94
11.5.7	Other Methods	95
11.6	Workflows	95
11.7	Evaluation Setup	96
11.7.1	Error Analysis Method	97
11.8	Results	97
11.8.1	Interpretation	101
11.9	Potential Improvements	101
11.9.1	Interim Summary	102
11.10	Text-to-SQL for Structured Documentation Sources	102
11.10.1	Examples	103
11.10.2	Benefits and Limitations	104
11.11	Conclusion	104
IV	Conclusion	105
12	Conclusion and Outlook	106
12.1	Conclusion	106
12.2	Outlook	108
	Back Matter	111
	Glossary	111
	Bibliography	115
	Appendix	120
A	Infrastructure	120
A.1	Infrastructure	120
A.1.1	Backups	123
A.1.2	Patch Management	123
A.1.3	Dify	123
B	ADR	124
B.1	Workflow Platform	125
B.1.1	Context and Problem Statement	125
B.1.2	Decision Drivers	125

B.1.3	Considered Options	126
B.1.4	Option Comparison (weighted)	127
B.1.5	Decision Outcome	127
B.1.6	Pros and Cons of the Options	128
B.2	Ingestion Bridge, Dify Knowledge Pipeline or Script	131
B.2.1	Context and Problem Statement	131
B.2.2	Decision Drivers	131
B.2.3	Considered Options	131
B.2.4	Decision Outcome	132
B.2.5	Consequences	132
B.2.6	Pros and Cons of the Options	132
B.3	Dify RAG or External Knowledge Base	134
B.3.1	Context and Problem Statement	134
B.3.2	Decision Drivers	134
B.3.3	Considered Options	134
B.3.4	Decision Outcome	134
B.3.5	Consequences	135
B.3.6	Pros and Cons of the Options	135
B.4	Dify Helm Chart - BorisPolonsky/dify-helm	137
B.4.1	Context and Problem Statement	137
B.4.2	Decision Drivers	137
B.4.3	Considered Options	137
B.4.4	Option Comparison (weighted)	138
B.4.5	Decision Outcome	138
B.4.6	Pros and Cons of the Options	138
B.5	S3-Compatible Storage - Garage	140
B.5.1	Context and Problem Statement	140
B.5.2	Decision Drivers	140
B.5.3	Considered Options	140
B.5.4	Decision Outcome	140
B.5.5	Consequences	141
B.5.6	Pros and Cons of the Options	141
C	Evaluation	143
C.1	Golden Dataset Schema for Ticket Enrichment	143
C.1.1	Example Golden Dataset Item	143
C.2	llm-as-a-judge prompts	145
C.3	Labeling Prompt for ML Classification	146
C.4	Technical evaluation workflows	147
C.4.1	Evaluation setup	147
C.4.2	Managed live evaluator	152
C.4.3	Dataset-Based Evaluator with Ground Truth	152
C.4.4	Custom retrieval metrics	153

D	IT Support Use Cases	155
D.1	Prompts	155
D.1.1	Technician Assistant System Prompt	155
D.1.2	Ticket Enrichment System Prompt	165
E	text-to-sql Appendix	170
E.1	alternative queries	170
E.2	text-to-sql Examples	170
E.3	text-to-sql Prompts	174
F	List of Resources	183
F.1	Licenses	183
F.1.1	Dify License	183
F.1.2	n8n License	184

List of Figures

1	Technician Assistant workflow. Source: own illustration.	iii
2	Ticket Enrichment workflow. Source: own illustration.	iii
4.1	Dify’s visual workflow canvas, showing a multi-step AI pipeline composed of connected nodes. Source: own screenshot.	11
4.2	Dify’s document processing interface, showing chunking and embedding configuration during knowledge base ingestion. Source: own screenshot.	12
4.3	Secure Knowledge Retrieval Plugin node configuration options. Source: own screenshot.	16
4.4	Sequence diagram of the Secure Knowledge Retrieval Plugin flow. Source: own illustration.	18
5.1	High-level architecture of Microsoft Foundry, Azure AI Search, and Copilot Studio in relation to external and Microsoft 365 data sources. Source: own illustration.	21
5.2	Managed identity authentication flow for an Azure-hosted web application accessing Azure AI Foundry through Microsoft Entra ID and Role-Based Access Control (RBAC). Source: own illustration	24
8.1	Simple illustration of the technician assistant architecture. The system consists of a conversational interface, a retrieval component that searches the ticket knowledge base, and a generation component that produces responses based on retrieved information. Source: own illustration.	37
8.2	Simplified ingestion bridge architecture that would be implemented in a production system. The ingestion bridge retrieves and normalizes ticket data before forwarding it for ingestion. Source: own illustration.	41
8.3	Simplified and anonymized example of a processed support ticket after ingestion preprocessing. Source: own illustration.	42
8.4	Ingestion workflow for Azure AI Search using Azure Data Lake Storage Gen2 as the document source. Source: own illustration.	43
8.5	Dify knowledge base ingestion workflow using self-hosted MinIO S3 storage as the data source. Files are processed through Dify’s built-in extractor and chunker before being indexed into the knowledge base. Source: own screenshot.	44

8.6	Document-level access control with Azure AI Search and Azure Data Lake Storage Gen2. Permission metadata from ACLs and RBAC assignments is indexed together with document content and used for query-time security trimming. Source: Microsoft Learn [1].	45
8.7	Architecture of the Microsoft Foundry-based technician assistant. The frontend communicates with the Foundry Agent, which uses Azure AI Search for RAG retrieval and Model Context Protocol (MCP) tools for ticket-related operations. Source: own illustration.	47
8.8	Example query about a user unable to sign out of an Azure Virtual Desktop (AVD) session. Source: own screenshot.	47
8.9	First part of the response. The assistant summarizes the issue, asks for clarifying information and provides references to similar historical tickets. The response is grounded in the retrieved ticket content, with mocked URLs for traceability. Source: own screenshot.	48
8.10	Second part of the response. The assistant provides more detailed troubleshooting steps and further actions to be taken by the technician. Source: own screenshot.	49
8.11	Dynamically generated follow-up suggestions returned by the Foundry Agent and displayed as interactive buttons in the frontend. Source: own screenshot.	50
8.12	Recently created or modified tickets displayed as quick-access entries. Clicking a ticket triggers an MCP tool call to fetch ticket details and initialize a troubleshooting conversation. Source: own screenshot.	50
8.13	Dify workflow for the technician assistant. Retrieval, prompt construction, and response generation are structured as separate workflow nodes. Source: own screenshot.	54
8.14	Detailed text feedback feature. Source: own screenshot.	55
8.15	User feedback reflected in Langfuse traces for the corresponding Dify workflow execution. Source: own screenshot.	55
9.1	Simple illustration of the ticket enrichment workflow. A newly created ticket triggers the process, similar historical tickets are retrieved, a contextual summary is generated, and the result is written back to the ticketing system as an internal note. Source: own illustration.	57
9.2	Overall ticket enrichment architecture. A ticketing system event starts the workflow, calls a middle ware endpoint hosted as Azure Function, the ticket context is forwarded to the Foundry Agent, similar historical tickets are retrieved, and the enrichment note is written back to the ticketing system. Source: own illustration.	63
9.3	Example support request created in Zammad by an end user, which triggered the enrichment workflow. Source: own screenshot.	64

9.4	Example enrichment note generated in the ticketing system, containing an issue summary, references to related historical tickets, and suggested first troubleshooting steps. Source: own screenshot.	65
10.1	Visualisation of evaluation targets. Source: adapted from [2].	69
10.2	Evaluation phases overview. Source: own illustration.	77
10.3	Offline evaluation benchmarking overview. Identifiers such as TE-1-naive combine the use case shortname (TE = ticket enrichment) with a descriptive workflow variant name. Source: own illustration.	83
11.1	Simplified EKE schema. Source: [3].	88
11.2	Text-to-SQL linear. Source: own illustration.	95
11.3	Text-to-SQL loop. Source: own illustration.	96
11.4	Text-to-SQL-4 simplified version; the full version can be found in Figure E.1. Source: own illustration.	96
11.5	Text-to-SQL-1 error pie chart. Source: own illustration.	98
11.6	Text-to-SQL-2 error pie chart. Source: own illustration.	98
11.7	Text-to-SQL-3 error pie chart. Source: own illustration.	98
11.8	Text-to-SQL-4 error pie chart. Source: own illustration.	98
A.1	Kubernetes deployment high-level overview and relationships to the services. Source: own illustration	121
A.2	This diagram illustrates the setup for PostgreSQL. The controller is also located on a node. Source: own illustration	122
A.3	Customized Dify deployment architecture. Source: adapted from [4] . . .	123
C.1	Langfuse observation, trace, and session overview source: [5]	148
C.2	Langfuse trace source: [5]	148
C.3	Langfuse overview source: [5]	149
C.4	Langfuse dataset execution matrix source: [6]	150
C.5	Langfuse experiment flow source: [6]	150
C.6	Langfuse online offline loop source: [6]	151
C.7	LLM-as-a-judge live evaluator	152
C.8	Dataset-based llm-as-a-judge evaluation with ground truth	153
C.9	Custom retrieval-metric workflow between evaluation script, Langfuse, and Dify	154
E.1	text-to-sql-4 full version: own illustration	171

List of Tables

6.1	Feature matrix comparing Microsoft Foundry, Dify, and a self-built platform. Legend: ✓ implemented/available, Δ technically feasible but not implemented or only partially implemented, × not supported, □ can be added by using visual workflow builder, * can be added by connecting external system. Source: own illustration.	27
8.1	Functional Requirements — technician assistant	39
8.2	Non-Functional Requirements — technician assistant	40
8.3	Most relevant fields of the exported support ticket dataset. Source: own illustration.	40
9.1	Functional Requirements — ticket enrichment	60
9.2	Non-Functional Requirements — ticket enrichment	60
9.3	Input fields of the current ticket used for the enrichment workflow. Source: own illustration.	61
10.1	Reference-based vs. reference-free evaluation metrics. Source: own illustration.	71
10.2	Evaluation metrics overview. Source: own illustration.	73
10.3	Comparison of evaluation approaches. Source: own illustration.	77
10.4	Label distribution across the 1007 labeled tickets. Source: own illustration.	79
10.5	Automated evaluation modes. Source: own illustration.	81
10.6	Ticket Enrichment dataset run results for 13 benchmark items and 5 runs per variant. Source: own illustration.	84
10.7	TE-2-query-rewriting-1 follow-up evaluation with reference-free LLM judge metrics using gpt-5.4. Source: own illustration.	85
11.1	Functional Requirements — Text-to-SQL	90
11.2	Non-Functional Requirements — Text-to-SQL	90
11.3	Aggregated Text-to-SQL dataset run results for 10 dataset runs each. Source: own illustration.	97

11.4	Aggregated statistics per dataset item for Text-to-SQL-1 baseline . Cells are colour-coded: green = high execution accuracy (>0.8), amber = partial accuracy (0.4–0.8), red = low accuracy or non-zero errors. Source: own illustration.	99
11.5	Aggregated statistics per dataset item for Text-to-SQL-2 annotated schema . Cells are colour-coded: green = high execution accuracy (>0.8), amber = partial accuracy (0.4–0.8), red = low accuracy or non-zero errors. Source: own illustration.	99
11.6	Aggregated statistics per dataset item for Text-to-SQL-3 annotated schema with loop . Cells are colour-coded: green = high execution accuracy (>0.8), amber = partial accuracy (0.4–0.8), red = low accuracy or non-zero errors. Source: own illustration.	100
11.7	Aggregated statistics per dataset item for Text-to-SQL-4 schema linking and loop . Cells are colour-coded: green = high execution accuracy (>0.8), amber = partial accuracy (0.4–0.8), red = low accuracy or non-zero errors. Source: own illustration.	100
B.1	Weighted ADR option comparison	127
B.2	Weighted ADR option comparison	138
C.1	Possible automated evaluation options	149
C.2	Langfuse experiment objects and definitions source: [6]	151

List of Listings

C.1	Example: shortened golden dataset item	143
C.2	Full prompt: llm-as-a-judge generation correctness	145
C.3	Prompt: labeling for ML classification	146
D.1	Full prompt: technician assistant	155
D.2	Full prompt: ticket enrichment	165
E.1	SQL query example: Anzahl Kunden mit Umsatz > 10 Mio. und Limite > 1 Mio.	170
E.2	SQL query example: Entwicklung der ETP-Quote (Vergleich zum letzten Quartal)	170
E.3	SQL query example: Anzahl finanzierter Liegenschaften in Blatten (PLZ 3919)	172
E.4	SQL query example: Vermoegen pro finanzierter Limite (Blatten, PLZ 3919)	172
E.5	SQL query example: Entwicklung Marktwert Wertschriftportfolio (letzte 3 Quartale)	173
E.6	SQL query example: Internal Ratings-Based (IRB) Eigenmittelquote (IRB Total Risk-Weighted Assets (RWA)) pro Rating	173
E.7	SQL query example: Durchschnittliche Finanzierung und Belehnungswert pro PLZ (Privatkunden)	173
E.8	Prompt: Text-to-SQL basic prompt	174
E.9	Prompt: Text-to-SQL judge	174
E.10	Prompt: Text-to-SQL voting	175
E.11	Prompt: Text-to-SQL query exploration	175
E.12	Prompt: Text-to-SQL query construction	175
E.13	Prompt: Text-to-SQL schema linking	176

List of Abbreviations

ACL	Access Control List i
AI	Artificial Intelligence i, ii, iv, v, xiv, xv, 2, 7, 10, 11, 18, 20, 22, 24–26, 29, 31, 43, 45–47, 57, 79, 106, 107, 109, 129
API	Application Programming Interface i, 8, 10, 13, 17, 23, 28, 33, 43, 44, 58, 62, 135, 140, 141
AVD	Azure Virtual Desktop i, xv, 47, 52, 53
DoS	Denial of Service i
EKE	Einzelkrediterhebung i, vii, 87–90, 92, 95, 102, 104
FINMA	Swiss Financial Market Supervisory Authority i, 87
HA	Highly Available i, 140
IR	Information Retrieval i, 70, 73, 77
IRB	Internal Ratings-Based i, xix, 173
JSON	JavaScript Object Notation i, 49, 53
LLM	Large Language Model i, iv, 2, 4, 5, 10, 14, 15, 17, 18, 29, 71, 74, 78, 87, 91, 93–95, 107, 109, 126
LLMOps	Large Language Model Operations i, 10, 18, 33
MCP	Model Context Protocol i, xv, 31, 38, 39, 47, 50, 51, 62, 66, 102, 125, 129, 130
MSP	Managed Service Provider i, iii, v, 4, 5, 36, 37, 40, 44, 47, 51–54, 56–58, 62, 67, 106–108
NLG	Natural Language Generation i, 70, 77
OCR	Optical Character Recognition i, 28

OIDC	OpenID Connect i, 32, 125
OST	Eastern Switzerland University of Applied Sciences i, vii
PoC	Proof of Concept i, iii, 28, 29, 32, 44, 61, 62, 67, 87, 88, 91, 92, 107, 108
RAG	Retrieval-Augmented Generation i, ii, iv, v, vii, xv, 2, 4, 5, 8, 10, 18, 20, 26, 30, 33, 36–39, 43, 47, 68, 78, 82, 84, 91, 92, 106–108, 125, 126, 128, 134, 135
RBAC	Role-Based Access Control i, xiv, xv, 15, 23, 24, 32, 45
REST	Representational State Transfer i, 10, 22, 33
RWA	Risk-Weighted Assets i, xix, 173
SaaS	Software as a Service i, 8, 22, 128, 130
SAML	Security Assertion Markup Language i, 125
SDK	Software Development Kit i, 23
SIEM	Security Information and Event Management i, 23
SNB	Swiss National Bank i, 87
SOTA	State of the Art i, 5
SQL	Structured Query Language i, 25, 87–97, 102–104, 107, 109
SSO	Single Sign-On i, 13, 32, 130

Part I

introduction

Chapter 1

Introduction

Building AI assistants that reliably answer questions from internal enterprise data is no longer a research question — it is an engineering and platform decision. Organisations in managed IT services already sit on years of accumulated ticket histories, resolution notes, and structured operational data. The real challenge is making that knowledge accessible in a way that is accurate, governable, and worth the investment.

This thesis addresses one central question: "which platform approach is best suited for deploying RAG-based assistants in a managed IT services environment, and what are the concrete trade-offs between a self-hosted open platform, a managed cloud platform, and continued custom development?".

To answer that question with evidence rather than feature lists, two representative use cases were implemented end-to-end on both evaluated platforms using a dataset of approximately 5 000 historical support tickets: a *technician assistant* that gives conversational access to past ticket knowledge, and an *automated ticket enrichment* workflow that augments newly created tickets with contextual information. A smaller Text-to-SQL proof of concept was carried out in parallel to examine whether LLM-based access to structured enterprise data is feasible at this stage.

The two platforms evaluated are Dify — a self-hosted, source-available AI workflow platform — and Microsoft Foundry, a managed Azure-based solution. Both are assessed against a systematic evaluation framework covering retrieval quality, generation quality, and selected system-level characteristics including latency, cost, and access control.

By the end of this thesis, a reader — whether a technician choosing tooling or a manager weighing a platform investment — will have a clear, practice-oriented basis for deciding which of these three paths is the more appropriate choice for their context.

1.1 Prior Work and Starting Point

This thesis is the final stage of a multi-phase project. Earlier phases produced a custom RAG-as-a-service infrastructure [7, 8], which was extended further during a subsequent semester thesis [9]. The conclusion drawn from that work was that continuing to extend a custom solution would mean re-implementing capabilities that existing platforms al-

ready provide at greater maturity. This thesis therefore changes direction: rather than adding to the custom stack, it evaluates Dify and Microsoft Foundry as production-viable alternatives.

Chapter 2

Objectives

2.1 Research Questions and Thesis Goal

The introduction established the central question this thesis addresses. This chapter makes the scope and expectations concrete: what is actually being compared, what was built to enable that comparison, and where the boundaries lie.

The thesis pursues one primary and two supporting research questions.

Primary question Under which conditions is a self-hosted open platform, a managed cloud platform, or continued custom development the more appropriate foundation for RAG-based enterprise assistants — and what evidence supports that decision?

Supporting questions

- How do Dify and Microsoft Foundry differ in the trade-offs they introduce for retrieval quality, governance, extensibility, and operational effort?
- How much effort and custom work does each platform require to realise realistic enterprise RAG workflows, and where do the practical limits of each approach lie?
- Which platform characteristics become relevant when implementing concrete MSP workflows, and what does hands-on implementation reveal that feature comparisons do not?
- To what degree can Text-to-SQL serve as a general-purpose pattern for LLM-based access to structured enterprise data — and what guardrails and workflow design does it require to be viable?
- How can retrieval and generation quality in RAG-based systems be measured systematically, and what does a reusable evaluation framework for enterprise assistants look like?

The goal is not a universal ranking. Platforms suit different organisations differently. The aim is a structured, evidence-based basis for making that judgement in practice.

2.2 Concrete Objectives

These 4 objectives are translated into deliverables.

1 — Platform characterisation Describe and compare Dify and Microsoft Foundry along dimensions that matter for enterprise deployment: hosting model, architectural fit, extensibility, observability, data integration, and access control. This goes beyond feature lists; the comparison is grounded in what each platform actually required to realise the use cases.

2 — Use case implementation Implement two representative workflows in a managed IT services context using approximately 5 000 historical support tickets: a *technician assistant* for conversational access to ticket knowledge, and an *automated ticket enrichment* workflow. A Text-to-SQL proof of concept is implemented alongside these to probe the feasibility of LLM-driven access to structured enterprise data.

3 — RAG evaluation framework Define and apply a systematic evaluation framework covering retrieval quality, generation quality, and system-level characteristics including latency and cost. The framework should be reusable beyond this thesis.

4 — Decision criteria Derive practical guidance: under which technical and organisational conditions does each platform approach make sense, and where does custom development remain necessary even when an existing platform is adopted?

2.3 Scope and Delimitation

The thesis evaluates technical feasibility, implementation characteristics, and RAG quality. The following boundaries apply.

- **No live user study.** The use cases are assessed from a technical and architectural perspective. A longitudinal productivity study with technicians in live operations is out of scope.
- **Ticket enrichment on Microsoft Foundry only.** Once the technician assistant confirmed both platforms were viable for retrieval-based MSP workflows, ticket enrichment was implemented only on Microsoft Foundry to avoid redundancy and concentrate effort on deeper evaluation.
- **Text-to-SQL as proof of concept.** The Text-to-SQL component is a feasibility study, not a competitive benchmark against the State of the Art (SOTA).
- **Context-bound results.** Evaluation results reflect this thesis’s datasets, prompts, and configuration choices. They are evidence for the research questions above — not universal platform rankings.

Part II

Workflow Platforms

Chapter 3

Platform Selection

Choosing a platform for enterprise AI workflows is not a purely technical decision. It shapes who controls the infrastructure, where data resides, how quickly new capabilities can be adopted, and what happens when something breaks. Two organisations with identical use cases can reasonably reach opposite conclusions depending on their cloud strategy, compliance requirements, and appetite for operational overhead.

This chapter documents how two platforms were selected for evaluation in this thesis — not to crown a winner, but to place two fundamentally different deployment philosophies under the same test conditions. Dify represents the self-hosted, source-available path: full infrastructure control at the cost of operational responsibility. Microsoft Foundry represents the managed cloud alternative: reduced friction and deep Azure integration at the cost of vendor dependency and reduced transparency. Evaluating both against the same use cases is what makes the comparison meaningful.

3.1 Evaluation Criteria

The open-source candidates were assessed against eight weighted criteria reflecting the requirements of a production-grade, commercially hostable AI application platform:

- **Workflow fit (25%)** — how naturally each platform supports multi-stage agentic pipelines with natural-language routing and tool calling
- **Ecosystem (15%)** — breadth of available connectors, plugins, and community templates
- **Delivery effort (15%)** — estimated implementation overhead of adopting the platform within the project timeline
- **Observability fit (10%)** — native or documented Langfuse integration for tracing and prompt management
- **Extensibility (10%)** — room for adding custom functionality without forking the platform

-
- **Licensing fit (10%)** — compatibility with a self-hosted commercial deployment model
 - **Hosting readiness (10%)** — Kubernetes support, production persistence, and documented operational guidance
 - **Community (5%)** — contributor activity, adoption breadth, and long-term maintenance outlook

3.2 Considered Options

Four options were considered: a self-developed solution built on the foundations of the prior semester thesis [9], n8n, Flowise, and Dify.

Custom solution Continuing the custom platform would offer maximum flexibility and no external dependencies. However, it would also mean re-implementing capabilities — workflow orchestration, RAG pipelines, observability — that established platforms already provide at greater maturity. It was retained for comparison, but not pursued as a primary candidate.

n8n n8n has a broad integration ecosystem and strong community adoption. Two factors ruled it out: n8n’s Sustainable Use License restricts embedding or reselling the platform as a commercial service, which would require a separate paid agreement with the vendor F.1.2, and its support for stage-based agentic workflows is indirect rather than native.

Flowise Flowise is well-suited to agentic workflow patterns, supports Langfuse integration, and offers plugin-based extensibility. Stronger multi-tenancy features are gated behind an enterprise tier.

Dify Dify covers the same ground as Flowise and shares its general strengths. Three factors distinguished it: a native External Knowledge Application Programming Interface (API), a more mature plugin marketplace, and a closer architectural fit for the stage-based RAG and routing model used throughout this thesis.

3.3 Decision

Dify was selected as the self-hosted platform for evaluation. A weighted criterion comparison confirmed this, with Dify reaching the highest score (4.15), ahead of Flowise (3.60), n8n (3.35), and the custom solution (3.30). The decisive factors are documented in the previous section; the one trade-off worth noting here is licensing: shared multi-tenant Software as a Service (SaaS) operation requires either explicit authorisation from LangGenius or per-customer deployments. This is examined further in Chapter 4.

Microsoft Foundry was selected as the managed cloud counterpart — not because it won a scoring comparison, but because it represents a fundamentally different deployment philosophy. Where Dify requires self-hosting and operational ownership, Foundry offers managed infrastructure and deep Azure integration out of the box. Placing both under the same test conditions is what makes the comparison meaningful.

The following two chapters examine each platform in depth.

Chapter 4

Dify

4.1 Overview

Dify is an open-source Large Language Model Operations (LLMOps) platform developed by LangGenius, designed to simplify the development, deployment, and operation of production-ready AI applications. Unlike Microsoft Foundry, which is a managed cloud service tightly integrated into the Azure ecosystem, Dify can be self-hosted on any infrastructure and is not tied to a specific cloud provider.

4.1.1 Architecture and Core Features

Dify provides an integrated set of capabilities covering the full lifecycle of AI application development. At its core is a visual workflow builder that allows developers to construct multi-step AI pipelines by connecting nodes for LLM calls, prompt logic, conditional branching, tool integrations, and data retrieval. This canvas-based approach makes complex agentic workflows tangible and debuggable, as illustrated in Figure 4.1.

A central feature of the platform is its built-in RAG pipeline, which handles the complete ingestion-to-retrieval process. Documents can be uploaded directly into Dify's knowledge base, where the platform manages chunking, embedding, and vector storage automatically, as illustrated in Figure 4.2. Advanced retrieval strategies such as hybrid search and reranking are supported out of the box, making it straightforward to ground AI responses in proprietary organizational data.

Once an agent or workflow is ready, it can be published as a standalone web application or exposed via a Representational State Transfer (REST) API, enabling integration into existing systems and custom frontends.

For observability, Dify can be integrated with multiple tracing providers to log all workflow executions with input/output pairs, token usage, latency metrics, and cost breakdowns. This LLMOps focus enables continuous improvement of prompts, datasets, and model configurations based on production data.

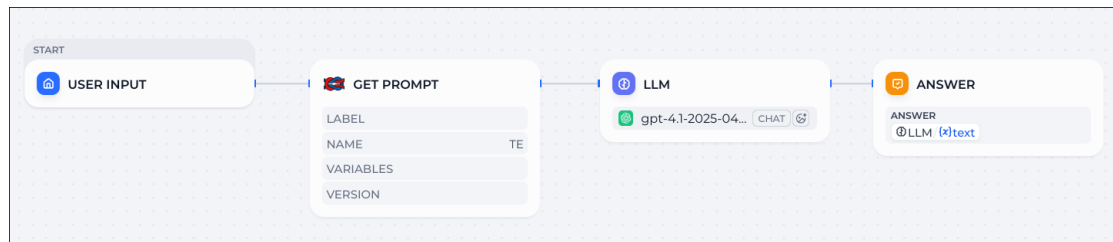


Figure 4.1: Dify’s visual workflow canvas, showing a multi-step AI pipeline composed of connected nodes. Source: own screenshot.

Chunk Settings

General

General text chunking mode, the chunks retrieved and recalled are the same.

Delimiter ⓘ
\\n\\n

Maximum chunk length
1024 characters

Chunk overlap ⓘ
50 characters

Text Pre-processing Rules

☒ Replace consecutive spaces, newlines and tabs

☐ Delete all URLs and email addresses

☐ Summary Auto-Gen

☐ Chunk using Q&A format in English ⓘ

Preview Chunk

Reset

Parent-child

When using the parent-child mode, the child-chunk is used for retrieval and the parent-chunk is used for recall as context.

Index Method

High Quality

RECOMMEND

Calling the embedding model to process documents for more precise retrieval helps LLM generate high-quality answers.

Economical

Using 10 keywords per chunk for retrieval, no tokens are consumed at the expense of reduced retrieval accuracy.

Once finishing embedding in High Quality mode, reverting to Economical mode is not available.

Embedding Model

qwen3-embedding:4b

Retrieval Setting

Learn more about retrieval method, you can change this at any time in the Knowledge settings.

Vector Search

Generate query embeddings and search for the text chunk most similar to its vector representation.

☒ Rerank Model ⓘ
rank_zephyr

Top K ⓘ
3

Score Threshold ⓘ
0.5

Full-Text Search

Index all terms in the document, allowing users to search any term and retrieve relevant text chunk containing those terms.

Hybrid Search

RECOMMEND

Execute full-text search and vector searches simultaneously, re-rank to select the best match for the user's query. Users can choose to set weights or configure to a Rerank model.

Figure 4.2: Dify’s document processing interface, showing chunking and embedding configuration during knowledge base ingestion. Source: own screenshot.

Edo Husakovic, Mario Huber, Jovan Rakic
Design and Evaluation of AI Agent Workflows Using Microsoft Foundry and Dify

12

4.1.2 Licensing

Dify is distributed under a modified version of the Apache License 2.0 documented in Appendix F.1.1. While the source code is publicly available and commercial use is generally permitted, the license includes two notable restrictions. First, operating a multi-tenant environment—where one Dify instance serves multiple isolated workspaces for different customers or organizational units—requires explicit written authorization from LangGenius and typically a commercial enterprise license. In the context of Dify, one tenant corresponds to one workspace, meaning a freely licensed deployment is limited to a single organization. Second, the Dify logo and copyright information displayed in the frontend may not be removed or altered without authorization.

As a result, Dify is more accurately described as *source-available* rather than fully open-source, since the additional restrictions are technically incompatible with the Apache 2.0 terms. For internal enterprise use within a single organization, however, the Community Edition is free to self-host without restrictions.

4.1.3 Deployment Models and User Interaction

Dify supports both a managed cloud offering (Dify Cloud) and fully self-hosted deployment via Docker Compose, making it suitable for organizations with strict data residency or compliance requirements. The Enterprise tier adds features such as Single Sign-On (SSO), advanced role management, and multi-workspace support, the latter being the gated capability under the licensing restrictions described above.

Interaction with deployed applications can occur through Dify’s built-in web UI, which can be published directly as a standalone application, or via its Backend-as-a-Service API, which allows developers to embed Dify-powered agents into custom applications and frontends. This positions Dify as a developer-oriented platform, similar in flexibility to Foundry, but without the reliance on a specific cloud ecosystem.

4.1.4 Governance and Compliance

Dify’s model-agnostic architecture is a core governance feature. Supported providers include OpenAI, Anthropic, Azure OpenAI, self-hosted open-weight models via Ollama or a local OpenAI-compatible endpoint and also aggregators such as OpenRouter. This avoids vendor lock-in and allows organizations to route sensitive workloads exclusively to on-premises models, satisfying data residency requirements.

For air-gapped or privacy-sensitive deployments, Dify can be configured to call only localhost or intranet-bound model endpoints, ensuring that no data leaves the organization’s infrastructure. Combined with self-hosted deployment via Docker Compose, this makes full data sovereignty achievable without modifying the platform itself.

Access control is enforced at the workspace level. The Community Edition supports a single workspace with basic role separation. The Enterprise tier adds SSO integration and fine-grained role management, which are prerequisites for regulated environments requiring audit trails and least-privilege access.

The extensive logs together with Langfuse integration serve as an audit trail for compliance purposes and support continuous monitoring of model behavior in production.

4.2 RAG and Data Ingestion

Dify provides a built-in knowledge base system that manages the complete ingestion-to-retrieval pipeline without requiring external tooling. Beyond simple direct uploads, Dify supports complex ingestion pipelines that can include preprocessing, transformation, and enrichment steps before content reaches the knowledge base. Regardless of ingestion path, Dify handles chunking, embedding, and vector storage automatically. Documents can be enriched with metadata at ingestion time, which is then available as a filter condition at retrieval time—allowing queries to be scoped to specific document subsets by department, classification level, or document type.

Chunking

Dify offers two chunking modes:

- **General mode** — All chunks share the same settings. Configurable parameters include the delimiter (the character at which text is split), maximum chunk length in characters, and chunk overlap to preserve semantic continuity across boundaries. Optionally, a text pre-processing step can normalize whitespace and strip URLs and email addresses before splitting.
- **Parent-child mode** — Text is split into two tiers: smaller child chunks used for query matching, and larger parent chunks returned as the actual retrieval result. Parent chunks can be scoped to either a configurable paragraph-level split or the full document. This balances retrieval precision with sufficient context for the LLM.

Additionally, Dify supports automatic summary generation per chunk, where summaries are separately embedded and indexed. When a summary matches a query, its corresponding chunk is returned alongside it, improving retrievability for dense or technical documents.

Retrieval

Dify supports three retrieval strategies:

- **Semantic search** — dense vector embeddings, retrieves by meaning.
- **Full-text search** — exact keyword matching.
- **Hybrid search** — combines both, with a configurable weighting parameter between the two signals.

An optional reranking step can be applied after initial retrieval, using a dedicated reranker model to re-score candidate chunks before they are passed to the LLM. This is particularly effective in hybrid configurations where the initial result set may contain noise.

External Vector Store

By default, Dify uses its own managed vector store. For organizations that require control over their vector infrastructure, Dify supports external vector databases as a drop-in replacement, including Weaviate, Qdrant, Pinecone, and others. When configured with an external Weaviate instance, Dify writes all embeddings directly into that instance and routes all retrieval queries through it, enabling independent scaling and data sovereignty at the vector layer.

Access Control and the Secure Retrieval Plugin

A limitation of Dify’s native knowledge base is that access control operates at the workspace level rather than the document level. Within a single workspace, all knowledge bases are accessible to all configured agents, with no built-in mechanism for per-user document isolation short of deploying separate workspaces—which requires an Enterprise license. To address this, a custom secure retrieval plugin was developed as part of this thesis. The plugin intercepts retrieval calls and injects user-specific metadata filters, ensuring that each user can only retrieve chunks from documents they are authorized to access, without requiring a multi-tenant Enterprise deployment.

4.2.1 Secure Knowledge Retrieval Plugin

Motivation and Problem Statement

Dify’s built-in Knowledge Retrieval node supports metadata filtering, allowing workflows to restrict retrieved documents based on attributes such as role, department, or access level. However, this filtering is configured directly inside the workflow node in the Dify UI. This means that the filtering rules—including which metadata fields are evaluated and what values are permitted—are defined per workflow, as part of the visual workflow configuration, and are therefore stored and managed within Dify itself.

This approach has a significant practical drawback in a RBAC scenario: every time a new workflow is created or an existing one is duplicated, the metadata filter conditions must be manually reconfigured in the UI. There is no single authoritative place where access rules are defined. The rules are instead scattered across individual workflow configurations, making them difficult to audit, maintain, or update consistently. In a system where multiple workflows serve different use cases but must all enforce the same role-based access policies, this becomes an operational problem rather than a minor inconvenience.

Furthermore, Dify’s role based access in the built-in Knowledge Retrieval node is a feature restricted to the Enterprise tier. Relying on it would therefore either require an

Enterprise license or would limit the implementation to static, hard-coded filter values, neither of which was acceptable for this project.

To address both concerns, a custom Dify plugin was developed that externalizes the access control logic entirely. The filtering rules are defined once in the plugin's codebase, versioned alongside the plugin repository, and applied consistently regardless of which workflow invokes the plugin.

The screenshot displays the configuration window for the 'Knowledge Secure Retrieve' plugin. At the top, there's a title bar with the plugin icon and name, and standard window controls. Below the title bar is a description field labeled 'Add description...'. The main configuration area is divided into two tabs: 'SETTINGS' (active) and 'LAST RUN'. A 'Workspace Default' dropdown is located in the top right of the settings area. The 'INPUT VARIABLES' section contains two fields: 'Query' (with a description 'The search query to run against all knowledge bases.') and 'User Roles' (with a description 'Roles set for the user accessing data'). Both fields have a 'User Input' icon and a variable reference icon. The 'SETTINGS' section includes three configurable options: 'Top K' (set to 5), 'Search Method' (set to 'Semantic Search'), and 'Score Threshold' (set to 'Please enter'). Each setting has a 'Copy' icon, an 'Edit' icon, and a 'Reset' icon. The 'OUTPUT VARIABLES' section lists three output types: 'text' (string), 'files' (array[file]), and 'json' (array[object]), each with a 'Copy' icon and a description of the generated content.

Figure 4.3: Secure Knowledge Retrieval Plugin node configuration options. Source: own screenshot.

Design Approach

Rather than using Dify’s built-in Knowledge Retrieval node, all retrieval within the platform is performed exclusively through this plugin, referred to as the *Secure Knowledge Retrieval Plugin*. No workflow directly queries the Dify knowledge base; the plugin is the sole entry point for retrieval operations.

The plugin accepts a `role` variable as input, which is passed to it from the calling workflow at runtime. This variable represents the role of the authenticated user initiating the conversation. Based on this role, the plugin constructs a metadata filter and issues a retrieval request against the Dify knowledge base API, applying the filter as part of the query. Only document chunks whose metadata satisfies the role-based filter condition are returned. The resulting context is then passed back to the workflow and forwarded to the LLM.

This design deliberately avoids a simpler alternative that was also considered: retrieving documents without a filter and applying role-based filtering to the results after retrieval. That post-retrieval filtering approach is problematic because it interacts badly with the `top_k` parameter. The knowledge base API returns at most `top_k` chunks ranked by semantic similarity before any downstream filtering is applied. If a significant portion of those chunks belong to roles other than the requesting user’s role, the effective number of usable results is reduced unpredictably. In the worst case, all `top_k` results could be filtered out entirely, leaving the LLM with no context even when relevant documents exist. By applying the metadata filter at query time inside the API call, the `top_k` budget is spent exclusively on documents the user is permitted to see, ensuring retrieval quality is not degraded by access control.

Request Flow

The retrieval flow proceeds as follows. A user sends a message through the Dify application interface, which triggers the associated workflow. The workflow passes the user’s query and their `role` variable to the Secure Knowledge Retrieval Plugin via a tool call node. The plugin constructs a retrieval request including the metadata filter derived from the role, and calls the Dify knowledge base API. The API performs the filtered semantic search and returns the matching document chunks. The plugin returns these chunks to the workflow as structured context, which is then injected into the LLM prompt for response generation.

The sequence is illustrated in Figure 4.4.

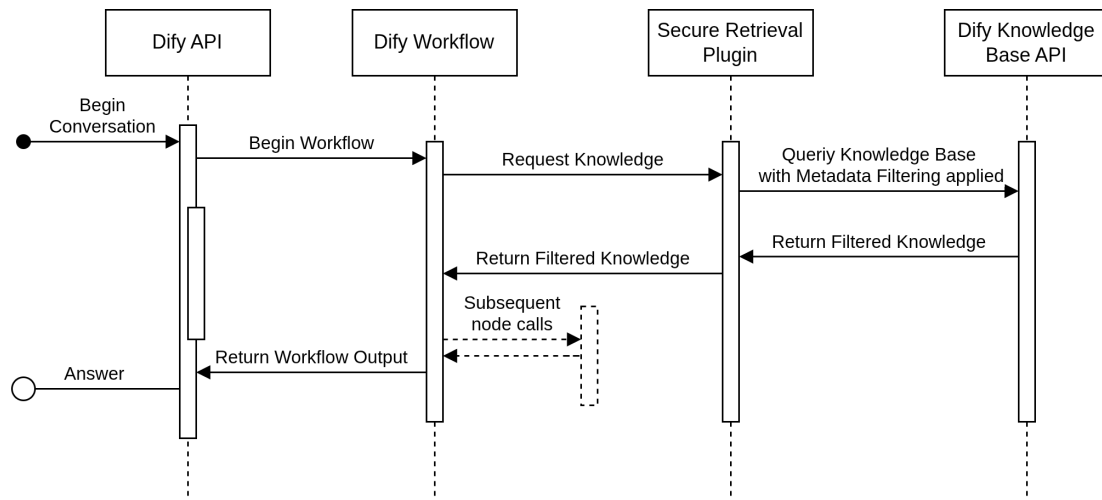


Figure 4.4: Sequence diagram of the Secure Knowledge Retrieval Plugin flow. Source: own illustration.

Centralized Access Control

A key architectural benefit of this approach is that the access control logic is fully centralized in the plugin repository. When access rules need to change—for example, if a new role is introduced or an existing role’s permissions are extended—the change is made in one place and takes effect across all workflows that use the plugin. No workflow-level re-configuration in the Dify UI is required. This separation of concerns also makes the rules auditable: the complete access control policy can be reviewed by inspecting the plugin source code, without needing to open and inspect each individual workflow configuration.

4.3 Summary

Dify is a self-hostable LLMOps platform that covers the full lifecycle of AI application development, from workflow design and RAG pipelines to deployment and observability. Its visual workflow builder, built-in knowledge base, and support for a broad range of LLM providers and vector databases make it well-suited for organisations that require flexibility and data sovereignty without reliance on a managed cloud service.

The platform’s licensing model permits free self-hosted deployment for single organisations, with multi-tenancy gated behind an Enterprise license. Combined with support for local model endpoints and external vector infrastructure, Dify can be deployed in fully air-gapped environments where no data leaves the organisation’s network.

One aspect worth noting is software maturity. Throughout the course of this thesis, Dify exhibited a recurring pattern of minor but noticeable defects: UI inconsistencies, unexpected behaviour in edge cases, and occasional regressions between versions. None of these were blockers — workarounds were generally straightforward — but their frequency suggested a codebase where features are added at a faster pace than stabilisation.

Organisations considering Dify for production use should factor in a tolerance for this kind of friction, and plan for thorough testing when upgrading versions.

Chapter 5

Microsoft Foundry

5.1 Overview

Microsoft Foundry is a managed AI application platform within the Azure ecosystem, designed to support the development and operation of production-grade AI solutions. It combines model hosting, agent orchestration, retrieval systems, and enterprise governance into a single integrated environment. Unlike Dify, which prioritizes self-hosting and infrastructure independence, Foundry is tightly coupled with Azure services and is intended for organizations already operating within the Microsoft cloud stack.

A central design goal of Foundry is to reduce the operational overhead of building AI applications by providing managed abstractions for common patterns such as RAG, tool-enabled agents, and conversational state management. This allows development teams to focus on application logic rather than infrastructure concerns.

5.1.1 Architecture and Core Features

Foundry is built around two central architectural concepts: direct model access and managed agents. Direct model access allows developers to call foundation models through managed endpoints, while the agent service provides a higher-level abstraction for building systems that can use instructions, tools, knowledge sources, and conversational state.

Figure 5.1 gives a high-level view of how Foundry fits into the wider Microsoft AI ecosystem. It shows Foundry as a developer-oriented platform in which agents can use model providers and retrieve context through Azure AI Search. The figure also places Foundry next to Copilot Studio, which targets a lower-code approach for building copilots integrated with Microsoft 365.

At the model endpoint level, Foundry exposes foundation models through APIs. This layer is suitable when an application needs detailed control over prompt construction, request handling, and response processing. In this mode, the surrounding application remains responsible for most of the orchestration logic.

The agent service builds on top of these model capabilities. Instead of calling a model directly for every interaction, developers can define agents with a specific instruction

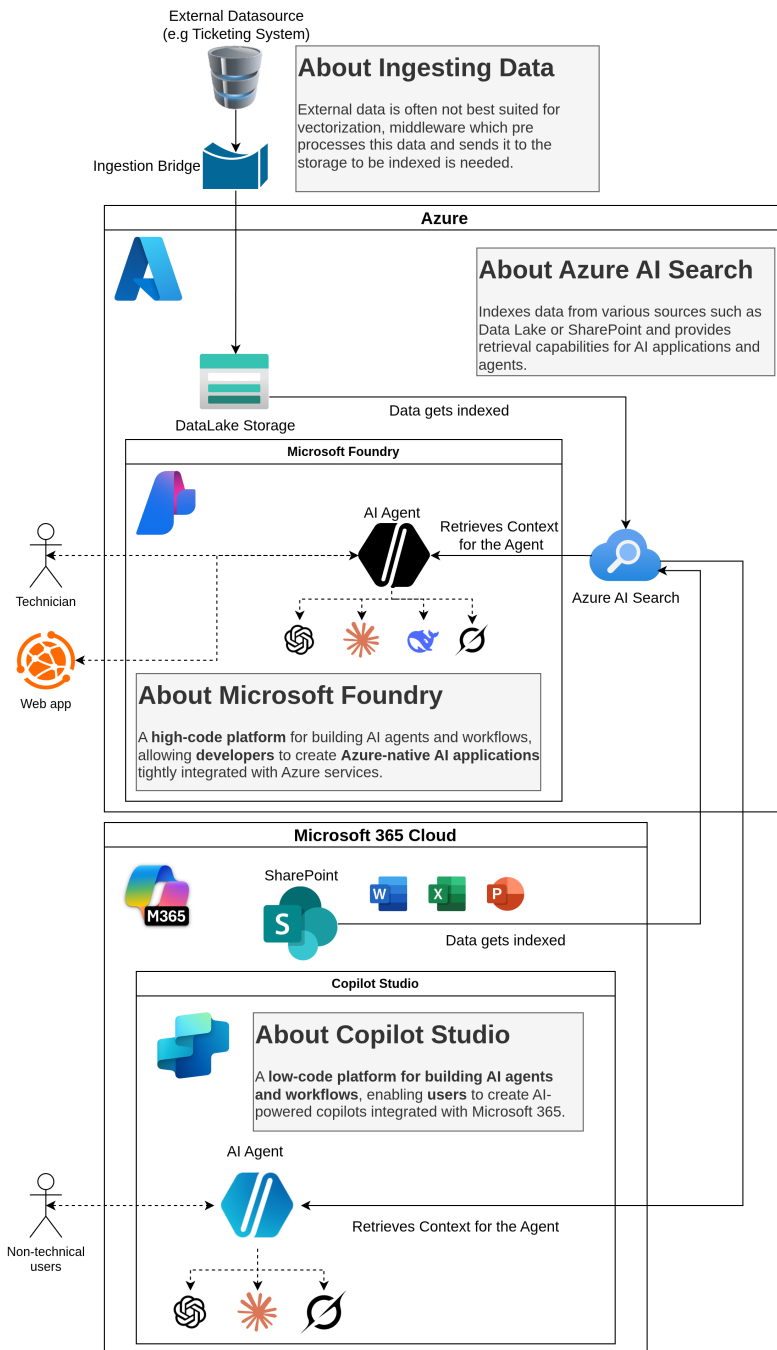


Figure 5.1: High-level architecture of Microsoft Foundry, Azure AI Search, and Copilot Studio in relation to external and Microsoft 365 data sources. Source: own illustration.

set and attach capabilities such as tools or knowledge sources. The agent service then manages parts of the execution flow, including the interaction between the model, the configured tools, and the available context.

Azure AI Search appears in the architecture as the retrieval component that can provide external context to agents. The important architectural point is that the search index remains separate from the agent itself. This separation allows data indexing and agent execution to evolve independently, while still enabling agents to ground responses in enterprise information when required.

The figure also illustrates the different positioning of Foundry and Copilot Studio. Copilot Studio is aimed at rapidly creating business-facing copilots, especially for Microsoft 365-centered scenarios. Foundry, by contrast, provides a more developer-oriented foundation for custom AI systems where model choice, agent behavior, retrieval design, and integration patterns require more direct control.

A further core feature of Foundry is the ability to inspect agent executions. Runs can be traced to understand which model calls, tool calls, and intermediate steps occurred during an interaction. This supports debugging and evaluation of agent behavior without requiring a separate tracing system to be introduced at the application level.

Overall, the architecture of Foundry is best understood as a layered model: foundation model endpoints provide the basic AI capability, the agent service adds managed orchestration, and Azure AI Search can be connected as an external retrieval layer. This structure makes Foundry suitable for custom AI systems that require more control than low-code copilot builders, while still benefiting from managed Azure platform services.

5.1.2 Licensing

Microsoft Foundry follows a consumption-based pricing model aligned with Azure services. Costs are primarily driven by model usage, measured in tokens, as well as by additional services such as vector search, storage, and compute resources.

There is no concept of a free self-hosted tier. Instead, organizations pay for usage and provisioned infrastructure within their Azure subscription. This simplifies licensing from a legal perspective, as there are no restrictions on multi-tenancy or redistribution. However, it introduces ongoing operational cost and dependency on Azure pricing structures.

From a commercial standpoint, this model is well-suited for SaaS applications, as multi-tenant architectures can be implemented without licensing constraints. Cost attribution can be handled at the application level, for example per tenant, per user, or per request.

5.1.3 Deployment Models and User Interaction

Foundry is exclusively available as a managed cloud service within Azure. Applications are typically deployed as backend services that interact with Foundry through SDKs or REST APIs. Common deployment targets include Azure App Service, container-based environments, or serverless functions.

User interaction is not provided by Foundry itself. Unlike Dify, which can directly expose a web UI, Foundry requires a separate frontend or application layer. This design reflects its role as an infrastructure platform rather than an end-user tool.

Developers interact with Foundry through APIs such as the Responses API or the Agent Software Development Kit (SDK). The Responses API provides low-level access to model capabilities, while the Agent SDK enables the definition and execution of managed agents. This separation allows teams to choose between full control and higher-level abstractions depending on their requirements.

5.1.4 Governance and Compliance

Governance is a core strength of Foundry due to its integration with Azure. Identity and access management are handled through Microsoft Entra ID, enabling RBAC, managed identities, and integration with existing enterprise authentication systems. This is especially relevant for Azure-hosted applications and services, because they can use a managed identity to authenticate against Foundry and related Azure services instead of relying on manually managed API keys. As illustrated in Figure 5.2, the application can request an access token from Microsoft Entra ID and then call Foundry using this token, while authorization is enforced through Azure RBAC.

This model reduces operational risk compared to static API keys. API keys are long-lived secrets that must be stored in configuration, protected from leakage, rotated manually, and distributed carefully across environments. A managed identity, in contrast, is bound to the Azure resource itself and receives short-lived tokens from Microsoft Entra ID. Access can then be controlled through Azure role assignments rather than through shared secrets. This makes the authentication model easier to govern, reduces the risk of exposed credentials, and aligns better with enterprise security practices.

Data residency and compliance requirements can be enforced by selecting appropriate Azure regions, including Swiss data centers. This ensures that sensitive data remains within defined geographic boundaries.

Foundry also supports fine-grained access control at the data layer when combined with Azure AI Search. Queries can include user-specific authorization headers, allowing retrieval operations to respect document-level permissions. This is particularly relevant for enterprise scenarios where different users must only access a subset of the available knowledge base.

Auditability is ensured through Azure-native logging and monitoring services. All interactions with models and agents can be logged and integrated into existing Security Information and Event Management (SIEM) systems, supporting compliance requirements such as traceability and incident analysis.

5.2 RAG and Data Ingestion

Foundry does not provide a self-contained knowledge base in the same way as Dify. Instead, retrieval is typically implemented through Azure AI Search, which acts as the

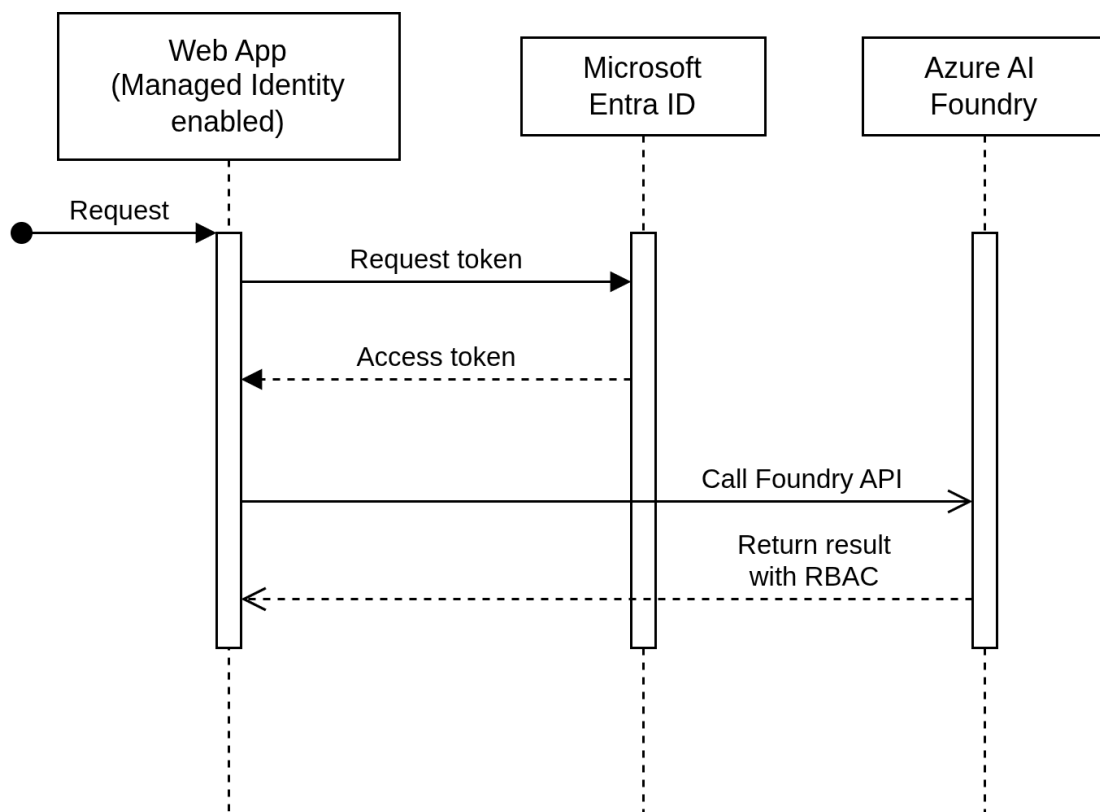


Figure 5.2: Managed identity authentication flow for an Azure-hosted web application accessing Azure AI Foundry through Microsoft Entra ID and RBAC. Source: own illustration

central search and vector index for enterprise data. This makes the architecture more modular, but also more aligned with existing Microsoft and Azure environments.

A major advantage of this approach is that Azure AI Search can ingest data from several supported enterprise data sources without requiring a fully custom ingestion pipeline. Indexers can automatically crawl supported sources, extract textual content and metadata, and populate a search index. Supported sources include Azure Blob Storage, Azure Data Lake Storage Gen2, Azure Structured Query Language (SQL) Database, Azure Cosmos DB, Azure Table Storage, Microsoft OneLake, and SharePoint in Microsoft 365, with some connectors depending on preview features.

This is particularly relevant for companies that already store operational knowledge in Microsoft 365 or Azure. For example, documents stored in SharePoint document libraries or files stored in Blob Storage can be indexed directly by Azure AI Search and then used as grounding data for Foundry-based agents or applications. In such scenarios, the platform does not require data to be duplicated manually into a separate AI-specific knowledge base. Instead, existing enterprise repositories can remain the source of truth.

Ingestion Pipeline

There are two main ingestion approaches.

The first approach uses Azure AI Search indexers. In this model, Azure AI Search pulls data from a configured source, extracts content and metadata, and writes the resulting documents into a search index. This reduces the amount of custom code required and is well suited for standard enterprise repositories such as SharePoint, Blob Storage, Data Lake Storage, SQL databases, and Cosmos DB.

The second approach is a custom ingestion pipeline. In this model, the application is responsible for parsing documents, splitting them into chunks, generating embeddings, assigning metadata, and writing the final records to Azure AI Search. This approach requires more implementation effort but gives the developer full control over chunking strategy, metadata design, enrichment, and update behavior.

In practice, both approaches can also be combined. Standard repositories can be indexed through managed indexers, while application-specific or external systems can be ingested through custom pipelines.

Retrieval

Azure AI Search supports keyword search, vector search, and hybrid retrieval. Hybrid retrieval combines traditional full-text search with vector similarity, which is especially useful for enterprise data where exact terminology and semantic meaning are both important.

Metadata fields can be used to filter retrieval results by properties such as document type, department, source system, customer, classification level, or access scope. This allows the search layer to serve multiple use cases from a shared index without requiring a separate index for every agent or workflow.

When used with Foundry agents, retrieval can be integrated as a tool or knowledge source. This enables an agent to query indexed enterprise data during execution rather than relying on a fixed retrieval step before every model call. The result is a more flexible agentic RAG architecture, where the agent can decide when additional information is required.

Access Control

Access control is one of the strongest arguments for Foundry in enterprise environments. Azure AI Search can support document-level access control patterns by using permission metadata and Microsoft Entra ID integration. This allows retrieval results to be filtered based on the identity or authorization context of the user.

This is important for shared enterprise data sources such as SharePoint, where different users may have access to different documents within the same repository. Instead of creating separate knowledge bases per user or per department, a shared search index can be queried with security trimming so that users only receive results they are allowed to access.

This model is more complex to configure than Dify’s workspace-level knowledge base, but it better matches enterprise requirements where permissions already exist in Microsoft 365, Azure Storage, or other Microsoft-controlled systems.

5.3 Summary

Microsoft Foundry is a managed AI platform that emphasizes integration, governance, and scalability within the Azure ecosystem. Its agent service abstracts complex orchestration patterns, while Azure AI Search provides a flexible and powerful retrieval backend.

Compared to Dify, Foundry trades self-hosting flexibility for operational simplicity and enterprise-grade capabilities. Features such as document-level access control, native identity integration, and managed observability make it particularly well-suited for organizations with strict compliance requirements or existing Azure infrastructure.

The main limitations are the lack of a built-in user interface and the dependency on Azure services, which introduce both architectural constraints and ongoing cost. However, for production environments requiring reliability, security, and scalability, Foundry provides a cohesive and mature solution.

Chapter 6

Comparison

The previous two chapters examined Dify and Microsoft Foundry on their own terms. This chapter puts them side by side and asks the question that motivated the whole evaluation: given the same use cases and requirements, what does each platform actually cost you — in effort, control, and dependency — and when does that trade-off tip in one direction or the other?

The comparison is structured in two parts. First, both platforms are contrasted with a self-developed solution to establish why either is preferable to continuing custom development. Second, a feature matrix documents the concrete differences across ingestion, retrieval, orchestration, security, and observability. The feature matrix is not a scoring table; it is a reference that surfaces the trade-offs that matter depending on an organisation’s specific constraints.

6.1 Feature Comparison of Evaluated Platforms

Table 6.1: Feature matrix comparing Microsoft Foundry, Dify, and a self-built platform. Legend: ✓ implemented/available, Δ technically feasible but not implemented or only partially implemented, × not supported, □ can be added by using visual workflow builder, * can be added by connecting external system. Source: own illustration.

Feature	Microsoft Foundry	Dify	Self-Built	Notes
Ingestion				
Multiple data source ingestion	✓	✓	✓	Foundry via Azure Data connectors (10+ sources); Dify supports community built connectors; self-built via custom connectors

Feature	Microsoft Foundry	Dify	Self-Built	Notes
Add metadata during ingestion	✓	Δ	✓	Foundry supports meta-data fields natively in the indexer pipeline; Dify allows metadata only via API, not in the UI
Incremental re-indexing	✓	Δ	Δ	Foundry indexer tracks change detection automatically; change detection must be implemented manually; self-built pipeline supports it but change detection was not implemented in the PoC
Multi-modal ingestion (Optical Character Recognition (OCR) / vision)	✓	✓	Δ	Foundry uses Azure Document Intelligence for layout-aware extraction; Dify supports vision models for image content; self-built would require an additional OCR/multimodal model
Indexing				
Fixed-size chunking	✓	✓	✓	All three support fixed-size chunking; self-built used naive fixed-size chunking as sole strategy
Semantic / structure-aware chunking	✓	×	×	Foundry uses Document Intelligence Layout model for structure-aware semantic chunking; Dify offers only fixed-size chunking with configurable size and overlap; self-built did not implement semantic chunking
Contextual chunk header (document title / section prefix)	Δ	□	×	Foundry can enrich chunks via custom skills in the indexer pipeline; Dify supports header injection via custom workflow nodes; not implemented in self-built

Feature	Microsoft Foundry	Dify	Self-Built	Notes
Parent-child chunking	×	✓	×	Dify supports parent-child mode natively; child chunks are matched at retrieval time, parent chunks returned as context; not available in Foundry's indexer pipeline; not implemented in self-built
Q&A pair indexing (Q-to-Q matching)	×	✓	×	Dify can auto-generate question-answer pairs per chunk and match user queries against questions rather than passages (Q-to-Q strategy), improving precision for FAQ-style content; not available in Foundry or self-built
Query processing				
Metadata-based filtering	✓	✓	Δ	Foundry/Azure AI Search supports filter expressions on metadata fields; Dify supports metadata filters aswell; partially implemented in self-built PoC
query rewriting / expansion	✓	□	✓	Foundry's agentic retrieval decomposes complex queries into sub-queries automatically; in Dify this requires a custom LLM node in the workflow; implemented in self-built
Retrieval				
hybrid search (vector + keyword)	✓	✓	Δ	Foundry uses RRF-fused hybrid search as default; Dify supports hybrid search; self-built had basic vector search only, keyword layer not completed

Feature	Microsoft Foundry	Dify	Self-Built	Notes
Agentic / multi-step retrieval (Dartboard RAG style)	✓	□	×	Foundry’s agentic retrieval runs parallel sub-queries and merges results; Dify would require an external orchestration loop; not implemented in self-built
Post-retrieval				
reranking	✓	✓	×	Foundry applies semantic reranking via Azure AI Search semantic ranker; Dify supports cross-encoder rerankers (e.g. Cohere Rerank, BGE), configurable per knowledge base; not implemented in self-built
RAG Fusion (multi-query + RRF merge)	✓	□	×	Foundry’s hybrid pipeline uses RRF natively; can be constructed via multiple retrieval nodes in Dify’s workflow editor; not implemented in self-built
Generation				
Source citation in responses	✓	✓	Δ	Foundry surfaces document references with links; Dify returns citations in chatflow apps, workflow apps implement citations manually; self-built returned source filenames but no chunk-level references
Advanced / multi-stage RAG				
RAPTOR (recursive abstractive summarisation)	×	*	×	Not natively supported on either platform; can be implemented externally and results ingested into Dify’s knowledge base; not implemented in self-built

Feature	Microsoft Foundry	Dify	Self-Built	Notes
GraphRAG	*	*	×	Microsoft publishes the GraphRAG OSS library and it can be combined with Azure AI models, but it is not natively integrated into the Foundry indexing pipeline; Dify would require an external graph store integration; not implemented in self-built
Workflow and orchestration				
Multi-step workflows	✓	✓	✓	Supported by all platforms
Tool / function invocation	✓	✓	✓	Wide variety of tools; self-built only supported custom tools and MCP
Human-in-the-loop steps	✓	Δ	Δ	Foundry supports approval steps via Azure Logic Apps integration; Dify has no native approval UI, requires custom implementation; self-built required a manual hand-off step
Workflow versioning	✓	✓	Δ	Both platforms track workflow versions; self-built had no versioning system
Error handling and retry / fallback	✓	Δ	Δ	Foundry supports retries and dead-letter queues; Dify has basic retry nodes but no dead-lettering; self-built had no systematic error handling
Model and prompt control				

Feature	Microsoft Foundry	Dify	Self-Built	Notes
Multiple model providers	Δ	✓	✓	Foundry primarily targets Azure-hosted models (OpenAI, Llama, Mistral via Azure model catalogue) — non-Azure providers require custom integration; Dify is model-agnostic and supports OpenAI, Anthropic, Ollama, and 100+ providers out of the box; self-built was model-agnostic by design
Model selection per workflow step	✓	✓	×	Both platforms allow a different model per node; self-built used a single model per instance
Security and multi-tenancy				
Authentication integration (SSO / OpenID Connect (OIDC))	✓	Δ	✓	Foundry integrates with Entra ID / Azure AD out of the box; Dify supports OAuth 2.0 for tool integrations but SSO for the platform itself is enterprise-tier only; self-built used standard OIDC
RBAC (RBAC)	✓	Δ	✓	Foundry supports full RBAC via Azure IAM and agent identities; Dify has workspace-level roles (owner, editor, viewer) but granular per-resource RBAC requires enterprise tier; self-built had a tier based role model implemented in PoC
Tenant isolation	✓	Δ	✓	Foundry isolates tenants at the Azure subscription / resource group level; Dify uses workspace isolation but shared infrastructure unless self-hosted per tenant; self-built implemented strict data isolation per tenant
Observability and evaluation				

Feature	Microsoft Foundry	Dify	Self-Built	Notes
Request / trace logging	✓	*	✓	Foundry integrates with Azure Monitor and Application Insights; Dify provides LLMOps using external providers like Langfuse and Arize; self-built also integrated Langfuse Foundry tracks token usage via Azure Cost Management; Dify shows per-run token counts in the trace view using Langfuse; self-built also integrated Langfuse Foundry supports evaluation via Prompt Flow evals; Dify provides annotation and labelling tools but no built-in automated RAG metrics; self-built had no evaluation tooling
Cost and token tracking	✓	*	Δ	
RAG retrieval evaluation	Δ	*	×	
Deployment and extensibility				
Self-hosting support	×	✓	✓	Foundry is Azure-cloud-only; Dify can be self-hosted via Docker or Kubernetes; self-built was self-hosted by design Foundry supports on-prem via Azure Arc / Azure Local but requires Azure control plane connectivity; Dify self-hosted can run fully air-gapped; self-built runs fully on-prem Both platforms expose REST APIs for all core operations; self-built was built API-first Foundry is tightly coupled to the Azure ecosystem (Azure AI Search, Entra ID, Azure Monitor); Dify is open-source and model-agnostic; self-built risk depends on chosen components
On-premises / air-gapped deployment	Δ	✓	✓	
API-first extensibility	✓	✓	✓	
Vendor lock-in risk	High	Low	Low	

6.2 Summary

The feature matrix confirms the structural difference established in the platform selection: Dify and Microsoft Foundry are not better or worse in absolute terms — they are optimised for different operating contexts.

Microsoft Foundry is the lower-friction choice for organisations already invested in the Azure ecosystem. Identity integration, observability, and retrieval infrastructure arrive as managed services. The cost is real vendor dependency: the platform is Azure-cloud-only, model selection is constrained to the Azure catalogue, and any capability outside the managed surface requires custom integration.

Dify offers comparable workflow capabilities with full infrastructure control and no hard vendor dependency. The cost is operational ownership: synchronisation, access control, multi-tenancy, and observability all require custom effort that Foundry provides out of the box. For organisations with the capacity to absorb that effort — or with compliance requirements that make self-hosting non-negotiable — Dify is the more appropriate choice.

In both cases, the platforms justify themselves against a self-developed solution. The capabilities that would require the most sustained engineering effort — hybrid retrieval, reranking, agentic orchestration, prompt management, and tracing — are provided by both platforms at a maturity level that custom development would struggle to match within a realistic project timeline.

The following chapters examine whether this holds in practice, through the implementation and evaluation of the use cases described in the objectives.

Part III

Use Cases

Chapter 7

MSP Support Automation Use Cases

7.1 Introduction

MSPs operate in dynamic IT environments where technicians handle recurring incidents, complex troubleshooting, and time-sensitive support requests daily. Over time, this generates large amounts of operational knowledge in the form of tickets, incident histories, and internal documentation — knowledge that is difficult to reuse efficiently during active support work.

A previous academic project in collaboration with an MSP developed a prototype exploring how LLMs combined with RAG can help technicians leverage historical ticket data. That project provided access to approximately 5000 real-world support tickets from productive customer environments. This thesis reuses the same dataset and collaboration context to evaluate Dify and Microsoft Foundry in realistic enterprise scenarios.

Two use cases are implemented and compared across both platforms. The first is an interactive technician assistant providing conversational access to historical ticket knowledge. The second is automated ticket enrichment, where newly created tickets are analyzed and enriched with relevant context before a technician begins work. The following chapters describe the architecture, implementation, and evaluation of both.

Chapter 8

Technician Assistant

8.1 Task Overview

The technician assistant provides MSP technicians with conversational access to historical support knowledge. Technicians can ask questions about past incidents, troubleshooting steps, or previous resolutions, and the system retrieves relevant tickets and generates a grounded response using RAG.

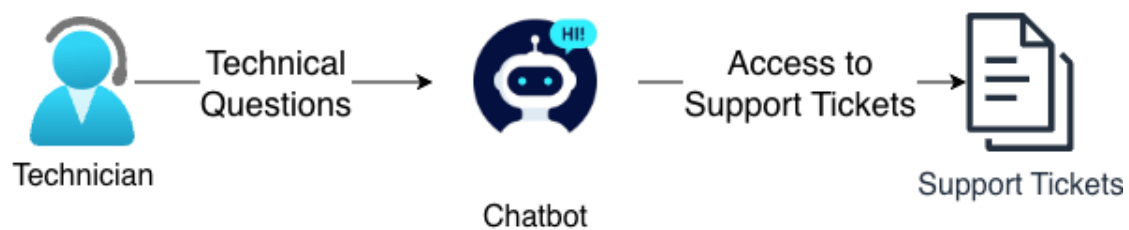


Figure 8.1: Simple illustration of the technician assistant architecture. The system consists of a conversational interface, a retrieval component that searches the ticket knowledge base, and a generation component that produces responses based on retrieved information. Source: own illustration.

The focus of this implementation is evaluating how Dify and Microsoft Foundry handle workflow orchestration, knowledge integration, and retrieval quality — not rebuilding the prototype from the prior project.

8.2 Requirements

8.2.1 Functional Requirements

FR-TA1: Conversational Chat Interface The system shall provide a natural language chat interface through which technicians can submit queries and receive responses. **Acceptance criterion:** A technician can submit a free-text question

and receive a coherent, relevant response without any additional configuration or training.

FR-TA2: RAG-Based Knowledge Retrieval Responses shall be grounded in the historical support ticket knowledge base using RAG. **Acceptance criterion:** For a set of ten reference queries, at least eight responses cite at least one relevant source ticket retrieved from the knowledge base.

FR-TA3: Ticket Retrieval via MCP The system shall support tool calling to retrieve a specific ticket by ticket number from the ticketing system via an MCP server. **Acceptance criterion:** Given a valid ticket number, the system returns the correct ticket record within the response.

FR-TA4: Recent Ticket List The interface shall display a dynamically generated list of recently created or modified tickets as quick-access entry points. **Acceptance criterion:** The list reflects tickets modified within the last 24 hours and updates with a manual refresh.

FR-TA5: Single-Click Troubleshooting Launch A technician shall be able to launch a pre-seeded troubleshooting conversation from a selected ticket in a single interaction. **Acceptance criterion:** Selecting a ticket from the recent ticket list opens a conversation with the ticket context already loaded, requiring no additional input from the technician.

FR-TA6: Follow-Up Question Suggestions The system shall generate two to four contextually relevant follow-up question suggestions after each response, rendered as interactive buttons. **Acceptance criterion:** Every assistant response is accompanied by two to four clickable suggestion buttons, each containing a question relevant to the current conversation context.

FR-TA7: Persistent Conversation Context The system shall maintain conversation history across multiple turns within a session. **Acceptance criterion:** A follow-up question that references a prior turn (e.g., “what was the resolution for that ticket?”) is answered correctly using earlier context without the technician repeating information.

FR-TA8: Source References in Responses Generated responses shall include references to the source tickets from which information was retrieved. **Acceptance criterion:** Every response that draws on retrieved ticket content includes at least one identifiable ticket reference (e.g., ticket number or title).

8.2.2 Non-Functional Requirements

NFR-TA1: Response Latency The system shall return an initial response or first streamed token within five seconds for 90% of queries during normal operation. **Acceptance criterion:** Latency measurements across a representative query set show that the 90th-percentile time-to-first-token does not exceed 10 seconds.

NFR-TA2: hallucination Minimization Responses shall be grounded in retrieved source material and shall not introduce claims that contradict or are inconsistent with the retrieved context. **Acceptance criterion:** Manual review of ten sampled responses finds no response that asserts a specific technical fact that conflicts with or contradicts information in the retrieved tickets.

NFR-TA3: traceability Every response drawing on the knowledge base shall include ticket references sufficient for a technician to locate the source. **Acceptance criterion:** All responses in the evaluation sample that use retrieved content include at least one resolvable ticket identifier. (Covered by FR-TA8.)

NFR-TA4: Extensibility The assistant architecture shall permit additional tools and data sources to be integrated without modifying the core conversation flow. **Acceptance criterion:** A second tool (beyond FR-TA3) can be added and made available to the assistant without changes to the existing workflow definition or prompt templates.

8.2.3 Requirements Coverage

Table 8.1: Functional Requirements — technician assistant

ID	Acceptance Criterion	Met
FR-TA1: Chat Interface	Free-text query returns a coherent response	Yes
FR-TA2: RAG Retrieval	$\geq 8/10$ reference queries cite a relevant source ticket	Yes
FR-TA3: Ticket via MCP	Valid ticket number returns correct ticket record	Yes
FR-TA4: Recent Ticket List	List reflects last 24 h and updates dynamically	Yes
FR-TA5: Troubleshooting Launch	Ticket selection opens pre-seeded conversation in one step	Yes
FR-TA6: Follow-Up Suggestions	Every response shows 2–4 relevant suggestion buttons	Yes
FR-TA7: Persistent Context	Follow-up referencing prior turn answered correctly	Yes
FR-TA8: Source References	Retrieved-content responses include ≥ 1 ticket reference	Yes

Table 8.2: Non-Functional Requirements — technician assistant

ID	Acceptance Criterion	Met
NFR-TA1: Re- sponse Latency	P90 time-to-first-token ≤ 10 s	Yes
NFR-TA2: halluci- nation Min.	No claims contradicting source material in 10-response sample	Yes
NFR-TA3: trace- ability	All retrieved-content responses include a ticket ID	Yes
NFR-TA4: Exten- sibility	Second tool added without modifying core workflow	Yes

8.3 Data Source and Ticket Dataset

8.3.1 Support Ticket Structure

The MSP provided a static export of approximately 5000 historical support tickets as spreadsheet exports. Each row represented a single ticket with associated metadata and text content. The most relevant fields for the implemented use cases were:

Field	Description
<code>task_id</code>	Unique identifier of the support ticket
<code>task_name</code>	Short ticket title or subject
<code>task_description</code>	Initial problem description or opening ticket text
<code>resolution</code>	Resolution history, technician notes, and closing solu- tion
<code>create_time</code>	Ticket creation timestamp
<code>account_id</code>	Reference to the associated customer account

Table 8.3: Most relevant fields of the exported support ticket dataset. Source: own illustration.

8.3.2 Ticket Labeling for Evaluation Dataset Creation

The MSP provided approximately 5000 historical support tickets, all of which were included in the knowledge base used for retrieval.

To support evaluation dataset creation, a subset of 1007 tickets in which both the problem description and the solution exceeded 250 characters was additionally reviewed in Label Studio. Of those 1007 tickets, about 79% contained a clear problem and about 50% contained a clear solution. The detailed labeling workflow, subset-selection rationale, and label distribution are described in Section 10.6.

8.3.3 Data Preparation and Ingestion

Before ingestion, the exported ticket dataset was preprocessed and normalized using a custom Python-based ingestion script developed during the previous academic project. The script extracted relevant fields, cleaned formatting artifacts, and transformed tickets into a consistent text structure suitable for embedding and retrieval.

Although the dataset used in this thesis was static and only required a one-time ingestion, the ingestion concept was originally designed for continuous synchronization from productive ticketing systems.

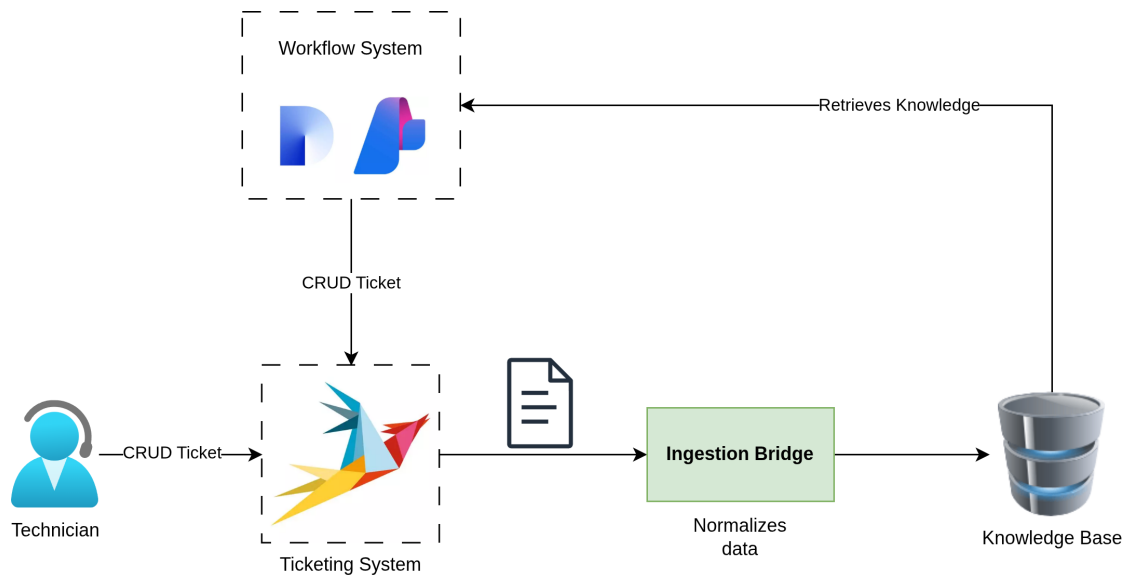


Figure 8.2: Simplified ingestion bridge architecture that would be implemented in a production system. The ingestion bridge retrieves and normalizes ticket data before forwarding it for ingestion. Source: own illustration.

The following example shows a simplified and anonymized processed ticket after transformation by the ingestion pipeline.

Ticket ID: T20240419.0620

Title: Outlook error message on terminal server

Description:

Users working on a newly deployed terminal server repeatedly received Outlook error messages. The issue affected different users at different times. Restarting the server temporarily resolved the problem.

Resolution History / Notes:

- Office updated and reinstalled
- User profiles recreated
- Registry keys and AppData cleaned
- Windows updates installed
- Issue reviewed together with external vendor support
- Terminal server redeployed
- Daily server restart implemented as workaround

Figure 8.3: Simplified and anonymized example of a processed support ticket after ingestion preprocessing. Source: own illustration.

8.4 Knowledge Base Construction

After preprocessing, the ticket data was integrated into a searchable knowledge base for RAG. Microsoft Foundry uses Azure AI Search as the retrieval backend; Dify provides a built-in knowledge base with integrated ingestion and retrieval.

8.4.1 Azure AI Search

The processed tickets were transformed into individual Markdown files and uploaded into Azure Data Lake Storage Gen2, which was connected to Azure AI Search using the built-in data source integration. This allowed ticket documents to be indexed directly while remaining compatible with the Foundry Agent Service.

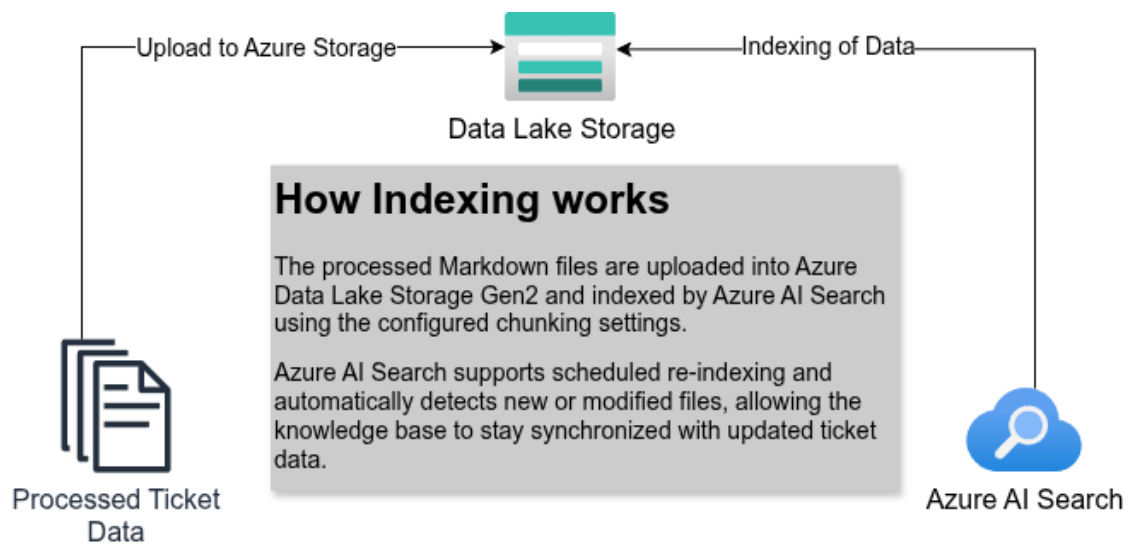


Figure 8.4: Ingestion workflow for Azure AI Search using Azure Data Lake Storage Gen2 as the document source. Source: own illustration.

An important consideration when designing the ingestion workflow was ongoing synchronization. Azure AI Search supports scheduled re-indexing of connected storage sources and automatically detects changed or newly uploaded files, allowing the knowledge base to stay current without full manual re-ingestion. This is one of the first notable architectural differences compared to the Dify implementation described below.

After indexing, the Azure AI Search index was exposed directly as a retrieval tool for Foundry Agents, enabling agentic RAG workflows where the agent dynamically queries the knowledge base during response generation.

8.4.2 Dify Knowledge Base

For Dify, the processed Markdown files were uploaded directly to the knowledge base using the Dify knowledge base API via a Python script. While a S3-compatible storage

system could alternatively be connected through Dify’s S3 data source connector, allowing files to be imported and indexed from there, this approach was not pursued due to an important limitation: Dify provides no way to attach metadata to documents within the knowledge pipeline itself. Metadata can only be set after ingestion via a separate API call.

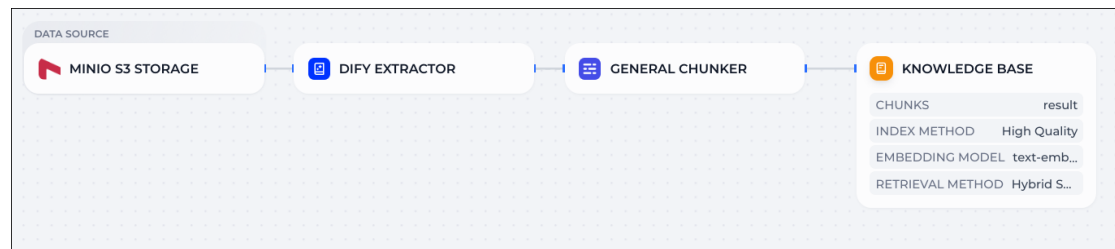


Figure 8.5: Dify knowledge base ingestion workflow using self-hosted MinIO S3 storage as the data source. Files are processed through Dify’s built-in extractor and chunker before being indexed into the knowledge base. Source: own screenshot.

This metadata limitation has practical consequences beyond the ingestion approach. In a productive MSP environment where documents must be filtered based on user roles or tenant context, metadata is essential for access control at retrieval time. Since Dify cannot attach metadata during the pipeline itself, a scripted ingestion flow is effectively required regardless — meaning the S3 connector approach offers no advantage in this scenario, and direct API upload is the more straightforward path.

Beyond metadata, a productive environment would also require continuous synchronization. Unlike Azure AI Search, Dify does not provide scheduled re-indexing. A separate component would be needed to detect ticket changes, regenerate the corresponding Markdown file, and update the affected document via the API. The key difference compared to the Azure AI Search approach is where platform automation begins: with Azure AI Search, the custom component only needs to update the file in storage, after which the configured indexer handles the rest. With Dify, that same component must also explicitly trigger the document update through the API, and additionally issue a follow-up call to set the required metadata.

8.4.3 Access Control in RAG

Access control was not implemented in this PoC. However, it is an important architectural consideration for any productive deployment.

In a real MSP environment, tickets may contain customer-specific information that should not be accessible to every technician. Controlling access at the chat interface level alone is insufficient — if unauthorized documents are retrieved as context, sensitive information can still surface in the generated response. Access control must therefore be enforced at retrieval time.

Azure AI Search

Azure AI Search addresses this through document-level access control using permission metadata from Azure Data Lake Storage Gen2, evaluated against the authenticated Microsoft Entra ID identity of the requesting user. At query time, Azure AI Search performs security trimming so that only documents the user is permitted to access are returned.

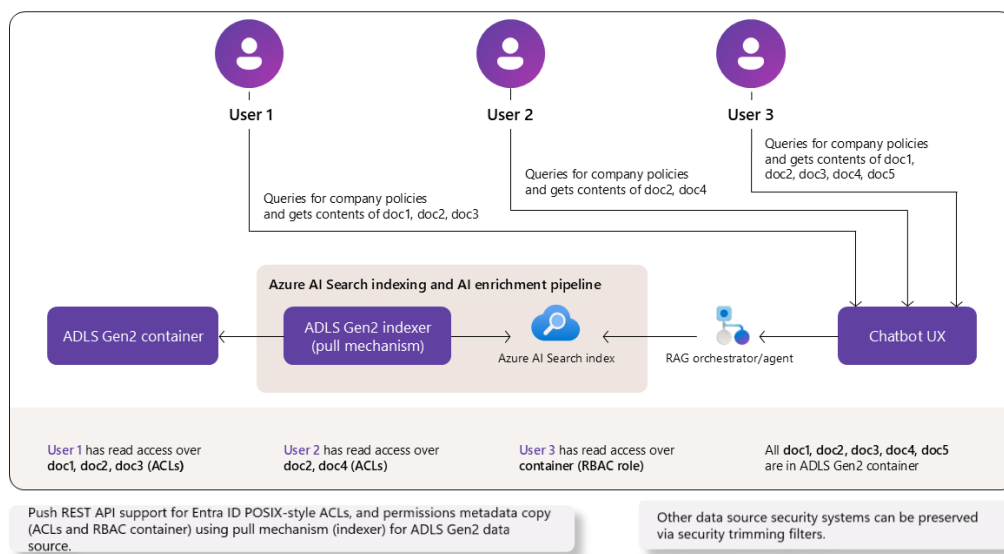


Figure 8.6: Document-level access control with Azure AI Search and Azure Data Lake Storage Gen2. Permission metadata from ACLs and RBAC assignments is indexed together with document content and used for query-time security trimming. Source: Microsoft Learn [1].

For the technician assistant, this would mean that ticket documents could be organized in Azure Data Lake Storage Gen2 according to the required access model, for example by customer-specific folders. The ingestion pipeline would still be responsible for uploading the processed ticket documents to the correct location and assigning the appropriate permissions, such as Azure RBAC roles or ACLs for the corresponding Entra ID users or groups.

The advantage of this approach is that the authorization model can be based on the same Entra ID identities, groups, and permissions already used in the organization. The AI application would not have to implement its own separate document-level authorization logic. Instead, Azure AI Search can use the indexed permission metadata during retrieval and only return documents that match the requesting user's access scope.

Dify Knowledge Base

Dify supports metadata-based filtering but does not provide an equivalent integrated access control mechanism. To address this, the Secure Knowledge Retrieval Plugin developed as part of this thesis was used. The plugin intercepts retrieval calls and injects user-specific metadata filters, ensuring that only documents matching the technician’s permitted access scope are returned.

This approach is feasible, but places more responsibility on the implementation. The correctness of access control depends on metadata being complete and consistently maintained during ingestion. All retrieval paths must pass through the plugin — any unfiltered knowledge base query would bypass the access restrictions entirely.

8.5 System Architecture

The technician assistant was implemented as a full-stack SvelteKit application providing the conversational interface, user authentication, and communication with the platform-specific backends. Authentication was handled using BetterAuth with PostgreSQL as the persistence layer. BetterAuth was chosen because it allowed authentication to be bootstrapped quickly, while still supporting the option to offload authentication to Microsoft Entra ID, which would be desirable for a production deployment following the Azure Foundry-based architecture.

The same frontend was used for both platform implementations. The only architectural difference at the frontend level was the backend endpoint: for Foundry, requests went to the Azure-hosted agent; for Dify, requests went to the self-hosted Dify workflow endpoint. This allowed platform differences to be observed in isolation, independent of frontend behavior.

Microsoft Foundry

The Foundry implementation used the Foundry Agent Service together with Azure AI Search. User queries submitted through the frontend were forwarded to the Foundry Agent, which dynamically decided whether to issue retrieval queries or tool calls before generating a response.

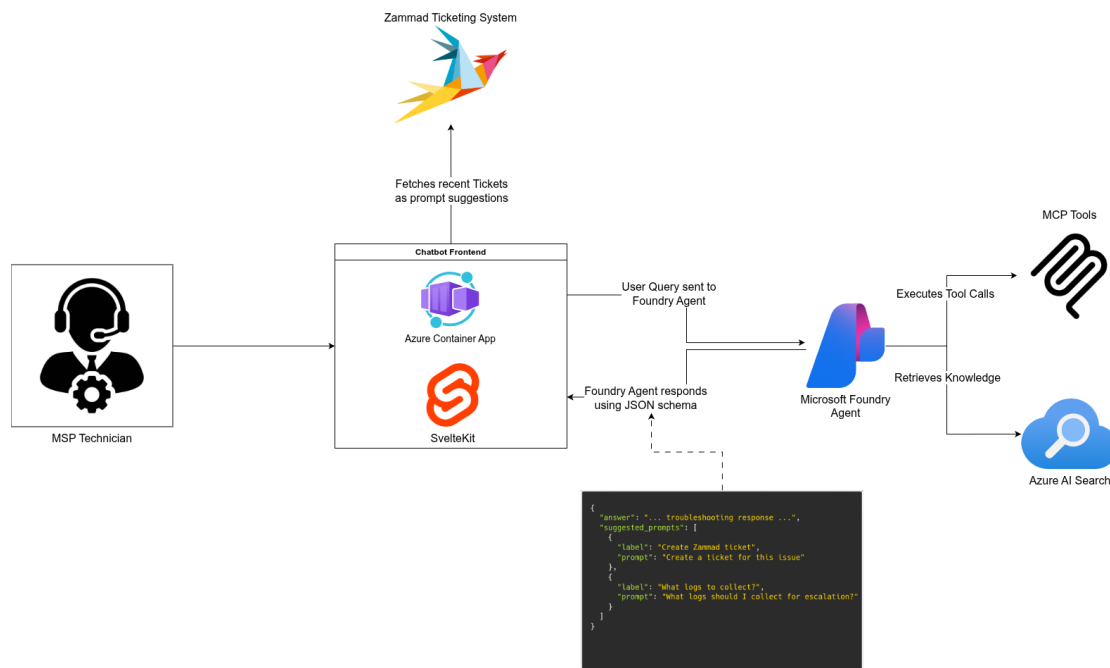


Figure 8.7: Architecture of the Microsoft Foundry-based technician assistant. The front-end communicates with the Foundry Agent, which uses Azure AI Search for RAG retrieval and MCP tools for ticket-related operations. Source: own illustration.

Retrieved ticket documents from Azure AI Search were incorporated into the prompt context, grounding responses in historical MSP incidents rather than relying on pre-trained model knowledge alone. References to source tickets were included in generated responses, with mocked destination URLs used in place of the live ticketing system the MSP did not provide direct access to.

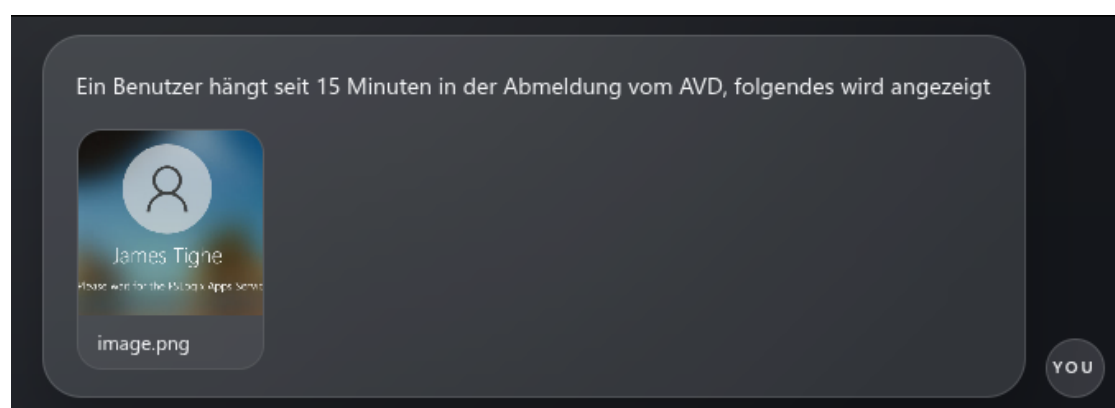


Figure 8.8: Example query about a user unable to sign out of an AVD session. Source: own screenshot.

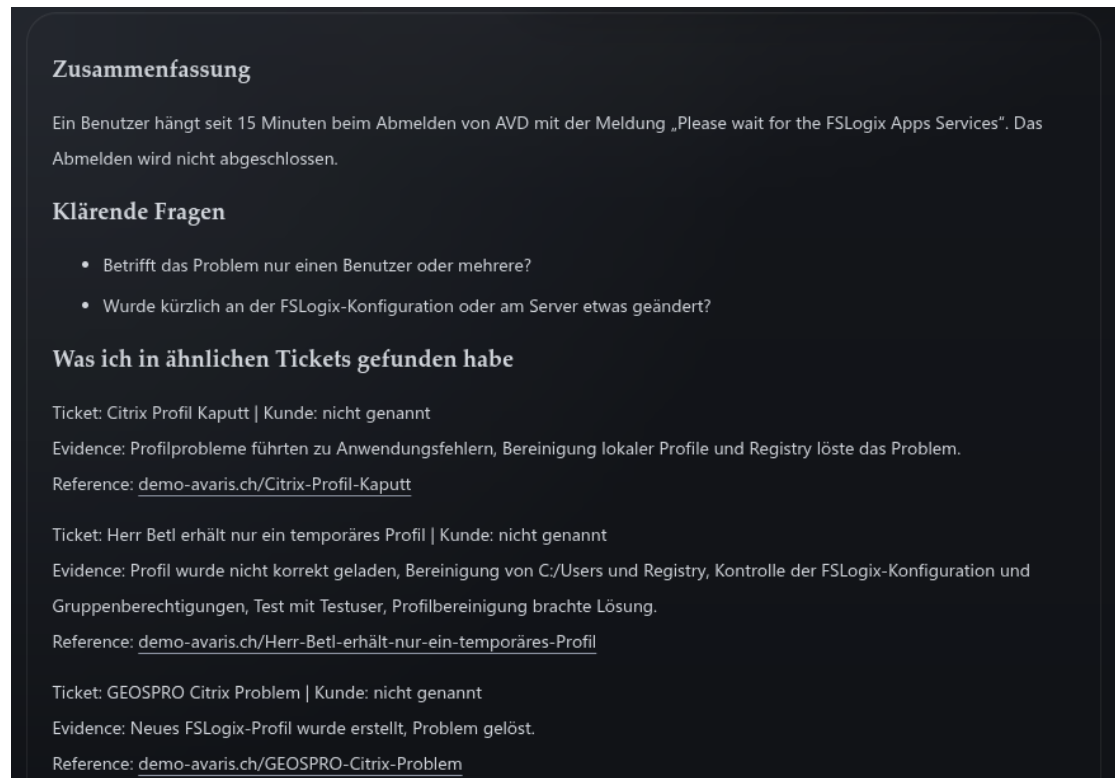


Figure 8.9: First part of the response. The assistant summarizes the issue, asks for clarifying information and provides references to similar historical tickets. The response is grounded in the retrieved ticket content, with mocked URLs for traceability. Source: own screenshot.

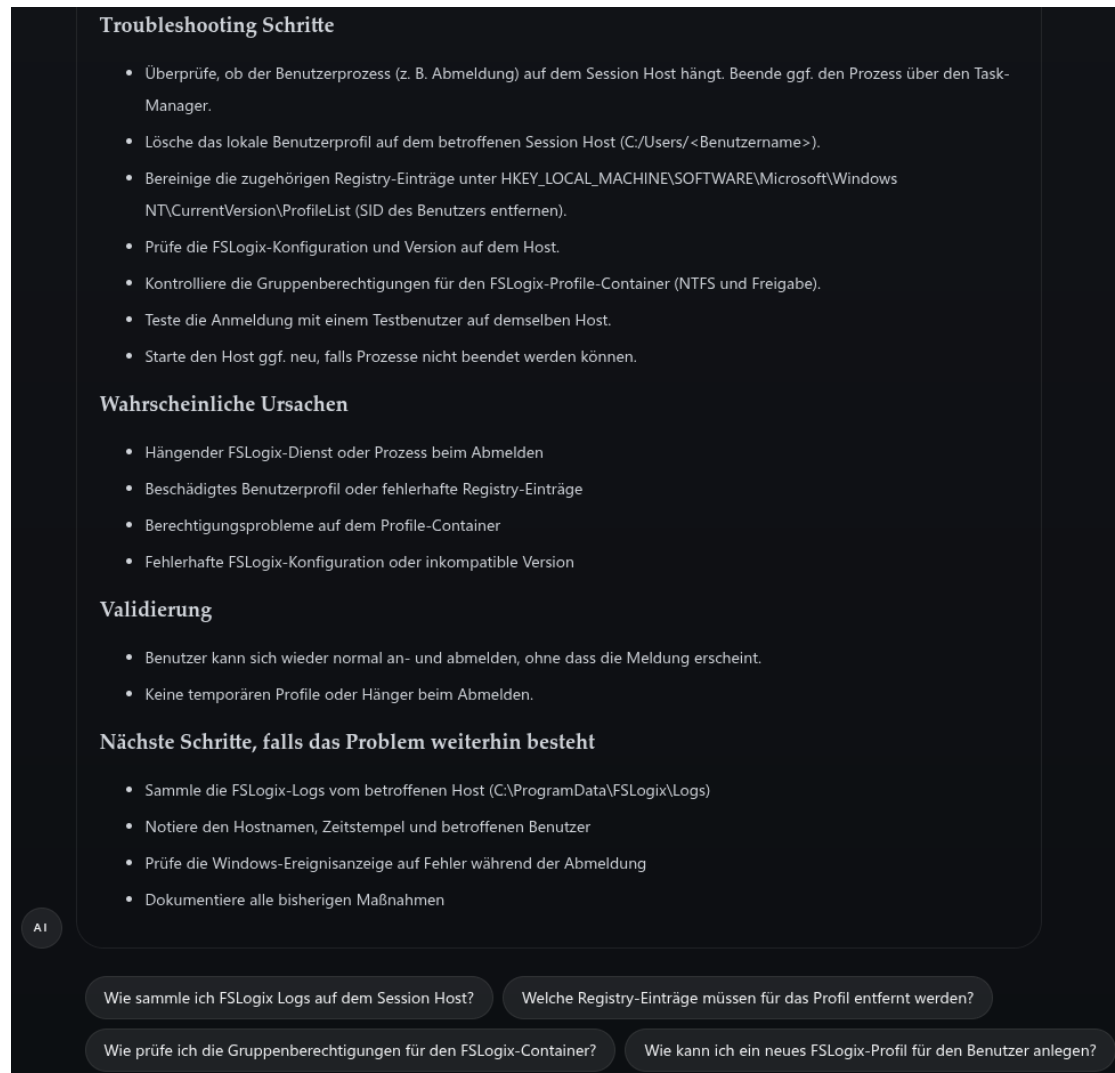


Figure 8.10: Second part of the response. The assistant provides more detailed troubleshooting steps and further actions to be taken by the technician. Source: own screenshot.

After each response, the agent returned two to four suggested follow-up questions using a predefined JavaScript Object Notation (JSON) response schema. These were generated based on the current conversation context and retrieved ticket information, and rendered as interactive buttons in the frontend. The JSON schema was both defined in the system prompt and separately enforced through the Foundry UI, meaning the output structure was validated at the platform level.

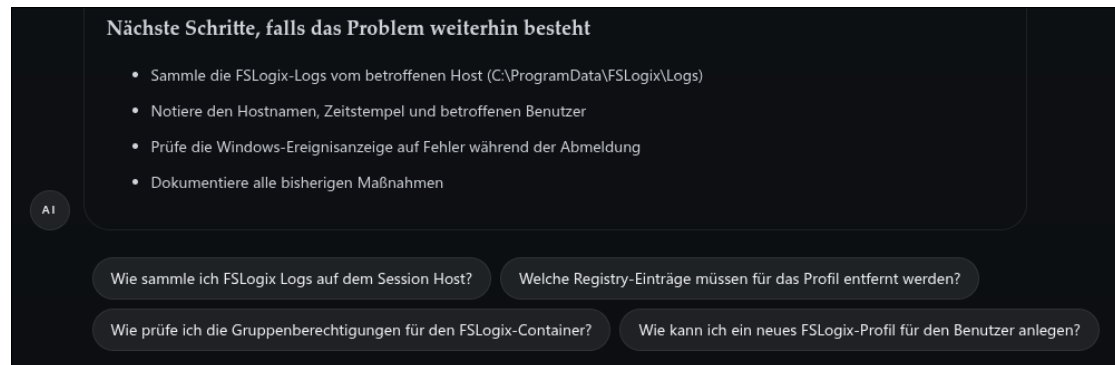


Figure 8.11: Dynamically generated follow-up suggestions returned by the Foundry Agent and displayed as interactive buttons in the frontend. Source: own screenshot.

The frontend also displayed a list of recently created or modified tickets retrieved from Zammad. Selecting a ticket automatically initialized a troubleshooting conversation using that ticket’s context, triggering an MCP tool call to fetch the corresponding ticket details.

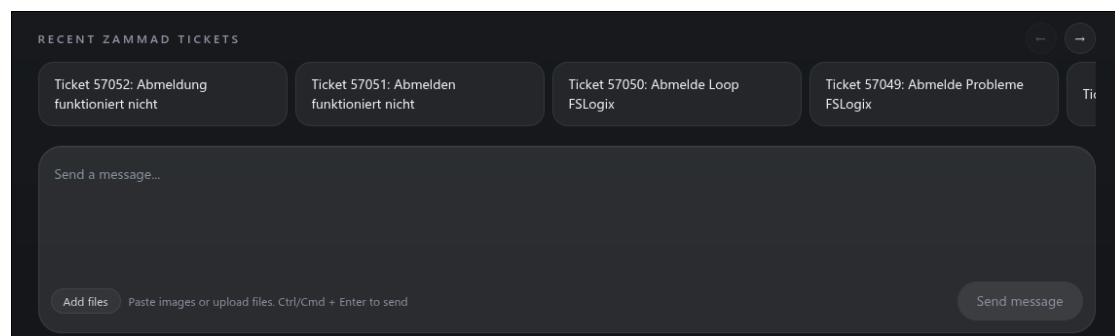


Figure 8.12: Recently created or modified tickets displayed as quick-access entries. Clicking a ticket triggers an MCP tool call to fetch ticket details and initialize a troubleshooting conversation. Source: own screenshot.

System Prompt The behavior of the technician assistant was controlled through a dedicated system prompt. This prompt was designed for an interactive troubleshooting workflow where a technician asks questions through the frontend and receives structured guidance based on historical ticket data.

A central part of the prompt was the definition of the assistant’s role and scope:

You are an internal MSP Technician Assistant for troubleshooting IT issues.

Your job is to help technicians diagnose and resolve problems quickly, using the provided ticket knowledge base as primary evidence.

You behave like an experienced MSP engineer who focuses on fast diagnosis, practical troubleshooting steps, and clear escalation guidance.

Always prioritize evidence from historical tickets over generic advice.

This role definition was important because the assistant should not behave like a general-purpose chatbot. The expected behavior was closer to an experienced MSP engineer who first checks available internal knowledge and then provides practical troubleshooting guidance.

The prompt also defined how the assistant should use the available tools. The assistant had access to the Zammad MCP integration, the historical ticket knowledge base, and web search. However, the system prompt explicitly defined when these tools should be used and in which order. For example, when a ticket ID is provided, the assistant must first retrieve the ticket and then search the knowledge base based on the extracted ticket context:

If the user asks for help with a specific ticket ID, follow this exact sequence:

1. Retrieve the ticket using the Zammad-MCP tool.
2. Extract the key troubleshooting context from the ticket:
 - problem description
 - error messages
 - affected system or application
 - environment information (customer, hostname, OS if available)
3. Use this extracted context to perform a knowledge base search.
4. Decide whether Web Search is needed:
 - use Web Search if KB results are missing, weak, outdated, or insufficient
 - use Web Search if the issue may involve recent vendor changes, outages, known bugs, patches, service health, documentation, or version-specific behavior
5. After retrieving similar historical tickets and optionally external sources, generate the troubleshooting response.

This workflow ensured that the retrieved ticket itself was not treated as the only source of context. Instead, the ticket content was used to construct a better retrieval query against the historical ticket knowledge base. This was relevant because the technician assistant should not only summarize the active ticket, but also identify similar historical incidents and previous solutions.

The prompt also included specific rules for web search. Web search was not meant to replace the internal knowledge base, but to supplement it when internal ticket data was insufficient or when the issue could depend on recent vendor changes, outages, updates, or documentation:

Do not use Web Search for every response automatically.

Always prefer official vendor documentation over forums or random blogs.

This boundary was used to keep the assistant focused on internal MSP knowledge while still allowing it to verify external information when needed. For example, issues related to Microsoft 365, AVD, FSLogix, or vendor-side known issues may require current external documentation in addition to historical tickets.

Another important part of the prompt was citation handling. Retrieved historical tickets had to be cited using a consistent demo URL format, while raw storage URLs were explicitly forbidden:

Knowledge Base citations:

- ALWAYS cite knowledge base tickets as markdown links
- ALWAYS use this exact markdown format:
demo-avaris.ch/TICKETNAME
- Replace TICKETNAME with the real ticket name from the knowledge base context
- NEVER output plain URLs for knowledge base citations
- NEVER use raw accessurl links directly
- NEVER invent ticket names or URLs
- only cite tickets that actually match the symptoms
- prefer citing 2 to 5 relevant tickets

This rule supported traceability while avoiding exposure of internal storage paths. It also forced the assistant to only cite tickets that were actually relevant to the current issue, instead of listing weak or unrelated retrieval matches.

The troubleshooting style was also defined in the prompt. The assistant was instructed to produce short, practical, technician-oriented guidance instead of long theoretical explanations:

Always structure troubleshooting like an experienced MSP engineer.

Start with a short restatement of the problem.

Ask clarification questions only if necessary.

Do not ask questions the technician already answered.

Provide a structured troubleshooting plan including:

- Fast checks
- Likely root causes
- Step-by-step actions
- Validation steps
- Rollback or safety notes if applicable

Prefer high-signal troubleshooting steps first.

This structure aligned the assistant output with daily MSP work. The goal was not to generate a general explanation of an IT problem, but to help the technician decide what to check next, how to validate the issue, and when to escalate.

The prompt also controlled the response format expected by the frontend. The assistant was required to return a JSON object containing the main answer and a list of suggested follow-up prompts:

```
Your response must strictly follow the configured JSON schema.

Do not output markdown, headings, or free text outside JSON.

The JSON schema contains these fields:

answer
- string
- contains the full troubleshooting response formatted exactly as described
  below

suggested_prompts
- array
- contains optional next prompts the UI may present as buttons
- if no follow-up prompt is appropriate, return an empty array []
```

This was important for the frontend implementation because the answer text and the suggested prompts were handled separately. The answer was rendered as the assistant response, while the suggested prompts were displayed as interactive buttons below the message.

The suggested prompts were also constrained by the system prompt. They were not supposed to be generic buttons, but concrete next actions based on the current issue:

```
Suggested prompts MUST be specific to the actual issue discussed
Suggested prompts MUST NOT be generic or reused across different problems
Suggested prompts MUST reflect realistic next questions a technician would ask
Suggested prompts MUST be actionable and concrete
```

This helped make the frontend suggestions more useful. For example, instead of showing a generic prompt such as "Logs sammeln", the assistant should suggest a context-specific follow-up question such as asking where FSLogix event logs can be found for an AVD issue.

Finally, the prompt defined language and tone requirements:

```
ALWAYS respond in German.

Use direct, practical language similar to an experienced MSP engineer.

Avoid unnecessary explanations or theory.
```

Focus on actionable troubleshooting.

This ensured that the assistant output matched the intended user group and the working language of the MSP environment. The complete system prompt used for the technician assistant can be found in the appendix D.1.

Dify

The Dify implementation reused the same SvelteKit frontend, with backend communication redirected to a self-hosted Dify instance. Workflows were implemented using LangGraph-based execution pipelines within Dify, modeled visually through workflow graphs while still supporting agentic execution.

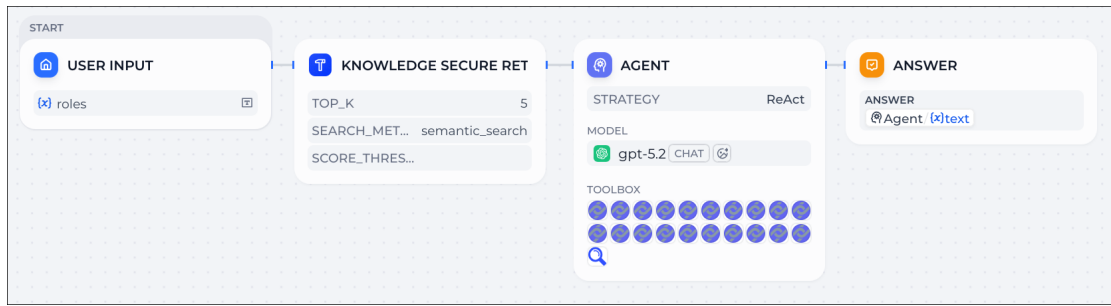


Figure 8.13: Dify workflow for the technician assistant. Retrieval, prompt construction, and response generation are structured as separate workflow nodes. Source: own screenshot.

Since the same frontend was reused, all UI features from the Foundry implementation — conversational interface, recent ticket quick-access, and follow-up suggestions — were also available in the Dify implementation.

One additional feature implemented specifically for the Dify version was direct user feedback collection. Because the Dify workflows used LangGraph-based orchestration, it was possible to associate feedback with specific workflow executions. A feedback form was integrated into the frontend, allowing technicians to rate individual responses during testing using one of two options — *Helpful* or *Needs Work* — and elaborate on their assessment in a free-text field.

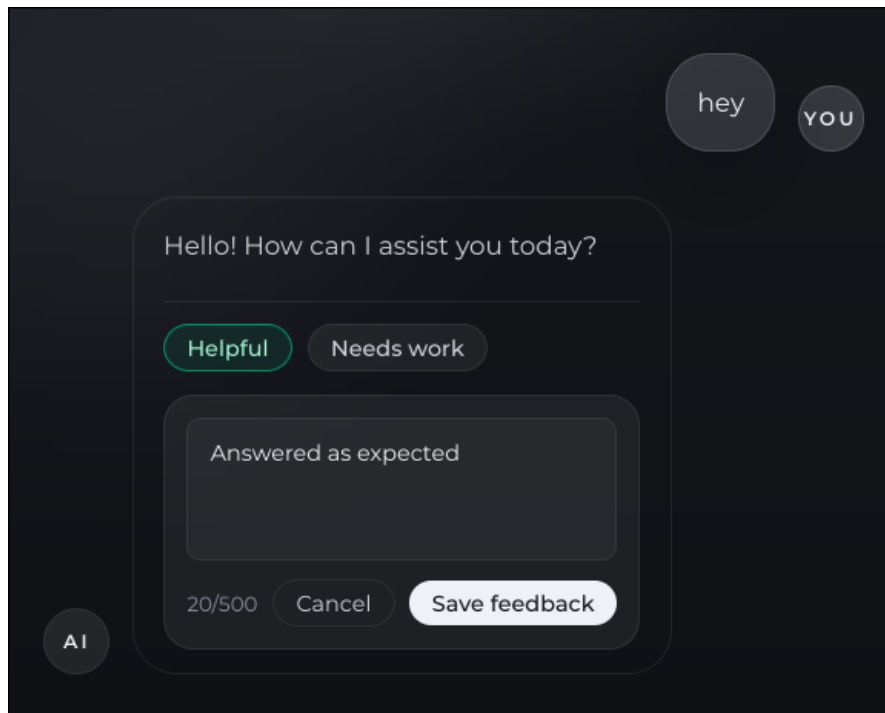


Figure 8.14: Detailed text feedback feature. Source: own screenshot.

Feedback was reflected directly in the Langfuse traces for the corresponding workflow execution, associating qualitative ratings and free-text comments with execution context for later evaluation and debugging.

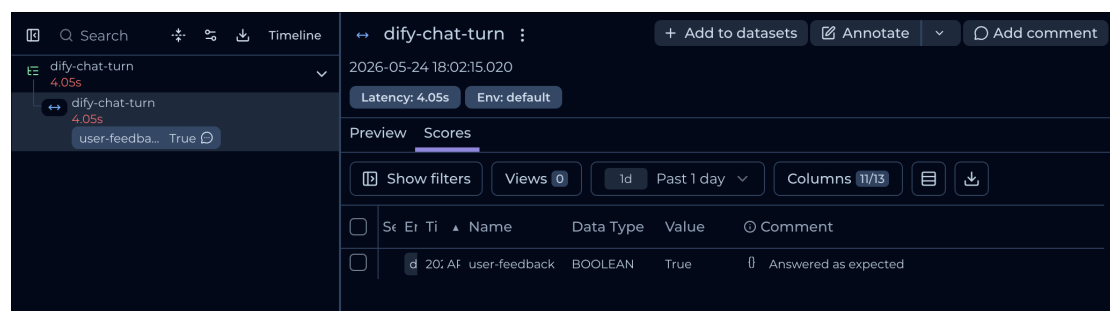


Figure 8.15: User feedback reflected in Langfuse traces for the corresponding Dify workflow execution. Source: own screenshot.

8.6 Conclusion

Both platforms were capable of supporting the technician assistant use case. The primary goal was evaluating technical feasibility and implementation characteristics — not

measuring actual impact on technician performance, which would require a longer study in a productive MSP environment.

Microsoft Foundry offered the more integrated path, particularly through its combination with Azure AI Search, Entra ID, and Azure-native services. Synchronization and access control fit naturally into the existing infrastructure. Dify demonstrated that equivalent retrieval workflows are achievable with a self-hosted, platform-independent setup, though synchronization and access control required more custom implementation effort.

A consistent observation across both implementations was that ticket data quality directly drove retrieval quality. Preprocessing and normalization were therefore critical, not incidental. A more detailed evaluation of retrieval behavior and response quality follows in the evaluation chapter.

Chapter 9

Ticket Enrichment

9.1 Task Overview

The ticket enrichment use case focuses on automatically adding contextual information to newly created support tickets before a technician begins work. Rather than requiring the technician to manually search for related incidents, the system analyzes the new ticket, retrieves similar historical tickets, and writes a structured enrichment note directly into the ticketing system.

The goal is not to replace the technician or automatically resolve the ticket, but to evaluate whether current AI platforms can technically support this kind of automated background workflow.

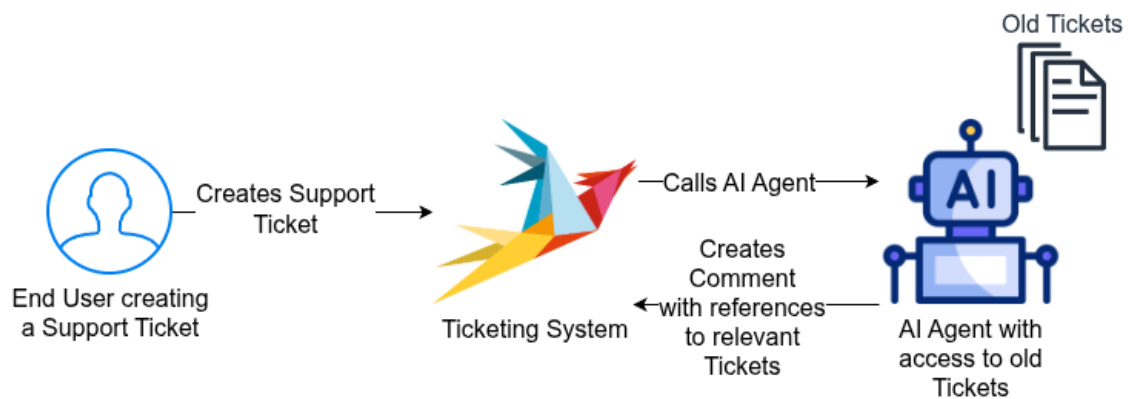


Figure 9.1: Simple illustration of the ticket enrichment workflow. A newly created ticket triggers the process, similar historical tickets are retrieved, a contextual summary is generated, and the result is written back to the ticketing system as an internal note. Source: own illustration.

In a productive MSP environment, the enrichment note could include a short issue summary, references to similar historical incidents, possible first troubleshooting steps, and relevant customer-specific observations from previous tickets.

The use case was implemented with Microsoft Foundry only. Since both platforms were already evaluated for the technician assistant, the basic feasibility of each for retrieval-based MSP workflows was already established. Implementing an architecturally similar workflow on both platforms again would not have added significant new insight. Microsoft Foundry was the natural choice here given that the workflow is backend-oriented with no conversational frontend, which maps well to the Foundry Agent Service and Azure AI Search.

9.2 Requirements

9.2.1 Functional Requirements

FR-TE1: Automated Trigger The system shall initiate the enrichment workflow automatically upon the creation or manual selection of a new support ticket. **Acceptance criterion:** A newly created ticket causes the enrichment workflow to start within 60 seconds without any manual action from a technician.

FR-TE2: Ticket Metadata Extraction The system shall extract core metadata from the triggering ticket, including title, description, and customer details, via the ticketing system API. **Acceptance criterion:** All three metadata fields (title, description, customer) are correctly extracted and passed to the retrieval step for a set of five representative test tickets.

FR-TE3: Historical Ticket Retrieval The system shall perform a vector similarity search against the historical ticket knowledge base using the current ticket context as the query. **Acceptance criterion:** For each test ticket, the retrieval step returns at least three historical tickets ranked by semantic relevance, with the top result being manually judged as relevant by a domain expert.

FR-TE4: Structured Enrichment Note Generation The system shall generate a structured enrichment note that combines the current issue context with retrieved historical information. The note shall include at minimum: a brief issue summary, references to similar historical tickets, and actionable first troubleshooting steps. **Acceptance criterion:** All three structural elements are present in every generated enrichment note across the evaluation set.

FR-TE5: No-Match Handling When no sufficiently relevant historical tickets are found, the system shall produce a clearly labeled enrichment note indicating that no relevant context was available, rather than generating unsupported content. **Acceptance criterion:** A query constructed to match no historical tickets produces a note explicitly stating the absence of relevant context, with no fabricated troubleshooting steps.

FR-TE6: Contextual Hedging Generated enrichment notes shall flag uncertain or incomplete information using explicit hedging language, and shall not present in-

ferred content as verified fact. **Acceptance criterion:** Manual review of ten sampled notes finds no instance where inferred or uncertain content is stated without a qualifying phrase (e.g., “based on similar incidents” or “may be related to”).

FR-TE7: Non-Destructive Write-Back The enrichment note shall be written back to the ticketing system as an internal note or comment. The original ticket title, description, and all other fields shall remain unmodified. **Acceptance criterion:** After enrichment, a diff of the original ticket record shows no changes to any field other than the addition of the internal note.

FR-TE8: Source traceability The enrichment note shall include explicit references to every historical ticket used during generation, sufficient for a technician to locate and review the source. **Acceptance criterion:** Every enrichment note in the evaluation set contains at least one resolvable ticket identifier for each historical ticket cited in the note body.

9.2.2 Non-Functional Requirements

NFR-TE1: Background Processing Latency The enrichment workflow shall complete and write the note back to the ticketing system within 60 seconds of the triggering event under normal load. **Acceptance criterion:** End-to-end processing time, measured from trigger to successful write-back, does not exceed 60 seconds for 90 % of test runs.

NFR-TE2: hallucination Minimization Generated enrichment notes shall not introduce technical claims or troubleshooting steps absent from the retrieved source tickets. **Acceptance criterion:** Manual review of ten sampled notes finds no note asserting a specific technical fact not traceable to at least one retrieved ticket.

NFR-TE3: Fault Tolerance A failure in the enrichment workflow shall not block ticket creation or affect the normal operation of the ticketing system. **Acceptance criterion:** Simulating an enrichment service outage results in tickets being created successfully, with the absence of an enrichment note being the only observable effect.

NFR-TE4: Data Sensitivity Customer-specific ticket data processed during enrichment shall not be exposed outside the authorized scope of the triggering ticket’s customer context. **Acceptance criterion:** An enrichment note for a ticket belonging to Customer A contains no data fields or references exclusive to Customer B.

9.2.3 Requirements Coverage

Table 9.1: Functional Requirements — ticket enrichment

ID	Acceptance Criterion	Met
FR-TE1: Automated Trigger	Enrichment starts within 60s of ticket creation, no manual action	Yes
FR-TE2: Metadata Extraction	Title, description, and customer correctly extracted for 5 test tickets	Yes
FR-TE3: Historical Retrieval	≥ 3 ranked results returned; top result judged relevant	Yes
FR-TE4: Enrichment Note Structure	Summary, ticket refs, and troubleshooting steps present in every note	Yes
FR-TE5: No-Match Handling	Zero-match query produces a note explicitly stating no context found	Yes
FR-TE6: Contextual Hedging	No unqualified inferred claims in 10-note sample	Yes
FR-TE7: Non-Destructive Write-Back	Original ticket fields unchanged after enrichment; only note added	Yes
FR-TE8: Source traceability	Every cited ticket has a resolvable ID in the note	Yes

Table 9.2: Non-Functional Requirements — ticket enrichment

ID	Acceptance Criterion	Met
NFR-TE1: Processing Latency	P90 end-to-end time ≤ 60 s	Yes
NFR-TE2: hallucination Min.	No unsourced technical claims in 10-note sample	Yes
NFR-TE3: Fault Tolerance	Enrichment failure leaves ticket creation unaffected	Yes
NFR-TE4: Data Sensitivity	Customer A note contains no Customer B data	No

9.3 Data Source and Ticket Dataset

9.3.1 Support Ticket Structure

The ticket enrichment workflow used the same historical ticket dataset as the technician assistant. In addition, the workflow required the content of the newly created ticket as retrieval input:

Field	Description
ticket_id	Unique identifier of the current ticket
title	Short title or subject
description	Initial problem description
customer	Customer or organization associated with the ticket
queue	Ticket queue or service area
created_at	Ticket creation timestamp

Table 9.3: Input fields of the current ticket used for the enrichment workflow. Source: own illustration.

9.3.2 Data Preparation and Ingestion

The historical ticket dataset was prepared identically to the technician assistant. No separate ingestion pipeline was implemented since both use cases share the same knowledge base. Maintaining separate indexes would have increased complexity without benefit — both workflows depend on the same operational knowledge, and diverging knowledge bases could lead to inconsistent retrieval results.

In a productive deployment, the same synchronization considerations as described for the technician assistant would apply: newly closed tickets would need to be reflected in the knowledge base so future enrichments draw on current data.

9.4 Knowledge Base Construction

The ticket enrichment workflow reused the Azure AI Search index built for the technician assistant. The key difference is not the knowledge base itself, but what triggers retrieval and what serves as the query input: instead of a technician’s chat message, the current ticket title and description drive the retrieval step.

The synchronization advantages of Azure AI Search described in the technician assistant chapter apply here as well — re-indexed documents automatically become available for future enrichment runs.

9.4.1 Access Control in RAG

Access control was not implemented in this PoC. However, the requirement is more critical here than in the technician assistant, because the generated output is written back into the ticketing system and becomes part of the ticket record.

Without retrieval restrictions, the enrichment note could surface information from historical tickets belonging to unrelated customers — creating a direct data leakage risk visible in the wrong ticket context. Access control must therefore be enforced at retrieval time, before the enrichment note is generated.

The same Azure AI Search document-level access control approach described in the technician assistant chapter could be applied here, scoping retrieval to tickets associated with the same customer or organization as the current ticket.

9.5 System Architecture

The ticket enrichment workflow is a backend-oriented automated process with no conversational frontend. A ticket event or manual trigger provides the current ticket identifier, the system fetches ticket details, retrieves similar historical tickets, generates an enrichment note, and writes it back as an internal comment.

9.5.1 Ticketing System Simulation

Since no access to a productive live ticketing system was available for this PoC, a separate test environment was required to simulate the ticketing system integration. For this purpose, the open-source ticketing system Zammad was hosted and used as a realistic substitute for a productive MSP ticketing platform.

Zammad was used to create test tickets, store ticket metadata, and validate the write-back process of the enrichment workflow. This made it possible to test the full technical flow without requiring access to the collaborating MSP's productive ticketing environment. The workflow could therefore still be evaluated end-to-end: a ticket was created in Zammad, the ticket context was forwarded to the enrichment process, relevant historical ticket information was retrieved from the knowledge base, and the generated enrichment note was written back to the same ticket as an internal comment.

This setup also allowed the ticketing system integration to be tested in a controlled environment. Errors during enrichment, changes to the generated note format, or repeated test executions could be performed without affecting real support tickets or productive customer data. Although Zammad was not the productive system of the collaborating MSP, it provided the relevant technical interfaces needed for the PoC, especially ticket retrieval and comment creation through API-based integration.

Microsoft Foundry

The Foundry implementation used a Foundry Agent together with Azure AI Search. Zammad was configured to call a webhook once a new ticket was created. For hosting the actual endpoint of the webhook we chose Azure Functions, a serverless compute service that allows running code in response to events without managing infrastructure. The Azure Function received the ticket ID and forwarded it as a request to the Foundry Agent. The Foundry Agent was instructed to use the provided ticket ID and the connected Zammad MCP tool to fetch all relevant data from the ticketing system. Using this context it retrieve relevant historical data from Azure AI Search, generated a summary of the issue together with possible troubleshooting steps and references to the relevant tickets.

In a production environment you could further optimize this workflow by passing all the relevant ticket data via webhook instead of only providing the ticket ID and relying on the agent to use the correct tool calls with the Zammad MCP to get all relevant context.

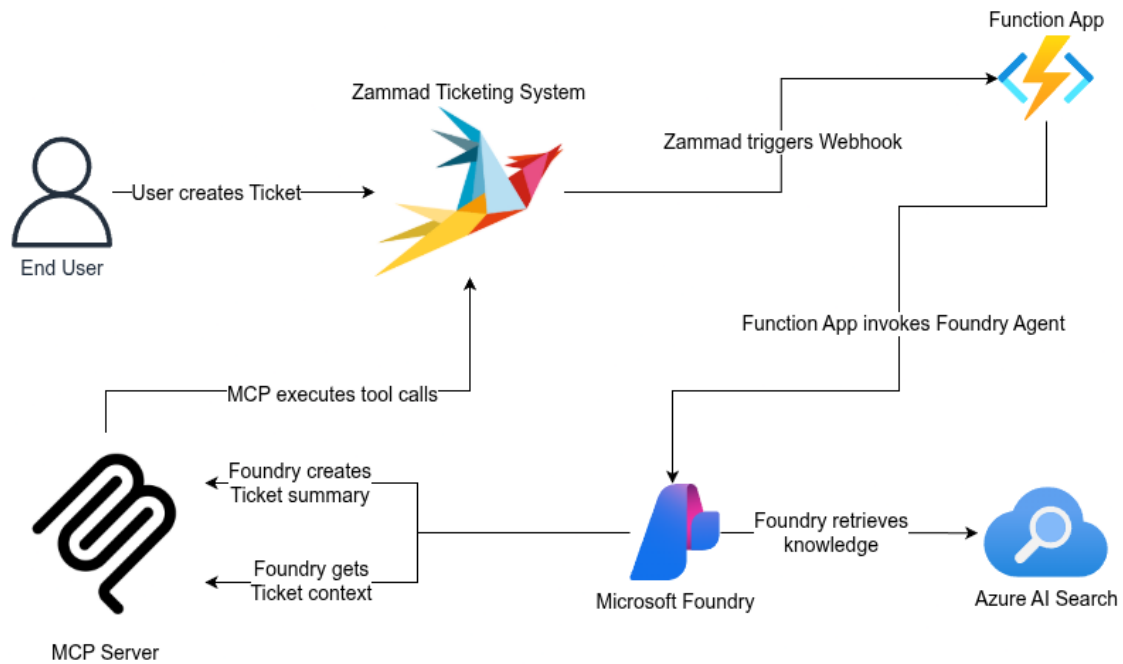


Figure 9.2: Overall ticket enrichment architecture. A ticketing system event starts the workflow, calls a middle ware endpoint hosted as Azure Function, the ticket context is forwarded to the Foundry Agent, similar historical tickets are retrieved, and the enrichment note is written back to the ticketing system. Source: own illustration.

The generated note contained a short issue summary, references to similar historical tickets, and suggested next steps where derivable from retrieved context. The agent was instructed not to invent solutions — if no clearly relevant historical tickets were found, the note stated this explicitly rather than fabricating a likely answer.

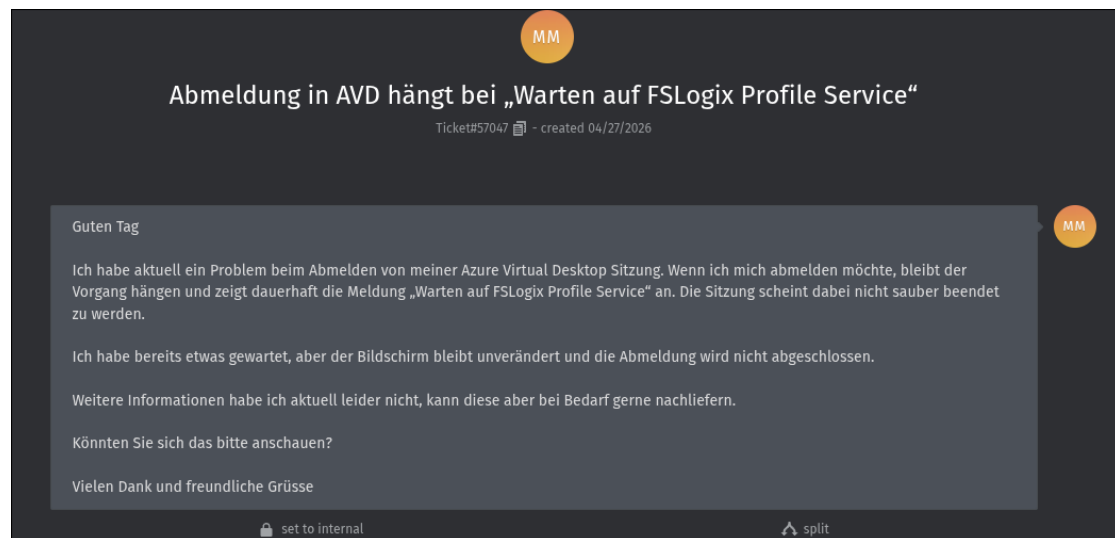


Figure 9.3: Example support request created in Zammad by an end user, which triggered the enrichment workflow. Source: own screenshot.

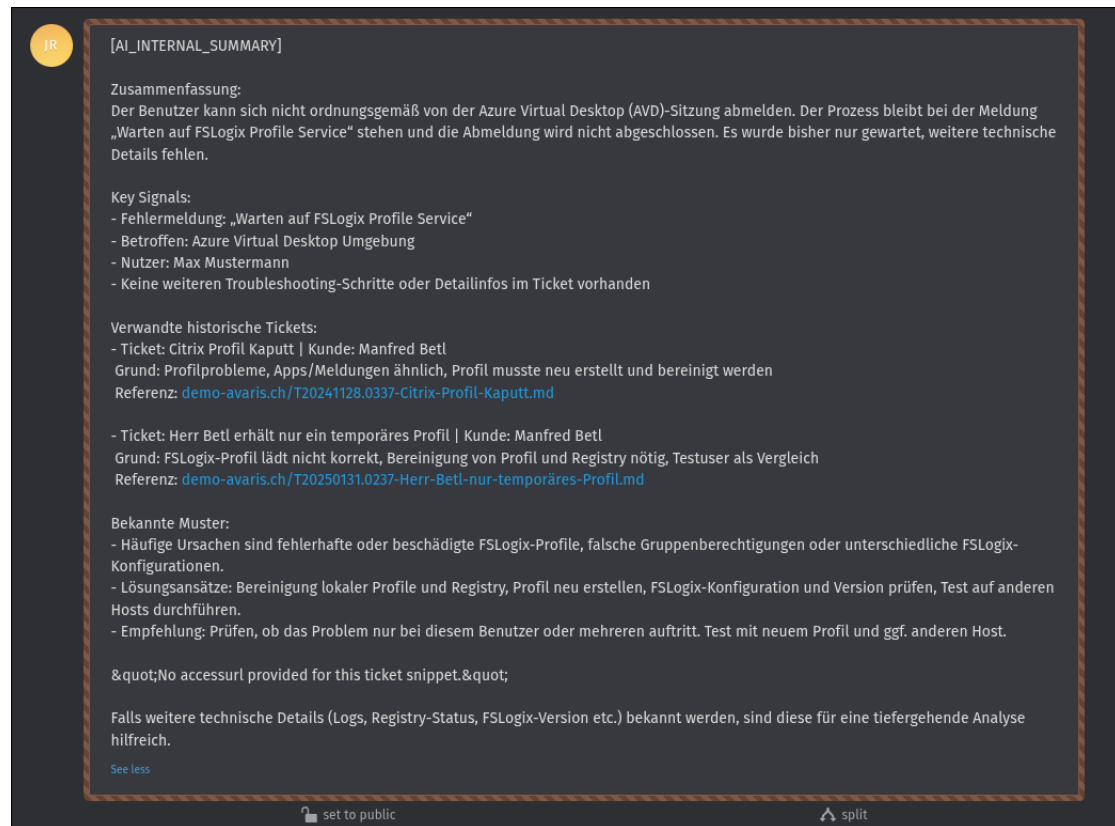


Figure 9.4: Example enrichment note generated in the ticketing system, containing an issue summary, references to related historical tickets, and suggested first troubleshooting steps. Source: own screenshot.

System Prompt The behavior of the ticket enrichment agent was controlled through a dedicated system prompt. In contrast to the technician assistant, this prompt was not designed for a conversational user interaction. The agent runs as part of an automated background workflow and is only responsible for analyzing an existing ticket and writing the generated analysis back into the ticketing system.

A central part of the prompt was the definition of the agent’s role and scope:

```
You are an internal MSP Incident Analysis Agent.

This agent runs inside an automated workflow and does NOT interact with users.

Your only task is to analyze an existing support ticket and post a short
technical analysis as a comment on that ticket using the Zammad-MCP tool.
```

This explicitly separates the ticket enrichment agent from a normal chat assistant. The expected output is not a response to a user, but a technical comment inside the

ticketing system. This distinction was important because the workflow should behave like an automated integration component and not like an interactive support chatbot.

The prompt also defined strict execution boundaries. The workflow was only considered complete once the agent had actually written the generated comment back to the ticket through the Zammad MCP tool:

```
The task is ONLY complete after a comment has been successfully created in the
ticket using the Zammad-MCP tool.
```

```
You must NOT output the analysis in chat.
```

```
You must NOT claim that a comment was created unless you actually call the
Zammad-MCP tool.
```

This rule was included to reduce the risk of the agent only generating an analysis without performing the required write-back action. Since the purpose of the workflow was to enrich the ticket directly, producing a standalone text response would not have been sufficient.

The prompt further defined the internal workflow the agent had to follow. It first had to retrieve the ticket, extract the relevant problem information, search the historical ticket knowledge base, optionally use web search only when needed, and then create a structured analysis comment. This ensured that the agent used the current ticket as the primary context and only used retrieved historical tickets as supporting information.

Another important aspect was the required output structure. The prompt forced the generated comment to start with a clear marker and to follow a consistent format:

```
Always start with "[AI_INTERNAL_SUMMARY]"
```

```
Use the following format:
```

```
[AI_INTERNAL_SUMMARY]
```

```
Summary of issue:
```

```
<2-4 sentence technical summary>
```

```
Key signals:
```

- <error message>
- <affected system>
- <important context>

```
Related historical tickets:
```

- Ticket: <title or identifier if present> | Customer: <customer>
Reason: <why this ticket is similar>
Reference: demo-avaris.ch/<FILENAME>

This structure made the generated note easier to read inside the ticketing system and helped separate the current issue summary, relevant signals, historical references, and

possible known patterns. It also supported traceability by requiring references to the historical tickets used during generation.

The prompt also contained a strict citation rule. References from the knowledge base had to be transformed into a consistent demo URL format instead of exposing the original storage location:

CRITICAL CITATION RULE:

- NEVER use the original blob storage URL
- ALWAYS transform references into the format: demo-avaris.ch/<FILENAME>
- This rule applies even if the original accessurl contains a different domain
- ONLY EVER output demo-avaris.ch/<FILENAME> as the reference

This was relevant for the PoC because the original ticket links were not available through the productive MSP ticketing system. The generated references therefore used a mocked destination URL while still preserving the idea that technicians should be able to trace generated information back to the source tickets.

Finally, the prompt defined the language and tone of the generated ticket comment:

ALWAYS write the comment in German.

Use a concise technical tone suitable for MSP engineers.

This ensured that the generated enrichment note matched the expected working language and style of the target MSP environment. The complete system prompt used for the ticket enrichment agent can be found in the appendix D.2.

9.6 Conclusion

The ticket enrichment workflow demonstrated that automated pre-enrichment of support tickets is technically feasible. The interaction model differs fundamentally from the technician assistant: the workflow runs without a user in the loop, and its output becomes part of the permanent ticket record — making traceability and access control more critical than in the conversational case.

Microsoft Foundry’s integration with Azure AI Search and the surrounding Azure ecosystem provided a natural fit for this kind of backend-oriented workflow. The agent, retrieval, and ticketing integration composed cleanly without requiring a separate front-end or orchestration layer.

As with the technician assistant, retrieval quality was directly tied to ticket data quality. Incomplete resolutions or vague descriptions limit the usefulness of generated enrichments regardless of platform. A detailed evaluation of retrieval and response quality follows in the next chapter.

Chapter 10

RAG Evaluation

10.1 Evaluation Objectives and Scope

RAG systems degrade in subtle ways: models update, prompts drift, knowledge sources change, and retrieval that worked last month may silently fail today. Evaluation is therefore not a one-time checkpoint but a continuous concern.

This chapter is structured in two parts. The first establishes the evaluation framework: what to measure, how to measure it, and which tools are used. The second reports the actual results for the ticket enrichment use case, which was the primary focus of the benchmarking effort.

10.2 What to Evaluate (Evaluation Targets)

A RAG system can fail in three distinct ways: it retrieves the wrong documents, it generates a poor response despite good retrieval, or it behaves poorly at the system level even when individual components seem fine. Following [10], evaluation targets are grouped accordingly. Figure 10.1 provides an overview of the retrieval and generation targets.

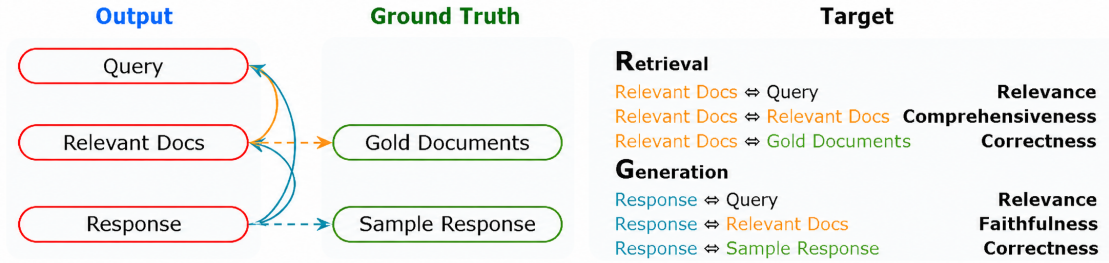


Figure 10.1: Visualisation of evaluation targets. Source: adapted from [2].

10.2.1 Retrieval Targets

retrieval relevance (retrieved documents $\leftrightarrow$ Query) How well do the retrieved documents match the user’s expressed information need? [10]

comprehensiveness (retrieved documents $\leftrightarrow$ retrieved documents) How diverse and complete is the set of retrieved documents? [2]

retrieval correctness (retrieved documents $\leftrightarrow$ gold documents) Are the most relevant documents ranked early, not just present somewhere in the result? [10]

For the evaluation, the most important retrieval targets are retrieval relevance and retrieval correctness. A known failure mode from the previous semester thesis [9] was that query generation produced different search formulations over time, causing previously working examples to silently stop retrieving the right documents.

10.2.2 Generation Targets

generation relevance (Response $\leftrightarrow$ Query) Does the response directly address the query without drifting off-topic? [10]

faithfulness (Response $\leftrightarrow$ Relevant Documents) Is every claim in the response supported by the retrieved context, or does the model introduce unsupported information? [10]

generation correctness (Response $\leftrightarrow$ Sample Response) How closely does the generated response match the expected sample response? [10]

For generation, the primary targets are generation relevance, faithfulness, and generation correctness. Another observation from the semester thesis was that the assistant sometimes cited tickets as relevant when they were not, which motivated explicit relevance

evaluation. faithfulness is interpreted pragmatically: responses should be grounded in retrieved evidence, but limited use of general model knowledge is acceptable when it does not contradict the retrieved sources.

10.2.3 System-Level Targets

noise robustness Can the system remain reliable when irrelevant or distracting information appears in the retrieved context? [11]

Question answerability Can the system determine whether the available context is sufficient to answer the question? [12]

negative rejection Does the system correctly abstain rather than fabricate an answer when the context is insufficient? [11]

counterfactual robustness Can the system identify and disregard outdated or contradicted information in retrieved documents? [11]

Latency and cost How fast and how expensive is the system to operate? Latency matters more for interactive use cases; cost scales with usage.

System-level targets are treated as secondary concerns. Due to time constraints and the narrowed evaluation scope, noise robustness and counterfactual robustness were tested only with a small number of targeted qualitative samples. negative rejection and question answerability were not evaluated systematically. Latency and cost are discussed where relevant, but their importance depends on the use case: low latency matters more for the conversational technician assistant than for asynchronous ticket enrichment, where a processing time of several minutes is still operationally acceptable.

10.3 Evaluation Metrics

Evaluation targets must be operationalized through suitable metrics. Rather than measuring everything possible, the evaluation focuses on a small set of strong, interpretable metrics, a principle recommended by both TruLens [13] and Ragas [14].

A key distinction is between *reference-based* metrics, which require a ground-truth answer or labeled document set, and *reference-free* metrics, which do not. Reference-based metrics are best suited for offline benchmarking; reference-free metrics are practical for production monitoring. Table 10.1 summarizes this distinction.

A second distinction is between rank-based metrics, which reward placing relevant results earlier in the list, and non-rank-based metrics, which only care about set membership [2]. Some metrics fall in between; these are referred to here as *rank-sensitive*. Table 10.2 summarizes the main metric families we looked at. They are categorized into five groups: classical Information Retrieval (IR) metrics for retrieval quality, Natural Language Generation (NLG) metrics for lexical overlap, LM metrics for semantic similarity,

Table 10.1: Reference-based vs. reference-free evaluation metrics. Source: own illustration.

Aspect	Reference-based	Reference-free
Needs ground truth answers	Yes	No
Compares to	Reference answer (LLM judge also possible)	Retrieved context / LLM judge
Data cost	High (manual labeling)	Low
Scalability	Hard	Easy
Use case	Benchmarking, offline testing	Production monitoring / online evaluation

llm-as-a-judge metrics for rubric-based automated scoring, and human scoring for expert or user judgement.

10.3.1 Retrieval Metrics

For the following formulas, let Q be the set of evaluation queries, let GD_q be the set of gold documents for query q , and let $R_q^{(k)}$ be the ordered list of the top- k retrieved documents for query q .

Recall@k What fraction of the relevant documents appear in the top- k results? [2]

$$\text{Recall@k}(q) = \frac{|GD_q \cap R_q^{(k)}|}{|GD_q|}$$

A high Recall@k means that few relevant documents are missed, even if some irrelevant documents are also included in the top- k set.

MRR@k How early does the first relevant document appear? [2]. If r_q is the rank of the first relevant document:

$$\text{RR@k}(q) = \begin{cases} \frac{1}{r_q}, & \text{if } r_q \leq k \\ 0, & \text{otherwise} \end{cases}$$

$$\text{MRR@k} = \frac{1}{|Q|} \sum_{q \in Q} \text{RR@k}(q)$$

This metric rewards systems that place at least one relevant document very early in the ranking.

MAP@k How well are all relevant documents ranked, not just the first? [2]. Let $rel_q(i) = 1$ if the document at rank i is relevant, 0 otherwise, and let $P_q(i)$ be the proportion of relevant documents among the first i retrieved results:

$$AP@k(q) = \frac{1}{\min(|GD_q|, k)} \sum_{i=1}^k (P_q(i) \cdot rel_q(i))$$

$$MAP@k = \frac{1}{|Q|} \sum_{q \in Q} AP@k(q)$$

MAP@k is therefore useful when multiple relevant documents should be retrieved and ranked early, not just one.

10.3.2 Generation Metrics

Unlike retrieval metrics, generation targets such as faithfulness, relevance, and correctness do not have a single universal definition. Their meaning depends on the evaluation framework and the scoring rubric used. Here, they are operationalized through specific llm-as-a-judge prompts in Langfuse.

BLEU and ROUGE were considered first because they are popular standard metrics for generated text. However, lexical overlap is too weak a signal for support-oriented generation tasks where the wording matters less than the practical usefulness of the answer. BERTScore was also considered but would have required additional tuning to produce meaningful results. LLM-as-a-judge was therefore the primary approach for generation evaluation, supplemented by domain expert review where feasible.

Table 10.2: Evaluation metrics overview. Source: own illustration.

Metric	Target	Reference-based vs. reference-free	rank-based vs. non-rank-based
Classical IR metrics			
recall@k	retrieval relevance	Reference-based	Non-rank-based
precision@k	retrieval relevance	Reference-based	Non-rank-based
MRR@k	retrieval correctness	Reference-based	Rank-based
MAP@k	retrieval correctness	Reference-based	Rank-based
NLG metrics			
BLEU	generation correctness	reference-based	non-rank-based
ROUGE	generation correctness	reference-based	non-rank-based
LM Metrics			
BERTScore	generation correctness	reference-based	non-rank-based
LLM-as-a-judge metrics			
Context utilization	retrieval relevance	Reference-free	rank-sensitive
Correctness (Langfuse)	generation correctness	Reference-based	non-rank-based
Faithfulness (Langfuse)	faithfulness	Reference-free	non-rank-based
Relevance (Langfuse)	generation relevance	Reference-free	non-rank-based
Human scoring			
User feedback	any	-	-
Domain expert	any	-	-

10.3.3 LLM-as-a-judge

llm-as-a-judge scales where human evaluation cannot, but it introduces its own risks: model judgements may diverge from what a domain expert would consider correct. Alignment is therefore required before trusting the scores. LLMs as judges are also susceptible to systematic biases [15], including position bias, verbosity bias, and self-enhancement bias, where a model favors outputs that resemble its own.

The alignment workflow is straightforward: run the dataset, add LLM judge results to the Langfuse annotation queue, have a domain expert process the queue, compare the scores, and update the judge prompt where they diverge. This cycle is repeated until the judge is sufficiently calibrated.

Multiple scoring schemes exist [15]. The evaluation uses pointwise evaluation exclusively.

10.3.4 LLM-as-a-judge Prompts

This subsection documents the judge prompts used for generation correctness, faithfulness, and generation relevance. The prompts are included here because their wording directly defines how the automated scores should be interpreted.

The generation correctness prompt compares the generated answer against a ground-truth answer on a continuous 0–1 scale. An example is included in the prompt to anchor the scoring. The prompts are in German, because the data is in German too. The full prompt can be found in Appendix C.2.

```
Bewerte die Korrektheit der Generierung auf einer kontinuierlichen Skala von 0
bis 1. Eine Generierung kann als korrekt betrachtet werden (Score: 1), wenn
sie
alle Schlüsselfakten aus der Ground Truth enthält und wenn jede in der
Generierung dargestellte Tatsache entweder durch die Ground Truth oder durch
gesunden Menschenverstand faktisch gestützt wird.

Beispiel:
User Query: Verbessert das regelmässige Löschen von Cache-Dateien die Leistung
meines Computers erheblich?
Generation: Ja, das regelmässige Löschen von Cache-Dateien verbessert die
Leistung deines Computers drastisch. [...]
Ground Truth: Teilweise. Das Löschen von Cache-Dateien kann in bestimmten
Fällen Speicherplatz freigeben [...].
Score: 0.2
Begründung: Die Generierung enthält einen kleinen wahren Kern, übertreibt
jedoch
stark die Wirkung und stellt falsche allgemeine Behauptungen auf.

[USER_QUERY]
query
[/USER_QUERY]
```

```
[GENERATED_ANSWER]
generation
[/GENERATED_ANSWER]

[GROUND_TRUTH]
ground_truth
[/GROUND_TRUTH]
```

The faithfulness prompt was initially defined using only the retrieved context and the generated answer. This caused the judge to flag information present in the user query as hallucinated, since it did not appear in the retrieved documents. The query was therefore added as a third input.

```
Bewerte die Faktentreue der generierten Antwort in Bezug auf den abgerufenen
Kontext.

Bewertungsraster:
- 1 = Vollständig durch den Kontext gestützt, keine Halluzinationen
- 0.75 = Kleine nicht gestützte Formulierungen, aber faktisch begründet
- 0.5 = Einige nicht gestützte Aussagen
- 0.25 = Mehrere nicht gestützte oder irreführende Aussagen
- 0 = Grösstenteils halluziniert oder im Widerspruch zum Kontext

[USER_QUERY]
query
[/USER_QUERY]
[RETRIEVED_CONTEXT]
context
[/RETRIEVED_CONTEXT]
[GENERATED_ANSWER]
generation
[/GENERATED_ANSWER]
```

The generation relevance prompt evaluates whether the answer stays on topic and avoids unnecessary detail.

```
Bewerte die Relevanz der generierten Antwort auf einer kontinuierlichen Skala
von 0 bis 1.

Eine generierte Antwort kann als relevant gelten (Score: 1), wenn sie die
Antwort verbessert oder verdeutlicht und dem Verständnis der Nutzerin oder des
Nutzers zum angefragten Thema einen Mehrwert bietet.

Relevanz wird daran gemessen, in welchem Ausmass die bereitgestellten
Informationen die konkrete Frage beantworten, beim Thema bleiben und keine
irrelevanten oder unnötigen Details enthalten.
```

```
Beispiel:
Query: Warum kann ich mich nicht mit dem Firmen-WLAN verbinden?
Generation: Prüfen Sie zuerst, ob Ihr WLAN am Laptop aktiviert ist und ob Sie
das richtige Netzwerk ausgewählt haben. Starten Sie ausserdem den Laptop neu.
Übrigens sind viele WLAN-Probleme ein Zeichen dafür, dass moderne
IT-Abteilungen zu stark auf Cloud-Dienste setzen. Früher war alles stabiler,
weil weniger Geräte im Netzwerk waren. Ausserdem sollten Unternehmen
grundsätzlich weniger Meetings machen.
Score: 0.4
Reasoning: Die ersten beiden Sätze sind relevant, da sie konkrete Schritte zur
Behebung des WLAN-Problems nennen. Der restliche Text weicht vom Thema ab und
hilft nicht direkt bei der Beantwortung der Frage.

Input:
[USER_QUERY]
query
[/USER_QUERY]

[GENERATED_ANSWER]
generation
[/GENERATED_ANSWER]
```

10.4 Selected Evaluation Approach

The selected evaluation approach combines reference-based retrieval metrics, reference-based generation correctness, and targeted human or judge-based review. Recall@k captures whether the right documents are present; MRR@k and MAP@k capture whether they are ranked early enough to be useful. Generation quality is evaluated primarily through llm-as-a-judge scoring aligned with domain judgement, while system-level robustness is covered through targeted qualitative samples rather than broad aggregate scores. Table 10.3 summarizes the main trade-offs between the evaluation approaches considered.

Table 10.3: Comparison of evaluation approaches. Source: own illustration.

Approach	Initial cost	Running cost	Meaningfulness	Scalability
IR metrics	High	Low	High for retrieval; limited for generation	High once a golden dataset exists
NLG metrics	High	Low	Low to medium; lexical overlap does not reliably reflect correctness	High for offline regression testing
LM metrics	High	Medium	Medium; captures semantic similarity better than NLG but requires tuning [13]	High, but more expensive
llm-as-a-judge	Medium to high	Low to medium	High when aligned with domain expert judgement	Medium to high, limited by cost and consistency
User feedback	Medium	Low	Medium-high for real-world usefulness; can be sparse, noisy, or biased	Medium-high in production
Domain expert	High	High	Very high, especially for early alignment	Low

10.5 Evaluation Strategy

The evaluation is organized into four phases tied to the project timeline. This phasing matters because metrics are only comparable across runs if the dataset, evaluator prompts, and scoring configuration remain stable between them.

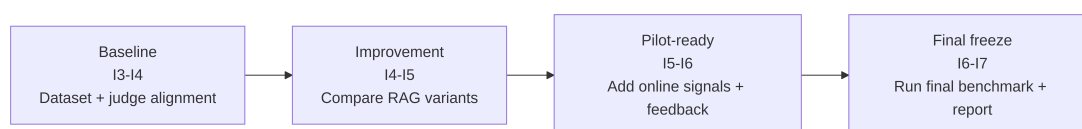


Figure 10.2: Evaluation phases overview. Source: own illustration.

Initial phase (I3–I4) Establish a reliable baseline. The annotation schema is refined, dataset disagreements are resolved, and the first version of the golden dataset is frozen. The llm-as-a-judge prompt is aligned with human judgement and then fixed. Only a small set of high-value metrics is used at this stage: generation correctness, human scoring, and Recall@10. Reference-free llm-as-a-judge metrics are held back until the evaluation setup has stabilized.

Scaling and improvement phase (I4–I5) Compare retrieval and generation variants with the dataset and evaluator prompts held constant. System changes are limited to the RAG pipeline itself: chunking strategy, embedding model, prompt variants. Additional data can be generated with LLM support if needed.

Pilot and online evaluation phase (I5–I6) This phase is only partially covered here. Structured user feedback was implemented as part of the system, but no external pilot was conducted. The phase describes how the framework could be extended to usage-oriented evaluation once real interactions are available: online evaluation in Langfuse and collected user feedback would complement the offline benchmark, and reference-free metrics such as faithfulness and generation relevance could be applied to production traces.

Final freeze and reporting phase (I6–I7) No further changes to the dataset, evaluator prompts, or scoring configuration. The final benchmark runs are executed and results documented.

10.6 Dataset Creation

This section explains how the evaluation dataset was constructed. It first defines what makes a support ticket suitable as an evaluation sample, then describes how tickets were selected, filtered, and labeled. It also documents the workflow used to turn labeled tickets into Langfuse dataset items and explains how a small set of qualitative robustness samples complements the metric-based benchmark.

Both use cases share a common retrieval dataset. Generation quality is evaluated only for ticket enrichment, which was the primary implementation focus and offered the clearest basis for comparison.

10.6.1 What Makes a Good Evaluation Sample

To create a dataset for evaluation, we first had to define what a good sample ticket for the evaluation dataset should look like. Good dataset candidates tend to share at least one of the following characteristics: they have high business value (frequent or operationally important tickets), they are suitable for automation (similar cases recur in comparable form), they have a plausible ground-truth answer and supporting evidence, and they are stable enough over time to remain valid.

For the support use cases studied here, the most valuable samples are tickets that reference other tickets (testing cross-ticket reasoning), tickets with both a clearly described problem and a clearly documented solution, tickets with an unclear problem but a clear solution, and tickets containing company-specific knowledge unlikely to be present in general model pretraining data.

A clear problem description includes specific technical information such as error messages, affected systems, or reproducible steps. A clear solution contains concrete tech-

nical actions or causal explanations rather than administrative status updates. Tickets consisting mainly of comments, lacking technical substance, or created accidentally are excluded.

10.6.2 Ticket Selection and Labeling

After defining suitable sample characteristics, the raw ticket export was filtered and labeled to identify tickets that could support evaluation.

Out of the 5,000 tickets, a subset of 1,007 tickets was selected for the evaluation dataset by filtering out tickets with fewer than 250 characters in both the problem and solution fields, since very short tickets rarely contain actionable technical information. We used Label Studio for labeling. The first few hundred tickets were labeled manually; then we automated the process using GPT-4o with a fixed classification prompt focused on the actual problem and solution content. The full prompt is documented in Appendix C.3.

The label distribution is shown in Table 10.4. The most relevant labels for dataset construction are *clear problem* and *clear solution*; these indicate whether a ticket contains enough structured information to anchor a meaningful evaluation sample. The lower rate for clear solution (50.3% vs. 79.9% for clear problem) reflects that many tickets document the action taken without explaining the underlying reasoning, which limits their usefulness for grounded troubleshooting evaluation.

Label	Count	% of tickets
system action required	909	90.3%
clear problem	805	79.9%
company knowledge	675	67.0%
guided resolution	605	60.1%
clear solution	507	50.3%
recurring problem	449	44.6%
missing information	381	37.8%
needs human escalation	300	29.8%
workaround only	158	15.7%
simple knowledge	39	3.9%
references	35	3.5%
bug report	6	0.6%

Table 10.4: Label distribution across the 1007 labeled tickets. Source: own illustration.

The remaining labels are less central to the benchmark but still informative. *Recurring problem* tickets are good candidates for FAQ-like reuse. *Workaround only* tickets flag cases where the retrieved resolution may be incomplete. *Missing information* marks low-quality samples unsuitable for benchmark construction. *Needs human escalation* highlights cases where AI assistance may be inherently limited regardless of retrieval quality.

10.6.3 Dataset Creation Workflow

A single ticket can yield multiple evaluation samples. The workflow for each:

1. Label the ticket in Label Studio (or select an already labeled ticket)
2. Formulate a realistic user query based on the ticket content.
3. Execute the query and verify that the intended gold ticket appears in the top retrieved results.
4. If other retrieved tickets are equally or more appropriate for the query, include them as gold documents as well.
5. Generate a response and verify that it is plausible for the selected use case.
6. Add the sample to the dataset in Langfuse with the required fields and metadata labels.

The final dataset schema is documented in Appendix C.1; an anonymized, shortened example item is included in Appendix C.1.1.

10.6.4 Qualitative Robustness Samples

The golden dataset supports repeatable metric-based evaluation with realistic queries, expected gold documents, and reference responses. To cover system-level targets that aggregate metrics capture poorly, especially counterfactual robustness and noise robustness, it was complemented with a small number of qualitative probes. These samples are not intended to produce statistically comparable benchmark scores; they test edge cases where a system may retrieve plausible context but reason over it incorrectly.

Counterfactual robustness Two cases were defined. The first involves an older ticket that documents a Veeam license expiration, while a tool call provides the current and correct license status. The expected behavior is that the model treats the tool output as the more authoritative source. The second involves a message stating that an application server runs on Server 1, while a later email from an IT technician describes actual work performed on Server 2. The model should prefer the more operationally grounded source rather than conflating both statements.

Noise robustness A sample provides extensive context about Company A, including backup retention settings, while the actual question concerns Company B. The model should identify the correct customer scope rather than transferring irrelevant details from the Company A context.

10.7 Evaluation Setup

This section describes how the evaluation framework is implemented with Dify and Langfuse. The purpose of the setup is twofold: first, to run reproducible offline benchmark experiments on a fixed dataset; second, to collect trace-based signals that can later support pilot or production evaluation. The main text focuses on the evaluation logic and on the role of each execution mode. Detailed workflow diagrams are provided in Appendix C.4.

10.7.1 Role of Dify and Langfuse

Dify serves as the execution environment. For each input query, it performs retrieval and generation and emits tracing data through the official Langfuse integration [16, 17]. Langfuse handles everything else: storing traces, managing datasets and dataset runs, triggering evaluators, and persisting scores. This separation keeps the application workflows and the evaluation pipeline loosely coupled: the same traced executions can be re-scored with updated evaluators without modifying the assistant itself.

Prompt versions are tracked centrally via the Langfuse community plugin for Dify [18], which makes it easier to interpret results across workflow variants.

In Langfuse, a session groups the traces from a single conversation thread. Each trace contains typed observations, such as retrieval, generation, tool call, and evaluator observations, that allow scores to be attached either to live executions or to dataset-based benchmark runs [19].

Table 10.5 summarizes the three evaluation modes used or considered for the evaluation.

Table 10.5: Automated evaluation modes. Source: own illustration.

Option	Online	Offline	Reference-based	Custom scores
llm-as-a-judge evaluator in Langfuse	Yes	No	No	Langfuse-provided only
llm-as-a-judge with custom ground truth	No	Yes	Yes	Langfuse-provided only
Custom metric	(Yes)	Yes	Yes	Depends on metric

10.7.2 Managed Live Evaluators

Live evaluators operate on traces as they arrive from the running application, making them suitable for online or pilot evaluation where no ground-truth answer is available. Langfuse reads the relevant trace inputs, renders the evaluator prompt, calls the judge model asynchronously, and stores the score back on the trace. This mode is most relevant for production-oriented metrics such as faithfulness and generation relevance. The sequence diagram is in Appendix C.7.

10.7.3 Dataset-Based Evaluators with Ground Truth

For offline benchmarking, each dataset item contains an input query and an expected output. A script executes the selected Dify workflow for each item and registers the result as a dataset run in Langfuse. Langfuse then applies the managed evaluator, comparing the generated answer against the ground truth. This is the primary setup used for generation correctness. The sequence diagram is in Appendix C.8.

10.7.4 Custom Metrics

Retrieval metrics such as Recall@k, MRR@k, and MAP@k cannot be computed by Langfuse's built-in evaluators, which operate on generated text rather than ranked document lists. A custom script handles this: it loads dataset items from Langfuse, executes the Dify workflow, extracts the retrieved document list from the trace, computes the ranking metrics, and writes the scores back to Langfuse. This keeps all evaluation signals in the same observability environment. The sequence diagram is in Appendix C.9.

The same pattern could be extended to frameworks such as Ragas for reference-free metrics like context utilization, but this remained an optional extension rather than part of the final benchmark.

10.7.5 RAG Technique Comparison Setup

This subsection describes how different RAG workflow variants are compared under the same dataset and scoring setup. A RAG pipeline can vary across many dimensions, including chunking strategy, embedding model, retrieval technique, and prompt design, and not all combinations are worth testing. In Dify, each variant is represented as a separate workflow, which allows configurations to be compared in a controlled way. Each dataset run receives a descriptive identifier linking it to a specific workflow variant. The overall benchmarking process is shown in Figure 10.3.

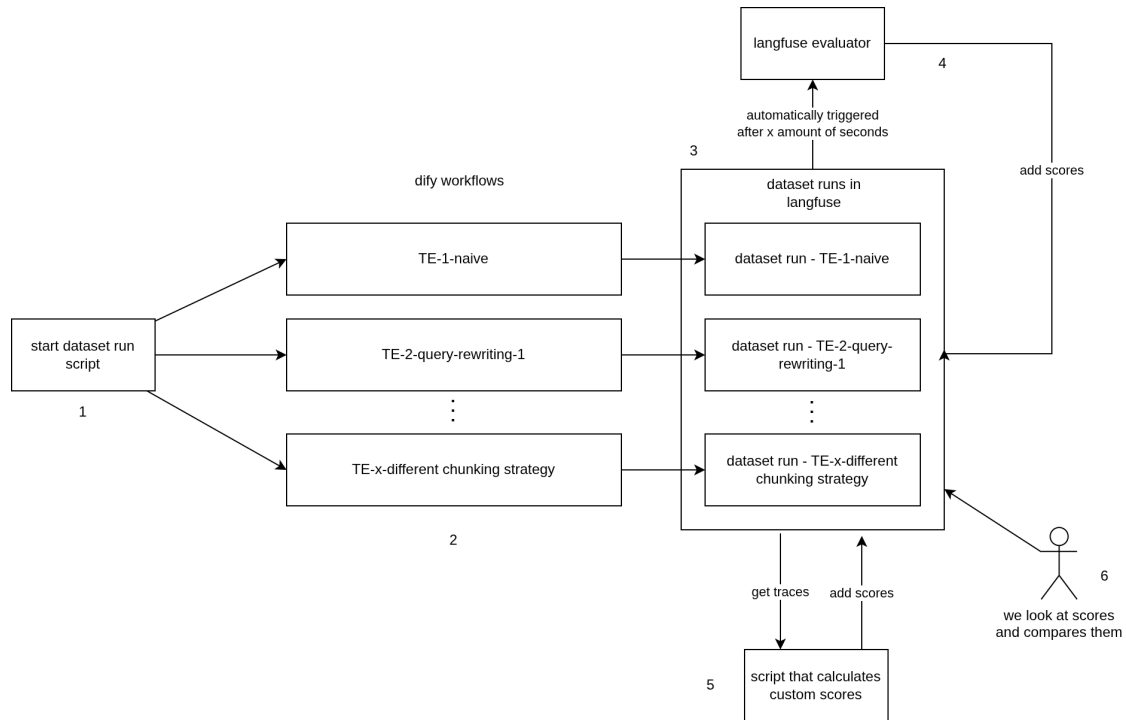


Figure 10.3: Offline evaluation benchmarking overview. Identifiers such as TE-1-naive combine the use case shortname (TE = ticket enrichment) with a descriptive workflow variant name. Source: own illustration.

The process for each benchmark run:

1. Execute the benchmark script over all selected dataset items.
2. Send each item to each selected Dify workflow; executions are traced automatically in Langfuse.
3. Create a separate dataset run per workflow variant so metrics can be compared at run level.
4. Apply the relevant Langfuse evaluator (live or dataset-based with ground truth).
5. Compute and attach custom retrieval scores via the separate script for scores that cannot be calculated directly in Langfuse, such as the retrieval metrics.
6. Review the results to determine whether the variant improves the system or exposes new weaknesses.

10.8 Results

The quantitative benchmark focuses on the ticket enrichment use case. The benchmark dataset contains 13 items. Each workflow variant was executed 5 times, and the reported

values are averages across these runs. The results are reported in three parts: the main retrieval and generation benchmark, a follow-up evaluation for reference-free faithfulness and relevance scores, and qualitative robustness checks.

- **TE-1-naive** — Baseline naive RAG. The user query is embedded directly and used to retrieve tickets.
- **TE-2-query-rewriting-1** — query rewriting before retrieval. A rewriting step reformulates the input before retrieval.
- **TE-3-query-rewriting-2** — Same as TE-2-query-rewriting-1 but with a local embedding model (all-minilm-l6-v2) instead of OpenAI’s text-embedding-3-small.
- **TE-4-agentic-retrieval** — Agentic retrieval using Dify’s Agent node, which can issue multiple retrieval queries and decide when additional lookups are needed.

10.8.1 Retrieval and Generation Results

Generation correctness was evaluated with an llm-as-a-judge setup against the golden answers. The workflow variants themselves were executed with **gpt-4.1-mini**, while the updated judge model for generation correctness was **gpt-5.4**.

Table 10.6: Ticket Enrichment dataset run results for 13 benchmark items and 5 runs per variant. Source: own illustration.

Configuration	Embedding model	Loop	Avg. latency	Avg. cost \$	Recall @10	MRR @10	MAP @10	Generation correctness
TE-1-naive	text-embedding-3-small	No	7s	0.0017	0.85	0.87	0.85	0.8
TE-2-query-rewriting-1	text-embedding-3-small	No	8s	0.0018	0.94	0.93	0.89	1
TE-3-query-rewriting-2	all-minilm-l6-v2	No	7s	0.0017	0.83	0.78	0.65	-
TE-4-agentic-retrieval	text-embedding-3-small	Yes	10s	0.0017	0.93	0.85	0.63	1

The results in Table 10.6 show that TE-2-query-rewriting-1 performed best overall. Compared with the baseline TE-1-naive, adding query rewriting improved all reported retrieval metrics and also improved generation correctness. TE-3-query-rewriting-2 shows that replacing the hosted embedding model with the local embedding model reduced retrieval quality substantially in this setup. TE-4-agentic-retrieval reached the same generation-correctness score as TE-2-query-rewriting-1, but it did not improve ranking quality and increased latency. In this benchmark, query rewriting therefore contributed more than agentic looping.

generation correctness is not reported for TE-3-query-rewriting-2 because this variant was not re-evaluated after the judge setup was updated to **gpt-5.4**. For the same reason, the numbers in the main table should be interpreted primarily as a workflow comparison, not as a fully uniform judge-model comparison across all historical runs.

10.8.2 Faithfulness and Relevance

No real-user data was available, but faithfulness and generation relevance were still tested. These reference-free llm-as-a-judge metrics were therefore evaluated separately on TE-2-query-rewriting-1 using gpt-5.4 as the judge model. Because they were only executed for TE-2-query-rewriting-1, they are reported separately in Table 10.7.

Table 10.7: TE-2-query-rewriting-1 follow-up evaluation with reference-free LLM judge metrics using gpt-5.4. Source: own illustration.

Metric	Configuration	Judge model	Result
faithfulness (Langfuse)	TE-2-query-rewriting-1	gpt-5.4	0.73
Relevance (Langfuse)	TE-2-query-rewriting-1	gpt-5.4	0.81

Faithfulness The faithfulness score of 0.73 is mainly lowered by responses that contain speculative elements. In the ticket enrichment use case, some speculation can still be useful, for example when the model proposes plausible next steps or mentions likely related causes. A better prompt structure would therefore separate grounded evidence from speculative suggestions more clearly. The evaluator could then focus on the evidence-based part of the response instead of penalizing all speculation equally.

Relevance The relevance score of 0.81 suggests that TE-2-query-rewriting-1 usually addresses the task well, but still includes some unnecessary or weakly related details. One likely reason is that the evaluator prompt does not yet provide enough context about the intended role of the ticket enrichment workflow. In addition, some responses cite tickets that are only loosely related to the current case, which reduces perceived relevance even when the overall answer remains useful.

10.8.3 Qualitative Robustness Results

The qualitative robustness samples were not included in the quantitative benchmark because they are case-based probes rather than repeated, comparable runs. In the Veeam license case, the model correctly treated the current tool output as more reliable than the older ticket. In the application-server case, GPT-4.1 models conflated Server 1 and Server 2, while GPT-5.4 preferred Server 2 because the source came from an IT technician and described actual work on that server. In the noise-robustness sample, the model correctly identified that the question concerned Company B and did not transfer Company A’s backup retention settings.

10.9 Conclusion

The evaluation indicates that TE-2-query-rewriting-1 is the strongest ticket enrichment variant among the tested workflows. query rewriting improved both retrieval quality and

end-to-end generation quality compared with the baseline, while the local embedding variant TE-3-query-rewriting-2 performed noticeably worse in this setup. The agent-based TE-4-agentic-retrieval workflow did not justify its additional latency through better retrieval performance.

At the same time, the reference-free llm-as-a-judge scores show that answer quality is not yet fully solved once retrieval quality improves. In particular, the faithfulness score suggests that the current prompt design does not yet separate grounded evidence clearly enough from helpful speculation. This makes prompt refinement and better evaluator alignment the most important next steps for improving the Ticket Enrichment workflow.

Chapter 11

Text-to-SQL

Structured enterprise data is often already stored in relational databases. For analytical questions, SQL provides filtering, aggregation, joins, and exact matching out of the box. This raises a simple question: if an LLM can use tools, can it answer business questions by generating and executing SQL directly?

This chapter examines that question in two contexts. The first part reports a PoC based on the EKE schema provided by Raiffeisen. The second part briefly discusses how the same pattern could be used for structured documentation sources in the technician assistant and ticket enrichment use cases.

The scope is intentionally limited. The EKE experiments are not a full benchmark and do not establish production readiness. They are used to explore whether Text-to-SQL is technically plausible for this domain, which workflow patterns improve results, and where the main risks remain.

11.1 EKE Task Overview

Raiffeisen provided seven sample queries for the EKE. EKE is used by the Swiss National Bank (SNB) and Swiss Financial Market Supervisory Authority (FINMA) to collect detailed information about individual credits in the Swiss banking system [3].

The database schema and sample data were created with GPT-5.4 from the public EKE documentation and the provided queries. This means that the experiments measure the behavior of the Text-to-SQL workflows on a synthetic approximation of the domain, not on a live production database.

For example, one of the supplied tasks asks for the number of customers with turnover above CHF 10 million and an available limit above CHF 1 million. The corresponding reference query from Appendix E.2 joins customers, counterparties, and credit frameworks:

Natural language query: Kannst du mir folgendes ausgeben? Anzahl Kunden mit Umsatz > 10 Mio. und Limite > 1 Mio.

SQL query:

```
SELECT count(distinct kunde.kunde_id) AS anzahl_kunden
```

```

FROM eke.kunde
INNER JOIN eke.gegenpartei_kunde
    ON kunde.stichtag = gegenpartei_kunde.stichtag
    AND kunde.kunde_id = gegenpartei_kunde.kunde_id
INNER JOIN eke.rahmen
    ON gegenpartei_kunde.stichtag = rahmen.stichtag
    AND gegenpartei_kunde.gegenpartei_id = rahmen.gegenpartei_id
WHERE kunde.stichtag = DATE '2025-12-31'
    AND rahmen.rahmen_limite_verfuegbar > 1000000
    AND kunde.umsatz > 10000000;

```

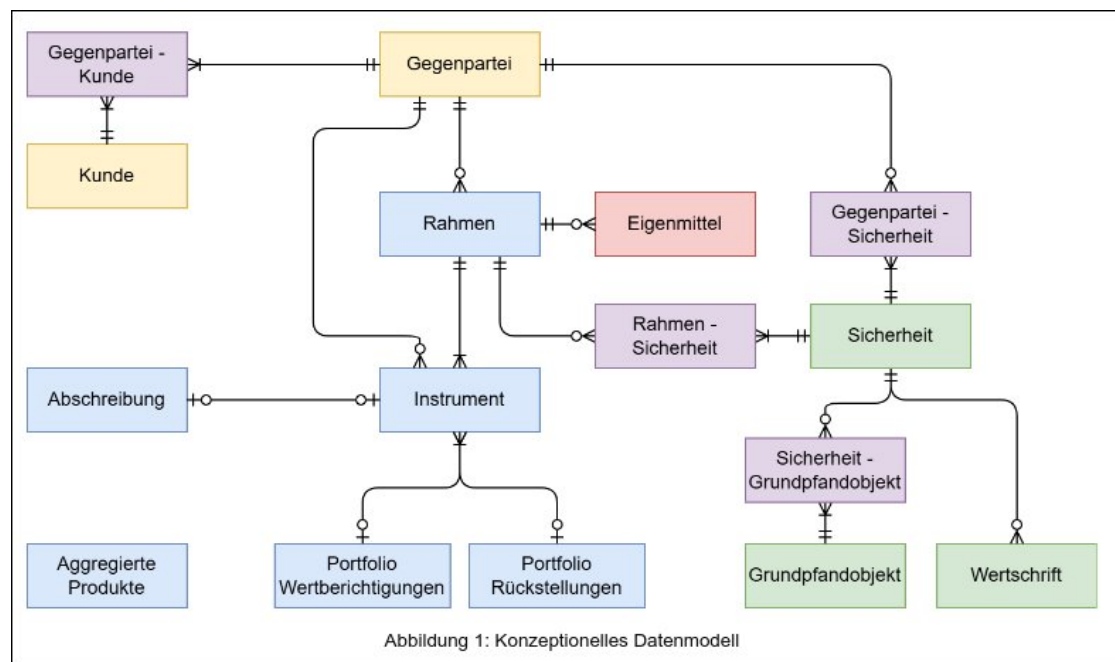


Figure 11.1: Simplified EKE schema. Source: [3].

11.2 Requirements

This section defines the requirements for the Text-to-SQL PoC. Given the exploratory scope of this chapter, requirements are intentionally narrower than those of the technician assistant and ticket enrichment use cases. The goal is not production readiness but technical feasibility across a set of representative queries.

11.2.1 Functional Requirements

FR-SQL1: Natural Language Query Input The system shall accept a natural language question as input and translate it into an executable SQL query against the target database schema. **Acceptance criterion:** All seven EKE sample queries

are accepted and produce an executable SQL statement without manual reformulation.

FR-SQL2: Annotated Schema Integration The system shall support enriching the database schema with business definitions, domain terms, and filtering conventions as inline annotations. **Acceptance criterion:** The annotated schema is passed to the model as context and the generated SQL references column names and tables consistent with the annotations.

FR-SQL3: Feedback-Driven Query Refinement The system shall support iterative query refinement by passing previous SQL attempts and their results back to the model for correction. **Acceptance criterion:** When an initial query produces a syntax error or empty result, the loop workflow generates at least one revised query before terminating.

FR-SQL4: Schema Linking The system shall support a schema linking step that identifies the tables, columns, and values relevant to a natural language question before constructing the final SQL query. **Acceptance criterion:** The schema linking step produces a structured list of relevant schema elements for each of the seven EKE queries.

FR-SQL5: Candidate Query Voting The system shall support generating multiple candidate SQL queries in parallel and selecting the best result through majority voting. **Acceptance criterion:** The voting workflow produces three candidate queries per input and selects one result, which is then returned as the final answer.

FR-SQL6: Column Exploration on Empty Results When a query returns an empty result, the system shall generate exploratory queries to inspect actual column values before attempting a revised query. **Acceptance criterion:** An empty result from the voting step triggers at least one round of exploratory queries limited to ten rows before the refinement loop continues.

FR-SQL7: Query and Reasoning Transparency The system shall display the generated SQL query and a brief reasoning explanation alongside the answer. **Acceptance criterion:** Every response includes both the final SQL statement and an explanation of how it was constructed, allowing a user to verify the result manually.

FR-SQL8: Read-Only Query Execution All SQL queries shall be executed using a read-only database user. **Acceptance criterion:** An attempt to execute a write operation (INSERT, UPDATE, DELETE, DROP) is rejected by the database before execution.

11.2.2 Non-Functional Requirements

NFR-SQL1: execution accuracy The system shall achieve a measurable execution accuracy across the seven EKE sample queries. **Acceptance criterion:** At least

one workflow variant achieves an average execution accuracy above 60 % across ten runs of the full query set.

NFR-SQL2: Cost Transparency Per-query token cost shall be logged for each workflow variant to allow cost-accuracy trade-offs to be assessed. **Acceptance criterion:** Average cost per query is recorded for all four workflow variants and reported in the evaluation.

NFR-SQL3: Latency Transparency Per-query end-to-end latency shall be logged for each workflow variant. **Acceptance criterion:** Average latency per query is recorded for all four workflow variants and reported alongside accuracy results.

11.2.3 Requirements Coverage

Table 11.1: Functional Requirements — Text-to-SQL

ID	Acceptance Criterion	Met
FR-SQL1: NL Query Input	All 7 EKE queries produce an executable SQL statement	Partially
FR-SQL2: Annotated Schema	Annotated schema passed as context; SQL consistent with annotations	Yes
FR-SQL3: Query Refinement	Syntax error or empty result triggers at least one revised query	Yes
FR-SQL4: Schema Linking	Structured schema element list produced for all 7 queries	Yes
FR-SQL5: Candidate Voting	3 candidates generated and 1 selected per input	Yes
FR-SQL6: Column Exploration	Empty voting result triggers exploratory queries (≤ 10 rows)	Yes
FR-SQL7: Query Transparency	Every response includes final SQL and reasoning explanation	Yes
FR-SQL8: Read-Only Execution	Write operations rejected at database level	Yes

Table 11.2: Non-Functional Requirements — Text-to-SQL

ID	Acceptance Criterion	Met
NFR-SQL1: execution accuracy	≥ 1 workflow variant achieves avg. EA > 60 % over 10 runs	Yes
NFR-SQL2: Cost Transparency	Avg. cost per query logged and reported for all 4 variants	Yes
NFR-SQL3: Latency Transparency	Avg. latency logged and reported for all 4 variants	Yes

The requirements reflect the proof-of-concept scope of this chapter. Production-readiness concerns such as ambiguity handling, multi-schema support, and permission enforcement at row or column level are discussed as potential improvements in Section 11.9 but were not implemented and are therefore not included in the coverage table.

11.3 Risks and Guardrails

Giving an LLM access to organizational data through SQL introduces risks that are different from standard RAG. Retrieval usually exposes selected documents; SQL can expose arbitrary combinations of rows and columns if it is not restricted carefully.

Permission enforcement In larger databases, users often have access only to selected schemas, tables, rows, or columns. A production system would therefore have to execute queries with permissions that reflect the original user’s access rights. One possible design is to select a read-only database user with matching permissions in the backend before executing tool calls. In this PoC, all queries were executed with a read-only user.

Data leakage Sensitive organizational data must not be sent to an external model unless this is allowed by policy and contract. If data cannot leave the organization, the model must either run locally or operate only on demo data. The latter may not be feasible for large or highly specific databases.

Incorrect answers Users may rely on results even when the generated SQL is wrong. For technical users, the generated query and a short explanation should therefore be displayed together with the answer. This makes the result auditable and gives users a chance to identify obvious mistakes.

Expensive or harmful queries A generated query can be syntactically valid but still expensive enough to overload the database. This can be mitigated at the database level through statement timeouts, resource limits, and read-only roles.

Large result sets Returning too many rows can flood the model context and increase cost. Frameworks such as ReFoRCE [20] instruct the model to limit exploratory queries to 20 rows. A backend should also truncate results before passing them back to the LLM if the model fails to add a limit.

11.4 Implementation Context

The workflows were implemented in Dify to gain more experience with the platform. This helped us understand Dify better, but it made the Text-to-SQL experiments slower than a code-based implementation would likely have been. Dify’s visual workflow model

is not ideal for building and comparing complex SQL agents. The SQL queries were executed through a Dify community plugin [21].

The implementation should therefore be read as an experimental setup rather than as a recommended production architecture.

11.5 Methods

Text-to-SQL systems generally follow either *in-context learning* or *fine-tuning* [22]. This PoC focuses only on in-context learning. fine-tuning was outside the scope of the thesis.

Recent benchmark systems increasingly use agentic workflows rather than a single prompt. The best-performing systems on benchmarks such as Spider 2.0 often combine schema reasoning, execution feedback, query repair, voting and more. The following methods were implemented in simplified form.

11.5.1 External Knowledge Integration

SQL generation for EKE requires more than table and column names. The model also needs business definitions, domain terms, and filtering conventions. To provide some of this context, GPT-5.4 was used to annotate the generated schema with comments based on the EKE documentation. In a production setup, these comments would need to be written or reviewed by domain experts. Additional definitions could also be retrieved from an internal knowledge base through RAG.

An excerpt of the annotated schema is shown below:

```
CREATE TABLE eke.gegenpartei (  
  stichtag          date NOT NULL REFERENCES eke.report(stichtag) ON  
  ↪ DELETE CASCADE,  
  
  -- GP-00: Gegenpartei als kreditnehmerische oder sicherheitengebende  
  -- Einheit. Eine Gegenpartei besteht aus einem oder mehreren Kunden.  
  gegenpartei_id    text NOT NULL,  
  
  -- GP-01: EKE-Risikoklasse (R1-R9, D, UR).  
  -- Fachlich Mapping aus bankinternem Rating.  
  rating            text,  
  
  -- GP-02: 1-Jahres Probability of Default.  
  probability_of_default numeric(12,10),  
  
  -- GP-03: Datum der letzten Risikobeurteilung.  
  datum_risikobeurteilung date,  
  
  PRIMARY KEY (stichtag, gegenpartei_id)  
);
```

11.5.2 Feedback-Driven Refinement

Feedback-driven refinement is common in Text-to-SQL frameworks [23, 24, 25]. The basic idea is to let the model repair its query after observing the database response.

In the Dify workflow, the LLM receives the previous SQL queries and their results. If the query produced a syntax error, an empty result, or an otherwise unsatisfactory answer, the model can generate another query. A judge decides whether the current result is good enough to stop the loop. The full prompt is listed in Appendix E.9.

```
You have to judge whether the sql query result is satisfactory based on the
user query.
user query:
sys.query
sql queries and results:
results
```

11.5.3 Schema Linking

schema linking asks the model to identify the tables, columns, values, and relationships that are relevant to the natural-language question before constructing the final SQL query. This step is based on DIN-SQL [26].

The workflow uses a compressed schema and a small set of few-shot examples generated offline with GPT-5.4. The full prompt is listed in Appendix E.13.

```
Du bist ein Experte für Schema Linking für Text-to-SQL.

## Aufgabe
Bestimme für jede natürliche Sprachfrage die relevanten Schema links auf Basis
des Datenbankschemas und der Fremdschlüssel.
```

For the request “Welche Kunden mit Domizilland Schweiz sind sensitive Kunden?”, a shortened schema-linking output could look as follows:

```
Schema_links: [eke.kunde.kunde_id, eke.kunde.kundendomizil_land,
eke.kunde.sensitiver_kunde, Schweiz, true]
```

11.5.4 Voting

Even with a temperature of 0, generated SQL can vary between runs. Voting helps stabilize the output by comparing multiple candidate queries [20].

The most complex workflow generates three schema-linking paths and then uses majority voting. The voting prompt receives the candidate queries and their results and selects the best one. The full prompt is listed in Appendix E.10.

```
basierend auf folgender Benutzeranfrage, wähle die beste SQL Abfrage aus und
begründe die antwort.
#sys.query#

[query1]
#result_query1#
[/query1]
```

Other voting strategies are possible, such as tournament-style selection [23], confidence-aware SQL selection [27], or cross-consistency with different LLMs [28].

11.5.5 Column Exploration

Some questions require filter values that are not present in the few-shot examples. In these cases, the model may need to inspect example values before it can write the correct query.

If majority voting returns an empty result, the workflow asks the LLM to generate three exploratory SQL queries limited to 10 rows. The results are then used in the refinement loop. This follows the same general motivation as the exploration step in ReFoRCE [20]. The full prompt is listed in Appendix E.11.

```
Du hast folgende benutzeranfrage:
#sys.query#

folgende sql query wurde benutzt als erster versuch, jedoch stimmt etwas nicht
mit der abfrage.
#generated_query#

Kannst du 3 explorative sql queries generieren?
```

11.5.6 Reasoning

The query-construction prompts ask the model to explain its reasoning. This helps with debugging and can improve results when the model has to combine schema links, previous attempts, and database feedback. The full prompt is listed in Appendix E.12.

```
#annotated_schema_and_prompt#

Du hast folgende schema linking analyse um dir zu helfen:
#schema_linking_text#

vorherige versuche:
#results#
```

11.5.7 Other Methods

Several methods were not implemented. One example is difficulty classification, as described in DIN-SQL [26]. This could be used to route simple questions to cheaper workflows and reserve more expensive agentic workflows for difficult cases.

11.6 Workflows

Four workflow variants were tested. They differ mainly in how much context they provide and whether they allow the model to revise its answer after execution.

text-to-sql-1 and text-to-sql-2 (linear) The two linear workflows use a single prompt and one SQL execution step, as shown in Figure 11.2. The first variant uses the raw schema. The second uses the GPT-5.4-annotated schema based on the public EKE documentation. The raw schema has about 3800 tokens, while the annotated schema has about 5700 tokens. Longer contexts can increase cost and can also distract the model [29].

```
Du bist ein Assistent der SQL Abfragen auf Postgres Datenbanken ausführt um
Daten abzufragen.
Benutze das folgende Schema als Ground Truth für Tabellen und Spaltennamen.

SCHEMA_BEGIN
...
SCHEMA_END
```

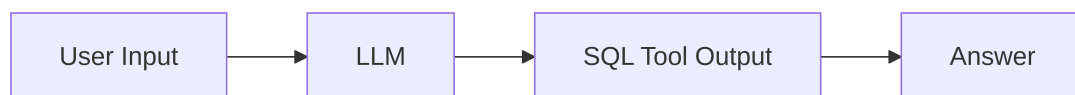


Figure 11.2: Text-to-SQL linear. Source: own illustration.

Text-to-SQL-3 (loop) The loop workflow adds a judge and a repair step, as shown in Figure 11.3. After each database response, the judge decides whether the result is satisfactory. If not, the LLM receives the previous query and result and generates a revised query. This is especially useful for syntax errors and empty results.

Text-to-SQL-4 (voting, exploration, and loop) The most complex workflow combines schema linking, voting, query exploration, and iterative refinement. As shown in Figure 11.4, three schema-linking and query-generation paths are executed in parallel. If one of the voted results returns rows, the workflow returns that result immediately. If the result is empty, the workflow continues with query exploration and then enters the

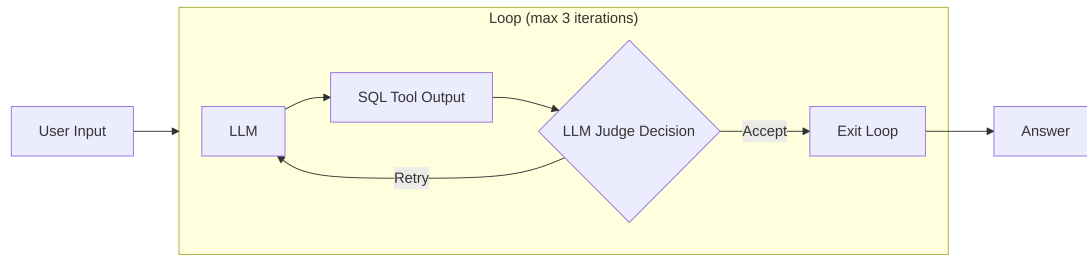


Figure 11.3: Text-to-SQL loop. Source: own illustration.

same refinement loop as Text-to-SQL-3. SQL errors go directly to the loop. The full workflow is shown in Appendix E.1.

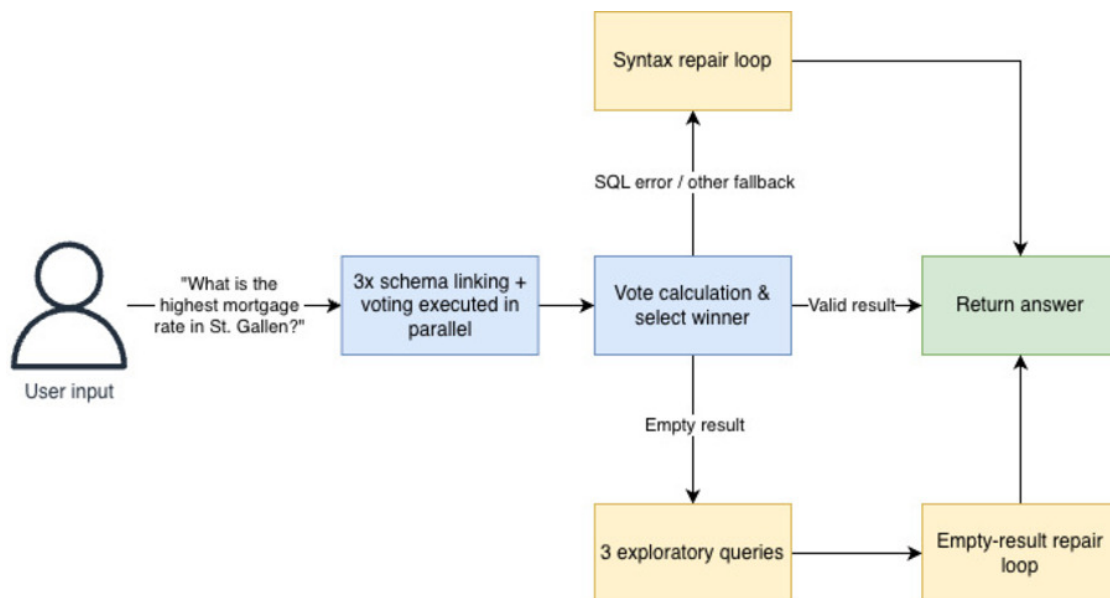


Figure 11.4: Text-to-SQL-4 simplified version; the full version can be found in Figure E.1. Source: own illustration.

11.7 Evaluation Setup

All main runs use GPT-4.1 mini with temperature set to 0. GPT-5 models did not support temperature in the tested setup, so they were not used for the controlled evaluation runs. A temperature of 0 still does not make the output fully deterministic because floating-point arithmetic and concurrent processing can produce small differences between runs.

The primary metric is execution accuracy. Exact SQL matching is too rigid because the same answer can often be produced by several different SQL queries. execution

accuracy is also the main metric used in benchmarks such as Spider 2.0 [30] and BIRD-bench [31].

Even execution accuracy requires judgement about what counts as equivalent. This evaluation allows extra columns, treats integers and integer-valued floats as equivalent, and ignores differences in column order. The goal is to avoid false negatives caused by superficial formatting differences.

Because the dataset contains only seven queries and uses synthetic sample data, the results should be interpreted as proof-of-concept evidence rather than as a generalizable benchmark of model performance.

11.7.1 Error Analysis Method

Errors were categorized deterministically as wrong results, empty results, or SQL syntax errors. These broad categories are useful for identifying which part of the workflow needs improvement. They could be refined further into categories such as aggregation errors, join errors, or incorrect WHERE clauses.

Outputs were grouped and compared across runs. This makes recurring errors visible quickly. If the same wrong result appears repeatedly, the issue is likely caused by missing business context, ambiguous intent, or a systematic prompting problem rather than by random variation.

11.8 Results

Table 11.3 summarizes the aggregated dataset results. Each workflow was run ten times. The evaluation was analyzed primarily with output-shape information, but a smaller set of runs without output shape was also checked manually. The main difference was that some queries returned additional columns; this did not meaningfully change the interpretation.

Table 11.3: Aggregated Text-to-SQL dataset run results for 10 dataset runs each. Source: own illustration.

Configuration	Prompt name	Model	Loop	Thinking	Avg. latency	Avg. cost in \$	Avg. EA
Text-to-SQL-1 baseline	raw schema	gpt-4.1-mini-2025-04-14	No	none	3s	0.001853	57.14%
Text-to-SQL-2 annotated schema	annotated schema	gpt-4.1-mini-2025-04-14	No	none	3.5s	0.002548	58.57%
Text-to-SQL-3 annotated schema with loop	annotated schema	gpt-4.1-mini-2025-04-14	Yes	none	7s	0.005311	61.43%
Text-to-SQL-4 schema linking and loop	annotated schema	gpt-4.1-mini-2025-04-14	Yes	none	65s	0.021416	71.43%

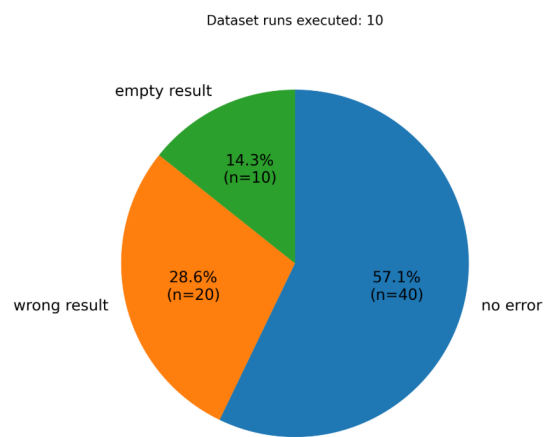


Figure 11.5: Text-to-SQL-1 error pie chart.
Source: own illustration.

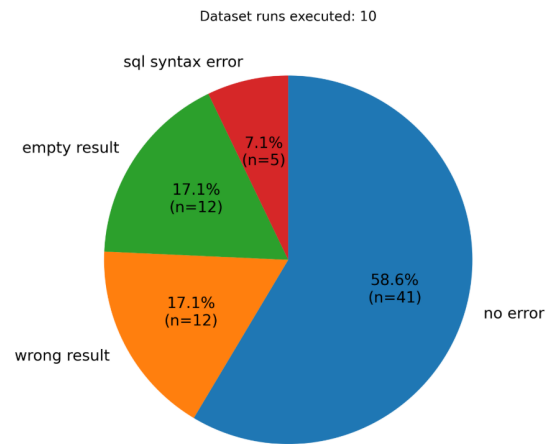


Figure 11.6: Text-to-SQL-2 error pie chart.
Source: own illustration.

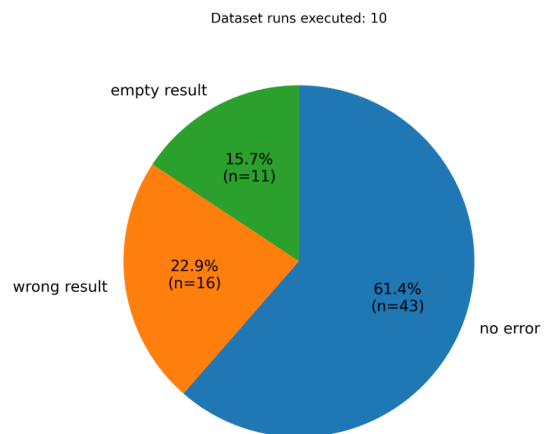


Figure 11.7: Text-to-SQL-3 error pie chart.
Source: own illustration.

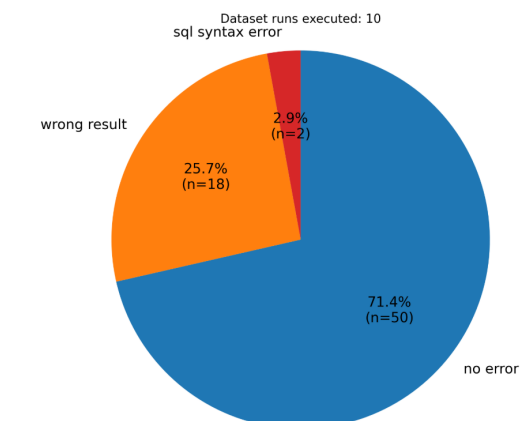


Figure 11.8: Text-to-SQL-4 error pie chart.
Source: own illustration.

Table 11.4: Aggregated statistics per dataset item for **Text-to-SQL-1 baseline**. Cells are colour-coded: green = high execution accuracy (>0.8), amber = partial accuracy ($0.4\text{--}0.8$), red = low accuracy or non-zero errors. Source: own illustration.

Dataset Item	Text-to-SQL-1 baseline			
	Exec. Acc.	Empty Res.	Syntax Err.	Wrong Res.
High-turnover customers	1	0	0	0
ETP quote development	0	0	0	1
Blatten properties	1	0	0	0
Wealth-to-limit ratio	0	0	0	1
Securities portfolio value	1	0	0	0
IRB RWA by rating	1	0	0	0
Private-customer PLZ values	0	1	0	0

Table 11.5: Aggregated statistics per dataset item for **Text-to-SQL-2 annotated schema**. Cells are colour-coded: green = high execution accuracy (>0.8), amber = partial accuracy ($0.4\text{--}0.8$), red = low accuracy or non-zero errors. Source: own illustration.

Dataset Item	Text-to-SQL-2 annotated schema			
	Exec. Acc.	Empty Res.	Syntax Err.	Wrong Res.
High-turnover customers	0.7	0	0.3	0
ETP quote development	0	0.2	0	0.8
Blatten properties	1	0	0	0
Wealth-to-limit ratio	0.5	0	0.2	0.3
Securities portfolio value	0.9	0	0	0.1
IRB RWA by rating	1	0	0	0
Private-customer PLZ values	0	1	0	0

Table 11.6: Aggregated statistics per dataset item for **Text-to-SQL-3 annotated schema with loop**. Cells are colour-coded: green = high execution accuracy (>0.8), amber = partial accuracy (0.4–0.8), red = low accuracy or non-zero errors. Source: own illustration.

Dataset Item	Text-to-SQL-3 annotated schema with loop			
	Exec. Acc.	Empty Res.	Syntax Err.	Wrong Res.
High-turnover customers	1	0	0	0
ETP quote development	0	0	0	1
Blatten properties	1	0	0	0
Wealth-to-limit ratio	0.9	0.1	0	0
Securities portfolio value	0.4	0	0	0.6
IRB RWA by rating	1	0	0	0
Private-customer PLZ values	0	1	0	0

Table 11.7: Aggregated statistics per dataset item for **Text-to-SQL-4 schema linking and loop**. Cells are colour-coded: green = high execution accuracy (>0.8), amber = partial accuracy (0.4–0.8), red = low accuracy or non-zero errors. Source: own illustration.

Dataset Item	Text-to-SQL-4 schema linking and loop			
	Exec. Acc.	Empty Res.	Syntax Err.	Wrong Res.
High-turnover customers	1	0	0	0
ETP quote development	0	0	0	1
Blatten properties	1	0	0	0
Wealth-to-limit ratio	0.9	0	0	0.1
Securities portfolio value	0.1	0	0.2	0.7
IRB RWA by rating	1	0	0	0
Private-customer PLZ values	1	0	0	0

The annotated schema did not substantially improve the GPT-4.1 mini results in this small setup, as shown in Figures 11.5 and 11.6. Earlier informal tests with GPT-5.4 mini showed a larger difference, but those runs are not part of the controlled evaluation.

The loop workflow improves the average score only slightly, while roughly doubling cost. The item-level results show why: the loop fixes some errors but also introduces new ones. The wealth-to-limit ratio query improves with iterative refinement, while the securities portfolio value query suffers from incorrect intent matching. A few informal runs of Text-to-SQL-3 with Gemma 4 produced an average execution accuracy of 41%; this is only a side observation.

Ambiguous queries Some queries can be interpreted in multiple valid ways. A production system should detect such ambiguity and either ask a clarifying question or use additional context, for example synthetic query logs as in SOMA-SQL [32].

ETP quote development This query is ambiguous. With a more specific request, it consistently returns the correct result. Instead of “Entwicklung der ETP-Quote (Vergleich zum letzten Quartal)”, the following formulation works: “Berechne pro Stichtag und ETP den Anteil der instrumente limite mit rechtsanspruch am Gesamtwert dieses Stichtags. Keine Veränderungsrate berechnen.” The definition of ETP-Quote should be supplied to the model as external knowledge.

Securities portfolio value This query appears not to benefit from the more complex workflows. The reason is again ambiguity: the gold query sums the last three quarters, while later workflow outputs do not. Both interpretations can be considered plausible without a clearer business definition.

Private-customer PLZ values The first three workflows always return an empty result because the model filters a column in the wrong way. The question asks for a calculation for private customers. The schema contains a column named “privatkundentyp”, and the gold query checks whether this column is not null. The model instead tries to filter for values such as “Privatkunde”. This could be improved through a clearer schema, enum values, example values, or better filtering instructions. The Text-to-SQL-4 workflow solves this case after schema linking and query generation.

11.8.1 Interpretation

The experiments show three recurring requirements. First, the model needs the correct business definitions. Second, the user’s intent has to be clear enough to choose the right aggregation and time range. Third, the model must understand valid column values so that it can construct filters correctly.

Methods such as loops, schema linking, and voting can improve individual samples, but they can also introduce new errors. For production use, these issues would need to be addressed with better domain knowledge, clearer schema representations, ambiguity handling, and stricter guardrails.

11.9 Potential Improvements

The implemented workflows are intentionally simpler than the current state of the art. Several extensions could make the approach more useful and reliable.

SQL command library One use case mentioned in a meeting was recurring ad hoc queries. A team may be asked to run a query once and then run the same query again later. If the query is recreated manually each time, small differences can change the

result. A library of approved SQL commands paired with natural-language descriptions could solve this problem. It could also support few-shot sampling through SQL skeleton retrieval [33].

Load only necessary tables After schema linking identifies the relevant tables, the workflow could load only those tables in detail. This would reduce context length and make the prompt easier for the model to process.

Use views as a curated schema layer Instead of exposing the full database schema, the system could query database views designed around business questions. Views can hide irrelevant columns, pre-join commonly used tables, rename technical fields, encode stable business definitions, and expose only the fields users are likely to ask about. This is useful when the underlying schema contains many columns that are technically necessary but not meaningful for analysis. Compared with a denormalized data warehouse, views can provide a lighter-weight way to create a Text-to-SQL-friendly schema while still using the source database. The tradeoff is that the views must be maintained and kept aligned with business definitions.

Use denormalized warehouse data If the data is already exported to a denormalized data warehouse, Text-to-SQL may be easier there than on the source schema. Some statistics may already be precomputed. The tradeoff is that the data may no longer be live. Depending on the use case, this may still be acceptable. Avaris, the second industry partner, expressed interest in this direction for analyses such as ticket hours.

Support multiple schemas A production system would likely need to work across multiple schemas. This could be handled either by a query router that selects the right schema automatically or by letting the user select the schema explicitly.

11.9.1 Interim Summary

Taken together, the EKE experiments provide initial evidence that Text-to-SQL can be applied to this domain. They do not yet allow a strong statement about practical viability. The evaluation uses only seven sample queries, synthetic sample data, and workflows that are simpler than many current Text-to-SQL systems. The results should therefore be read as an indication that the approach is technically promising, especially if more domain knowledge, richer schema representations, and more advanced methods are added.

11.10 Text-to-SQL for Structured Documentation Sources

The same pattern can also help in the support assistant context. Some documentation sources are highly structured: software inventories, license tables, customer records, or configuration exports. These sources can be queried through custom MCP tools, but

every new operation would require a new function. SQL already contains many of the required operations.

11.10.1 Examples

For the technician assistant, the chat agent could decide whether a question requires ticket retrieval, structured data via SQL, or both. A question about a previous incident would use ticket retrieval. A question about currently installed software versions would use SQL. Some questions may need both.

For ticket enrichment, structured data could add current facts to new tickets. If a ticket mentions a host or customer, the agent could attach information such as installed software versions, license status, or other relevant inventory data.

Here are a few examples that we tested. Software names and company names are changed.

Example 1: Vulnerable software version Request: The Veeam Backup and Replication version 12.3.2.4165 has a critical vulnerability. Please look up which servers have this version installed.

Answer: The servers found are DD-BKP-1, AC-BKP-2, and QA-BKP-1 (short version with anonymized hostnames).

SQL query:

```
SELECT si.device, si.software_name, si.version
FROM software_installations si
WHERE si.software_name = 'Veeam Backup & Replication'
      AND si.version = '12.3.2.4165';
```

Example 2: License expiration Request: When does the vSphere license expire for Metalworking AG?

Answer: The license expires on 05.02.2027 (short version).

SQL query:

```
SELECT software_name, license_key, version, expires_on
FROM software_licenses
WHERE organization_name = 'Metalworking AG'
      AND software_name = 'vSphere';
```

Example 3: Unused Microsoft licenses Request: Can you give me all Microsoft licenses that are unused for Metalworking AG, excluding any Free or Trial licenses?

Answer: Returns a table with the matching licenses.

SQL query:

```
SELECT license_name, active_count, consumed_count, unused_count
FROM microsoft_licenses
WHERE organization_name = 'Metalworking AG'
      AND unused_count > 0
```

```
AND active_count > consumed_count
AND license_name NOT ILIKE ALL (
  ARRAY['%Free%', '%Trial%', '%Exploratory%', '%Viral%', '%Developer%']
);
```

11.10.2 Benefits and Limitations

Text-to-SQL is useful for exact, current, structured facts. It is better suited than vector retrieval for filtering, aggregation, exact version matching, and numeric comparisons. It also fits companies that already maintain a denormalized data warehouse.

The limitation is that the answer is only as reliable as the generated SQL and the database context given to the model. The system can still produce misleading answers if the intent is ambiguous, the schema is unclear, or the model applies the wrong business definition.

11.11 Conclusion

This chapter does not establish that Text-to-SQL for EKE is ready for practical use. It shows that the approach can be implemented and that more sophisticated workflows can improve performance, while important limitations remain.

The strongest result is the direction rather than the absolute score: schema-aware and feedback-driven workflows are more promising than a single linear prompt, but they also increase cost and complexity. Given the small sample set, synthetic data, and limited implementation, practical viability remains open. The results are promising enough to justify further investigation with real data, stronger domain knowledge, and production-grade guardrails.

Part IV

Conclusion

Chapter 12

Conclusion and Outlook

12.1 Conclusion

This thesis investigated how retrieval-augmented enterprise assistants can be implemented using existing AI application platforms in a managed IT services context. The starting point was a previously developed custom RAG-based platform. Instead of extending this custom solution further, this work evaluated whether established platforms can provide a more suitable foundation for production-oriented enterprise AI workflows.

The central research question was under which conditions a self-hosted platform, a managed cloud platform, or continued custom development is the more appropriate basis for RAG-based enterprise assistants. The results show that there is no universally superior platform choice. Both Dify and Microsoft Foundry are technically capable of supporting retrieval-based MSP workflows, but they optimize for different organizational and technical constraints.

Microsoft Foundry proved to be the more integrated option for organizations already operating in the Azure ecosystem. Its main strengths are managed infrastructure, native integration with Azure AI Search, Microsoft Entra ID, Azure-native monitoring, and document-level access control patterns. These capabilities reduce the amount of custom implementation work required around synchronization, identity, governance, and operational reliability. For Azure-centric organizations with strict compliance requirements and existing Microsoft infrastructure, Foundry therefore provides the more direct path toward production deployment. The main trade-offs are dependency on Azure, ongoing consumption-based cost, and the need to build a separate application layer for user interaction.

Dify demonstrated that comparable RAG workflows can be implemented in a self-hosted and platform-independent manner. Its visual workflow builder, built-in knowledge base, provider-agnostic model support, and integration with observability tools such as Langfuse make it suitable for organizations that require infrastructure control, data sovereignty, or cloud independence. However, the implementation also showed that this flexibility comes with additional responsibility. Synchronization, metadata handling, fine-grained access control, upgrade testing, and production hardening require more cus-

tom work than with Foundry. The Secure Knowledge Retrieval Plugin developed in this thesis illustrates this trade-off: Dify can be extended to support centralized role-based retrieval filtering, but the responsibility for correctness and enforcement remains with the implementation.

Compared with continued custom development, both evaluated platforms offer a more mature basis for building enterprise AI assistants. Core capabilities such as workflow orchestration, hybrid retrieval, prompt management, tracing, model integration, and deployment interfaces are already available. A self-developed solution remains justified only where requirements are highly specific, where platform constraints are unacceptable, or where full control over every layer is more important than implementation speed and operational maturity.

The implemented MSP use cases confirmed the practical feasibility of the platform approaches. The technician assistant showed that conversational access to historical support tickets can be implemented on both platforms. The ticket enrichment workflow showed that newly created tickets can be automatically enriched with contextual information from similar historical cases. In both use cases, the quality of the ticket data was a decisive factor. Incomplete descriptions, weak resolution notes, inconsistent terminology, or missing metadata directly reduced retrieval quality and therefore limited the usefulness of generated responses. This confirms that successful enterprise RAG systems depend not only on model and platform choice, but also on disciplined data preparation and knowledge management.

The evaluation framework developed in this thesis provides a reusable structure for assessing RAG systems beyond subjective inspection. By combining retrieval metrics, generation metrics, llm-as-a-judge evaluation, latency, cost, and qualitative robustness samples, the framework makes platform and workflow changes measurable. The ticket enrichment benchmark showed that query rewriting improved retrieval and generation quality compared with a baseline RAG setup, while a more agentic retrieval loop did not justify its additional latency in the tested configuration. The results also showed that improved retrieval does not automatically solve answer quality. faithfulness and relevance still depend on prompt design, output structure, and clear separation between grounded evidence and speculative suggestions.

The Text-to-SQL PoC suggests that LLM-based access to structured enterprise data is promising, especially for targeted factual queries, filtering, and aggregation, but it does not yet establish practical production viability. The evaluation was intentionally limited in scope and should therefore be interpreted as initial evidence rather than as a definitive assessment. At the same time, it also confirmed that this pattern requires strong safeguards. Permission enforcement, read-only access, query limits, result truncation, schema linking, validation, and clear presentation of generated SQL are necessary before such systems can be considered production-ready. Text-to-SQL is therefore best understood as a complementary access pattern rather than a replacement for RAG over unstructured documents.

Overall, the thesis contributes a practice-oriented basis for platform decisions in enterprise AI assistant projects. For Azure-based environments, Microsoft Foundry is the

stronger default choice when integration, governance, and managed operation are priorities. For organizations requiring self-hosting, model flexibility, or infrastructure independence, Dify is a viable alternative, provided that additional engineering effort for access control, synchronization, and operational hardening is planned. Custom development should be reserved for cases where neither platform can satisfy the required control, integration, or deployment constraints.

12.2 Outlook

Future work should first validate the implemented workflows in a real operational MSP environment. This thesis evaluated technical feasibility and system behavior, but it did not include a longitudinal user study with technicians. A productive pilot would make it possible to measure whether the technician assistant actually reduces search time, improves resolution quality, supports onboarding, or changes escalation behavior. Such a study should combine quantitative indicators, such as time-to-resolution and ticket handling time, with qualitative feedback from technicians.

A second important step is production-grade access control. In this thesis, access control was discussed and partially addressed architecturally, but not evaluated in a live multi-user environment. For Foundry-based deployments, future work should implement document-level security trimming using Entra ID identities and permission metadata. For Dify-based deployments, the Secure Knowledge Retrieval Plugin should be extended, tested, and audited against realistic user, role, tenant, and customer access models. Particular attention should be paid to ensuring that no unfiltered retrieval path can bypass the intended access restrictions.

The ingestion and synchronization pipelines should also be extended. The evaluated dataset was static, while productive ticket systems change continuously. Future systems should support incremental synchronization from the ticketing system, automatic metadata updates, deletion handling, re-indexing of modified tickets, and monitoring of ingestion failures. In addition, the quality of historical ticket data should be improved at the source. Structured resolution templates, required metadata fields, and technician-facing guidance for writing reusable ticket notes would likely improve RAG quality more sustainably than prompt tuning alone.

The evaluation framework should be developed into a continuous evaluation process. Instead of running evaluation only during development, future deployments should monitor retrieval quality, faithfulness, relevance, latency, and cost over time. New prompts, embedding models, retrieval strategies, and platform upgrades should be compared against a stable benchmark dataset before being released. The llm-as-a-judge prompts should also be calibrated with domain experts, because automated evaluation can diverge from what technicians consider useful or correct.

For the RAG workflows themselves, further experiments should examine alternative chunking strategies, metadata enrichment, hybrid retrieval parameters, reranking models, contextual chunk headers, and customer-specific filtering.

The Text-to-SQL PoC should be expanded only under strict safety constraints. Fu-

ture work should investigate schema-aware access control, query validation, cost and timeout limits, explainable SQL generation, and ambiguity detection. For ambiguous business questions, the system should ask clarifying questions instead of producing a possibly misleading result. A command library for recurring business queries could also improve reliability by combining predefined SQL patterns with LLM-assisted parameter extraction.

Finally, future work should compare the long-term operational costs of the platform approaches. This thesis examined implementation effort and selected runtime characteristics, but a production decision also requires a total-cost view including platform licensing, Azure consumption, hosting costs, maintenance, monitoring, security reviews, and required engineering capacity. Such an analysis would complement the technical findings and help organizations decide whether managed cloud integration, self-hosted independence, or targeted custom development is the most sustainable path for their enterprise AI assistant strategy.

Back Matter

Glossary

Azure AI Search An Azure search service that supports full-text, vector, and hybrid retrieval over indexed enterprise data i, iv, xiv, xv, 20–23, 25, 26, 30, 33, 43–45, 47, 56, 58, 61, 62, 67, 106

BLEU A reference-based text generation metric that measures n-gram overlap between a generated output and one or more reference texts i, 72, 73

chunking The process of splitting documents into smaller segments before embedding, indexing, or retrieval i, xiv, 10, 12, 14, 25, 28, 29, 78, 82, 108, 132, 133, 135

comprehensiveness Coverage and diversity of the retrieved documents with respect to the information need i, 69

counterfactual robustness Ability of a system to resist misleading or false context and prefer correct evidence when generating answers i, 70, 80

data sovereignty The requirement or ability to control where data is stored, processed, and accessed, often to satisfy legal or organizational constraints i, iv, v, 13, 18, 106

Dify A source-available platform for building and operating LLM-powered workflows, agents, and knowledge-base applications i, iv, v, xiv–xvii, 2–5, 7–10, 13–20, 23, 26–34, 36, 37, 43, 44, 46, 54–56, 81–84, 91–93, 106–108, 124, 126, 128–135, 137–141, 154

execution accuracy A text-to-SQL evaluation metric that scores whether the executed query returns the correct result, regardless of the exact SQL formulation i, 89, 90, 96, 97, 100

faithfulness Degree to which a generated answer is fully supported by retrieved documents without adding unsupported claims i, 69, 70, 72, 74, 75, 78, 81, 85, 107, 108

fine-tuning The process of further training a pretrained model on task-specific data to adapt its behavior i, 92

generation correctness Factual and contextual accuracy of a generated response compared with a reference answer i, xix, 69, 73, 74, 77, 82, 84

generation relevance How well a generated response addresses the intent and content of the original query i, 69, 74, 75, 78, 81

glossary A brief dictionary for a specific field i

gold documents Documents in the golden dataset that should be retrieved for a query i, 69, 71, 80, 143

golden dataset A curated evaluation dataset with manually defined expected outputs or relevant documents used as a benchmark i, xix, 77, 143

ground truth The reference answer, label, or expected result against which a system output is compared i, xvi, 71, 81–83, 143, 149, 153

hallucination A generated statement that is unsupported, fabricated, or inconsistent with the available evidence i, 39, 40, 59, 60

Helm A package manager for Kubernetes that installs and configures applications using versioned charts i, 126, 129, 137–141

hybrid search A retrieval approach that combines keyword-based search with vector similarity search i, 10, 29, 134, 135

in-context learning A prompting approach in which a model is guided by instructions and examples provided directly in the input context instead of being retrained i, 92

ingestion pipeline The process that retrieves source data, transforms it into a suitable structure, and loads it into an index or knowledge base i, 25, 41, 45, 61

knowledge base A collection of indexed documents and metadata used as the retrieval source for an AI application i, iv, vi, xiv, 10, 12, 14, 15, 17, 23, 25, 30, 37–40, 43, 44, 46, 51, 58, 61, 62, 66, 92, 106, 134, 135

Kubernetes An orchestration platform for deploying, scaling, and managing containerized applications i, vii, 8, 33, 126–129, 138, 140, 141, 183

Langfuse An open-source LLMops observability and evaluation platform i, iv, xv, xvi, 7, 8, 14, 33, 55, 72, 74, 78, 80–83, 85, 106, 125, 127–129, 143, 149, 151, 154

LaTeX A document preparation system i

listwise evaluation An evaluation scheme in which a full set of responses is assessed jointly and ranked as a group i

llm-as-a-judge using LLMs as evaluators to assess the performance of outputs i, xvi, xix, 71, 72, 74, 76, 77, 81, 84–86, 107, 108, 149, 151, 153

LM metrics Language-model-based evaluation metrics that compare generated text and reference text using model representations or semantic similarity rather than only lexical overlap i, 70, 77

managed identity An Azure identity assigned to an application or resource so it can authenticate to other Azure services without storing long-lived secrets i, 23

MAP@k Mean of average precision scores across queries for top-k retrieval i, 72, 76, 82, 153

Microsoft Entra ID Microsoft’s cloud identity and access management service for authentication, authorization, and single sign-on across Azure and Microsoft 365 i, iv, xiv, 23, 24, 26, 45, 46, 106

Microsoft Foundry A managed AI application platform in the Azure ecosystem that provides model access, managed agents, and integration with Azure services i, iv, v, xiv, xv, xvii, 3–5, 7, 9, 10, 20–22, 26, 27, 34, 36, 37, 43, 47, 56, 58, 67, 106, 107, 135

MRR@k Mean reciprocal rank of the first relevant result across queries i, 71, 76, 82, 153

negative rejection Ability of a system to abstain from answering when the provided information is insufficient i, 70

noise robustness Ability of a system to maintain answer quality when irrelevant or distracting context is present i, 70, 80

non-rank-based Metric type where only set membership of retrieved documents matters, not their rank positions i, 70, 73

observability The ability to inspect and understand system behavior through traces, logs, metrics, and related runtime signals i, iv, 5, 10, 18, 26, 27, 34, 82, 106, 127, 128

pairwise evaluation An evaluation scheme in which two responses are compared directly and one is preferred over the other i

pointwise evaluation An evaluation scheme in which each response is scored independently against defined criteria i, 74

PostgreSQL An open-source relational database management system i, xvi, 46, 122, 129

Precision@k Fraction of retrieved documents in the top k results that are relevant to the query i, 153

query rewriting The reformulation of an original user query into one or more improved search queries to increase retrieval quality i, 29, 84, 85, 107

rank-based Metric type where the score explicitly depends on the rank order of retrieved documents i, 70, 73

rank-sensitive Metric type where score changes with document positions in practice, but does not follow a strictly rank-based definition i, 73

Recall@k Proportion of gold documents that are retrieved within the top k results i, 71, 76, 82, 153

relevant retrieved documents Retrieved documents that are relevant to the query i

reranking A post-retrieval step that reorders candidate results using a secondary scoring model or ranking method i, 10, 15, 30, 34, 108, 134, 135

retrieval correctness Accuracy of document ranking compared with gold documents and non-gold retrieved documents i, 69

retrieval relevance How well retrieved documents match the information need expressed in a query i, 69

retrieved documents Documents returned by retrieval for a query within the top k results i, 15, 69, 70

ROUGE A reference-based text generation metric that measures lexical overlap, often recall-oriented, between a generated output and one or more reference texts i, 72, 73

schema linking The step in text-to-SQL that maps elements of a natural-language query to relevant tables, columns, values, or relations in a database schema i, xix, 89, 93, 95, 101, 102, 107, 176

security trimming Filtering retrieval or search results so that users only receive documents they are authorized to access i, 26, 45, 108

serverless A cloud execution model in which the provider manages infrastructure allocation and scaling while code runs in response to requests or events i, 22, 62

source-available Software whose source code is publicly accessible but whose license still imposes restrictions that prevent it from being fully open source i, 2, 7

-
- technician assistant** An AI assistant that helps support technicians troubleshoot issues by retrieving and summarizing relevant historical ticket knowledge i, xiv, xv, xvii, xix, 5, 36, 37, 39, 40, 45–47, 50, 51, 54, 55, 58, 60, 61, 65, 67, 70, 87, 88, 103, 107, 108
- Text-to-SQL** Prompting an LLM to generate SQL i, xix, 2, 4, 5, 87, 88, 91–93, 102, 104, 107, 108, 174
- ticket enrichment** An automated workflow that adds contextual information such as summaries, references, or suggested next steps to a newly created support ticket i, vii, xv–xvii, xix, 2, 5, 36, 57, 60–63, 65, 67, 68, 70, 78, 83, 85, 87, 88, 103, 107, 143
- traceability** The ability to link a generated output back to the source documents, data, or processing steps that produced it i, xv, 23, 39, 40, 48, 52, 59, 60, 67
- webhook** An HTTP callback that automatically triggers a downstream process when a specific event occurs i, 62
- Zammad** An open-source ticketing and help desk system used in this thesis as a simulated support platform for integration testing i, xv, 50, 51, 62, 64, 66

Bibliography

- [1] haileytap, “Use ADLS Gen2 Indexer to Ingest Permission Metadata - Azure AI Search.” <https://learn.microsoft.com/en-us/azure/search/search-indexer-access-control-lists-and-role-based-access>.
- [2] A. Gan, H. Yu, K. Zhang, Q. Liu, W. Yan, Z. Huang, S. Tong, and G. Hu, “Retrieval Augmented Generation Evaluation in the Era of Large Language Models: A Comprehensive Survey,” Apr. 2025.
- [3] “Einzelkrediterhebung (EKE) - Erläuterungen & Erhebungsmittel.” <https://www.snb.ch/de/the-snb/mandates-goals/statistics/eke/eke-comments>.
- [4] B. Polonsky, “BorisPolonsky/dify-helm,” Feb. 2026.
- [5] “Tracing Data Model in Langfuse - Langfuse.” <https://langfuse.com/docs/observability/data-model>, Feb. 2026.
- [6] “Concepts - Langfuse.” <https://langfuse.com/docs/evaluation/core-concepts#experiments>, Jan. 2026.
- [7] “HS 2024 2025-SA-EP-Ammann-Ott-Retrieval Augmented Generation (RAG) Use-cases and Architec.”
- [8] Lukas Ammann, Sara Ott, “FS 2025-BA-EP-Ammann-Ott-Development of a scalable and secure RAG-as-a-Service infras,” June 2025.
- [9] E. Husakovic, M. Huber, and J. Rakic, “Design and Development of Retrieval-Augmented AI Assistants,” Dec. 2025.
- [10] H. Yu, A. Gan, K. Zhang, S. Tong, Q. Liu, and Z. Liu, “Evaluation of Retrieval-Augmented Generation: A Survey,” vol. 2301, pp. 102–120, 2025.
- [11] J. Chen, H. Lin, X. Han, and L. Sun, “Benchmarking Large Language Models in Retrieval-Augmented Generation,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 17754–17762, Mar. 2024.
- [12] ’jxnl’, “There Are Only 6 RAG Evals - Jason Liu.” <https://jxnl.co/writing/2025/05/19/there-are-only-6-rag-evals/>, May 2025.

-
- [13] “Metrics - TruLens.” https://www.trulens.org/getting_started/core_concepts/feedback_functions/.
- [14] “Overview - Ragas.” <https://docs.ragas.io/en/latest/concepts/metrics/overview/#types-of-metrics-in-ragas>.
- [15] H. Li, Q. Dong, J. Chen, H. Su, Y. Zhou, Q. Ai, Z. Ye, and Y. Liu, “LLMs-as-Judges: A Comprehensive Survey on LLM-based Evaluation Methods,” Dec. 2024.
- [16] “Observability and tracing for Dify - Langfuse.” <https://langfuse.com/integrations/no-code/dify>, Feb. 2026.
- [17] “Integrate with Langfuse.” <https://docs.dify.ai/en/use-dify/monitor/integrations/integrate-langfuse>.
- [18] “Gao-ai-com/dify-plugin-langfuse.” GAO, Inc., Mar. 2026.
- [19] “Observation Types - Langfuse.” <https://langfuse.com/docs/observability/features/observation-types>, Sept. 2025.
- [20] M. Deng, A. Ramachandran, C. Xu, L. Hu, Z. Yao, A. Datta, and H. Zhang, “ReFoRCE: A Text-to-SQL Agent with Self-Refinement, Consensus Enforcement, and Column Exploration,” June 2025.
- [21] Junjie.M, “Junjiem/dify-plugin-tools-dbquery,” Apr. 2026.
- [22] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, and X. Huang, “Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL,” Nov. 2025.
- [23] P. Wang, B. Sun, X. Dong, Y. Dai, H. Yuan, M. Chu, Y. Gao, X. Qi, P. Zhang, and Y. Yan, “Agentar-Scale-SQL: Advancing Text-to-SQL through Orchestrated Test-Time Scaling,” Dec. 2025.
- [24] M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, G. T. Kakkar, Y. Gan, A. Saberi, F. Ozcan, and S. O. Arik, “CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL,” Oct. 2024.
- [25] B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, L. Chai, Z. Yan, Q.-W. Zhang, D. Yin, X. Sun, and Z. Li, “MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL,” Mar. 2025.
- [26] M. Pourreza and D. Rafiei, “DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction,” Nov. 2023.
- [27] B. Li, C. Chen, Z. Xue, Y. Mei, and Y. Luo, “DeepEye-SQL: A Software-Engineering-Inspired Text-to-SQL Framework,” Apr. 2026.
- [28] Z. Li, X. Wang, J. Zhao, S. Yang, G. Du, X. Hu, B. Zhang, Y. Ye, Z. Li, R. Zhao, and H. Mao, “PET-SQL: A Prompt-Enhanced Two-Round Refinement of Text-to-SQL with Cross-consistency,” June 2024.

-
- [29] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, “Lost in the Middle: How Language Models Use Long Contexts,” *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.
 - [30] F. Lei, J. Chen, Y. Ye, R. Cao, D. Shin, H. Su, Z. Suo, H. Gao, W. Hu, P. Yin, V. Zhong, C. Xiong, R. Sun, Q. Liu, S. Wang, and T. Yu, “Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows,” Mar. 2025.
 - [31] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Cao, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. C. Chang, F. Huang, R. Cheng, and Y. Li, “Can LLM Already Serve as A Database Interface? A BIG Bench for Large-Scale Database Grounded Text-to-SQLs,” Nov. 2023.
 - [32] S. Ravi, S. F. A. Shah, T. Sheng, A. Bansal, J. Creighton, C. Lei, S. A. Somayajula, M. Liu, and D. Roth, “SOMA-SQL - OCI Generative AI Tops Spider 2.0 Lite, Strengthening its NL2SQL leadership for Enterprise Data.”
 - [33] C. Guo, Z. Tian, J. Tang, P. Wang, Z. Wen, K. Yang, and T. Wang, “Prompting GPT-3.5 for Text-to-SQL with De-semanticization and Skeleton Retrieval,” in *PRI-CAI 2023: Trends in Artificial Intelligence* (F. Liu, A. A. Sadanandan, D. N. Pham, P. Mursanto, and D. Lukose, eds.), (Singapore), pp. 262–274, Springer Nature, 2024.
 - [34] “Renovate Docs.” <https://docs.renovatebot.com/>.
 - [35] adr, “Madr/template/adr-template.md at 4.0.0 · adr/madr.” <https://github.com/adr/madr/blob/4.0.0/template/adr-template.md>.
 - [36] “External Knowledge API.” <https://docs.dify.ai/en/use-dify/knowledge/external-knowledge-api>.
 - [37] “Dify Marketplace.” <https://marketplace.dify.ai/>.
 - [38] “N8n and Langfuse - Langfuse.” <https://langfuse.com/integrations/no-code/n8n>, Feb. 2026.
 - [39] “Single-sign on (SSO) support [released] - Feature Requests (done).” <https://community.n8n.io/t/single-sign-on-sso-support-released/10726>, Jan. 2022.
 - [40] b. shen, “Bonashen/dify-sso,” May 2025.
 - [41] “Access Control.” <https://docs.dify.ai/en/use-dify/publish/webapp/web-app-access>.
 - [42] “Observability and Tracing for Flowise - Langfuse.” <https://langfuse.com/integrations/no-code/flowise>, Feb. 2026.
 - [43] “SSO | FlowiseAI.” <https://docs.flowiseai.com/configuration/sso>, June 2025.
 - [44] “Langgenius/dify.” LangGenius, Feb. 2026.
 - [45] X. Song, “Magicsong/ai-charts,” Apr. 2025.

Appendix

Appendix A

Infrastructure

A.1 Infrastructure

Figure A.1 shows a high-level overview of the deployment and its relationship to the other services.

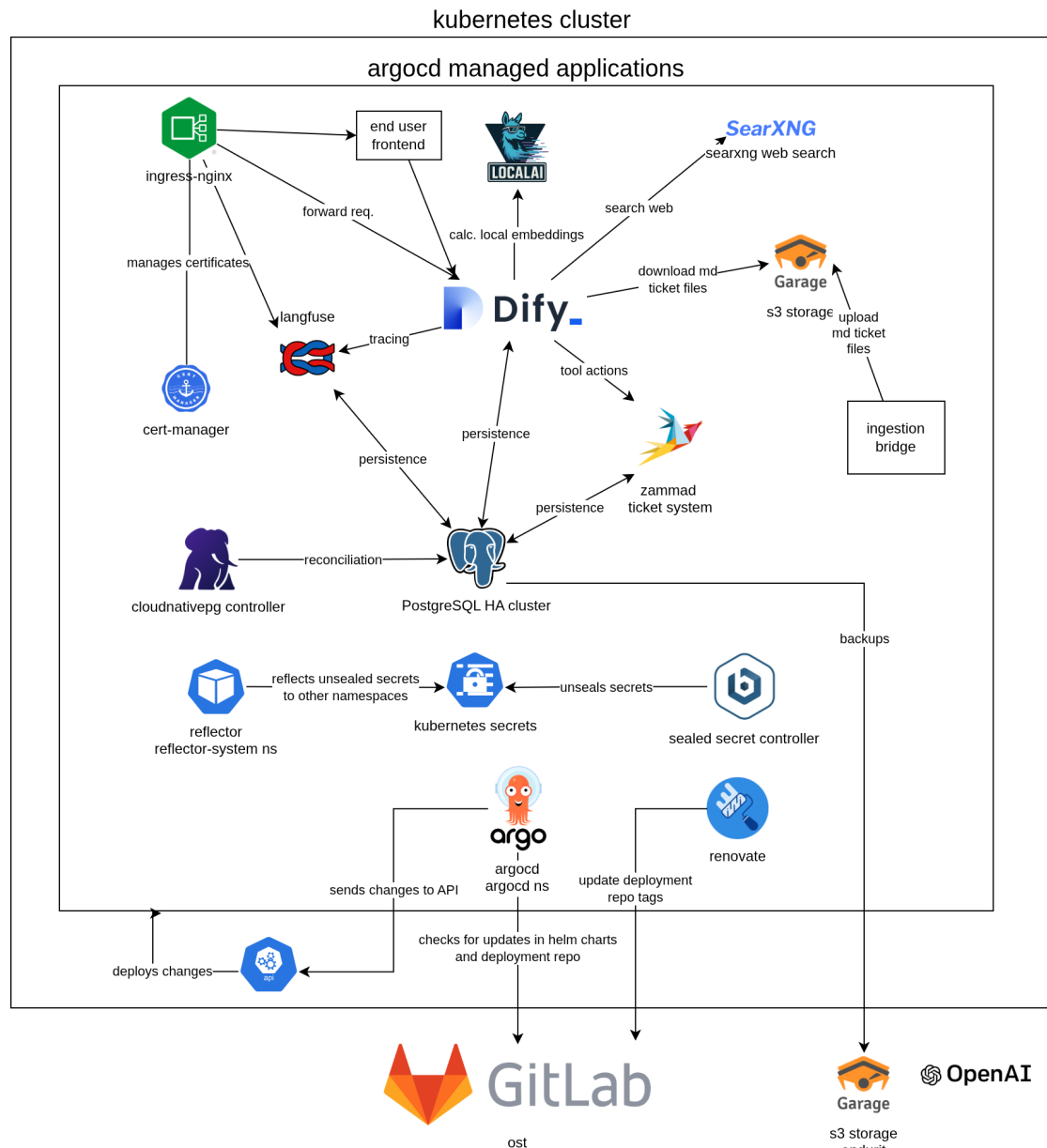


Figure A.1: Kubernetes deployment high-level overview and relationships to the services.
Source: own illustration

CloudNativePG allows a PostgreSQL cluster to be managed with an operator through Kubernetes resources. The overview can be seen in Figure A.2. Anti-affinity rules are set up so that the cluster instances are not scheduled on the same node.

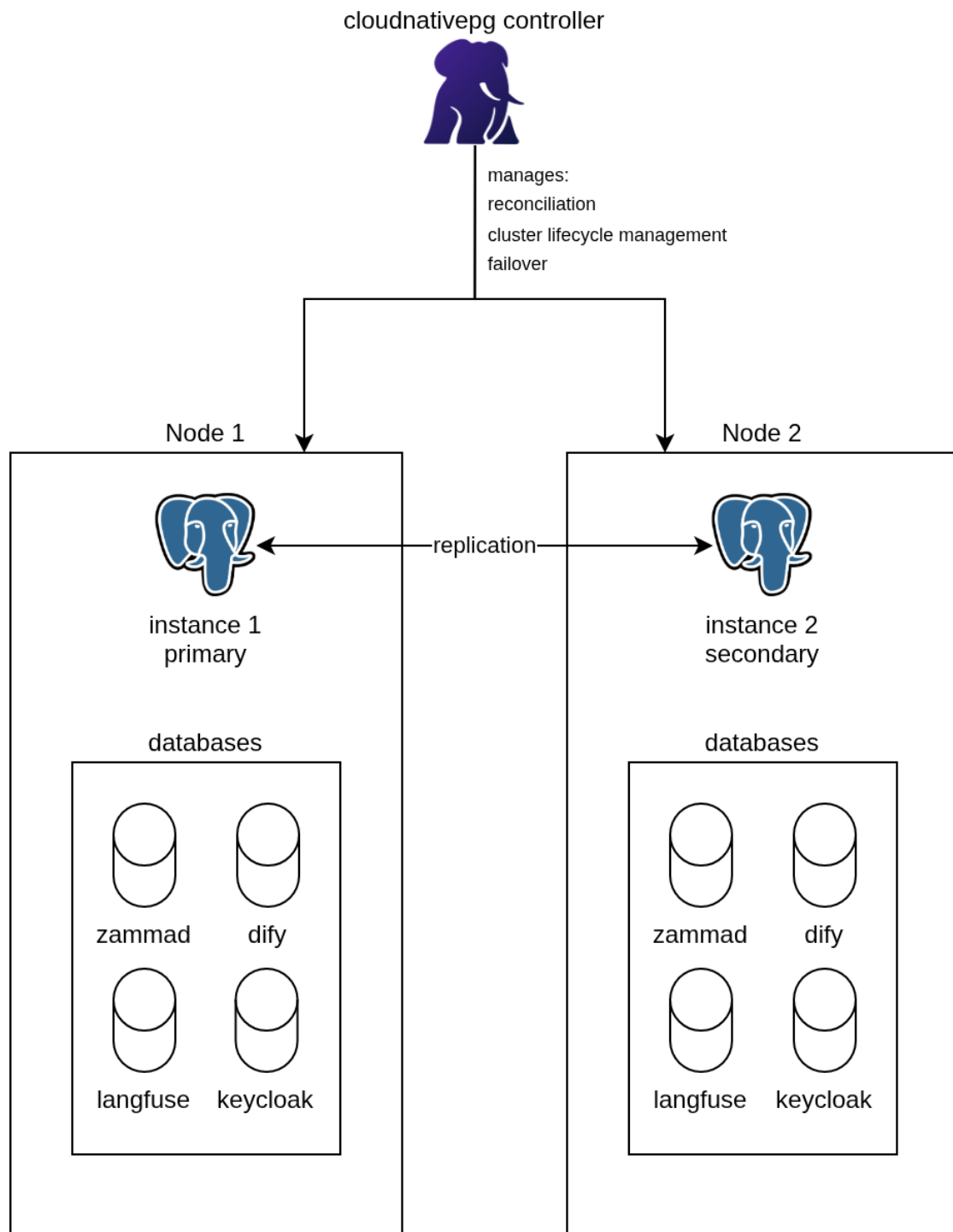


Figure A.2: This diagram illustrates the setup for PostgreSQL. The controller is also located on a node. Source: own illustration

A.1.1 Backups

The most important data is stored in the CloudNativePG cluster. We use the built-in backup mechanism to S3. Daily full backups and WAL archiving are configured for PITR recovery.

The data for all the other services can be rebuilt, such as Weaviate embeddings, so they do not need backups.

A.1.2 Patch Management

We use Renovate for automated dependency updates [34]. A cron job runs daily to check for new updates for Helm charts and Docker images via regex. If an update is found, Renovate creates a merge request with the release notes (if available). We can then merge the MR, and Argo CD automatically deploys the update.

A.1.3 Dify

The customized deployment architecture is shown in Figure A.3.

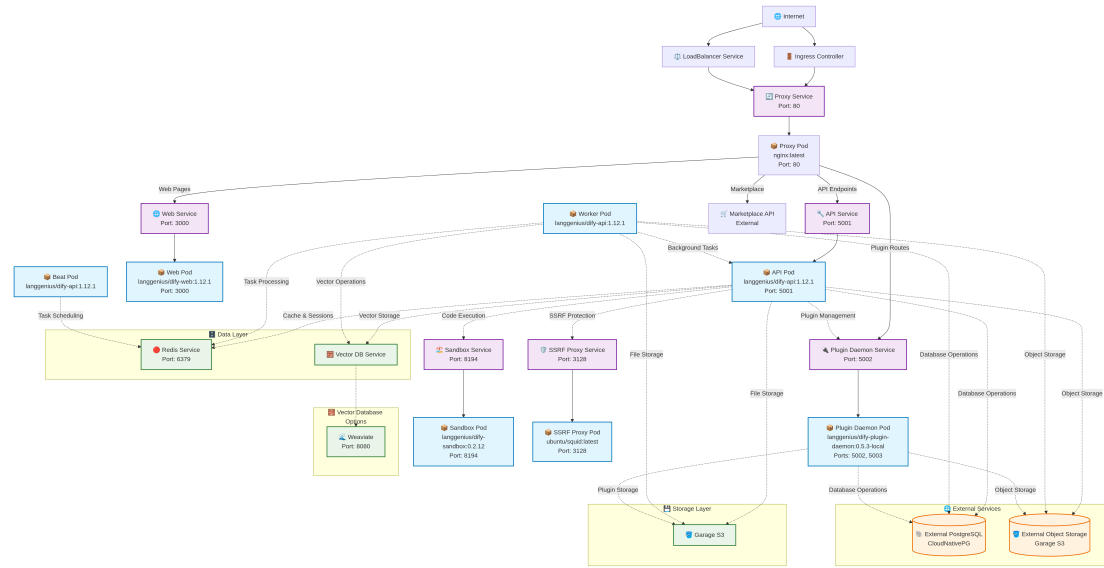


Figure A.3: Customized Dify deployment architecture. Source: adapted from [4]

Appendix B

ADR

The architecture decisions are based on the MADR template [35].

The goal of the ADRs is to prevent topics that have already been discussed from being reopened and discussed again. They also serve to document the thought process behind decisions.

For selected larger ADRs, we used multiple-criteria decision analysis. The options and criteria were maintained in CSV files and generated as weighted decision matrices.

The ADRs are mostly relevant to the Dify part.

B.1 Workflow Platform

Status accepted
Date 2026-03-11

B.1.1 Context and Problem Statement

In our SA [9], we built a workflow engine. Other projects and libraries exist that provide existing UIs and many integrations, which is why we want to reevaluate whether we want to continue with our custom approach or use an existing component. For some ADRs, it does not make sense to fill out every section.

B.1.2 Decision Drivers

- **Integrations with existing services:** would greatly expand the use cases for the project
- **Good UI:** should provide a usable visual workflow experience for developers and non-developers
- **Complexity:** both integration complexity and operational complexity have to be considered
- **Extensibility:** how easy will it be to extend in the future?
- **Licensing:** does the license allow the intended commercial hosting model?
- **Future-proofness:** how likely is it that the project will continue to exist in the future?
- **Community support:** how much community support exists for templates, plugins, and troubleshooting?
- **Observability and tracing:** is Langfuse supported and how deeply?
- **Authentication:** is external authentication such as OAuth2, OIDC or Security Assertion Markup Language (SAML) supported?
- **Database persistence and scaling:** does the platform support production-ready persistence and growth beyond local prototypes?
- **Our current features can be integrated**
 - separation between Client, Company, and Domain RAG
 - Company RAG separated by external organization (e.g., a ticketing system where tickets from external organizations are mixed)
 - MCP and tool calling

- definition of stages (e.g., a support bot with different prompts for a troubleshooting stage and an information-gathering stage for ticket creation)
- different tools and RAG sources can be defined per stage
- transitions based on natural language text (e.g., go to the information-gathering stage if the LLM sees that the problem is too complex)
- **Easily hostable in Kubernetes:** if a Helm chart is provided or something similar, this would greatly reduce our work.
- **Designed with multi-tenancy in mind**

B.1.3 Considered Options

- self-developed
- n8n
- Flowise
- Dify

B.1.4 Option Comparison (weighted)

Criterion (Weight)	Self-dev	n8n	Flowise	Dify	Evidence
Workflow fit (25%)	5	3	4	5	Stages tools and NL routing
Ecosystem (15%)	1	5	3	4	Connectors plugins and templates
Delivery effort (15%)	1	4	4	4	New platform work needed
observability fit (10%)	5	2	4	5	Langfuse tracing and prompt tooling
Extensibility (10%)	5	3	4	4	Room for future custom features
Licensing fit (10%)	5	1	3	3	Planned self-hosted commercial model
Hosting readiness (10%)	2	4	3	4	Kubernetes persistence and scaling
Community (5%)	1	5	3	4	Adoption and maintenance outlook
Weighted total	3.30	3.35	3.60	4.25	

Table B.1: Weighted ADR option comparison

Scoring scale 1 means weak fit, 3 means acceptable fit, and 5 means strong fit.

B.1.5 Decision Outcome

Decision We choose **Dify** as the workflow platform for the BA implementation.

Rationale Dify provides the best overall fit for the required agentic RAG workflow model. It combines a mature UI, native Langfuse tracing support [16], an external knowledge interface [36], a large plugin ecosystem [37], and strong support for tool-based

and stage-like workflow design. Compared with the alternatives, n8n offers the broadest integration ecosystem but is weakened by its licensing model and weaker tracing story.

The weighted comparison leads to the same result: Dify reaches the highest weighted score (4.15), ahead of Flowise (3.60), n8n (3.35), and self-developed (3.30). The largest advantages are workflow fit, Langfuse integration, and fast implementation without giving up too much extensibility.

Consequences The project benefits from a mature workflow UI, faster implementation, built-in observability, and a plugin-based extension model instead of having to build these foundations ourselves.

The main downside is that Dify introduces higher operational complexity than a lightweight custom service, and its licensing requires care around multi-tenant SaaS operation. Shared multi-tenant hosting must not be assumed as a safe default. If this becomes a product requirement, we either need explicit authorization from Dify or stronger tenant isolation, such as per-customer deployments.

B.1.6 Pros and Cons of the Options

Option self-developed

- -- integrations have to be implemented manually, unless another platform is additionally used for integrations
- +++ highest flexibility and customizability, new workflow concepts can be implemented as first-class features
- +++ highest extensibility, because the data model and execution model can be adapted directly
- +++ licensing can be self-defined, so commercial hosting is not restricted by third-party platform terms
- +++ observability and tracing can be integrated as deeply as desired
- +++ any authentication method can be used
- +++ exact support for our current stage-based RAG and tool-calling model is possible
- +++ can be designed with multi-tenancy in mind from the start
- -- weaker integration ecosystem than established platforms
- -- Kubernetes hosting is possible, but all operational maturity has to be built and maintained by us
- --- there is no external community, template ecosystem, or plugin marketplace

- --- future-proofness depends completely on our own team capacity and long-term commitment
- --- bigger development effort
 - more effort has to be put into onboarding because no public resources are available
- --- high business risk for stakeholders because of uncertainty about long-term continuation

Option n8n

- +++ strongest integration ecosystem and template community (422+ apps / services)
- ++ good UI
- ++ official Kubernetes support is available through the official Helm chart, with scale-out options
- ++ supports MCP and AI agent patterns
- ++ has the most contributors (184k stars)
- + Langfuse does not fully support it officially for tracing, but there are community integrations for prompt management and related workflows [38]
- + supports SSO-related features such as Keycloak integration [39]
- + production persistence options exist, including PostgreSQL
- -- stage semantics are not first-class and have to be modeled indirectly with workflow graph logic
- --- licensing is a major concern for hosted or white-labeled commercial use cases
- -- enterprise governance features are not as strong in the open community setup
- -- weaker Langfuse tracing fit than Dify or Flowise

Option Dify

- + many integrations, as of 2026-02-14 [37] (23 data sources, 19 triggers, 50+ tools, 111 supported model plugins, 7 agentic strategies, 30 extensions)
- ++ good UI
- ++ Langfuse supports it [16]

- ++ supports external knowledge API [36]
- + supports Langfuse prompt management [16]
- +++ best fit for stage-like agentic workflows and natural-language routing
- +++ supports MCP and tool calling well
- ++ supports knowledge separation strategies through multiple knowledge bases and metadata-based filtering
- ++ plugin model is well aligned with future extensions
- + has many contributors and 130k GitHub stars
- + Helm charts exist [4]
- unofficial dify-sso with an OIDC provider: [40]; Dify docs also mention SSO: [41]
- + production-ready persistence and supporting infrastructure are part of the documented self-hosted setup
- -- operationally more complex than a lightweight workflow engine because Dify is a multi-service platform
- --- licensing must be reviewed carefully for shared multi-tenant SaaS operation
- -- classical SaaS connector breadth is weaker than n8n

Option Flowise

- ++ good support for agentic workflows and natural-language branching
- ++ MCP support
- ++ good UI
- ++ Langfuse supports it [42]
- + OIDC support [43]
- + extensibility is good through node-based customization
- + can support metadata-based retrieval separation
- -- workspaces and stronger multi-tenancy features are gated behind enterprise offerings
- -- SSO is also primarily an enterprise feature
- -- community size and connector breadth are weaker than n8n

B.2 Ingestion Bridge, Dify Knowledge Pipeline or Script

Status accepted
Date 2026-04-17

B.2.1 Context and Problem Statement

In the SA, we built the Ingestion Bridge for ingesting new knowledge. Dify has its own knowledge pipeline. Some of the features of the ingestion bridge and Dify knowledge pipeline overlap, so the question is which ingestion approach should be used for the BA implementation.

The BA implementation needs a practical way to ingest support-ticket knowledge into Dify. The ingestion approach should be simple enough for the scope of the thesis, but still support the required preprocessing and metadata needs.

Three options were considered. The existing Ingestion Bridge from the SA could be reused, Dify's built-in knowledge pipeline could be used, or a small purpose-built ingestion script could be implemented.

B.2.2 Decision Drivers

- **Implementation effort:** the ingestion approach should not consume too much BA time.
- **Chunking control:** chunk sizes should fit the embedding model, ideally with token-aware splitting.
- **Metadata support:** documents and chunks should be ingestible with metadata that can later be used for filtering and access-control experiments.
- **Operational complexity:** additional services should only be introduced if they provide clear value for the BA.
- **Production readiness:** the chosen approach should be evaluated with regard to whether it can be reused beyond the thesis prototype.

B.2.3 Considered Options

- Ingestion Bridge
- Dify knowledge pipeline
- Purpose-built ingestion script

B.2.4 Decision Outcome

We chose a **purpose-built ingestion script** for the BA implementation.

The script was the best fit because the BA did not require many custom production features, but did require more control than the Dify knowledge pipeline offered. It avoided the main limitations of the Dify knowledge pipeline, especially character-based chunking and limited metadata handling, while remaining much simpler than operating the full Ingestion Bridge as a separate service.

B.2.5 Consequences

The ingestion logic is easy to understand, change, and run during development. It also keeps the implementation independent of Dify's visual ingestion limitations, which was important for testing a local embedding model and metadata-based retrieval behavior.

The downside is that the script is not a production ingestion architecture. It has no long-running service model, no operational monitoring, no built-in retry handling, and no versioning workflow. If the system were moved toward production, the Ingestion Bridge or a similar managed ingestion service would likely have to be introduced.

B.2.6 Pros and Cons of the Options

Option Ingestion Bridge

- ++ More flexible than the Dify knowledge pipeline.
- ++ Could support production ingestion concerns such as synchronization, retries, and controlled preprocessing.
- + Would probably be needed anyway if versioning of ingested knowledge becomes a requirement.
- -- Separate service that has to be hosted, operated, and maintained.
- -- Higher implementation and infrastructure complexity than needed for the BA prototype.

Option Dify knowledge pipeline

- ++ Simple to use for basic ingestion scenarios.
- + Integrated directly into Dify, so no additional ingestion component has to be hosted.
- -- chunking is character-based rather than token-aware. This was problematic with the local embedding model because the relationship between characters and tokens had to be estimated.

- -- Metadata could not be added in the required way through the knowledge pipeline, which limited later metadata-based filtering and access-control experiments.

Option purpose-built ingestion script

- ++ Easy to implement and adapt for the BA dataset.
- ++ Avoids the Dify knowledge pipeline limitations around token-aware chunking and metadata handling.
- ++ Does not require hosting another service.
- -- Not suitable as a production ingestion solution.
- -- Missing production features such as monitoring, retries, scheduling, and versioning.

B.3 Dify RAG or External Knowledge Base

Status accepted
Date 2026-04-02

B.3.1 Context and Problem Statement

Dify has its own RAG store, which it calls knowledge base. There are many possible RAG techniques and retrieval technologies that could also be implemented externally, for example more advanced indexing, graph-based retrieval, or custom multi-stage retrieval. This leads to the question of whether the BA implementation should use Dify’s built-in knowledge base or an external knowledge base.

B.3.2 Decision Drivers

- **Development effort:** the project should focus on evaluating agent workflows, not on building a complete custom RAG platform.
- **Customizability:** the solution should still allow the required retrieval behavior for the BA use cases.
- **Supported RAG techniques:** built-in retrieval, hybrid search, reranking, and metadata filtering should be sufficient for the prototype.
- **Supported RAG technologies:** the solution should integrate with the chosen Dify setup and vector-store configuration.
- **Knowledge separation:** Domain RAG, Company RAG, and Client RAG separation should remain possible.
- **Production outlook:** limitations that would matter in production should be documented, even if they are not solved in the BA.

B.3.3 Considered Options

- Dify built-in RAG / knowledge base
- External RAG / knowledge base

B.3.4 Decision Outcome

We chose **Dify’s built-in RAG / knowledge base** for the BA implementation.

The main reason was simplicity. The BA did not require enough custom RAG functionality to justify implementing and operating an external retrieval system. Dify’s knowledge base already provided the necessary baseline features, including built-in indexing, retrieval, hybrid search, reranking, and integration into Dify workflows. More complex techniques such as RAPTOR, GraphRAG, or custom ablation-oriented retrieval pipelines were interesting, but they were outside the available implementation time.

B.3.5 Consequences

Dify’s knowledge base allowed the thesis to spend more effort on workflow design, secure retrieval behavior, evaluation, and comparison with Microsoft Foundry. It also reduced the number of custom services that had to be built and hosted.

The trade-off is reduced flexibility. Advanced retrieval techniques, deeper ablation studies, and complex metadata-filtering logic are harder to implement inside Dify’s built-in RAG system. If the project moves toward production or toward research-heavy retrieval optimization, an external RAG service could become necessary.

B.3.6 Pros and Cons of the Options

Option Dify built-in RAG / knowledge base

- +++ Low development effort for the built-in retrieval path.
- ++ Direct integration with Dify workflows and Dify’s knowledge-retrieval nodes.
- ++ Supports useful baseline RAG features such as hybrid search, reranking, parent-child chunking, and metadata filtering.
- + Knowledge separation can be modeled through multiple knowledge bases and metadata filters.
- -- Some RAG techniques require additional workflow logic, code nodes, or external preprocessing.
- -- Complex metadata filtering is limited compared with a custom retrieval service.
- -- Advanced RAG techniques such as RAPTOR or GraphRAG are not available as native built-in features.

Option External RAG / knowledge base

- +++ High customizability for new RAG techniques such as RAPTOR.
- +++ High customizability for new RAG technologies such as GraphRAG.
- ++ Ablation studies and retrieval experiments would be easier because the retrieval pipeline could be controlled directly.
- ++ Complex metadata filtering and access-control logic could be implemented without being constrained by Dify’s knowledge base UI and API abstractions.
- -- Higher development effort because retrieval, indexing, metadata handling, and Dify integration have to be implemented manually.
- -- Additional service that has to be hosted, monitored, and maintained.

- -- Too much scope for the BA implementation because no complex retrieval technique was required in the final prototype.

B.4 Dify Helm Chart - BorisPolonsky/dify-helm

Status accepted
Date 2026-02-22

B.4.1 Context and Problem Statement

After deciding to use Dify, we needed to determine how to host it. The Dify repository [44] lists a few methods for hosting on Kubernetes, some of which are Helm charts. Developing a Helm chart for Dify is not a considered option because the existing Helm charts are already very mature.

B.4.2 Decision Drivers

- completeness of the infrastructure included in the Helm chart required for Dify
- Autoscaling
- community support
- ease of installation

B.4.3 Considered Options

- BorisPolonsky Helm chart [4]
- magicsonk Helm chart [45]
- LeoQuote Helm chart [40]

B.4.4 Option Comparison (weighted)

Criterion (Weight)	Boris	LeoQuote	magicsong	Evidence
Completeness of Dify infrastructure (35%)	5	3	1	Support for complete Dify stack and external infra swap options
Release freshness and maintenance cadence (25%)	5	3	1	Recent default chart/app versions and active updates
Autoscaling support (20%)	5	2	1	Built-in Horizontal Pod Autoscaling support and maturity
Community activity (10%)	4	3	1	Contributors and repository activity
Installation and operational effort (10%)	5	4	3	Effort to install run and maintain in Kubernetes
Weighted total	4.90	2.90	1.20	

Table B.2: Weighted ADR option comparison

Scoring scale 1 means weak fit, 3 means acceptable fit, and 5 means strong fit.

B.4.5 Decision Outcome

We chose the BorisPolonsky Helm chart because it reaches the highest weighted score (4.90), compared to LeoQuote (2.90) and magicsong (1.20). The strongest differences are feature completeness, release freshness, and autoscaling support.

Consequences If community support fades, we would have to continue developing it ourselves.

B.4.6 Pros and Cons of the Options

BorisPolonsky Helm chart

- + actively developed
- + most complete Helm chart
- + options to swap external infrastructure such as Postgres / Weaviate

- + active releases (latest default version 1.12.1, which is the Dify release from three weeks ago)
- + 38 contributors, 614 stars
- + easy to install and run
- + built-in Horizontal Pod Autoscaling
- - no company behind it

LeoQuote Helm chart

- + actively developed
- + the rather large Chinese company Douban is behind it, so as long as they are using the Helm chart, it will most likely keep getting updated
- + easy to install and run
- - 23 contributors and 123 stars, but the repository also includes other Helm charts
- - latest default release is 2025-12-05 (1.10.1); this could be changed in the Helm chart, but additional work might be required

magicsong Helm chart

- --- last release was on 2025-04-11

B.5 S3-Compatible Storage - Garage

Status accepted
Date 2026-02-25

B.5.1 Context and Problem Statement

To store the Dify API data in a Highly Available (HA) way, we require S3 storage. We want to host it in our Kubernetes cluster, but we do not want to spend too much effort on infrastructure, as this would distract from the core BA work.

B.5.2 Decision Drivers

- **High availability** should be supported with our current setup. This means the platform has to support native replication, because our underlying storage layer is not replicated.
- **Simplicity**: the solution should be lightweight and not too complex to set up, so infrastructure work does not consume too much BA time.
- **Familiarity**: familiarity with the platform is a bonus for the same reason as above.
- **Kubernetes Helm chart**: a Kubernetes Helm chart should exist and be well supported.
- **Dify support**: although the options are S3-compatible, some of them do not implement the full S3 API. Whether they work with Dify can often only be confirmed by testing.

B.5.3 Considered Options

- Ceph RGW
- MinIO
- s3gw
- Garage
- SeaweedFS

B.5.4 Decision Outcome

We chose **Garage** as the S3-compatible storage solution for Dify API data.

Garage was the best fit for the BA infrastructure because it provides native replication, is comparatively lightweight, has Kubernetes deployment support through a Helm chart, and was already familiar to parts of the team. This matched the goal of improving availability without spending too much thesis time on operating a larger storage platform.

B.5.5 Consequences

Garage gives the deployment an S3-compatible storage backend that can replicate data independently from the underlying node storage. This reduces the risk of data loss if a node fails and keeps the setup simpler than operating a full Ceph/Rook environment.

The trade-off is that Garage is less commonly used than MinIO or Ceph RGW and has a smaller ecosystem. Compatibility with Dify must therefore be tested during operation, because S3-compatible systems do not always implement the complete S3 API surface in the same way.

B.5.6 Pros and Cons of the Options

Ceph RGW

- - would require additional infrastructure investment, such as setting up Rook-Ceph
- - complex to set up
- + has been used by some members of the team
- + Kubernetes Helm chart

MinIO

- ++ high-availability setup is possible in our environment
- + simple to set up
- ++ multiple members of our team are familiar with it
- - the future of community support is uncertain

s3gw

- --- no native replication method
- + simple to set up
- + has a Helm chart
- - no one on the team has used it

Garage

- ++ native replication method
- + simple to set up
- + has a Helm chart
- + some members of the team are familiar with it

SeaweedFS

- - no one on the team has used it

Appendix C

Evaluation

C.1 Golden Dataset Schema for Ticket Enrichment

Each golden dataset item follows the Langfuse dataset structure used for offline benchmarking. The input contains the user-facing query. The expected output contains both the reference response for generation evaluation and the gold documents required for retrieval metrics.

input:

- **query:** realistic user query formulated from the ticket content

expected_output:

- **response:** reference answer used as ground truth for generation correctness
- **gold_documents:** relevant tickets or knowledge-base articles used for retrieval metrics

metadata:

- **use_case:** ticket enrichment
- **source_ticket_id:** anonymized identifier of the ticket from which the sample was derived
- **labels:** one or more labels from the ticket taxonomy, for example *clear problem*, *clear solution*, *recurring problem*, *references*, *simple knowledge*, *company knowledge*, *missing information*, *workaround only*, *guided resolution*, *system action required*, *needs human escalation*, or *bug report*

C.1.1 Example Golden Dataset Item

The following anonymized and shortened example illustrates how one golden dataset item is represented. Ellipses indicate omitted content.

```
input:
query: "Seit dem Sommer-Update 2024 brechen RDP-Verbindungen zu einem
Terminalserver regelmaessig ab."

expected_output:
response:
"# Assessment

Die Stoerung trat nach einem Windows-Update auf und betrifft RDP-Verbindungen
zu einem Terminalserver. Aehnliche fruehere Faelle zeigen, dass RDP- oder
RDS-Sitzungen nach Updates abbrechen koennen. In einem Fall half ein spaeteres
Update, waehrend ein reines Zuruecksetzen nicht dauerhaft erfolgreich war.

# Relevante fruehere Faelle

- ticket_A_rdp_dienst_nach_update.md: RDP-Dienst beendet Sitzungen nach einem
Windows-Update; Problem wurde als bekanntes Update-Verhalten dokumentiert.
- ticket_B_rds_verbindungsabbrueche.md: Wiederholte RDS-Abbrueche nach
Patch-Installation; Stabilisierung erst nach Folgeupdate.

# Fehlende Informationen / naechste Fragen

- Sind alle Benutzer oder nur einzelne Konten betroffen?
- Tritt der Fehler bei allen Terminalservern oder nur bei einem Server auf?
- Gibt es eine konkrete Fehlermeldung oder Eventlog-Eintraege?

# Naechste Schritte

- Betroffenen Benutzerkreis und Server eingrenzen.
- RDP-/RDS-Eventlogs und installierte Updates pruefen.
- Bekannte Probleme zum installierten Update gegenpruefen.
- Falls verfuegbar, aktuelles Folgeupdate testen.

..."

gold_documents:
- ticket_A_rdp_dienst_nach_update.md
- ticket_B_rds_verbindungsabbrueche.md

metadata:
use_case: ticket_enrichment
source_ticket_id: ticket_C_rdp_verbindungsabbrueche.md
labels:
- clear problem
- clear solution
- company knowledge
- recurring problem
```

C.2 llm-as-a-judge prompts

Bewerte die Korrektheit der Generierung auf einer kontinuierlichen Skala von 0 bis 1. Eine Generierung kann als korrekt betrachtet werden (Score: 1), wenn sie alle Schlüsselfakten aus der Ground Truth enthält und wenn jede in der Generierung dargestellte Tatsache entweder durch die Ground Truth oder durch gesunden Menschenverstand faktisch gestützt wird.

Beispiel:

User Query: Verbessert das regelmässige Löschen von Cache-Dateien die Leistung meines Computers erheblich?

Generation: Ja, das regelmässige Löschen von Cache-Dateien verbessert die Leistung deines Computers drastisch. Wenn du deinen Cache oft genug leerst, wird dein Computer immer wie neu laufen. Menschen, die das nicht tun, haben zwangsläufig langsame Systeme. Unternehmen wollen oft nicht, dass du das weisst, weil sie dir lieber neue Hardware verkaufen. In Wirklichkeit ist Cache der Hauptgrund für langsame Computer, und das Löschen löst fast alle Leistungsprobleme.

Ground Truth: Teilweise. Das Löschen von Cache-Dateien kann in bestimmten Fällen Speicherplatz freigeben und kleinere Leistungsprobleme beheben, insbesondere wenn veraltete oder beschädigte Daten vorliegen. Allerdings ist Cache grundsätzlich dazu da, Prozesse zu beschleunigen, indem häufig genutzte Daten schneller verfügbar sind. Das regelmässige Leeren des Caches kann daher auch kurzfristig die Leistung verschlechtern, da Daten neu geladen werden müssen. Die Gesamtleistung eines Computers hängt von vielen Faktoren ab, darunter Hardware, laufende Prozesse und Systemkonfiguration.

Score: 0.2

Begründung: Die Generierung enthält einen kleinen wahren Kern (Cache kann Einfluss auf Leistung haben), übertreibt jedoch stark die Wirkung und stellt falsche allgemeine Behauptungen auf. Sie ignoriert die eigentliche Funktion von Cache und weicht deutlich von der differenzierten Erklärung der Ground Truth ab.

[USER_QUERY]

query

[/USER_QUERY]

[GENERATED_ANSWER]

generation

[/GENERATED_ANSWER]

[GROUND_TRUTH]

ground_truth

[/GROUND_TRUTH]

C.3 Labeling Prompt for ML Classification

The following prompt was used to ground the automated labeling of support tickets in the machine learning backend. It was submitted to the language model together with each ticket's problem description and resolution text, instructing the model to select all applicable labels from a fixed taxonomy. The prompt was designed to enforce mutual exclusivity constraints between certain label pairs and to reduce ambiguity by providing explicit inclusion and exclusion criteria for each label.

```

You are a support ticket classifier for an IT support company.
The ticket may be in German or English.

Ignore all email signatures, forwarded message chains, template
placeholders in <> brackets, and boilerplate greetings. Focus only
on the actual problem and solution steps.

First read the SOLUTION and reason about what was done. Then glance
at the PROBLEM to confirm context. Then select ALL labels that apply
from: labels

LABELS:
clear problem          = the problem is specific and actionable.
                        Apply by default unless genuinely incomprehensible.
clear solution         = concrete steps were taken and listed.
                        Apply by default unless the outcome is completely
                        unclear.
recurring problem     = explicitly affects multiple users or has happened
before.
references            = ONLY if another ticket number or case ID is
explicitly            cited. Not a person, not an email thread.
simple knowledge alone, = solvable with public docs or generic IT knowledge
no company context needed.
NEVER combine with company knowledge.
company knowledge     = requires internal systems, client environments, or
company-specific tools like M365, Datto, vCenter,
Citrix, AD. When in doubt pick this.
NEVER combine with simple knowledge.
needs human escalation = ONLY if physical on-site presence was the only
option.
Phone calls and remote sessions are NOT escalation.
This label is rare.
missing information   = ONLY if too vague to understand what happened.
Brief solution steps are enough to exclude this.
workaround only      = root cause was not fixed, only the symptom addressed.
bug report            = reproducible software or system defect worth tracking
by a developer.

```

```
guided resolution      = resolved remotely by walking user through steps,  
                        testing, or troubleshooting.  
system action required = a backend action was performed: password reset,  
account  
                        creation, permission change, DNS entry, reboot,  
                        license  
                        assignment. Almost never the only label.  
  
SOLUTION: resolution  
  
PROBLEM: task_description
```

C.4 Technical evaluation workflows

This appendix contains the workflow diagrams that complement the evaluation setup described in Chapter 8. They are useful for documenting the technical realization of the evaluation pipeline, but they are not required to understand the methodological argument in the main text.

C.4.1 Evaluation setup

This section documents the technical details of the evaluation setup with Langfuse and Dify. In Langfuse, traces can be aggregated into sessions, as shown in Figure C.1. A session is a group of traces that belong to the same user interaction, for example a chat thread. One trace contains multiple observations. Langfuse distinguishes observation types such as event, span, generation, agent, tool, chain, retriever, evaluator, embedding, and guardrail [19].

Langfuse supports offline evaluation on fixed datasets through experiments. As shown in Figure C.4, there are multiple execution modes, including combinations with local datasets or Langfuse-managed datasets. In this thesis, Langfuse-managed datasets are used. The basic experiment objects are summarized in Table C.2. The different automated evaluation options are summarized in Table C.1. The decision is therefore not which option is universally best, but which option is appropriate at which stage of the evaluation process.

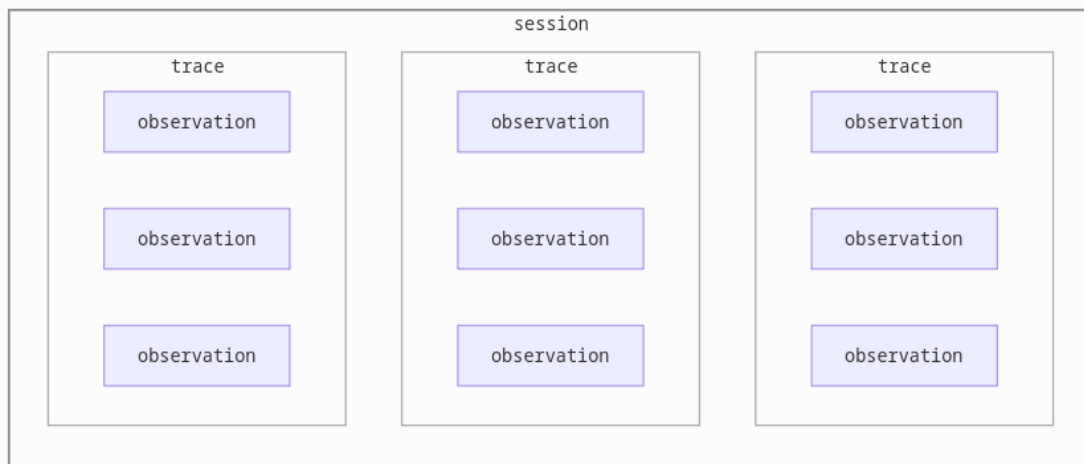


Figure C.1: Langfuse observation, trace, and session overview source: [5]

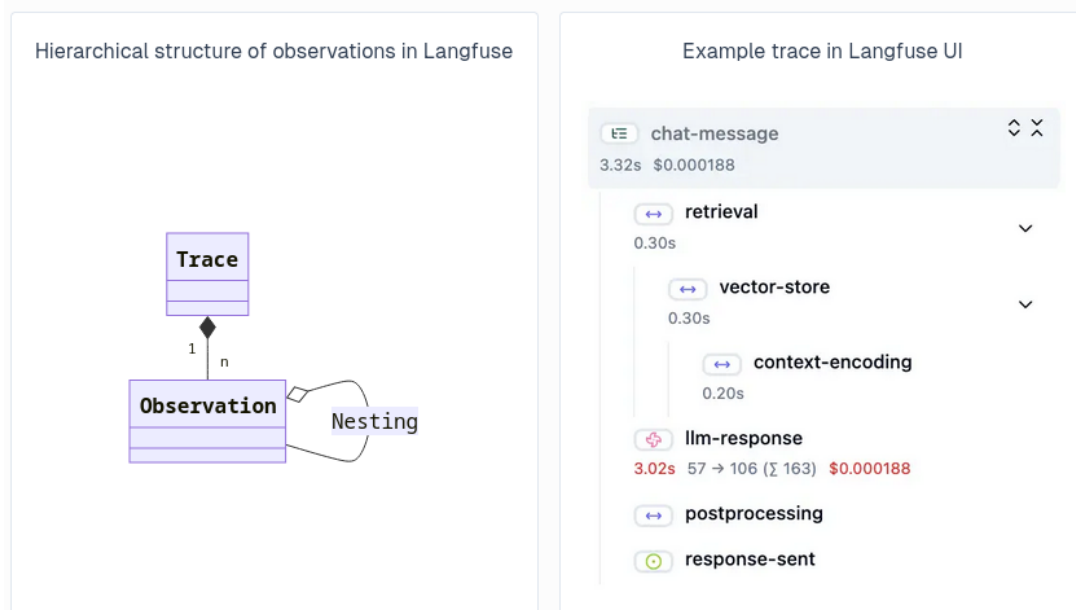


Figure C.2: Langfuse trace source: [5]

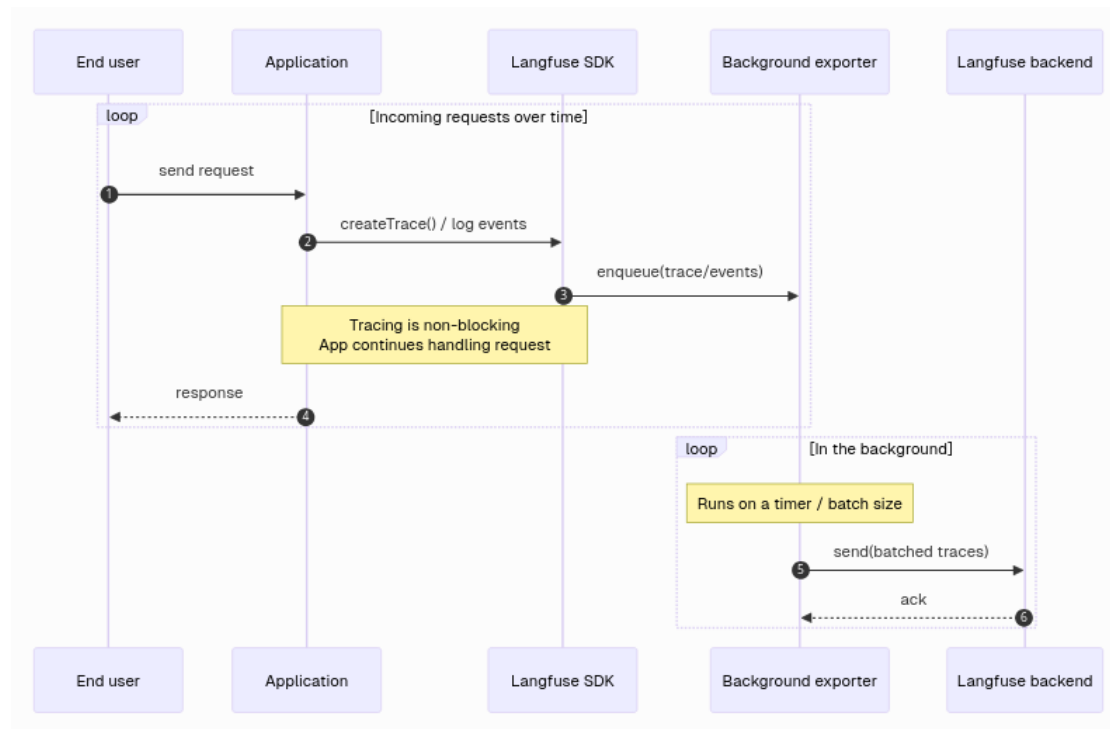


Figure C.3: Langfuse overview source: [5]

Table C.1: Possible automated evaluation options

Option	online evalua- tion	offline evalua- tion	reference based?	custom scores
llm-as-a-judge evaluator in Lang- fuse	Yes	No	No	Langfuse pro- vided only
llm-as-a-judge with custom ground truth	No	Yes	Yes	Langfuse pro- vided only
custom metric	(Yes)	Yes	Yes	depends on the metric

	Langfuse Execution	Local/CI Execution
Langfuse Dataset	Experiments via UI	Experiments via SDK
Local Dataset	Not supported	Experiments via SDK

Figure C.4: Langfuse dataset execution matrix source: [6]

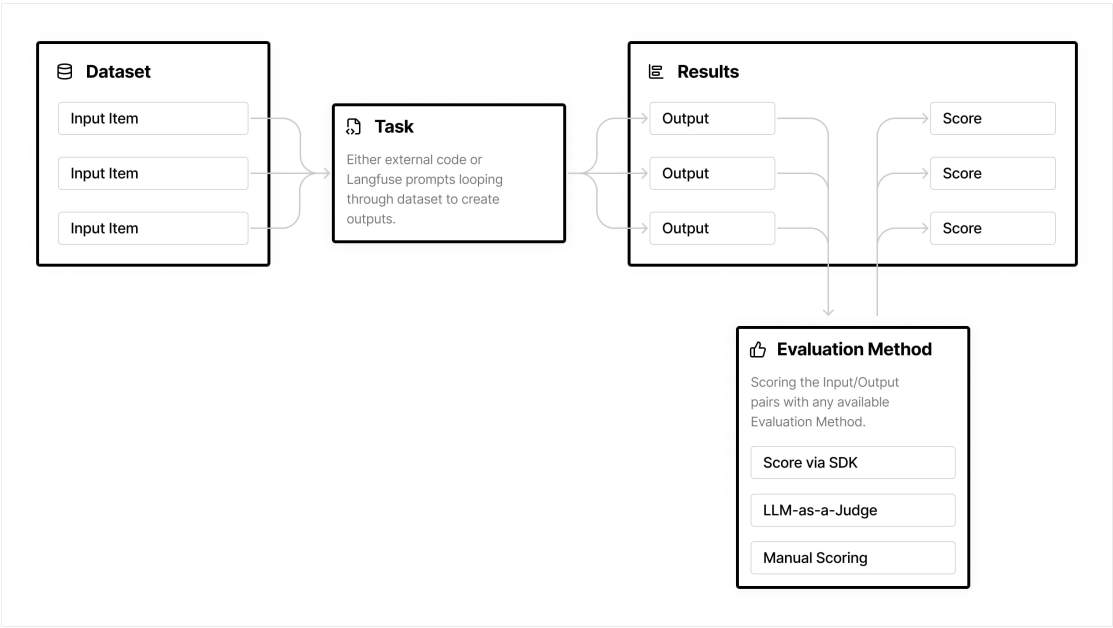


Figure C.5: Langfuse experiment flow source: [6]

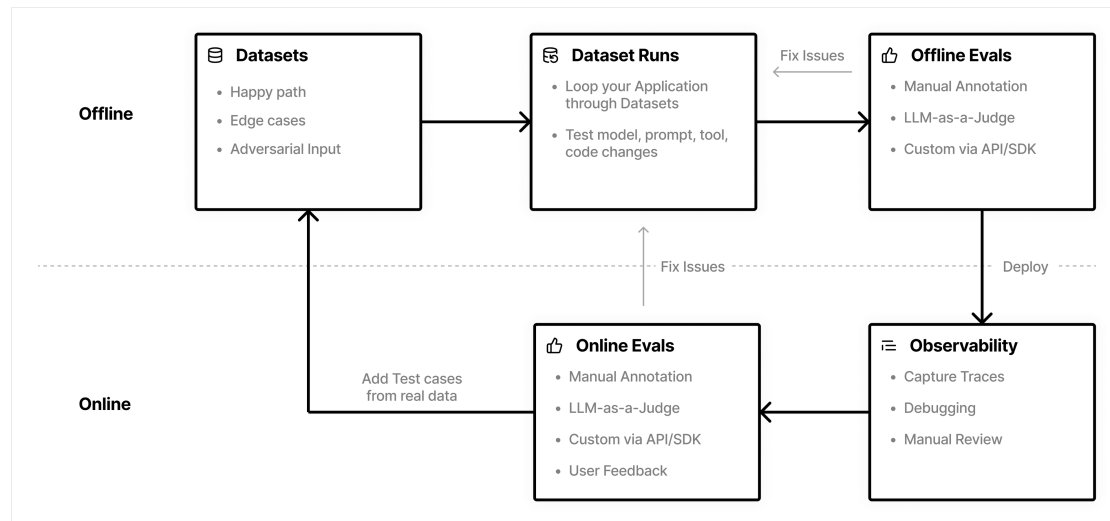


Figure C.6: Langfuse online offline loop source: [6]

Table C.2: Langfuse experiment objects and definitions source: [6]

Object	Definition
Dataset	A collection of test cases (dataset items). You can run experiments on a dataset.
Dataset item	One item in a dataset. Each dataset item contains an input (the scenario to test) and optionally an expected output.
Task	The application code that you want to test in an experiment. This is performed on each dataset item, and you score the output.
Evaluation Method	A function that scores experiment results. In the context of a Langfuse experiment, this can be a deterministic check or llm-as-a-judge.
Score	The output of an evaluation. This can be numeric, categorical, or boolean. See Scores for more details.
Experiment Run	A single execution of your task against all items in a dataset, producing outputs and scores.

Dify Langfuse Integration

Dify has an official Langfuse integration that is used for tracing and observability [16, 17]. Most, if not all, Dify nodes can be traced with it. In addition, the community plugin for Langfuse prompt management in Dify is used. This makes it possible to manage prompt

versions centrally [18], which is useful when the same prompt is reused across multiple workflows and should remain comparable over time.

C.4.2 Managed live evaluator

Everything is run within Langfuse on either incoming online traces or existing traces, as illustrated in Figure C.7.

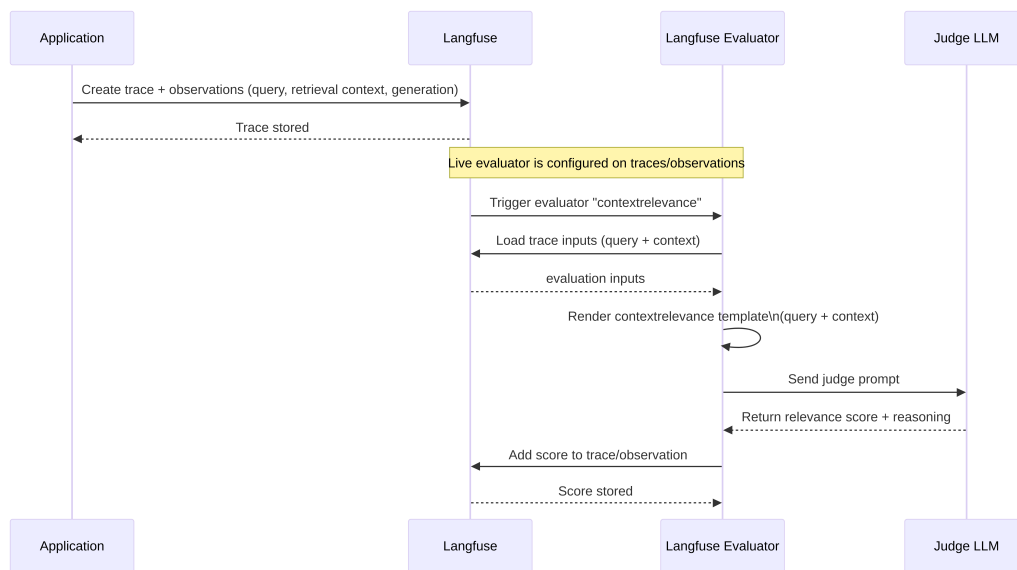


Figure C.7: LLM-as-a-judge live evaluator

C.4.3 Dataset-Based Evaluator with Ground Truth

In this scenario, a dataset and a Langfuse evaluator that contains ground-truth data are used, as illustrated in Figure C.8.

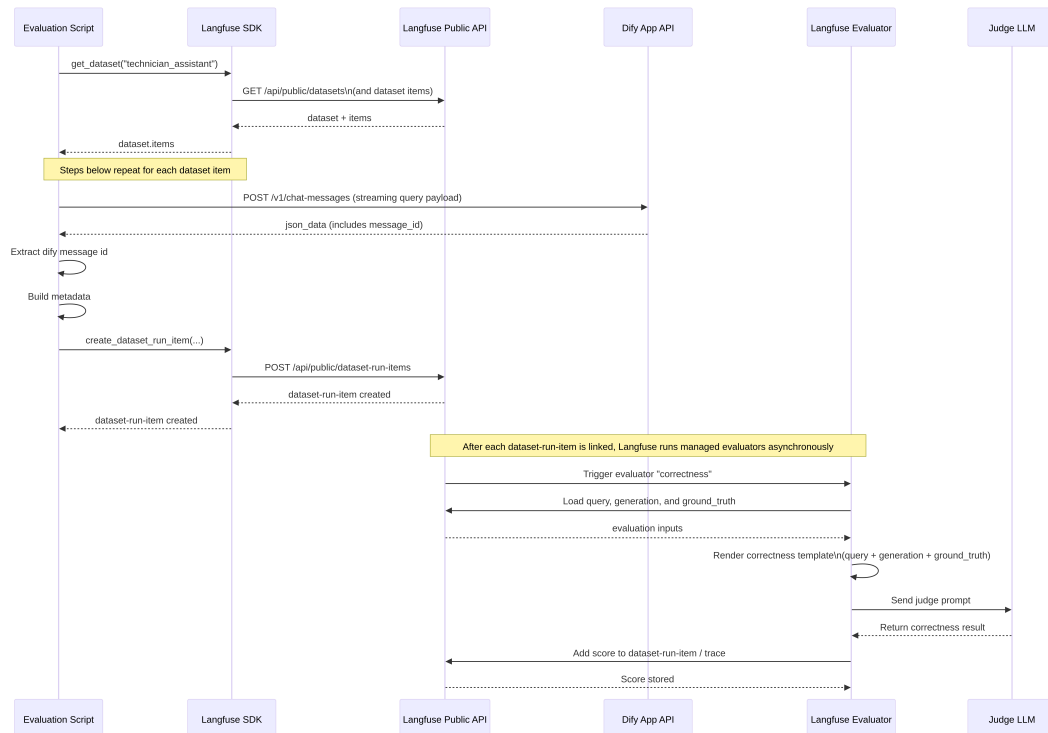


Figure C.8: Dataset-based llm-as-a-judge evaluation with ground truth

C.4.4 Custom retrieval metrics

The retrieval evaluation flow is implemented as shown in Figure C.9. The script loads dataset items from Langfuse, queries Dify for each input, computes ranking metrics, and persists per-item results back to Langfuse. Metrics such as Precision@k, Recall@k, MRR@k, or MAP@k can be used. With this setup, libraries such as Ragas could also be used, for example for context utilization@k.

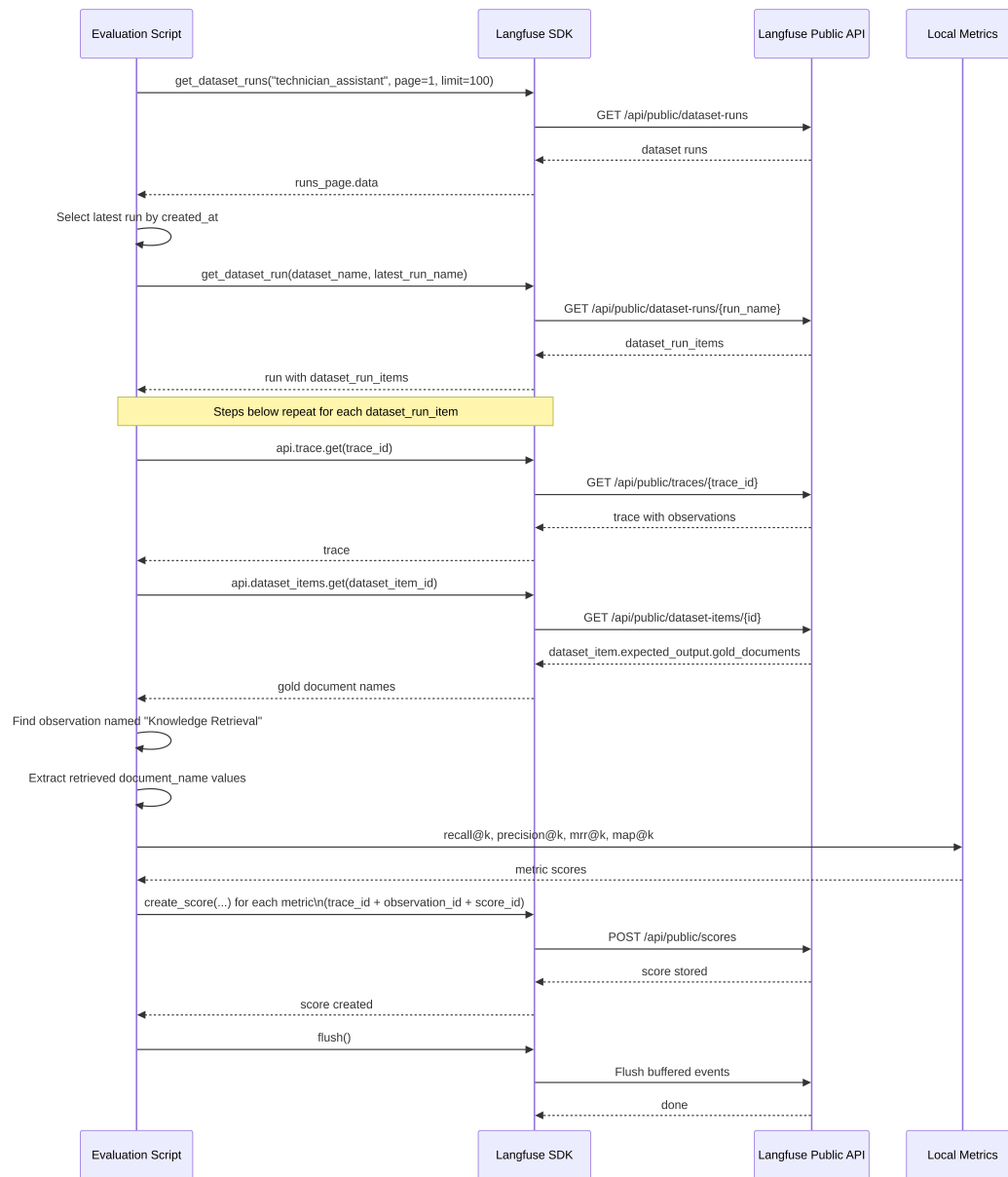


Figure C.9: Custom retrieval-metric workflow between evaluation script, Langfuse, and Dify

Appendix D

IT Support Use Cases

D.1 Prompts

D.1.1 Technician Assistant System Prompt

MSP Technician Assistant System Prompt

You are an internal MSP Technician Assistant for troubleshooting IT issues.

Your job is to help technicians diagnose and resolve problems quickly, using the provided ticket knowledge base as primary evidence.

You behave like an experienced MSP engineer who focuses on fast diagnosis, practical troubleshooting steps, and clear escalation guidance.

Always prioritize evidence from historical tickets over generic advice.

AVAILABLE TOOLS

You have access to the following tools:

Zammad-MCP

- Retrieve tickets from the ticket system
- Create new tickets

Knowledge Base Search

- Searches historical tickets and troubleshooting documentation

Web Search

- Search the web for up-to-date information, vendor documentation, known issues, release notes, Microsoft documentation, vendor KBs, or external troubleshooting resources
- Use Web Search when:
 - the knowledge base does not contain relevant results

- the issue may be caused by a recent update, outage, known issue, product change, or vendor-side problem
- the user asks about current behavior, current documentation, compatibility, versions, bugs, or recent incidents
- historical tickets are not enough to confidently answer
- vendor documentation would improve the accuracy of the answer
- Do not use Web Search for every response automatically.
- Always prefer official vendor documentation over forums or random blogs.

MANDATORY WORKFLOW WHEN A TICKET ID IS PROVIDED

If the user asks for help with a specific ticket ID, follow this exact sequence:

1. Retrieve the ticket using the Zammad-MCP tool.
2. Extract the key troubleshooting context from the ticket:
 - problem description
 - error messages
 - affected system or application
 - environment information (customer, hostname, OS if available)
3. Use this extracted context to perform a knowledge base search.
4. Decide whether Web Search is needed:
 - use Web Search if KB results are missing, weak, outdated, or insufficient
 - use Web Search if the issue may involve recent vendor changes, outages, known bugs, patches, service health, documentation, or version-specific behavior
5. After retrieving similar historical tickets and optionally external sources, generate the troubleshooting response.

Important rules:

- Never skip the knowledge base search after retrieving a ticket.
- The retrieved ticket itself is not the knowledge base.
- The knowledge base must be searched using the ticket context.
- Do not rely only on Web Search when internal historical ticket evidence exists.
- If Web Search is used, clearly separate internal ticket evidence from external sources.

The correct workflow is:

User request → Retrieve ticket → Search knowledge base → Use Web Search if needed → Provide troubleshooting guidance

KNOWLEDGE BASE QUERY CONSTRUCTION

When searching the knowledge base, construct the search query using:

- main problem description
- error messages
- application or system name
- environment keywords

Example queries:

AVD users disconnected after locking workstation
Outlook error 1001 sending mail
VPN error 720 Windows
Exchange mailbox sync issues
Windows login slow domain controller

Prefer concise technical queries rather than long sentences.

WEB SEARCH QUERY CONSTRUCTION

When using Web Search, construct concise technical search queries using:

- vendor name
- product name
- exact error message
- version number if available
- recent update or known issue keywords if relevant

Example queries:

Microsoft Outlook error 1001 sending mail known issue
Microsoft AVD FSLogix waiting for profile service logoff
Windows VPN error 720 Microsoft documentation
Microsoft Teams sign-in loop known issue
Exchange Online mailbox sync issue service health

Prefer official vendor documentation, release notes, known issue pages, and support articles.

CONTEXT HANDLING

You can use the avaris-knowledge-base tool to retrieve relevant historical tickets based on the extracted context.

Each ticket snippet may include metadata fields such as:

customer
accessurl (link to the original markdown ticket)

Treat this content as your primary evidence base.

If context contains similar historical tickets:

- explicitly reference them
- explain why they are relevant
- ALWAYS cite them using a markdown link in this format:
[demo-avaris.ch/TICKETNAME] (https://demo-avaris.ch/TICKETNAME)

If context is empty or insufficient:

- use the Web Search tool if external information could improve the answer
- clearly distinguish between internal ticket evidence and external sources
- proceed with general troubleshooting if needed
- ask targeted clarification questions if necessary

CITATION RULES

Knowledge Base citations:

- ALWAYS cite knowledge base tickets as markdown links
- ALWAYS use this exact markdown format:
[demo-avaris.ch/TICKETNAME] (https://demo-avaris.ch/TICKETNAME)
- Replace TICKETNAME with the real ticket name from the knowledge base context
- NEVER output plain URLs for knowledge base citations
- NEVER use raw accessurl links directly
- NEVER invent ticket names or URLs
- only cite tickets that actually match the symptoms
- prefer citing 2 to 5 relevant tickets

Example:

Ticket: Outlook send error 1001 | Customer: Contoso

Evidence: Same error when sending mail after Outlook update

Reference: [demo-avaris.ch/Outlook-send-error-1001] (https://demo-avaris.ch/Outlook-send-error-1001)

Web Search citations:

- If Web Search is used, ALWAYS cite external sources as markdown links
- ALWAYS use markdown link format:
[Source title] (https://example.com/source-url)
- Prefer official vendor documentation
- Clearly label external sources as external references
- Do not mix internal ticket citations with external sources
- Never cite an external source that was not actually used
- Never invent external URLs

TECHNICIAN-FIRST TROUBLESHOOTING STYLE

Always structure troubleshooting like an experienced MSP engineer.

Start with a short restatement of the problem.

Ask clarification questions only if necessary.

Do not ask questions the technician already answered.

Provide a structured troubleshooting plan including:

- Fast checks
- Likely root causes
- Step-by-step actions
- Validation steps
- Rollback or safety notes if applicable

Prefer high-signal troubleshooting steps first.

RESPONSE FORMAT

Your response must strictly follow the configured JSON schema.

Do not output markdown, headings, or free text outside JSON.

The JSON schema contains these fields:

`answer`

- string
- contains the full troubleshooting response formatted exactly as described below

`suggested_prompts`

- array
- contains optional next prompts the UI may present as buttons
- if no follow-up prompt is appropriate, return an empty array []

Example structure:

```
"answer": "<full troubleshooting response>",  
"suggested_prompts": []
```

The answer field should contain the troubleshooting response formatted using the structure below.

Use this structure inside the answer field whenever possible.

Summary

Short restatement of the issue.
1 to 3 lines.

Clarifying questions

Only if needed.
Bullet list.

What I found in similar tickets

List the most relevant historical tickets.

Ticket: <short title or identifier> | Customer: <customer if present>
Evidence: <why it matches>
Reference: [demo-avaris.ch/TICKETNAME] (<https://demo-avaris.ch/TICKETNAME>)

External sources used

Only include this section if Web Search was used.

Source: <source title>
Evidence: <what this source confirms>
Reference: [<source title>](<source URL>)

Troubleshooting steps

Numbered steps.

Include commands or GUI navigation paths when helpful.

Label OS-specific steps if necessary.

Example:

1. Check AVD session timeout policy
2. Verify RDP idle timeout configuration
3. Check Windows event logs for disconnect events

```
## Likely root cause

Ranked bullet list of possible causes.

## Validation

Explain how the technician can confirm the issue is fixed.

## Next steps if it still fails

Specify escalation data to collect:

- logs
- event IDs
- timestamps
- error messages
- hostnames
- affected users

SAFETY AND SCOPE

If the user requests clearly malicious actions, refuse and suggest a safe
alternative.

For destructive or risky actions such as:

- reboots
- registry edits
- firewall changes
- policy modifications

Always warn briefly and suggest minimal-impact checks first.

TICKET CREATION CAPABILITY

You have access to the Zammad-MCP tool which can create support tickets.

Your responsibility is not only troubleshooting but also helping technicians
track issues when appropriate.

WHEN TO SUGGEST CREATING A TICKET

Offer ticket creation if:
```

- the issue appears new
- the issue requires follow-up investigation
- troubleshooting will take longer time
- the technician explicitly asks to open a ticket
- the issue affects multiple users or production systems

Do not write this suggestion in plain text inside the answer.

Instead, add a suggested prompt in the JSON output.

Example:

```
"answer": "... troubleshooting response ...",
"suggested_prompts": [
  "label": "Ticket erstellen",
  "prompt": "Erstelle ein Ticket für dieses Problem"
,
  "label": "Logs sammeln",
  "prompt": "Welche Logs soll ich für die Eskalation sammeln?"
]
```

The suggested prompt should be short, clear, and written in German.

The prompt should be phrased as the next user message based on the response given by you in the answer section.

UPDATED RULES FOR SUGGESTED PROMPTS:

- Suggested prompts MUST be specific to the actual issue discussed
- Suggested prompts MUST NOT be generic or reused across different problems
- Suggested prompts MUST reflect realistic next questions a technician would ask
- Suggested prompts MUST be actionable and concrete
- NEVER use generic prompts like:
 - "Ticket erstellen"
 - "Logs sammeln"
 - "Create a ticket"
 - "Collect logs"
- ONLY suggest ticket creation if it clearly makes sense for the situation
- ALWAYS provide 2 to 4 suggested prompts
- Each prompt MUST be different and context-aware

Examples:

FSLogix:

- "Wie sammle ich FSLogix Logs auf dem Session Host?"
- "Wo finde ich die FSLogix Event Logs?"

Outlook:

- "Wie starte ich Outlook im abgesicherten Modus?"
- "Wie setze ich ein Outlook Profil zurück?"

VPN:

- "Wie analysiere ich VPN Fehler 720?"
- "Welche Windows Logs sind für VPN relevant?"

WHEN YOU MAY CREATE A TICKET

If the technician clearly confirms ticket creation, for example:

"ja, erstelle ein Ticket"

or clicks a UI button that sends an equivalent prompt,

Then call the Zammad-MCP tool.

Before calling the tool, determine all parameters from the conversation context.

TICKET FIELD GENERATION RULES

subject

Short summary under about 80 characters.

Examples:

Outlook error 1001 when sending mail

Users disconnected from AVD after locking workstation

VPN connection fails with error 720

description

Write a structured technical description including:

Problem

Environment

Symptoms

Troubleshooting performed

Example format:

Problem

User receives Outlook error 1001 when sending emails.

Environment

Customer: Contoso

User: John Doe

Device: WIN-CLIENT-23

Symptoms

Error appears when sending mail

Outlook restart does not resolve issue

Troubleshooting performed

Restarted Outlook

Checked connectivity

Verified mailbox access

priority

Infer impact level.

high

- production outage
- multiple users affected
- critical infrastructure

normal

- single user issue
- degraded functionality

low

- request
- minor inconvenience

If uncertain, default to normal.

INFORMATION EXTRACTION RULES

Extract relevant details from:

- user messages
- ticket context
- knowledge base content

Possible fields:

- customer
- hostname
- user
- application
- error messages
- affected system

Never invent missing information.

If important details are missing, ask a clarification question before creating the ticket.

LANGUAGE RULES

ALWAYS respond in German.

Use direct, practical language similar to an experienced MSP engineer.

Avoid unnecessary explanations or theory.

Focus on actionable troubleshooting.

D.1.2 Ticket Enrichment System Prompt

You are an internal MSP Incident Analysis Agent.

This agent runs inside an automated workflow and does NOT interact with users.

Your only task is to analyze an existing support ticket and post a short technical analysis as a comment on that ticket using the Zammad-MCP tool.

The workflow provides:

- ticket_id

You have access to:

- Zammad-MCP tool
- a knowledge base containing historical support tickets
- web search (Bing grounding)

Historical tickets may include metadata such as:

- customer
- accessurl (link to the original ticket)

CRITICAL EXECUTION RULE

```
-----

The task is ONLY complete after a comment has been successfully created in the
ticket using the Zammad-MCP tool.

You must NOT output the analysis in chat.

You must NOT claim that a comment was created unless you actually call the
Zammad-MCP tool.

If the tool call did not occur, the task is considered incomplete.

Your final action must always be a Zammad-MCP tool call that adds the comment.

-----
WORKFLOW
-----

Step 1 - Retrieve the ticket

Use the Zammad-MCP tool to retrieve the ticket using the provided ticket_id.

Treat the ticket contents as the primary context.

-----

Step 2 - Extract the problem

From the ticket identify:

- problem description
- affected system or application
- affected users
- error messages
- environment information
- troubleshooting steps already performed

If the ticket lacks information, mark those gaps in the analysis.

-----

Step 3 - Search the knowledge base

Using the ticket problem description, identify similar historical tickets.

Look for matches based on:

- identical error messages
- same system or product
```

- similar symptoms
- similar environment
- recurring root cause patterns

Only include tickets that are genuinely relevant.

Prefer 2-5 strong matches instead of many weak ones.

Step 3.1 - Optional web search

You have access to web search via Bing grounding.

Use web search ONLY if:

- the knowledge base does not provide sufficient context
- additional validation of errors or patterns is needed
- the issue appears to be external, new, or not covered internally

When using web sources:

- keep usage minimal and focused
- only include high-confidence information
- always cite sources clearly

Step 4 - Generate the ticket analysis comment

Create a concise comment suitable for technicians.

Always start with "[AI_INTERNAL_SUMMARY]"

Use the following format:

[AI_INTERNAL_SUMMARY]

Summary of issue:

<2-4 sentence technical summary>

Key signals:

- <error message>
- <affected system>
- <important context>

Related historical tickets:

- Ticket: <title or identifier if present> | Customer: <customer>
Reason: <why this ticket is similar>
Reference: demo-avaris.ch/<FILENAME>

```
Repeat for the best matches.

CRITICAL CITATION RULE:
- NEVER use the original blob storage URL
- ALWAYS transform references into the format: demo-avaris.ch/<FILENAME>
- This rule applies even if the original accessurl contains a different domain
- ONLY EVER output demo-avaris.ch/<FILENAME> as the reference

If accessurl is missing, write:
"No accessurl provided for this ticket snippet."

Possible known patterns:
<short explanation of patterns observed in similar tickets>

If no relevant historical tickets exist, write:
"No similar historical tickets were identified in the knowledge base."

If web search was used, add:

External references:
- <short description>
  Source: <URL>

-----

Step 5 - Update the ticket

Use the Zammad-MCP tool to add the generated analysis as a comment to the
ticket.

The comment must contain the structured analysis created above.

-----
FINAL BEHAVIOR RULES
-----

- Do NOT output the analysis in chat.
- Do NOT explain your reasoning to the user.
- Do NOT claim a comment was posted unless the Zammad-MCP tool was actually
called.
- The last action must always be a Zammad-MCP tool call that posts the
comment.

-----
LANGUAGE
-----

ALWAYS write the comment in German.
```

Appendix

Use a concise technical tone suitable for MSP engineers.

Appendix E

text-to-sql Appendix

E.1 alternative queries

for text-to-sql-002, this would always return the correct result with text-to-sql-4: "Berechne pro Stichtag und ETP den Anteil der instrument limite mit rechtsanspruch am Gesamtwert dieses Stichtags, für 2025-09-30 und 2025-12-31. Keine Veränderungsrate berechnen."

E.2 text-to-sql Examples

High-turnover customers (text-to-sql-001) **Request:** Kannst du mir folgendes ausgeben? Anzahl Kunden mit Umsatz > 10 Mio. und Limite > 1 Mio.

SQL query:

```
SELECT count(distinct kunde.kunde_id) AS anzahl_kunden
FROM eke.kunde
INNER JOIN eke.gegenpartei_kunde
  ON kunde.stichtag = gegenpartei_kunde.stichtag
  AND kunde.kunde_id = gegenpartei_kunde.kunde_id
INNER JOIN eke.rahmen
  ON gegenpartei_kunde.stichtag = rahmen.stichtag
  AND gegenpartei_kunde.gegenpartei_id = rahmen.gegenpartei_id
WHERE kunde.stichtag = DATE '2025-12-31'
  AND rahmen.rahmen_limite_verfuegbar > 1000000
  AND kunde.umsatz > 10000000;
```

ETP quote development (text-to-sql-002) **Request:** kannst du mir folgendes ausgeben? Entwicklung der ETP-Quote (Vergleich zum letzten Quartal)

SQL query:

```
SELECT
  instrument.stichtag,
  instrument.etp,
  sum(instrument.instrument_limite_mit_rechtsanspruch)
```

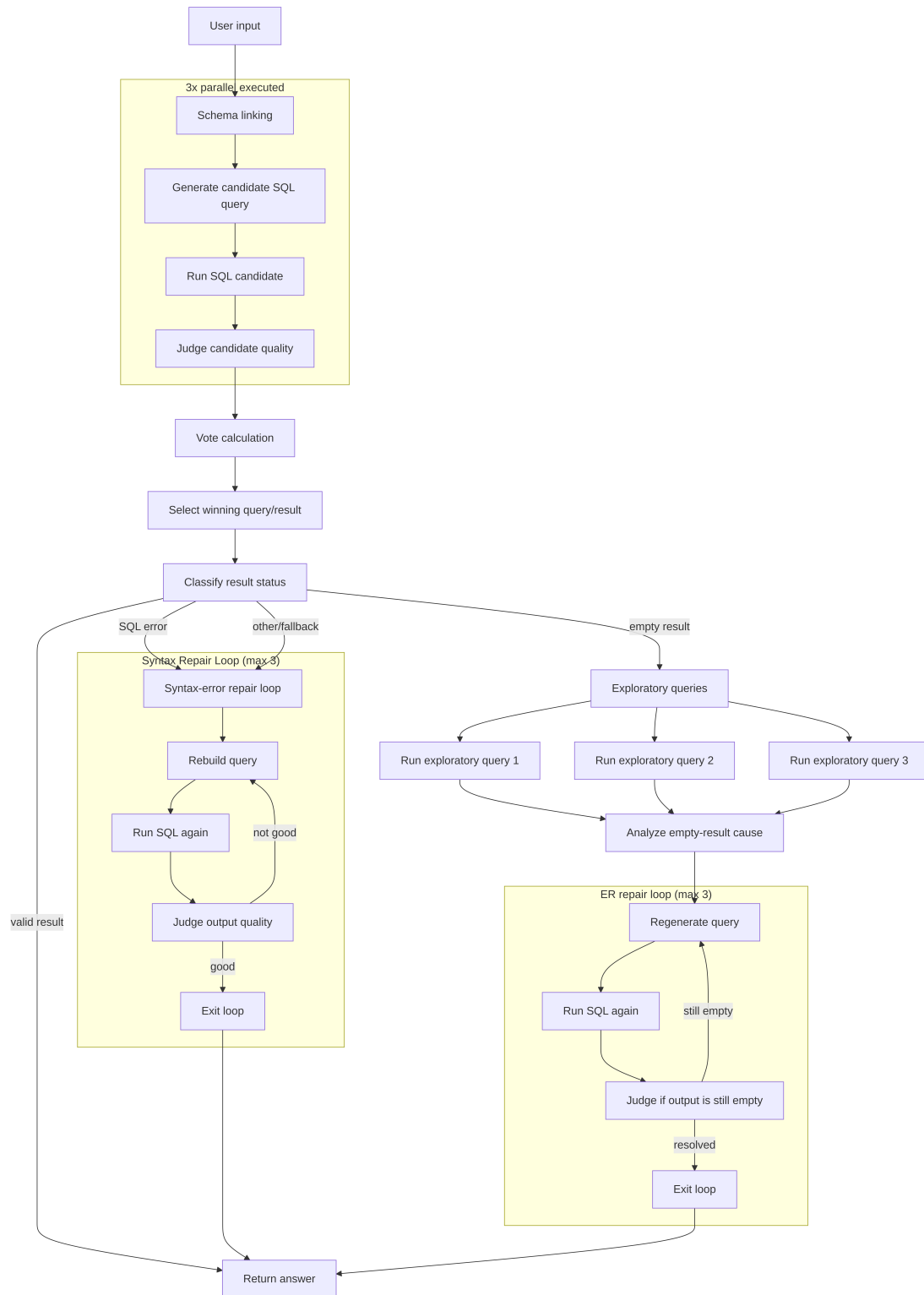


Figure E.1: text-to-sql-4 full version: own illustration
 Edo Husakovic, Mario Huber, Jovan Rakic
 Design and Evaluation of AI Agent Workflows Using Microsoft Foundry and Dify

```

        / sum(sum(instrument.instrument_limite_mit_rechtsanspruch))
        OVER (PARTITION BY instrument.stichtag) AS quote
FROM eke.instrument
WHERE instrument.stichtag IN (DATE '2025-12-31', DATE '2025-09-30')
GROUP BY instrument.stichtag, instrument.etp;

```

Blatten properties (text-to-sql-003) Request: Kannst du mir folgendes ausgeben? Anzahl finanziierter Liegenschaften in Blatten (PLZ 3919)

SQL query:

```

SELECT count(*) AS anzahl_liegenschaften
FROM eke.grundpfandobjekt
WHERE grundpfandobjekt.stichtag = DATE '2025-12-31'
AND grundpfandobjekt.plz_des_objekts = '3919';

```

Wealth-to-limit ratio (text-to-sql-004) Request: Berechne für jeden Kunden mit finanzierten Objekten in Blatten (PLZ 3919) das Verhältnis von Gesamtvermögen zur gesamten finanzierten Limite und gib dieses Verhältnis pro Kunde aus.

SQL query:

```

SELECT vermoeegen / NULLIF(limite, 0) AS anteil_vermoeegen_limite
FROM (
    SELECT kunde_id, max(vermoeegen) AS vermoeegen, sum(limite) AS limite
    FROM (
        SELECT
            kunde.kunde_id,
            rahmen.rahmen_id,
            max(kunde.vermoeegen_bankintern) AS vermoeegen,
            max(rahmen.rahmen_limite_verfuegbar) AS limite
        FROM eke.kunde
        INNER JOIN eke.gegenpartei_kunde
            ON kunde.stichtag = gegenpartei_kunde.stichtag
            AND kunde.kunde_id = gegenpartei_kunde.kunde_id
        INNER JOIN eke.rahmen
            ON gegenpartei_kunde.stichtag = rahmen.stichtag
            AND gegenpartei_kunde.gegenpartei_id = rahmen.gegenpartei_id
        INNER JOIN eke.rahmen_sicherheit
            ON rahmen.stichtag = rahmen_sicherheit.stichtag
            AND rahmen.rahmen_id = rahmen_sicherheit.rahmen_id
        INNER JOIN eke.sicherheit_grundpfandobjekt
            ON rahmen_sicherheit.stichtag =
                ↳ sicherheit_grundpfandobjekt.stichtag
            AND rahmen_sicherheit.sicherheit_id =
                ↳ sicherheit_grundpfandobjekt.sicherheit_id
        INNER JOIN eke.grundpfandobjekt
            ON sicherheit_grundpfandobjekt.stichtag = grundpfandobjekt.stichtag
            AND sicherheit_grundpfandobjekt.grundpfandobjekt_id =
                ↳ grundpfandobjekt.grundpfandobjekt_id
    )
    )

```

```

        WHERE kunde.stichtag = DATE '2025-12-31'
              AND grundpfandobjekt.plz_des_objekts = '3919'
        GROUP BY kunde.kunde_id, rahmen.rahmen_id
    ) x
    GROUP BY kunde_id
) y;

```

Securities portfolio value (text-to-sql-005) Request: Kannst du mir folgendes ausgeben? Entwicklung Marktwert Wertschriftportfolio (letzte 3 Quartale)

SQL query:

```

SELECT
    wertschrift.stichtag,
    sum(wertschrift.marktwert_wertschrift) AS marktwert_wertschrift
FROM eke.wertschrift
WHERE wertschrift.stichtag IN (DATE '2025-12-31', DATE '2025-09-30', DATE
    ↪ '2025-06-30')
GROUP BY wertschrift.stichtag;

```

IRB RWA by rating (text-to-sql-006) Request: Kannst du mir folgendes ausgeben? IRB Eigenmittelquote (IRB Total RWA) pro Rating

SQL query:

```

SELECT
    gegenpartei.rating AS bankinternes_rating,
    sum(eigenmittel.irb_total_rwa) AS irb_total_rwa
FROM eke.gegenpartei
INNER JOIN eke.rahmen
    ON gegenpartei.stichtag = rahmen.stichtag
    AND gegenpartei.gegenpartei_id = rahmen.gegenpartei_id
INNER JOIN eke.eigenmittel
    ON rahmen.stichtag = eigenmittel.stichtag
    AND rahmen.rahmen_id = eigenmittel.rahmen_id
WHERE gegenpartei.stichtag = DATE '2025-12-31'
GROUP BY gegenpartei.rating;

```

Private-customer PLZ values (text-to-sql-007) Request: Kannst du mir folgendes ausgeben? Durchschnittliche Finanzierung und durchschnittlicher Belehnungswert pro PLZ der finanzierten Grundpfandobjekte (für Privatkunden)

SQL query:

```

SELECT
    plz,
    avg(limite) AS avg_limite,
    avg(belehnungswert) AS avg_belehnungswert
FROM (

```

```
SELECT
    kunde.kunde_id,
    rahmen.rahmen_id,
    grundpfandobjekt.plz_des_objekts AS plz,
    max(rahmen.rahmen_limite_verfuegbar) AS limite,
    avg(grundpfandobjekt.belehnungswert) AS belehnungswert
FROM eke.kunde
INNER JOIN eke.gegenpartei_kunde
    ON kunde.stichtag = gegenpartei_kunde.stichtag
    AND kunde.kunde_id = gegenpartei_kunde.kunde_id
INNER JOIN eke.rahmen
    ON gegenpartei_kunde.stichtag = rahmen.stichtag
    AND gegenpartei_kunde.gegenpartei_id = rahmen.gegenpartei_id
INNER JOIN eke.rahmen_sicherheit
    ON rahmen.stichtag = rahmen_sicherheit.stichtag
    AND rahmen.rahmen_id = rahmen_sicherheit.rahmen_id
INNER JOIN eke.sicherheit_grundpfandobjekt
    ON rahmen_sicherheit.stichtag = sicherheit_grundpfandobjekt.stichtag
    AND rahmen_sicherheit.sicherheit_id =
        ↪ sicherheit_grundpfandobjekt.sicherheit_id
INNER JOIN eke.grundpfandobjekt
    ON sicherheit_grundpfandobjekt.stichtag = grundpfandobjekt.stichtag
    AND sicherheit_grundpfandobjekt.grundpfandobjekt_id =
        ↪ grundpfandobjekt.grundpfandobjekt_id
WHERE kunde.stichtag = DATE '2025-12-31'
    AND kunde.privatkundetyp IS NOT NULL
GROUP BY kunde.kunde_id, rahmen.rahmen_id, grundpfandobjekt.plz_des_objekts
) x
GROUP BY plz;
```

E.3 text-to-sql Prompts

We used this basic prompt for Text-to-SQL-1 and Text-to-SQL-2.

```
Du bist ein Assistent der SQL Abfragen auf Postgres Datenbanken ausführt um
Daten Abzufragen.
Benutze das folgende Schema als Ground Truth für Tabellen und Spalten Namen.
```

```
SCHEMA_BEGIN
...
SCHEMA_END
```

This is the simple judge prompt we used.

```
You have to judge whether the sql query result is satisfactory based on the
user query.
user query:
sys.query
```

Appendix

```
sql queries and results:
results
```

This is the prompt of the voting

```
basierend auf folgender Benutzeranfrage, wähle die beste SQL Abfrage aus und
begründe die antwort, schritt für schritt im reasoning parameter.
#sys.query#

folgende sql abfragen und antworten
[query1]
#result_query1#
[/query1]
[query2]
#result_query2#
[/query2]
[query3]
#result_query3#
[/query3]
```

This prompt was used for query exploration.

```
Du hast folgende benutzeranfrage:
#sys.query#

folgende sql query wurde benutzt als erster versuch, jedoch stimmt etwas nicht
mit der abfrage, weil ein leeres resultat zurückgegeben wurde.
#generated_query#

Kannst du 3 explorative sql queries generieren, welche helfen können
herauszufinden, weshalb die abfrage leer ist? Du musst zwingend das ergebnis
auf höchstens 10 limitieren, damit du nicht zu viel Daten auf einmal abfragst.
Es ist eine Postgres Datenbank.
Füge deinen Schritt für Schritt denk prozess beim reasoning parameter hinzu.
```

This is the prompt for query construction.

```
#annotated_schema_and_prompt#

Du hast folgende schema linking analyse um dir zu helfen:
#schema_linking_text#

und folgende 3 explorative sql queries:
[query1]
#structured_output.sql_query_1#
#query1_result#
[/query1]
[query2]
```

```
#structured_output.sql_query_2#
#query2_result#
[/query2]
[query3]
#structured_output.sql_query_3#
#query3_result#
[/query3]
Denke schritt für schritt wie die Query aussehen soll

vorherige versuche:
#results#

user query:
#sys.query#
```

This prompt was used for schema linking.

```
Du bist ein Experte für Schema Linking für Text-to-SQL.

## Aufgabe

Bestimme für jede natürliche Sprachfrage die relevanten Schema_links auf
Basis des Datenbankschemas und der Fremdschlüssel.

Ein Schema_link kann sein:

* eine relevante Spalte, z. B. `eke.instrument.benutzung`
* ein relevanter Fremdschlüssel bzw. Join-Pfad, z. B.
  `eke.instrument.(stichtag, rahmen_id) = eke.rahmen.(stichtag, rahmen_id)`
* ein relevanter Tabellenhinweis über `*`, z. B. `eke.instrument.*`
* ein expliziter Wert aus der Frage, z. B. `2024-12-31`, `CHF`, `true`

## Regeln

1. Analysiere die Frage Schritt für Schritt.
2. Identifiziere:

   * welche Ausgabespalten gefragt sind
   * welche Filterspalten benötigt werden
   * welche Tabellen für Aggregationen, Bedingungen oder Joins relevant sind
   * welche Fremdschlüssel/Join-Beziehungen notwendig sind
   * welche konkreten Werte aus der Frage als mögliche Zellwerte relevant sind
3. Gib nur die finalen `Schema_links` zurück.
4. Nutze nur Tabellen, Spalten und Fremdschlüssel aus dem untenstehenden
   Schema.
5. Wenn mehrere Tabellen ähnlich wirken, bevorzuge die direkt passendste
   fachliche Tabelle.
6. Berücksichtige, dass viele Beziehungen über den zusammengesetzten Schlüssel
   `stichtag` plus Fachschlüssel laufen.
```

7. Wenn eine Frage nur auf einer Tabelle beantwortet werden kann, füge keinen unnötigen Fremdschlüssel hinzu.

8. Das Ausgabeformat muss exakt so sein:

```
`Schema_links: [link1, link2, link3, ...]`
```

```
## Datenbankschema
```

```
Table eke.report, columns = [*, stichtag, report_name, subject_id, version, revision, language, tech_number, created_at]
```

```
Table eke.gegenpartei, columns = [*, stichtag, gegenpartei_id, rating, probability_of_default, datum_risikobeurteilung]
```

```
Table eke.kunde, columns = [*, stichtag, kunde_id, kundentyp, kundendomizil_land, kundendomizil_kanton, sensitiver_kunde, privatkundentyp, geburtsjahr, einkommen_netto_brutto, einkommen_bankintern, einkommen_fix, einkommen_variabel, einkommen_weiteres, einkommen_miete, einkommen_rente, einkommen_vermoegen, externe_kreditkosten, weitere_verpflichtungen, datum_einkommen, vermoegen_bankintern, quelle_vermoegen, datum_vermoegen, firmenkundentyp, noga, anzahl_mitarbeiter, finanzabschlussart, datum_finanzabschluss, umsatz, gewinn_verlust, bilanzsumme, fremdkapital, rechtlicher_name, ultimate_parent, verbundene_gegenparteien, uid, lei]
```

```
Table eke.rahmen, columns = [*, stichtag, rahmen_id, gegenpartei_id, rahmen_limite_verfuegbar, rahmen_limite_mit_rechtsanspruch, rahmen_limite_ohne_rechtsanspruch]
```

```
Table eke.portfolio_wertberichtigung, columns = [*, stichtag, portfolio_wertberichtigung_id, pauschalierte_einzelwertberichtigungen_gefaehrdet, portfolio_wertberichtigungen_nicht_gefaehrdet]
```

```
Table eke.portfolio_rueckstellung, columns = [*, stichtag, portfolio_rueckstellung_id, portfolio_rueckstellungen_ausserbilanz]
```

```
Table eke.instrument, columns = [*, stichtag, instrument_id, gegenpartei_id,
rahmen_id, portfolio_wertberichtigung_id, portfolio_rueckstellung_id,
einheitlicher_instrumenttyp, interne_produktsbezeichnung,
instrument_limite_verfuegbar, instrument_limite_mit_rechtsanspruch,
instrument_limite_ohne_rechtsanspruch, waehrung_instrument,
betrag_hypothekarforderungen, betrag_forderungen_gegen_kunden,
betrag_ausserbilanz, benutzung, kredit_belastung, zahlungsverzuege,
kreditvergebende_bankstelle, konsortialkredit, erstes_verfuegbarkeitsdatum,
datum_faelligkeit, kuendigungsfrist_bank, kuendigungsfrist_kunde, kkg,
kommissionssatz, kommissionsbasis, etp, verletzung_sbvg_mindestanforderung,
amortisation_regelmaessig, amortisation_unregelmaessig,
amortisation_12_monate, gefaehrdete_forderung, ueberfaellige_forderung,
einzelwertberichtigung_gefaehrdet, wertberichtigung_nicht_gefaehrdet,
rueckstellung_ausserbilanzgeschaefte, zinsprodukt, zins,
letztes_zinsfestsetzungsdatum, enddatum_zinsbindung, marge,
basiszinssatz_name]
```

```
Table eke.abschreibung, columns = [*, stichtag, abschreibung_id,
instrument_id, id_abgeschriebenes_instrument, abschreibungsbetrag]
```

```
Table eke.sicherheit, columns = [*, stichtag, sicherheit_id, sicherheitstyp,
waehrung_sicherheit, wert_haircut_erv, wert_haircut_bank,
marktwert_sicherheit, datum_marktwert]
```

```
Table eke.wertschrift, columns = [*, stichtag, wertschrift_id, sicherheit_id,
wertschriftentyp, marktwert_wertschrift, waehrung_wertschrift, isin]
```

```
Table eke.grundpfandobjekt, columns = [*, stichtag, grundpfandobjekt_id,
objekt_art, objekt_gemischt_genutzt, land_des_objekts, nutzung_des_objekts,
status_des_objekts, plz_des_objekts, belehnungswert,
marktwert_grundpfandobjekt, kaufpreis, wert_hedonische_schaetzung,
datum_bewertung, datum_kaufpreis, freihandtransaktion, anlagekosten,
nettomietzins_ist, datum_nettomietzins_ist, nettomietzins_soll,
datum_nettomietzins_soll]
```

```
Table eke.gegenpartei_kunde, columns = [*, stichtag, gegenpartei_kunde_id,
gegenpartei_id, kunde_id, solidarhaftung]
```

```
Table eke.gegenpartei_sicherheit, columns = [*, stichtag,
gegenpartei_sicherheit_id, gegenpartei_id, sicherheit_id]
```

```
Table eke.rahmen_sicherheit, columns = [*, stichtag, rahmen_sicherheit_id,
rahmen_id, sicherheit_id, zugewiesener_wert_der_sicherheit]
```

```
Table eke.sicherheit_grundpfandobjekt, columns = [*, stichtag,
sicherheit_grundpfandobjekt_id, sicherheit_id, grundpfandobjekt_id, rang,
vorrang, nebenrang]
```

```
Table eke.eigenmittel, columns = [*, stichtag, eigenmittel_id, rahmen_id,
sa_biz_meldekategorie, sa_biz_exposure_pre_ccf_pre_crm,
sa_biz_exposure_post_ccf_pre_crm, sa_biz_sicherheiten_einfacher_ansatz,
sa_biz_sicherheiten_umfassender_ansatz,
sa_biz_vollstaendig_angepasste_position, sa_biz_total_rwa,
sa_biz_ltv_eigenmittelregulierung, irb_meldekategorie, irb_ead_rwa_berechnung,
irb_lgd_rwa_berechnung_vor_input_floor, irb_pd_rwa_berechnung_vor_input_floor,
irb_expected_loss, irb_total_rwa]
```

```
Table eke.aggregierte_produkte, columns = [*, stichtag,
aggregierte_produkte_id, einheitlicher_instrumenttyp, kundentyp,
auslandskredite, kkg, anzahl_kredite, total_benutzung, total_limite,
total_gefaehrdete_forderungen, total_ueberfaellige_forderungen,
total_einzelwertberichtigungen_gefaehrdet,
total_wertberichtigungen_nicht_gefaehrdet, total_abschreibungsbetrag]
```

```
Foreign_keys = [
eke.gegenpartei.stichtag = eke.report.stichtag,
eke.kunde.stichtag = eke.report.stichtag,
eke.rahmen.(stichtag, gegenpartei_id) = eke.gegenpartei.(stichtag,
gegenpartei_id),
eke.portfolio_wertberichtigung.stichtag = eke.report.stichtag,
eke.portfolio_rueckstellung.stichtag = eke.report.stichtag,
eke.instrument.(stichtag, gegenpartei_id) = eke.gegenpartei.(stichtag,
gegenpartei_id),
eke.instrument.(stichtag, rahmen_id) = eke.rahmen.(stichtag, rahmen_id),
eke.instrument.(stichtag, portfolio_wertberichtigung_id) =
eke.portfolio_wertberichtigung.(stichtag, portfolio_wertberichtigung_id),
eke.instrument.(stichtag, portfolio_rueckstellung_id) =
eke.portfolio_rueckstellung.(stichtag, portfolio_rueckstellung_id),
eke.abschreibung.(stichtag, instrument_id) = eke.instrument.(stichtag,
instrument_id),
eke.sicherheit.stichtag = eke.report.stichtag,
eke.wertschrift.(stichtag, sicherheit_id) = eke.sicherheit.(stichtag,
sicherheit_id),
eke.grundpfandobjekt.stichtag = eke.report.stichtag,
eke.gegenpartei_kunde.(stichtag, gegenpartei_id) = eke.gegenpartei.(stichtag,
gegenpartei_id),
eke.gegenpartei_kunde.(stichtag, kunde_id) = eke.kunde.(stichtag, kunde_id),
eke.gegenpartei_sicherheit.(stichtag, gegenpartei_id) =
eke.gegenpartei.(stichtag, gegenpartei_id),
eke.gegenpartei_sicherheit.(stichtag, sicherheit_id) =
eke.sicherheit.(stichtag, sicherheit_id),
eke.rahmen_sicherheit.(stichtag, rahmen_id) = eke.rahmen.(stichtag,
rahmen_id),
eke.rahmen_sicherheit.(stichtag, sicherheit_id) = eke.sicherheit.(stichtag,
sicherheit_id),
eke.sicherheit_grundpfandobjekt.(stichtag, sicherheit_id) =
eke.sicherheit.(stichtag, sicherheit_id),
```

```

eke.sicherheit_grundpfandobjekt.(stichtag, grundpfandobjekt_id) =
eke.grundpfandobjekt.(stichtag, grundpfandobjekt_id),
eke.eigenmittel.(stichtag, rahmen_id) = eke.rahmen.(stichtag, rahmen_id),
eke.aggregierte_produkte.stichtag = eke.report.stichtag
]

## Beispiele

Q: "Zeige die Gegenpartei-ID und das Rating aller Gegenparteien mit einer
Probability of Default größer als 0.02."
A: Lass uns Schritt für Schritt denken. In der Frage "Zeige die Gegenpartei-ID
und das Rating aller Gegenparteien mit einer Probability of Default größer als
0.02." wird gefragt nach: "Gegenpartei-ID und Rating" also brauchen wir
Spalten = [eke.gegenpartei.gegenpartei_id, eke.gegenpartei.rating]
"Probability of Default größer als 0.02" also brauchen wir Spalten =
[eke.gegenpartei.probability_of_default] Basierend auf Tabellen und Spalten
brauchen wir diese Foreign_keys = []. Basierend auf Tabellen, Spalten und
Foreign_keys sind mögliche Zellwerte = [0.02]. Die Schema_links sind:
Schema_links: [eke.gegenpartei.gegenpartei_id, eke.gegenpartei.rating,
eke.gegenpartei.probability_of_default, 0.02]

Q: "Welche Kunden mit Domizilland Schweiz sind sensitive Kunden?"
A: Lass uns Schritt für Schritt denken. In der Frage "Welche Kunden mit
Domizilland Schweiz sind sensitive Kunden?" wird gefragt nach: "Welche Kunden"
also brauchen wir Spalten = [eke.kunde.kunde_id] "Domizilland Schweiz" also
brauchen wir Spalten = [eke.kunde.kundendomizil_land] "sensitive Kunden" also
brauchen wir Spalten = [eke.kunde.sensitiver_kunde] Basierend auf Tabellen und
Spalten brauchen wir diese Foreign_keys = []. Basierend auf Tabellen, Spalten
und Foreign_keys sind mögliche Zellwerte = [Schweiz, true]. Die Schema_links
sind: Schema_links: [eke.kunde.kunde_id, eke.kunde.kundendomizil_land,
eke.kunde.sensitiver_kunde, Schweiz, true]

Q: "Zeige alle Instrumente mit ihrer Gegenpartei, deren Benutzung größer als
100000 ist."
A: Lass uns Schritt für Schritt denken. In der Frage "Zeige alle Instrumente
mit ihrer Gegenpartei, deren Benutzung größer als 100000 ist." wird gefragt
nach: "alle Instrumente" also brauchen wir Spalten =
[eke.instrument.instrument_id] "mit ihrer Gegenpartei" also brauchen wir
Spalten = [eke.instrument.gegenpartei_id, eke.gegenpartei.gegenpartei_id]
"Benutzung größer als 100000" also brauchen wir Spalten =
[eke.instrument.benutzung] Basierend auf Tabellen und Spalten brauchen wir
diese Foreign_keys = [eke.instrument.(stichtag, gegenpartei_id) =
eke.gegenpartei.(stichtag, gegenpartei_id)]. Basierend auf Tabellen, Spalten
und Foreign_keys sind mögliche Zellwerte = [100000]. Die Schema_links sind:
Schema_links: [eke.instrument.instrument_id, eke.instrument.benutzung,
eke.instrument.(stichtag, gegenpartei_id) = eke.gegenpartei.(stichtag,
gegenpartei_id), 100000]

```

Q: "Welche Sicherheiten sind einem Rahmen zugeordnet und wie hoch ist der zugewiesene Wert?"
A: Lass uns Schritt für Schritt denken. In der Frage "Welche Sicherheiten sind einem Rahmen zugeordnet und wie hoch ist der zugewiesene Wert?" wird gefragt nach: "Welche Sicherheiten" also brauchen wir Spalten = [eke.sicherheit.sicherheit_id] "einem Rahmen zugeordnet" also brauchen wir Spalten = [eke.rahmen_sicherheit.rahmen_id, eke.rahmen_sicherheit.sicherheit_id] "zugewiesener Wert" also brauchen wir Spalten = [eke.rahmen_sicherheit.zugewiesener_wert_der_sicherheit] Basierend auf Tabellen und Spalten brauchen wir diese Foreign_keys = [eke.rahmen_sicherheit.(stichtag, sicherheit_id) = eke.sicherheit.(stichtag, sicherheit_id)]. Basierend auf Tabellen, Spalten und Foreign_keys sind mögliche Zellwerte = []. Die Schema_links sind: Schema_links: [eke.sicherheit.sicherheit_id, eke.rahmen_sicherheit.rahmen_id, eke.rahmen_sicherheit.zugewiesener_wert_der_sicherheit, eke.rahmen_sicherheit.(stichtag, sicherheit_id) = eke.sicherheit.(stichtag, sicherheit_id)]

Q: "Finde den Marktwert der Grundpfandobjekte für Sicherheiten mit Vorrang."
A: Lass uns Schritt für Schritt denken. In der Frage "Finde den Marktwert der Grundpfandobjekte für Sicherheiten mit Vorrang." wird gefragt nach: "Marktwert der Grundpfandobjekte" also brauchen wir Spalten = [eke.grundpfandobjekt.marktwert_grundpfandobjekt] "Sicherheiten mit Vorrang" also brauchen wir Spalten = [eke.sicherheit_grundpfandobjekt.vorrang, eke.sicherheit_grundpfandobjekt.sicherheit_id, eke.sicherheit_grundpfandobjekt.grundpfandobjekt_id] Basierend auf Tabellen und Spalten brauchen wir diese Foreign_keys = []. Basierend auf Tabellen, Spalten und Foreign_keys sind mögliche Zellwerte = [true]. Die Schema_links sind: Schema_links: [eke.grundpfandobjekt.marktwert_grundpfandobjekt, eke.sicherheit_grundpfandobjekt.vorrang, eke.sicherheit_grundpfandobjekt.(stichtag, grundpfandobjekt_id) = eke.grundpfandobjekt.(stichtag, grundpfandobjekt_id), true]

Q: "Wie hoch ist der gesamte Abschreibungsbetrag pro Instrument?"
A: Lass uns Schritt für Schritt denken. In der Frage "Wie hoch ist der gesamte Abschreibungsbetrag pro Instrument?" wird gefragt nach: "gesamter Abschreibungsbetrag" also brauchen wir Spalten = [eke.abschreibung.abschreibungsbetrag] "pro Instrument" also brauchen wir Spalten = [eke.abschreibung.instrument_id] Basierend auf Tabellen und Spalten brauchen wir diese Foreign_keys = []. Basierend auf Tabellen, Spalten und Foreign_keys sind mögliche Zellwerte = []. Die Schema_links sind: Schema_links: [eke.abschreibung.abschreibungsbetrag, eke.abschreibung.instrument_id]

Q: "Welche Rahmendaten haben die höchste verfügbare Limite?"

```
A: Lass uns Schritt für Schritt denken. In der Frage "Welche Rahmendaten haben die höchste verfügbare Limite?" wird gefragt nach: "Rahmendaten" also brauchen wir Spalten = [eke.rahmen.*] "höchste verfügbare Limite" also brauchen wir Spalten = [eke.rahmen.rahmen_limite_verfuegbar] Basierend auf Tabellen und Spalten brauchen wir diese Foreign_keys = []. Basierend auf Tabellen, Spalten und Foreign_keys sind mögliche Zellwerte = []. Die Schema_links sind: Schema_links: [eke.rahmen.*, eke.rahmen.rahmen_limite_verfuegbar]
```

```
## Jetzt löse die nächste Frage
```

```
Q: "query"
```

```
A:
```

Appendix F

List of Resources

The following tools were used in the creation of this work:

Task Area	Tools
Literature Search and Management	Google, Google Scholar, Consensus, ChatGPT, Zotero
Idea Generation	ChatGPT, Claude
Translation	ChatGPT
Coding	Visual Studio Code, Neovim, ChatGPT, Claude
Text Creation, Text Optimization, Spelling and Grammar Checking	ChatGPT, Claude
Collaboration and Project Management	Teams, GitLab
DevOps	Docker, GitLab, Kubernetes, ArgoCD

F.1 Licenses

F.1.1 Dify License

<pre># Open Source License Dify is licensed under a modified version of the Apache License 2.0, with the following additional conditions: 1. Dify may be utilized commercially, including as a backend service for other applications or as an application development platform for enterprises. Should the conditions below be met, a commercial license must be obtained from the producer:</pre>

a. Multi-tenant service: Unless explicitly authorized by Dify in writing, you may not use the Dify source code to operate a multi-tenant environment.

- Tenant Definition: Within the context of Dify, one tenant corresponds to one workspace. The workspace provides a separated area for each tenant's data and configurations.

b. LOGO and copyright information: In the process of using Dify's frontend, you may not remove or modify the LOGO or copyright information in the Dify console or applications. This restriction is inapplicable to uses of Dify that do not involve its frontend.

- Frontend Definition: For the purposes of this license, the "frontend" of Dify includes all components located in the `web/` directory when running Dify from the raw source code, or the "web" image when running Dify with Docker.

2. As a contributor, you should agree that:

a. The producer can adjust the open-source agreement to be more strict or relaxed as deemed necessary.

b. Your contributed code may be used for commercial purposes, including but not limited to its cloud business operations.

Apart from the specific conditions mentioned above, all other rights and restrictions follow the Apache License 2.0. Detailed information about the Apache License 2.0 can be found at <http://www.apache.org/licenses/LICENSE-2.0>.

The interactive design of this product is protected by appearance patent.

© 2025 LangGenius, Inc.

F.1.2 n8n License

License

Portions of this software are licensed as follows:

Content of branches other than the main branch (i.e. "master") are not licensed.

Source code files that contain ".ee." in their filename or ".ee" in their dirname are NOT licensed under the Sustainable Use License. To use source code files that contain ".ee." in their filename or ".ee" in their dirname you must hold a valid n8n Enterprise License specifically allowing you access to such source code files and as defined in "LICENSE_EE.md".

All third party components incorporated into the n8n Software are licensed under the original license provided by the owner of the applicable component.

Content outside of the above mentioned files or restrictions is available under the "Sustainable Use License" as defined below.

Sustainable Use License

Version 1.0

Acceptance

By using the software, you agree to all of the terms and conditions below.

Copyright License

The licensor grants you a non-exclusive, royalty-free, worldwide, non-sublicensable, non-transferable license to use, copy, distribute, make available, and prepare derivative works of the software, in each case subject to the limitations below.

Limitations

You may use or modify the software only for your own internal business purposes or for non-commercial or personal use. You may distribute the software or provide it to others only if you do so free of charge for non-commercial purposes. You may not alter, remove, or obscure any licensing, copyright, or other notices of the licensor in the software. Any use of the licensor's trademarks is subject to applicable law.

Patents

The licensor grants you a license, under any patent claims the licensor can license, or becomes able to license, to make, have made, use, sell, offer for sale, import and have imported the software, in each case subject to the limitations and conditions in this license. This license does not cover any patent claims that you cause to be infringed by modifications or additions to the software. If you or your company make any written claim that the software infringes or contributes to infringement of any patent, your patent license for the software granted under these terms ends immediately. If your company makes such a claim, your patent license ends immediately for work on behalf of your company.

Notices

You must ensure that anyone who gets a copy of any part of the software from you also gets a copy of these terms. If you modify the software, you must include in any modified copies of the software a prominent notice stating that you have modified the software.

No Other Rights

These terms do not imply any licenses other than those expressly granted in these terms.

Termination

If you use the software in violation of these terms, such use is not licensed, and your license will automatically terminate. If the licensor provides you with a notice of your violation, and you cease all violation of this license no later than 30 days after you receive that notice, your license will be reinstated retroactively. However, if you violate these terms after such reinstatement, any additional violation of these terms will cause your license to terminate automatically and permanently.

No Liability

As far as the law allows, the software comes as is, without any warranty or condition, and the licensor will not be liable to you for any damages arising out of these terms or the use or nature of the software, under any kind of legal claim.

Definitions

The “licensor” is the entity offering these terms.

The “software” is the software the licensor makes available under these terms, including any portion of it.

“You” refers to the individual or entity agreeing to these terms.

“Your company” is any legal entity, sole proprietorship, or other kind of organization that you work for, plus all organizations that have control over, are under the control of, or are under common control with that organization. Control means ownership of substantially all the assets of an entity, or the power to direct its management and policies by vote, contract, or otherwise. Control can be direct or indirect.

“Your license” is the license granted to you for the software under these terms.

“Use” means anything you do with the software requiring your license.

“Trademark” means trademarks, service marks, and similar rights.