HTL Leonding
next level

HÖHERE TECHNISCHE
BUNDESLEHRANSTALT
LEONDING

Development of an MCP Server with Semantic Search for LeoWiki

DIPLOMA THESIS

submitted for the

Reife- und Diplomprüfung

at the

Department of Informatik

Submitted by:
Jan Ritt
Imre Obermüller

Supervisor:
Rainer Stropek

Project Partner:
HTL Leonding (LeoWiki)

Leonding, April 2026

I hereby declare that I have composed the presented paper independently and on my own, without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such.

This paper has neither been previously submitted to another authority nor has it been published yet.

The presented paper is identical to the electronically transmitted document.

Leonding, April 2026 Jan Ritt & Imre Obermüller

Abstract

English Abstract

This diploma thesis presents the development of a Model Context Protocol (MCP) server that enables semantic search functionality for LeoWiki, the internal wiki system of HTL Leonding. The project addresses the limitations of traditional keyword-based search by implementing a vector database with German-optimized text embeddings, allowing users to find relevant content through natural language queries via AI assistants (Claude, ChatGPT, Mistral Le Chat).

The work is divided into two individual contributions: Imre Obermüller focuses on the MCP server implementation, including transport protocols (stdio, HTTP Streamable) and tool development. Jan Ritt concentrates on the embedding pipeline, retrieval optimization for German content, OAuth 2.1 authentication integration, and deployment via Docker Compose on the Raspberry Pi.

The resulting system comprises an MCP server compatible with popular AI clients (Claude, ChatGPT, Mistral Le Chat), a benchmarked embedding system that raises the Mean Reciprocal Rank from 0.051 (DokuWiki keyword search with natural-language queries) to 0.872 (semantic search)—a 17-fold improvement—and a secure role-based access control layer integrated with Scalekit. The best-performing local model (Octen-Embedding-4B, MRR 0.883) could not be deployed on the target hardware (Raspberry Pi 5); the production system therefore uses the API-based OpenAI `text-embedding-3-large` (MRR 0.872).

Keywords: MCP, Model Context Protocol, Semantic Search, Vector Database, Text Embeddings, RAG, DokuWiki, LeoWiki, OAuth 2.1, RBAC

Deutsche Kurzfassung

Diese Diplomarbeit beschreibt die Entwicklung eines Model Context Protocol (MCP) Servers, der semantische Suchfunktionalität für LeoWiki, das interne Wiki-System der

HTL Leonding, ermöglicht. Das Projekt adressiert die Einschränkungen der traditionellen Stichwortsuche durch die Implementierung einer Vektordatenbank mit für Deutsch optimierten Text-Embeddings, wodurch Benutzer relevante Inhalte durch natürlichsprachige Anfragen über gängige KI-Assistenten (Claude, ChatGPT, Mistral Le Chat) finden können.

Die Arbeit gliedert sich in zwei individuelle Beiträge: Imre Obermüller konzentriert sich auf die MCP-Server-Implementierung, einschließlich Transportprotokolle (stdio, HTTP Streamable) und Tool-Entwicklung. Jan Ritt fokussiert auf die Embedding-Pipeline, Retrieval-Optimierung für deutsche Inhalte, OAuth 2.1-Authentifizierungsintegration und Deployment via Docker Compose auf dem Raspberry Pi.

Der entwickelte MCP-Server ist mit populären KI-Clients (Claude, ChatGPT, Mistral Le Chat) kompatibel und umfasst ein evaluiertes Embedding-System, das den Mean Reciprocal Rank von 0,051 (DokuWiki-Stichwortsuche mit natürlichsprachigen Anfragen) auf 0,872 (semantische Suche) steigert—eine 17-fache Verbesserung—sowie ein sicheres rollenbasiertes Zugriffskontrollsystem integriert mit Scalekit. Das leistungsstärkste lokale Modell (Octen-Embedding-4B, MRR 0,883) konnte auf der Zielhardware (Raspberry Pi 5) nicht betrieben werden; das Produktivsystem nutzt daher das API-basierte OpenAI `text-embedding-3-large` (MRR 0,872).

Schlagwörter: MCP, Model Context Protocol, Semantische Suche, Vektordatenbank, Text-Embeddings, RAG, DokuWiki, LeoWiki, OAuth 2.1, RBAC

Contents

1 Introduction

1.1 Problem Statement

HTL Leonding, a secondary technical college in Upper Austria, operates LeoWiki as its central knowledge management platform. Built on DokuWiki – a file-based wiki system without a relational database [1] – the platform covers curricula, examination regulations, administrative procedures, and organizational information (see Section 3.1 for corpus details). The wiki serves three stakeholder groups with distinct information needs: students seeking examination schedules and learning resources, teachers maintaining curricula and internal documentation, and administrators managing the system.

Despite this knowledge base, the built-in DokuWiki search relies exclusively on keyword matching via an inverted index without semantic interpretation [2]; Section 3.3 documents the mechanism in detail. When a student asks "When is the next exam in mathematics?", the search engine does not recognize the semantic equivalence between "exam" (*Schularbeit*) and "test date" (*Prüfungsdatum*). Furthermore, DokuWiki's AND logic in conjunction with its stopword filter renders natural-language queries – the expected search behavior for users without knowledge of DokuWiki's namespace syntax – largely ineffective.

A systematic evaluation on 78 question-answer pairs (Section 3.3) confirms this limitation: even with manually optimized keywords, the default search achieves a Mean Reciprocal Rank (MRR) of only 0.365, while natural-language questions yield an MRR of 0.051.

This gap motivates the central research objective: developing a semantic search system that bridges the vocabulary mismatch between natural-language queries and wiki content.

1.2 Objectives

The goal of this diploma thesis is the development of a Model Context Protocol (MCP) server that provides semantic search functionality for LeoWiki. By integrating the server into AI assistants such as Claude, ChatGPT, and Mistral Le Chat, users can find relevant wiki content through natural-language dialogue rather than keyword-based search.

Three research questions guide the work:

FF1 – Search Quality. How effectively can a semantic search based on sentence embeddings replace keyword-based search in an educational wiki (LeoWiki)?

Evidence: Quantitative comparison on a defined test question catalog. Primary metric: MRR. Secondary metric: Precision@10. The search-quality evaluation is documented in Section 6.5.

FF2 – MCP as Integration Standard. How suitable is the Model Context Protocol (MCP) [3] as an interface between a specialized search server and heterogeneous AI applications?

Evidence: Compatibility matrix of popular AI clients, practical interoperability tests with at least three clients, and a qualitative assessment of the specification's maturity. Client tests and limitations are summarized in Section 5.10.

FF3 – Embedding Model Selection for German. Which sentence embedding model achieves the best retrieval quality for German-language educational content on resource-limited hardware?

Evidence: Comparison of five embedding models (four local, one API-based) on a test corpus of LeoWiki pages. Metrics: MRR, NDCG@10, P@5, Recall@10, and Hit Rate (tabulated in Section 6.1 and Appendix .6).

Note on sub-question labels. Each research question is operationalized through chapter-level sub-questions that appear in the respective individual chapters:

Prefix	Chapter	Maps to
FF1–FF3	Chapters 1 & 7 (shared)	Main research questions
I1–I5	Chapter 5 (Imre Obermüller)	FF2, FF1
J1–J8	Chapter 6 (Jan Ritt)	FF3, FF1

The prefix **I** denotes sub-questions from the MCP server chapter; **J** denotes sub-questions from the embedding and retrieval chapter. Each chapter section notes which FF question a given sub-question addresses (e.g. "operationalized as I5" or "sub-question J2").

1.3 Scope and Limitations

The scope covers the full lifecycle from data ingestion to production deployment: a five-stage offline pipeline (Chapter 4), the MCP server implementation (tools, resources, prompts, transport protocols, middleware), the retrieval system (vector database, similarity search), the authentication and authorization layer (OAuth 2.1 via Scalekit, RBAC), and the containerized deployment (Docker Compose on a Raspberry Pi).

The following topics are explicitly **not in scope**:

- The SYP projects Leonidas and Leonie – separate school-year projects that are organizationally distinct from this diploma thesis.
- Continuous deployment to the Raspberry Pi – not required by the ABA and intentionally deferred in favor of deeper evaluation work. A continuous integration pipeline for automated testing is described in Section 5.11.
- Corpus-wide parametric significance testing (e.g., ANOVA across all five models simultaneously) – pairwise non-parametric tests (Wilcoxon signed-rank) and bootstrap confidence intervals are reported for all direct comparisons in Chapter 6, but a full multi-comparison correction framework is outside the scope of this thesis.
- Real-time content synchronization – the system operates on a periodically refreshed offline index rather than live-updating embeddings.

1.4 Methodology

The project follows a **Design Science Research** approach [4], combining iterative artifact development with empirical evaluation. The methodology comprises three interleaved phases:

1. **Requirements Engineering and Technology Selection.** Stakeholder needs were elicited through analysis of the existing wiki usage patterns. Technology alternatives for embedding models, vector databases, identity providers, and transport protocols were systematically evaluated using structured comparison matrices (Chapter 3).

2. **Iterative Implementation.** The system was developed in milestones aligned with the ABA schedule: MCP server with stdio transport (December 2025), semantic search integration (January 2026), HTTP Streamable transport (February 2026), OAuth2 authentication (March 2026), and Docker Compose deployment (March 2026; npm as an alternative channel was evaluated per ABA but not adopted, Section 6.7.4). Each milestone produced a testable artifact.

3. **Evaluation-First Approach.** A ground-truth dataset of 78 question-answer pairs was constructed early (generated via Claude Opus 4.6, Anthropic, accessed 2026-02-17; manually verified by both authors against the wiki content for source page existence, answer derivability, and question clarity; see Section 6.8.4 for the verification procedure) and reused across all evaluations: embedding model comparison (FF3), chunk size optimization, hybrid vs. dense retrieval, and the keyword baseline comparison (FF1). This shared test corpus ensures cross-evaluation comparability and eliminates confounding variables from differing test sets.

The work is divided into two individual contributions: **Imre Obermüller** focuses on the MCP server implementation, including JSON-RPC 2.0 messaging, capability negotiation, transport protocol evaluation, tool development, and client compatibility analysis (Chapter 5). **Jan Ritt** concentrates on the embedding pipeline, retrieval optimization for German content, the Qdrant vector database schema, OAuth2 authentication integration with Scalekit, and Docker-based deployment (Chapter 6).

1.5 Structure of this Thesis

This thesis is structured as follows:

Part I: Shared Chapters

- **Chapter 1** presents the problem statement, objectives, scope, and methodology.
- **Chapter 2** covers text embeddings, German-language challenges for NLP, the Model Context Protocol, and authentication/authorization foundations.
- **Chapter 3** analyzes the current LeoWiki search limitations, stakeholder requirements, security needs, and technology alternatives.
- **Chapter 4** describes the overall system architecture, component responsibilities, interface design, and deployment topology.

Part II: Individual Contribution – Imre Obermüller

- **Chapter 5** details the MCP server implementation: JSON-RPC messaging, transport protocols, tool development, resources, prompts, middleware, authentication, and client compatibility analysis.

Part III: Individual Contribution – Jan Ritt

- **Chapter 6** covers the embedding model evaluation, chunking strategy, Qdrant vector database implementation, retrieval optimization, search quality evaluation, Scalekit authentication, deployment, and testing.

Part IV: Shared Chapter

- **Chapter 7** summarizes the achieved results against each research question, discusses limitations and lessons learned, outlines future work, and presents the overall conclusion.

2 Theoretical Foundations

2.1 Text Embeddings

2.1.1 From Tokens to Sentences

The foundation of modern embedding methods is the Transformer architecture [5]. Its central mechanism – self-attention – enables consideration of the entire input context when processing each token. Unlike recurrent architectures (RNN, LSTM), all positions of a sequence are processed in parallel, enabling faster training and better modeling of long-range dependencies. Self-attention explains why contextualized embeddings outperform the static vectors of earlier approaches.

Static Token Embeddings

The first generation of neural word representations – in particular Word2Vec [6] – assigns each word in the vocabulary exactly one fixed vector. This vector is learned from co-occurrence statistics of large text corpora and encodes semantic similarities: words with similar meanings are positioned close together in vector space. The central limitation of static embeddings is that polysemy cannot be captured. The word “bank” receives the same vector regardless of whether it refers to a financial institution or a park bench.

Contextualized Token Embeddings

BERT [7] addresses the polysemy problem by computing a context-dependent vector for each token. Through the bidirectional Transformer encoder, each token receives a vector reflecting the entire sentence context. Thus, “bank” in “I go to the bank” is represented differently than in “The bank grants a loan.” However, BERT primarily produces token-level embeddings – it generates a matrix of vectors (one vector per token), not a single vector for the entire sentence.

Sentence Embeddings and the Bi-Encoder Paradigm

For retrieval tasks, a single vector representing an entire text passage (sentence, paragraph, or chunk) is required. While BERT can produce a sentence vector via mean pooling or the `[CLS]` token, this strategy has been shown to yield suboptimal results. Reimers and Gurevych [8] demonstrated that BERT-based sentence similarity via cross-encoding delivers high quality but is impractical for retrieval: finding the most similar pair in a collection of 10,000 sentences requires approximately 50 million inference computations – around 65 hours [8].

Sentence-BERT (SBERT) [8] introduces the bi-encoder paradigm: query and document are independently processed through the same encoder, and the resulting sentence vectors are compared via cosine similarity. Training uses Siamese networks with contrastive learning, where the model learns to place semantically similar sentences close together and dissimilar sentences far apart in vector space. For retrieval, all document vectors can be pre-computed and indexed in a vector database. At query time, the query is encoded once, and the nearest neighbors are found via approximate nearest-neighbor search in milliseconds [9].

The development since SBERT has led to a series of high-performing models. OpenAI's `text-embedding-3-large` [10] represents the API-based variant with 3,072-dimensional vectors. BGE-M3 [11] pursues a multi-granularity approach that unifies dense, sparse, and ColBERT retrieval in a single model. These models are systematically compared on the MTEB benchmark [12], enabling informed model selection.

Suitability for LeoWiki Retrieval

For the semantic search in the LeoWiki of HTL Leonding, sentence embeddings were chosen because the task – given a natural-language query, find the most relevant wiki sections – aligns with the bi-encoder retrieval paradigm. Token embeddings would be unsuitable for two reasons:

1. **Scalability:** Cross-encoding (pairwise comparison) does not scale to hundreds of wiki pages with thousands of chunks. Pre-computed sentence vectors, in contrast, enable sub-second response times.
2. **Semantic granularity:** The search unit is a text passage (chunk), not an individual word. Sentence embeddings capture the meaning of the entire passage, while token embeddings encode only word-level semantics.

The concrete model selection (FF3) and chunk size optimization are empirically investigated in Section 6.1 and Section 6.2.3, respectively.

Retrieval-Augmented Generation. Retrieval-Augmented Generation (RAG), introduced by Lewis et al. [13], combines a language model with an external non-parametric memory – a retrieval system – to ground responses in verifiable documents rather than relying solely on knowledge stored in model parameters. The LeoWiki pipeline implements the ingestion and retrieval phases; generation is handled by the AI client. Figure 2.1 illustrates the progression from static token to sentence-level embeddings. Figure 2.2 summarizes the three-phase RAG process.

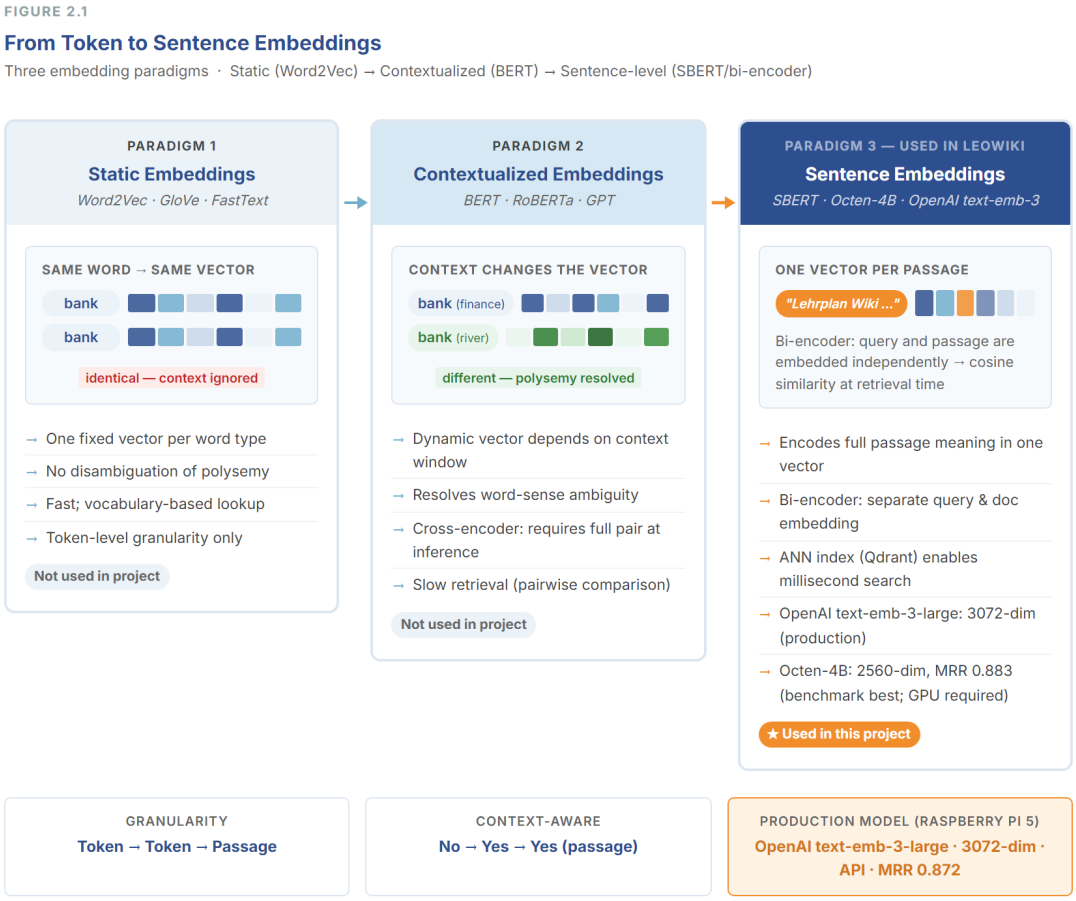


Figure 2.1: From token to sentence embeddings: static (Word2Vec), contextualized (BERT), and sentence-level (SBERT) paradigms. Sentence embeddings enable bi-encoder retrieval at scale.

Table 2.1 summarizes the key differences among the three embedding paradigms.

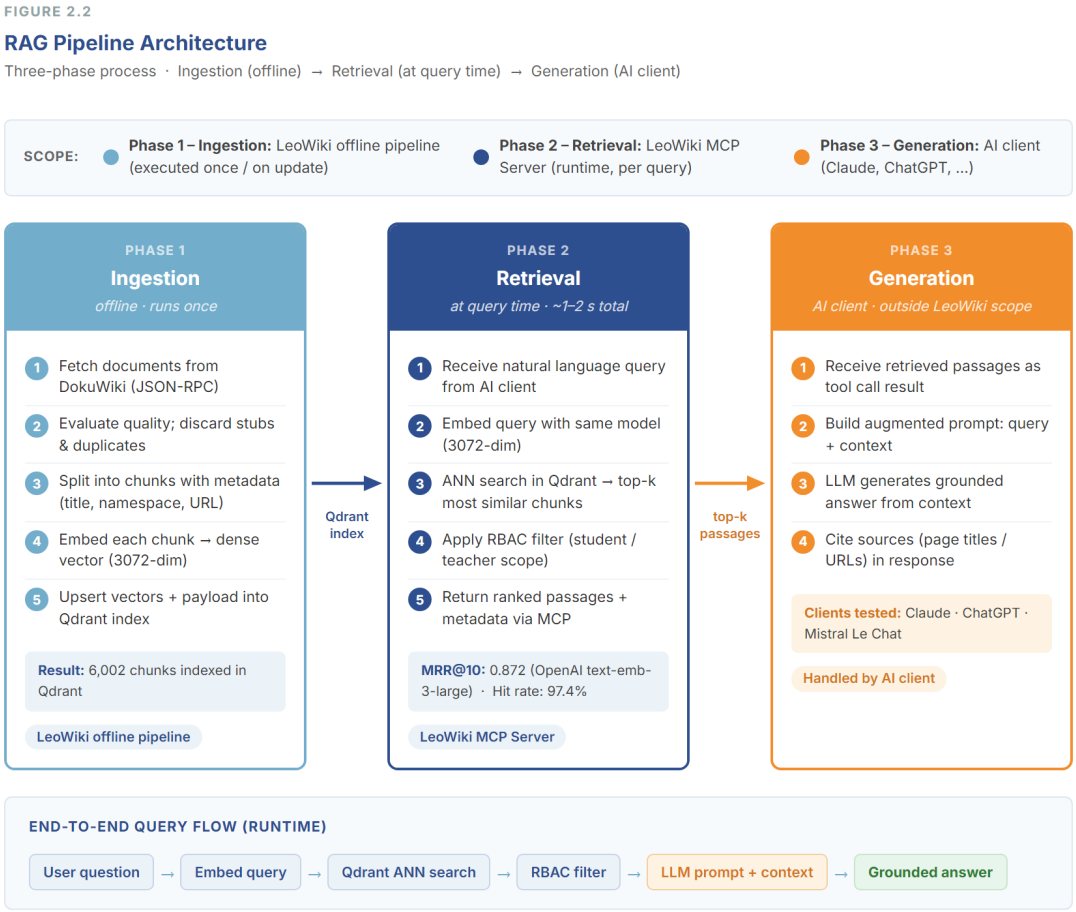


Figure 2.2: RAG pipeline architecture [13]: ingestion (offline), retrieval (at query time), and generation (handled by the AI client). The LeoWiki system covers ingestion and retrieval; the client performs generation.

Table 2.1: Comparison of embedding paradigms: static, contextualized, and sentence-level.

Property	**Static (Word2Vec)**	**Contextualized (BERT)**	**Sentence (SBERT)**
Granularity	Word	Token (in context)	Sentence / paragraph
Polysemy	No	Yes	Yes
Retrieval-suitable	Limited	No (cross-encoder)	Yes (bi-encoder)
Pre-computation	Yes	Token vectors only	Yes (sentence vectors)
Search latency	$O(N)$	$O(N \cdot \text{seq\_len})$	$O(\log N)$ with ANN [9]

2.1.2 Challenges for German Language Models

German morphology poses three challenges for embedding-based retrieval: compound words, case inflection, and flexible word order.

Compound Words

German forms compound words by concatenating multiple word stems into a single word: "Donaudampfschifffahrtsgesellschaft" (Danube steam shipping company), "Semesterpruefungsvorbereitung" (semester exam preparation), or "Stundenplankoordinator" (timetable coordinator). For embedding models based on subword tokenization (e.g., WordPiece, BPE), this leads to two problems:

1. **Tokenization fragmentation:** Long compounds are split into multiple subword tokens that individually carry little meaning. A model trained primarily on English corpora may segment "Semesterpruefung" into fragments that are no longer semantically associated with "Semester" or "Pruefung" (exam).
2. **Vocabulary coverage:** The subword vocabulary of a primarily English-trained model contains German compounds as whole tokens only if they appear frequently enough in the training corpus. Domain-specific compounds from the school context (e.g., "Kollegiumssitzung" – faculty meeting, "Maturaprüfungstermin" – diploma exam date) are typically not covered.

Case Inflection and Morphology

German features four grammatical cases (nominative, genitive, dative, accusative), three genders (masculine, feminine, neuter), and rich verb inflection. A single concept can appear in various morphological forms: "des Lehrers" (of the teacher), "dem Lehrer" (to the teacher), "den Lehrer" (the teacher, accusative). While contextualized models such as BERT [7] can resolve these variants in context, this requires the model to have seen sufficient German training data to learn the morphological patterns.

Word Order Flexibility

Unlike English (SVO order), German permits considerable flexibility in word order. In particular, the verb-second position in main clauses, verb-final position in subordinate clauses, and the bracket structure ("hat... gemacht" – has... done) cause semantically

related elements to be distributed across wide distances within a sentence. Self-attention-based models [5] can in principle model such long-range dependencies but benefit from language-specific training.

Multilingual vs. Monolingual Models

These properties motivate the model comparison in FF3. Two strategies are available:

Multilingual models. Models such as BGE-M3 [11] are trained on corpora in over 100 languages and can be deployed without language-specific adaptation. The advantage lies in broad coverage; the disadvantage is that the model must distribute its capacity across many languages – a trade-off termed the "Curse of Multilinguality" [14, 15]. For languages with a smaller share of the training corpus, this can lead to quality degradation.

German-optimized models. Since 2024, the MTEB benchmark [12] includes a dedicated German retrieval leaderboard (RTEB deu, beta) that evaluates retrieval quality specifically for German texts. Models such as `bflhc/Octen-Embedding-4B` (4.0B parameters, 2,560 dimensions) and `telepix/PIXIE-Rune-v1.0` (568M parameters, 1,024 dimensions) achieve high scores on this leaderboard (accessed 2026-02-17) and potentially offer better coverage of German morphology. The Sentence-Transformers library (v3.3 in the evaluation stack) [16] provides the technical infrastructure for training and inference.

Decompounding as a Preprocessing Strategy

A common approach to improving retrieval quality for German texts is decompounding – the automatic decomposition of compounds into their constituents ("Stundenplankoordinator" → "Stunden," "Plan," "Koordinator"). This strategy is recommended in the literature particularly for traditional search systems (BM25/TF-IDF), as it increases the lexical overlap between query and document.

Decompounding was considered as a preprocessing strategy but not implemented. The decision rests on two considerations:

1. We assume that subword tokenization partially addresses compound splitting; empirical verification is outside the scope of this thesis.
2. The additional implementation effort (decompounding dictionary, error handling for proper nouns and technical terms) is disproportionate to the expected quality gain for sentence-level retrieval.

Implications for Model Selection

Research question FF3 calls for a systematic comparison of five embedding models on LeoWiki content. The evaluation (Section 6.1) compares both multilingual models (BGE-M3, OpenAI `text-embedding-3-large`) and German-optimized models on a test corpus of real wiki pages from HTL Leonding. Whether German-optimized models outperform on domain-specific data is tested empirically in Chapter 6.

2.2 Model Context Protocol

2.2.1 Protocol Overview

The integration of external knowledge sources into AI applications requires a communication protocol that enables both structured data retrieval and dynamic tool interaction between a language model and a data source. With the Model Context Protocol (MCP) [3], Anthropic introduced a protocol standard in 2024 specifically designed for this use case. Unlike classical API protocols, MCP follows a tool-based paradigm: the server does not expose resources in the REST sense but declares capabilities that an AI client can invoke. Communication is organized not around entities but around actions: "search for X," "read document Y," "navigate to Z" [3].

MCP defines two transport mechanisms: **stdio** (standard input/output) for local process-to-process communication and **HTTP Streamable** for network-based scenarios. Both transports use JSON-RPC 2.0 [17] as the message format. The stdio variant enables direct integration into CLI-based AI clients without network overhead, while HTTP Streamable ensures interoperability with web-based clients.

Without MCP, integrating N AI clients with M data sources requires $N \times M$ custom adapters—each combination needs its own connector code. MCP addresses this combinatorial problem by introducing a standardized protocol layer: each client implements one MCP client, each data source implements one MCP server, reducing the total

integration effort from $O(N \cdot M)$ to $O(N + M)$. Figure 2.3 illustrates this architectural advantage.

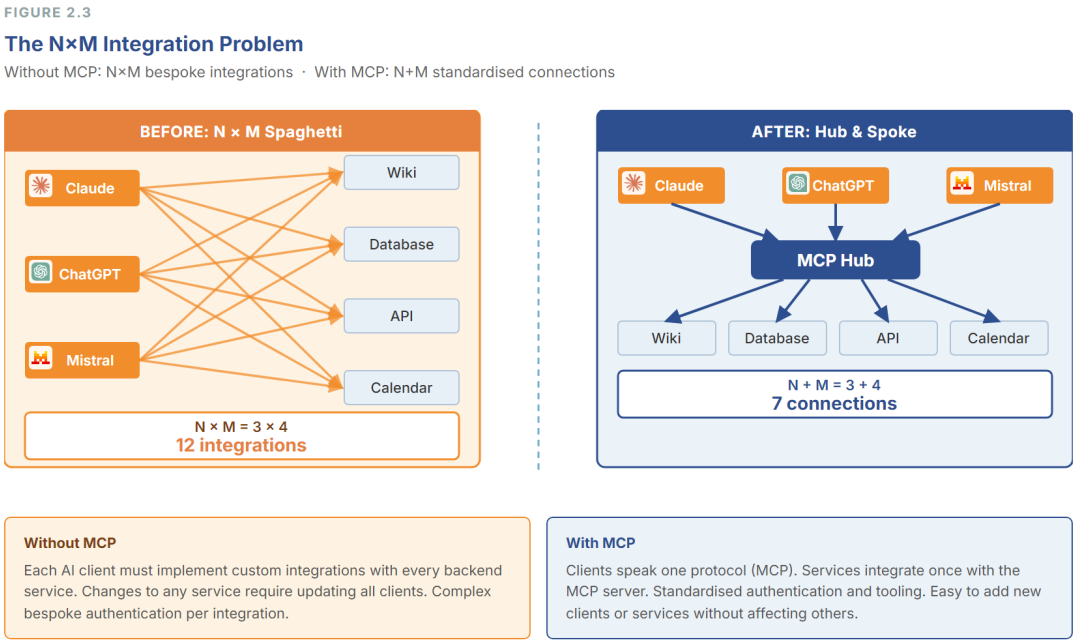


Figure 2.3: The $N \times M$ integration problem (left) vs. the MCP hub-and-spoke model (right). MCP eliminates custom adapter code by standardizing the communication protocol.

A distinguishing feature of MCP is the explicit session lifecycle model. Communication begins with an initialization phase in which client and server negotiate their capabilities. Thereafter, a persistent session is maintained within which multiple tool calls take place. This model corresponds to the natural flow of an AI conversation, where a language model interacts with the same data source across multiple turns. Figure 2.4 depicts the MCP architecture and session flow.

2.2.2 Architectural Comparison: MCP vs REST vs GraphQL vs OData

To understand the architectural positioning of MCP, it is compared below along eight dimensions with the established integration protocols REST [18], GraphQL [19], and OData [20].

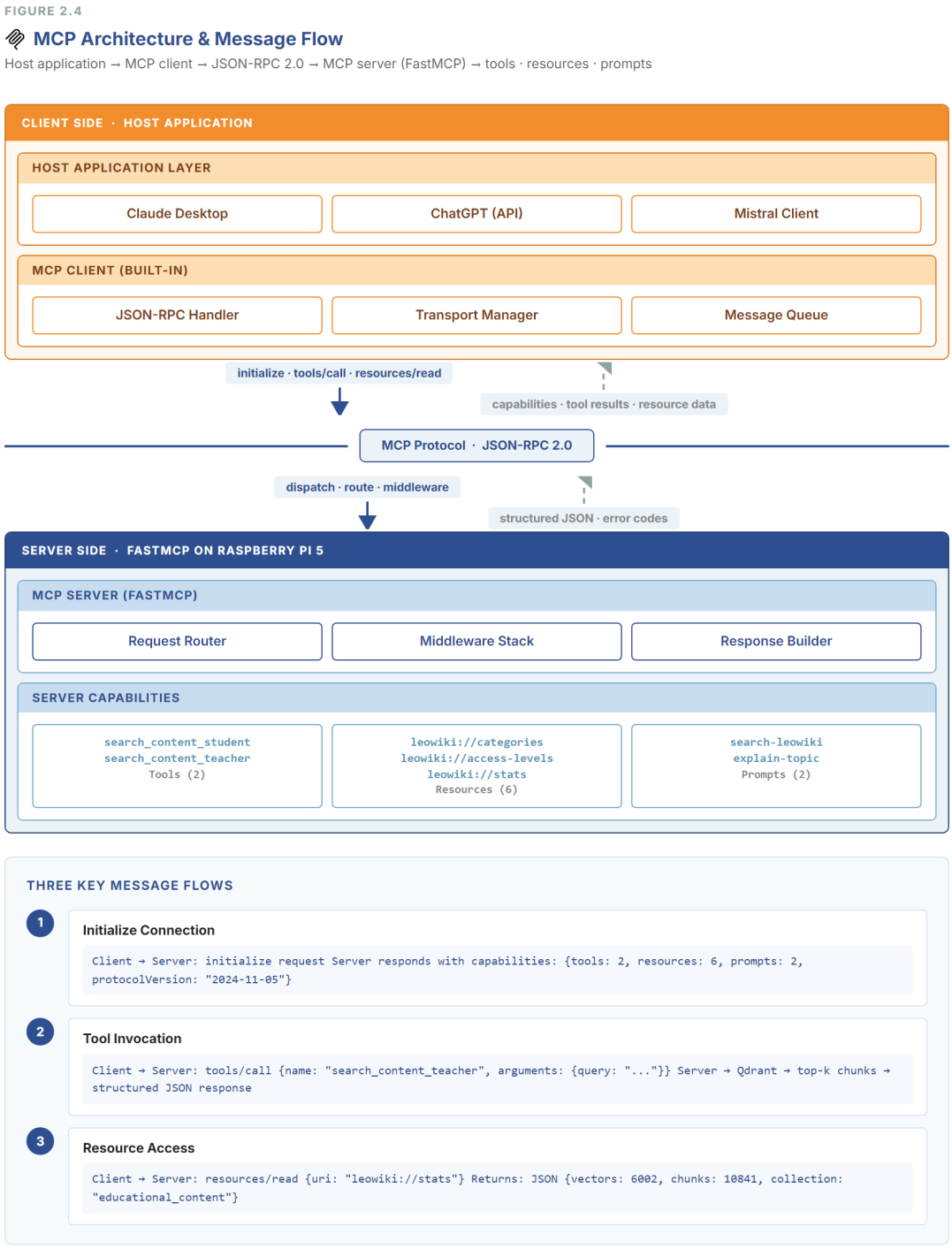


Figure 2.4: MCP architecture: client-server session with initialization, tool discovery, and JSON-RPC 2.0 messaging over stdio or HTTP Streamable transport.

Paradigm

REST follows the resource-oriented architectural style: each entity is identified by a URI, and interaction occurs via standardized HTTP methods (GET, POST, PUT, DELETE) [18]. GraphQL pursues a query-based approach where the client declaratively specifies the desired data structure [19]. OData combines both approaches: resource-oriented URLs with a standardized query language (`$filter`, `$expand`, `$select`) [20]. MCP, in contrast, is a protocol for tool provisioning where the server declares capabilities that an AI client can invoke [3].

Schema and Typing

REST uses OpenAPI/Swagger for interface description, though its use is optional. GraphQL enforces a strongly typed schema (SDL – Schema Definition Language) that simultaneously serves as documentation and validation basis [19]. OData defines its schema via the Common Schema Definition Language (CSDL) [20]. MCP types its tools via JSON Schema [3]: each tool declares its input parameters as a JSON Schema object. The crucial difference is that in MCP, the schema is not primarily intended for developers but for the language model – it must be able to infer from the tool description and schema when and how to call a tool.

Discovery

REST ideally relies on HATEOAS (Hypermedia as the Engine of Application State), where links in responses point to related resources [18]. In practice, HATEOAS is rarely fully implemented; instead, an OpenAPI specification serves as a static directory. GraphQL offers introspection: the client can query the complete schema via a query [19]. OData provides a `$metadata` document [20]. MCP implements discovery as an integral part of the protocol: the `tools/list` call returns all available tools with descriptions and JSON schemas at runtime. This occurs automatically during the initialization phase and can be dynamically updated via the `tools/list_changed` notification [3].

Suitability for AI Tool Integration

This is the dimension in which MCP differs fundamentally from the three comparison protocols. REST, GraphQL, and OData were designed for human-to-machine or

machine-to-machine communication. An AI client that wants to use a REST API requires:

- A manual tool definition (function calling schema)
- A wrapper that executes the API calls
- Error handling for each endpoint
- Context management across multiple calls

MCP abstracts this layer: the server declares its capabilities in a format that language models can directly interpret. MCP reduces the manual integration effort through natural-language tool descriptions, JSON Schema parameters, and a structured session lifecycle [3]. Since its introduction, MCP has been adopted by AI clients including Claude Desktop (developed by Anthropic, which also created MCP), as well as the independent third-party products Cursor and VS Code Copilot [21].

Summary Comparison

Table 2.2 summarizes the comparison along all eight dimensions.

Table 2.2: Architectural comparison: MCP, REST, GraphQL, and OData along eight dimensions.

Dimension	**REST**	**GraphQL**	**OData**	**MCP**
Paradigm	Resource-oriented	Query-based	Resource + query	Tool-/capability-based
Transport	HTTP	HTTP (de facto)	HTTP	stdio, HTTP Streamable
State	Stateless	Stateless	Stateless	Session lifecycle
Schema	OpenAPI (opt.)	SDL (mandatory)	CSDL	JSON Schema (per tool)
Discovery	HATEOAS / OpenAPI	Introspection	`$metadata`	`tools/list` (dynamic)
Streaming	SSE (add-on)	Subscriptions	Delta links	SSE + progress notif.
AI suitab.	Low (wrapper req.)	Medium	Low	Native
Batching	Manual	Nested req.	`$expand`	LLM-guided seq.

The comparison shows that MCP is not a competitor to REST, GraphQL, or OData in classical API design but rather a complementary layer for AI-specific integration. The design decisions – session lifecycle, natural-language tool descriptions, dynamic

discovery, and JSON-RPC 2.0 as the message format [17] – address specific requirements of language model–tool interaction that are not covered by the established protocols. For the present work, this comparison justifies the choice of MCP as the integration protocol for the LeoWiki server: the task – a language model should issue natural-language search queries to a wiki knowledge base – aligns with the tool-calling paradigm for which MCP was designed.

2.3 Authentication and Authorization

2.3.1 OAuth2/OIDC for Server-Side Integration

The semantic search in LeoWiki must not make all content equally accessible to all users. The DokuWiki of HTL Leonding uses a namespace-based permission model (ACL – Access Control List) that governs access to pages and namespaces by user groups. An integration of semantic search must respect this permission model: a student must not be able to find exam questions intended only for teachers.

The OAuth 2.0 Framework

OAuth 2.0 [22] defines an authorization framework that enables a third-party application (client) to access protected resources on behalf of a user without knowing the user's credentials. The framework specifies four roles: Resource Owner (user), Client (application), Authorization Server (token issuer), and Resource Server (API). The central mechanism is the issuance of access tokens, which are transmitted as Bearer tokens [23] in HTTP requests.

Authorization Code Flow with PKCE

For server-side web applications – such as the MCP server with HTTP Streamable transport – the Authorization Code Flow is the recommended grant type [22]. The flow comprises the following steps:

1. The client redirects the user to the Authorization Server.
2. The user authenticates and grants authorization.
3. The Authorization Server sends an authorization code to the client (via redirect URI).

4. The client exchanges the code for an access token (and optionally a refresh token).

OAuth 2.1 [24] (`draft-ietf-oauth-v2-1-12`, October 2025; still under revision) mandates PKCE (Proof Key for Code Exchange) for all clients – not only for public clients. PKCE protects against authorization code interception attacks by having the client generate a cryptographic code verifier, whose hash (code challenge) is sent with the authorization request. During token exchange, the client must present the original verifier, which only it knows.

OpenID Connect as Identity Layer

OAuth 2.0 is an authorization, not an authentication protocol – it defines what a client may do, not who the user is. OpenID Connect (OIDC) [25] extends OAuth 2.0 with an identity layer:

- **ID Token:** A JWT (JSON Web Token) containing user attributes (claims): subject (sub), name, email, group memberships.
- **UserInfo Endpoint:** A standardized endpoint providing additional user information.
- **Standardized Scopes:** `openid` (mandatory), `profile`, `email`, `groups`.

For the wiki integration, OIDC is essential because the authorization decision (which namespaces may the user access?) is based on the user's identity and group membership. The ID token provides this information in machine-readable form.

2.3.2 RBAC to Namespace Mapping

The RBAC Model

Role-Based Access Control [26] defines an authorization model in which permissions are assigned not directly to users but to roles. Users are assigned to roles and thereby inherit the permissions associated with that role. The model comprises four core components:

- **Users:** Persons accessing the system.
- **Roles:** Named collections of permissions (e.g., "student," "teacher," "administrator").
- **Permissions:** Allowed actions on objects (e.g., "read on namespace exam:").

- **Sessions:** Activated roles of a user at runtime.

Mapping to DokuWiki Namespaces

The DokuWiki ACL system assigns permission levels (0=none, 1=read, 2=edit, ..., 255=admin) to user groups at the namespace level. This model maps directly to RBAC. Table 2.3 shows the assignment.

Table 2.3: Mapping of RBAC roles to DokuWiki groups and namespace permissions.

RBAC Role	DokuWiki Group	Typical Namespace Permissions
Student	@student	Read on public namespaces (wiki:, info:)
Teacher	@teacher	Read/write on all pedagogical namespaces
Administrator	@admin	Full access on all namespaces incl. admin: and config:

For semantic search, this means: before returning search results, the MCP server must verify the ACL permissions of the requesting user and return only chunks from namespaces for which the user possesses at least read permission. The implementation of this filter mechanism is described in Chapter 6, Section 6.6.

Architecture Summary

The authentication and authorization architecture for the LeoWiki integration is based on three layers:

1. **Authentication:** OAuth 2.1 Authorization Code Flow with PKCE (via an external authorization server).
2. **Identity:** OpenID Connect ID tokens with group claims (assignment to student/teacher/administrator).
3. **Authorization:** RBAC model mapping OIDC groups to DokuWiki namespace ACLs.

This architecture is designed to enforce the existing permission model of LeoWiki and reduce the risk of unauthorized cross-namespace access.

3 Requirements Analysis

3.1 LeoWiki Content Analysis

3.1.1 Content Volume and Structure

The LeoWiki of HTL Leonding is based on DokuWiki version 2024-02-06b "Kaos" [1] (cf. Section 1.1). The limitations of the built-in search are analyzed in Section 3.3; here only a summary is given: full-text indexing matches exact strings against an inverted index without semantic interpretation [2].

As part of the offline pipeline, the entire data inventory of LeoWiki was captured and preprocessed. The corpus comprises 196 pages across 23 namespaces as well as 320 media files. The pages cover organizational documents, examination information, curricula, and internal administrative content.

3.1.2 Content Characteristics

The content of LeoWiki is predominantly German-language and encompasses both structured documents (tables, lists, regulations) and running text. The German language frequently forms compound words (e.g., "Maturaarbeit" – diploma work, "Semesterpruefung" – semester exam, "Lehrerkonferenz" – teacher conference), which pose a particular challenge for the search function. DokuWiki-specific markup must be removed or transformed during preprocessing before embeddings can be generated.

3.2 User Requirements

3.2.1 User Roles

Based on an analysis of DokuWiki ACL group configurations (file `conf/acl.auth.php`, five role groups in the production instance) and review of which namespaces are exposed to students versus teachers, three primary stakeholder groups were identified for the LeoWiki MCP system, whose needs determine the requirements:

Students The largest user group of LeoWiki. They require fast access to learning materials, examination information, and organizational documents. Their search queries are frequently formulated as natural-language questions (e.g., "When is the next exam in mathematics?"). Access should be restricted exclusively to student-relevant content.

Teachers Teachers use the wiki both as consumers (looking up regulations, curricula) and as authors. They require access to extended content, including internal documents and administrative information not visible to students.

Administrators Responsible for system maintenance and configuration. They require access to system statistics, query metrics, and vector database management.

3.2.2 Access Patterns

The use cases were derived from the stakeholder requirements and documented system interactions. The identified use cases are summarized in Table 3.1. Students primarily access the wiki in read mode and formulate their queries as natural-language questions. Teachers act both as consumers and authors and require extended access to internal documents. Administrators monitor the system state and perform data updates as needed.

Table 3.1: Identified use cases for the LeoWiki MCP system.

Use Case	Actor	Description
UC-1	Student	Semantic search for wiki content via an AI client
UC-2	Teacher	Extended search with access to internal documents
UC-3	Admin	View system statistics and query metrics
UC-4	Admin	Update the vector database via pipeline
UC-5	All	Authentication via OAuth 2.1 / Scalekit

3.2.3 Search Requirements

From the stakeholder needs, the following requirements for the search function emerge:

- Natural-language queries must be semantically interpreted rather than merely performing exact keyword matching.

- Synonyms, paraphrases, and semantically related terms should be considered during search.
- Search results must be weighted and sorted by relevance.
- German compound words must be handled correctly.
- Access to search results must be controlled on a role-based basis.

3.3 Existing Search Limitations

3.3.1 DokuWiki Default Search

The DokuWiki search operates at the token level: a document is recognized as a hit only if the exact search terms appear in the text. The search engine is based on an inverted index that maps words to the documents in which they appear. No semantic interpretation of the query takes place [2]. The sorting of search results uses a simple heuristic based on the frequency and position of hits within the document. A learning-based relevance weighting, as offered by modern retrieval systems based on text embeddings, is not available [8].

3.3.2 Identified Limitations

Four weaknesses of the existing DokuWiki search were identified:

1. **Missing semantic interpretation.** The semantic equivalences listed in Section 3.2.3 are not recognized. For example, searching for "Schularbeit Termin" (exam date) does not find pages that instead contain "Pruefungsdatum" (test date) or "Klausurtermin" (examination date). This behavior is known in the information retrieval literature as the *vocabulary mismatch problem* [2].
2. **No handling of German compound words.** The DokuWiki search does not decompose compounds into their constituents. Searching for "Matura" finds pages containing "Maturaarbeit" only if substring search is supported – the default indexing yields unreliable results.
3. **No query contextualization.** Search queries in natural language (e.g., "How does the registration for the diploma thesis work?") are not interpreted as

questions. Instead, individual words are searched independently in the index. The order and context of terms are lost.

4. **No learning-based relevance weighting.** DokuWiki uses a simple heuristic for sorting search results. A semantic relevance weighting based on text embeddings is not available.

As part of the keyword baseline evaluation (cf. Chapter 6), the existing DokuWiki search was systematically evaluated with a test corpus of 78 question-answer pairs. The results quantitatively confirm the described weaknesses:

- **MRR with full questions:** 0.051 – the correct wiki page appears on average far beyond the first results when a natural-language question is used as the search query.
- **MRR with optimal keywords:** 0.365 – even with manually chosen optimal search terms, the MRR is below the MRR 0.872 achieved by the dense retrieval system (cf. Chapter 6).
- **Precision@10 with full questions:** 0.010 – the relevant page is almost never found among the first ten search results. Precision@10 is reported as the secondary metric for FF1 (cf. Section 1.2); the embedding model evaluation in Chapter 6 additionally reports P@5.

The discrepancy between question-based search (MRR 0.051) and keyword-based search (MRR 0.365) illustrates the core problem: users formulate queries as questions, but the system expects exact keywords. The quantitative gaps above therefore motivate an embedding-based search integrated via the Model Context Protocol [3], as evaluated in Chapter 6.

3.4 Security Requirements

3.4.1 Authentication Requirements

Authentication is performed via OAuth 2.1 with an external identity provider [22]. Three identity provider candidates were compared (cf. Table 3.2):

After evaluating the three alternatives, the team selected **Scalekit** based on its focus on SSO integration for educational institutions, low setup effort, and availability as a managed service. Keycloak's Java runtime imposes a significant memory overhead

Table 3.2: Comparison of identity provider alternatives (as of February 2026).

Criterion	Scalekit	Auth0	Keycloak
Type	Managed SaaS	Managed SaaS	Self-hosted
SSO for education	Yes (core focus)	Yes (enterprise)	Yes (config-intensive)
Setup effort	Low (SDK)	Medium	High (Java infrastructure)
ARM64 support	n/a (cloud)	n/a (cloud)	Yes, but resource-intensive
Cost	Free tier sufficient	Free tier limited	Free (self-hosted)

(typically several hundred megabytes at idle [27]), exceeding the available memory budget on the Raspberry Pi 5, so it was ruled out; Auth0's enterprise-oriented pricing model made it less suitable for a school project. Role assignment is performed via JWT claims in the OAuth 2.1 flow [22, 25].

3.4.2 Authorization Requirements

Access to search results is controlled on a role-based basis [26]. Three roles are defined:

Student Restricted access to student-relevant content.

Teacher Extended access to all content, including internal documents.

Admin Full access to all content and system administration functions.

For security reasons, two separate MCP tools are implemented: `search_content_student` (student-relevant content) and `search_content_teacher` (all content). This separation prevents RBAC circumvention through parameter manipulation (Security by Design).

3.4.3 Data Protection Requirements

The security architecture comprises the following measures:

- TLS termination via Caddy with automatic certificate renewal through Let's Encrypt [28, 29].
- Qdrant bound to localhost only in the deployment stack (Section 4.5).
- Network separation: `mcp-frontend` (public) and `mcp-backend` (internal).

- Security headers (HSTS, X-Frame-Options, X-Content-Type-Options).
- The `AuditLoggingMiddleware` is designed to support GDPR-aligned data minimization: user identifiers are SHA-256 hashed and no personal data is stored in the embeddings. No formal GDPR audit has been performed.

3.5 Functional Requirements

3.5.1 MCP Integration

FR-1: Semantic search via MCP. The system must provide semantic search functionality that can be integrated into AI clients (e.g., Claude Desktop, Cursor, VS Code) via the Model Context Protocol [3]. The search should understand natural-language queries and identify relevant wiki pages using text embeddings [8].

FR-2: Multi-client support. The system must be accessible via two transport protocols: `stdio` for local clients (e.g., Claude Desktop) and HTTP Streamable for web-based clients. Both transports follow the MCP specification [3].

Table 3.3 compares three integration protocol options.

Table 3.3: Comparison of integration protocol alternatives (as of February 2026).

Criterion	MCP	Custom REST API	GraphQL
AI client integration	Native (Claude, Cursor, VS Code)	Manual per client	Manual per client
Tool discovery	Automatic	Not available	Schema-based
Capability negotiation	Yes	No	No
Transport protocols	stdio, HTTP Streamable	HTTP	HTTP
Ecosystem maturity	Growing (est. 2024)	Mature	Mature

MCP was chosen as integration protocol [3]. It provides standardized mechanisms for tool discovery and capability negotiation that are absent in generic API protocols. A custom REST API would require separate integration for each client. GraphQL offers a schema but does not address the specific communication patterns required for AI tool integration [19].

Limitations of MCP. MCP is a relatively recent standard: the specification underwent three revisions within a year (2024-11-05, 2025-03-26, 2025-11-25), requiring repeated server updates. Not all AI clients implement the full feature set (cf. Section 5.10). Alternative approaches – such as OpenAI function calling with custom tool definitions, or framework-bound solutions like LangChain Skills – were not selected because they lack standardized tool discovery across multiple clients. MCP was chosen because it is the only evaluated option with both standardized tool discovery and session lifecycle management; its limitations are documented throughout Chapters 5 and 7.

3.5.2 Search Functionality

FR-3: Two separate search tools. Two separate MCP tools are implemented: `search_content_student` (student-relevant content) and `search_content_teacher` (all content), following the RBAC separation principle defined in Section 3.4.2.

FR-4: Offline pipeline for data preparation. The preparation of wiki content is performed as an offline five-stage pipeline; the stages are defined in Chapter 4. The pipeline operates independently of the runtime system and can be executed manually or automatically as needed.

Four vector database candidates were evaluated (cf. Table 3.4):

Table 3.4: Comparison of vector database alternatives (as of February 2026).

Criterion	Qdrant	Milvus	Pinecone	ChromaDB
Language	Rust	Go + C++	Managed	Python
Docker-native	Yes	Yes (complex)	No (cloud)	Yes (limited)
ARM64 support	Yes	Experimental	n/a	Yes
Hybrid search	Yes, native	Yes	No	No
RBAC	Collections	Partitions	Yes	No
License	Apache 2.0	Apache 2.0	Proprietary	Apache 2.0

Qdrant best matched the deployment constraints [30]. Qdrant provides native ARM64 Docker images, an integrated hybrid search implementation, and a collection-based architecture that maps directly to the RBAC requirements of this project. These concrete properties match the Raspberry Pi deployment constraints better than the alternatives evaluated. Milvus requires additional infrastructure components (etcd, MinIO) that exceed the Raspberry Pi's resource budget. Pinecone was excluded as a

cloud-only solution, since the system is to be operated on-premises. ChromaDB does not provide sufficient RBAC support.

Five embedding model candidates were compared for German-language wiki content; Table 3.5 lists four of the five models; Octen-Embedding-4B is introduced in Chapter 6. Language support is a relevant selection criterion [12].

Table 3.5: Comparison of embedding model alternatives (as of February 2026).

Criterion	**text-emb.-3-large**	**bge-m3**	**Snowflake Arctic**	**Octen-4B**
Provider	OpenAI (API)	BAAI (local)	Snowflake (local)	Octen (local)
Dimensions	3072	1024	1024	2560
MTEB (deu)	n/a	60.24	65.48	73.25
Cost	\$0.13/1M tok.	Free	Free	Free

OpenAI `text-embedding-3-large` [10] is used as the primary embedding model for the pipeline (cf. Section 4.2.2). The API stability and simple HTTP-based integration favor the commercial solution; the costs are negligible for a corpus of 196 wiki pages (\$0.14 for the production run). As part of research question FF3, five models are systematically compared (cf. Chapter 6), including four local models from the MTEB(deu) leaderboard and OpenAI as a paid reference.

3.5.3 Administration

FR-5: System monitoring. Administrators should be able to view system status and usage statistics via dedicated MCP tools (`get_collection_stats`, `get_query_statistics`, `health_check`).

Three reverse proxy options were compared (cf. Table 3.6):

Table 3.6: Comparison of reverse proxy alternatives (as of February 2026).

Criterion	**Caddy**	**Nginx**	**Traefik**
Auto-TLS	Yes, zero-config	Yes, via Certbot	Yes, native
Config effort	Minimal	Medium	Medium
SSE support	Yes (`flush_interval`)	Yes (manual)	Yes
ARM64	Yes	Yes	Yes

The deployment constraints – automatic TLS, minimal configuration, and native SSE support for HTTP Streamable transport – led to **Caddy** as the reverse proxy [28].

3.6 Non-Functional Requirements

3.6.1 Performance

NFR-1: Response time. Semantic search queries must be answered within 2 seconds (measured from receipt of the MCP request to return of results). This includes the vector search in Qdrant as well as result formatting.

NFR-2: Availability. The runtime system (MCP server, Qdrant, Caddy) must be operated as a Docker Compose stack that is automatically restored after a host system restart (`restart: unless-stopped`) [31].

3.6.2 Scalability

NFR-3: Reproducibility. All pipeline results must be tagged with a timestamp, configuration hash, and code version (`manifest.json`) to ensure traceability of data processing. The pipeline is designed so that it can be re-executed fully or incrementally when wiki content changes.

3.6.3 Maintainability

NFR-4: Maintainability. System maintainability is ensured through the following measures:

- Configuration via YAML files (`config/env.yaml`), secrets in separate `.token` files.
- Docker-based deployment with clearly defined service boundaries.
- Watchdog service for automated data updates.

NFR-5: Data protection (GDPR). The data protection measures defined in Section 3.4.3 apply, including GDPR-aligned audit logging (SHA-256 hashed identifiers) and the exclusion of personal data from embeddings. No formal GDPR audit has been performed.

4 System Architecture

4.1 System Overview

4.1.1 High-Level Architecture

The overall system consists of two architecturally separated areas: an offline pipeline for data preparation and a runtime system for answering semantic search queries. This separation is a deliberate design principle – the MCP server does not connect directly to LeoWiki at runtime but accesses exclusively the prepared vector data in Qdrant. Figure 4.1 provides an overview of both phases.

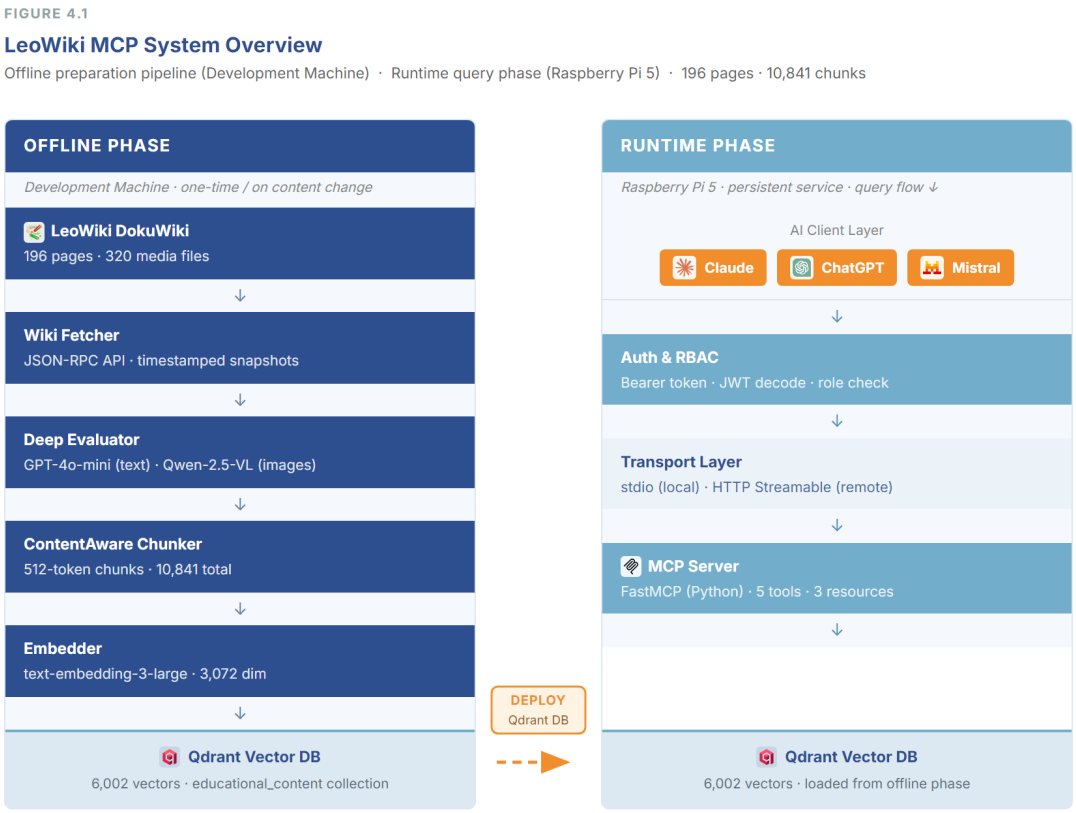


Figure 4.1: LeoWiki MCP system overview. The offline phase (left) prepares the knowledge base on a development machine: wiki content is fetched, evaluated, chunked, embedded, and indexed into Qdrant. The resulting vector database is deployed to the Raspberry Pi 5 (right), where the runtime phase serves semantic search queries via the MCP protocol to AI clients.

The offline pipeline is executed on a development machine and transforms the LeoWiki content into vectorized data. The runtime system, comprising four Docker services on a Raspberry Pi, exposes this data via the Model Context Protocol (MCP) for AI clients. This separates data preparation from production operation.

4.1.2 Data Flow

The data flow in the system can be divided into two phases: the preparation phase (offline pipeline) and the query phase (runtime system).

Preparation Phase: Five-Stage Pipeline

Data preparation follows a sequential five-stage pipeline. The production-oriented runs in this thesis used a wiki snapshot fetched on 2026-02-25, with RAG preprocessing on 2026-02-26 and embedding on 2026-02-26. The full evaluation corpus (10,841 chunks) was produced from pipeline runs on 2026-02-17 using a snapshot fetched on 2026-02-16.

1. **Wiki Fetcher:** Communicates with LeoWiki via the DokuWiki XML-RPC API [32]. The fetcher wraps each XML-RPC call in a Python helper that returns JSON-structured data for downstream pipeline stages. Page contents (DokuWiki syntax), metadata (JSON), HTML renderings, and media files are downloaded. The output is placed in a timestamped directory.

2. **Deep Evaluation:** Classifies content using external LLM APIs. OpenAI `gpt-4o-mini` [33] (model snapshot `gpt-4o-mini-2024-07-18`, Chat Completions API, accessed 2026-02-17) classifies text, and `qwen/qwen2.5-vl-7b` via LM Studio (local OpenAI-compatible endpoint; LM Studio build not pinned in the repository) performs vision-based analysis of media files. The result is a routing configuration for the next stage.

3. **RAG Preprocessing:** Transforms DokuWiki syntax into Markdown with YAML front matter. The output comprises 196 preprocessed pages and 320 media descriptions. Each document receives structured metadata.

4. **Embeddings Creator:** Segments the Markdown documents context-aware into chunks and converts them into 3,072-dimensional vectors using the embedding model described in Section 4.2.2. The output is in JSONL format.

5. **Deploy:** Loads the JSONL file into the Qdrant vector database. Three modes are available: direct upsert via the Qdrant REST API, export to a monitored directory (watchdog mode), or SCP transfer to the Raspberry Pi.

Each stage produces a clearly defined intermediate format. This makes the pipeline modular: individual stages can be re-executed independently without running the entire chain. Figure 4.2 shows the five stages with their concrete parameters and outputs. Table 4.1 documents the data formats between stages.

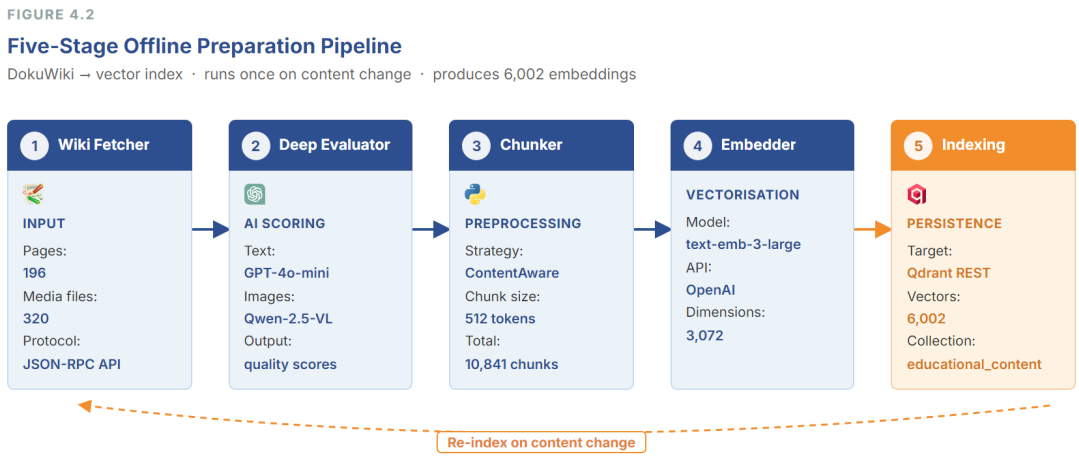


Figure 4.2: Five-stage offline preparation pipeline with concrete parameters. Wiki content is fetched via XML-RPC (196 pages, 320 media), quality-evaluated by LLMs, chunked into up to 10,841 segments of 512 tokens (evaluation corpus), embedded into 3,072-dimensional vectors, and indexed in Qdrant (6,002 vectors, production run with stricter length filter).

The pipeline produces different output sizes at each stage. The wiki fetcher retrieves 196 content pages and 320 media files (516 raw documents in total). The deep evaluation stage filters exactly 80 documents classified as empty, spam, or out of scope ($516 - 436 = 80$), yielding 436 Markdown files for downstream processing. The chunking step generates up to 10,841 text segments from the evaluation corpus, which uses all remaining documents to maximize model-comparison coverage. The production embedding run applies stricter content-length filtering and produces 6,002 vectors indexed in Qdrant. This intentional discrepancy – evaluation corpus (10,841 chunks) versus production index (6,002 vectors) – is discussed in detail in Section 6.3.4.

Query Phase: Runtime System

A typical search query at runtime traverses the following steps:

Table 4.1: Data formats between pipeline stages.

Transition	Format	Description
Stage 1 → 2	TXT, JSON, HTML	Raw wiki content, metadata, HTML renderings
Stage 2 → 3	YAML	Routing configuration with classification results
Stage 3 → 4	Markdown + YAML	Preprocessed documents with structured metadata
Stage 4 → 5	JSONL	One JSON object per line with embedding vector

1. The AI client sends an MCP tool request to the domain `leowiki-mcp.stream`.
2. Caddy v2.9.1 terminates TLS and forwards the request to the MCP server.
3. The middleware validates the JWT token via Scalekit and checks the RBAC permissions.
4. The MCP server performs a vector search in Qdrant.
5. The results are returned to the client as an MCP response.

The entire chain must complete within 2 seconds (NFR-1). The generation phase is handled by the AI client itself – the MCP server delivers exclusively the relevant context data.

4.2 Component Architecture

4.2.1 MCP Server Component

The MCP server forms the central component of the runtime system. It is implemented in Python 3.13 using FastMCP 3.0 and processes MCP requests via JSON-RPC 2.0 [17, 34]. The server exposes 5 tools, 6 resources, and 2 prompts.

The processing of each request traverses a four-stage middleware chain:

1. **Request Logging Middleware:** Generates correlation IDs and captures response times for monitoring.
2. **User Context Middleware:** Extracts JWT claims from the bearer token and provides the user context.
3. **RBAC Enforcement Middleware:** Checks access authorization at the tool level based on role claims.

4. **Audit Logging Middleware:** Logs access in pseudonymized fashion for traceability. No formal GDPR audit was performed; the SHA-256 hashing of user identifiers serves as a first-level pseudonymization measure.

The server supports two transport protocols per the MCP specification [3]: **stdio** for local clients such as Claude Desktop (communication via stdin/stdout) and **HTTP Streamable** for web-based clients (port 8000 internally, Server-Sent Events for asynchronous notifications). Additionally, an HTTP bridge (`http_bridge.py`) exists that connects stdio-based clients with a remote HTTP Streamable server.

Figure 4.3 provides an overview of the three-tier component architecture, showing how AI clients communicate with the FastMCP server layer via the MCP protocol, and how the server queries the Qdrant vector database.

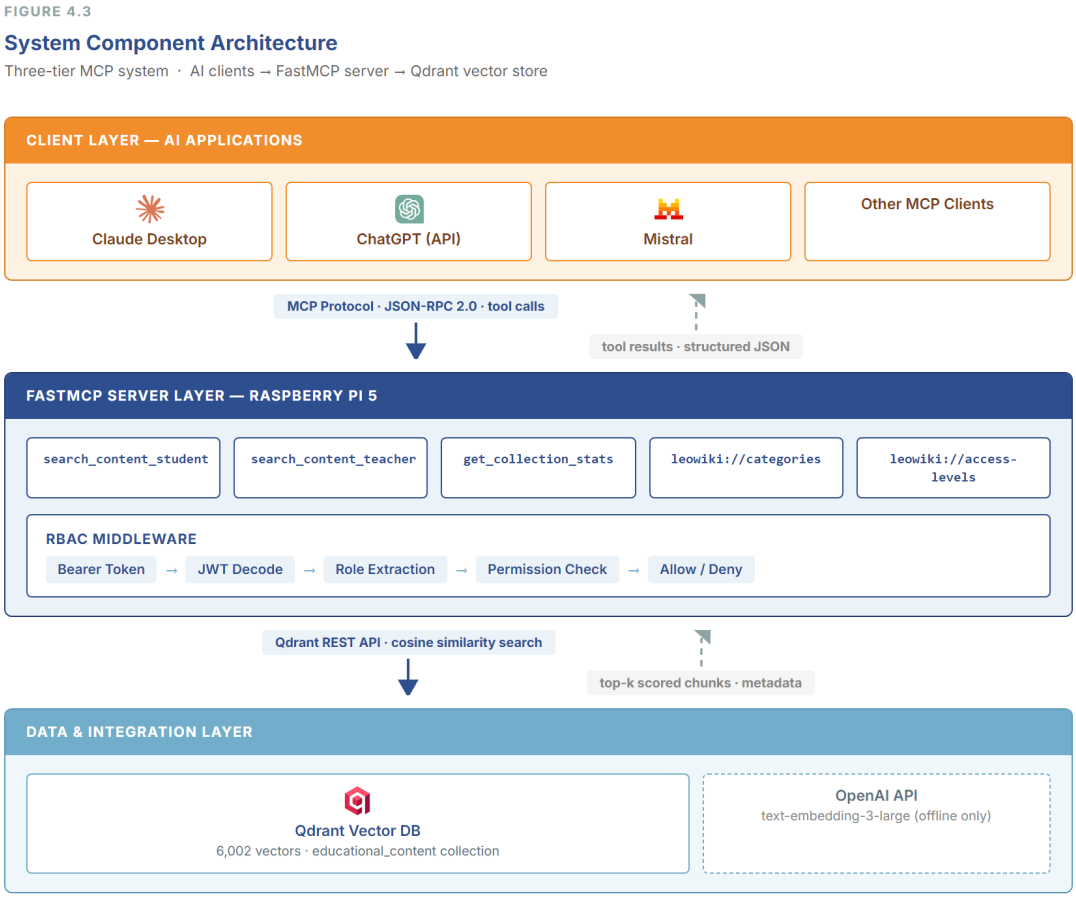


Figure 4.3: System component architecture. AI clients communicate with the FastMCP server via the MCP protocol (JSON-RPC 2.0). The server applies RBAC middleware to validate bearer tokens and enforce role-based access, then queries the Qdrant vector database using cosine similarity search. The OpenAI API is used exclusively during offline indexing (dashed border).

4.2.2 Embedding Service Component

The embedding service is implemented as the fourth stage of the offline pipeline. The `ContentAwareChunker` segments Markdown documents context-aware while respecting semantic boundaries (headings, paragraphs, code sections). The resulting chunks are converted via the OpenAI API using the `text-embedding-3-large` model into 3,072-dimensional vectors [10].

Processing occurs in batch mode with a batch size of 20 chunks per API request and integrated rate limiting. The output is stored in JSONL format, where each line is a JSON object with the fields `id` (UUID), `text` (chunk content), `embedding` (3,072-dimensional vector), and `metadata` (title, page ID, namespace, chunk index, access rights).

4.2.3 Retrieval Service Component

The retrieval service is integrated into the MCP server and communicates via the Qdrant REST API on port 6333 [30]. For an incoming search query, the search text is vectorized with the same embedding model and a cosine similarity search is performed on the Qdrant collection `educational_content` [2].

Results are returned as structured JSON objects, where each hit contains the fields `text` (chunk content), `score` (similarity value), and `metadata` (title, namespace, page ID, access rights). Filtering by access rights is performed already at the Qdrant level via payload filters, so that students receive exclusively content released for them.

4.2.4 Authentication Component

Authentication is performed via OAuth 2.1 through Scalekit [22]. AI clients send a JWT bearer token with every request. The User Context Middleware validates the token via the Scalekit endpoint and extracts the contained role claims (`student`, `teacher`, `admin`).

The server uses separate search tools instead of parameterized access control: `search_content_student` returns exclusively public content, while `search_content_teacher` also includes internal content, as specified in Section 3.4.2. The RBAC Enforcement Middleware checks at the tool level whether the user's role permits access to the requested tool.

4.2.5 Wiki Connector Component

The Wiki Connector is implemented as the first stage of the offline pipeline and communicates with LeoWiki (`leowiki.htl-leonding.ac.at`) via the DokuWiki XML-RPC API [32]. Authentication uses a bearer token stored in a separate `.token` file.

The connector downloads four data types:

- **Page contents:** Raw DokuWiki syntax as TXT files
- **Metadata:** Structured JSON files with author information, timestamps, and access rights
- **HTML renderings:** Rendered page versions for quality control
- **Media files:** PDFs, images, and other embedded resources

As noted in Section 4.1.1, the wiki connector operates exclusively during offline preparation. Updates require re-executing the pipeline.

4.3 Technology Decisions

4.3.1 Vector Database: Qdrant

After evaluating Qdrant alongside the alternatives in Table 3.4, Qdrant v1.14.1 was chosen as the vector database [30]. ChromaDB lacked native payload filtering for RBAC enforcement; Milvus and Pinecone exceeded the deployment constraints. The key decision criteria for Qdrant were:

- **Resource efficiency:** Qdrant is implemented in Rust and is suitable for operation on a Raspberry Pi with limited memory.
- **Payload filtering:** Native support for metadata-based filtering, which is required for role-based access control at the chunk level.
- **Cosine similarity:** Support for the cosine distance metric for semantic search [2].
- **REST API:** Simple integration via HTTP without additional client libraries.
- **Docker support:** Official Docker image available for ARM64 architecture.

The Qdrant instance manages a collection `educational_content` with 3,072-dimensional vectors and cosine similarity as the distance metric.

4.3.2 Authentication Provider: Scalekit

Scalekit was chosen over Auth0 and Keycloak as the identity provider because it offers a free tier sufficient for the project's scale, requires no self-hosted infrastructure (unlike Keycloak), and provides OAuth 2.1 with JWT-based authentication [22] out of the box. The Python integration uses `scalekit-sdk-python` ≥2.4.0 (configured February 2026). The integration enables:

- **Role-based claims:** JWT tokens contain role claims (`student`, `teacher`, `admin`), which are used directly for the RBAC decision.
- **Token validation:** Standards-compliant validation of bearer tokens without implementing custom cryptography.
- **Decoupling:** User management resides outside the MCP server, which increases maintainability.

4.3.3 Embedding Model Selection

The production deployment uses the embedding model described in Section 4.2.2. The evaluation in Chapter 6.1 shows that the locally executable model Octen-Embedding-4B achieves the best ranking quality (MRR 0.883), but requires 7.7 GB VRAM. Since the target platform (Raspberry Pi 5, 8 GB RAM, no GPU) cannot run the 4B model locally, OpenAI was selected for production (MRR 0.872; Δ 1.1 percentage points). The detailed justification of this engineering trade-off is provided in Section 6.1.4.

4.3.4 Runtime Environment

The runtime system is based on the following technology decisions:

- **Python 3.13:** The MCP server is implemented in Python 3.13. The choice of Python follows from the availability of the FastMCP framework and extensive library support for NLP tasks. The evaluation stack uses Python 3.11 (see Appendix .6).
- **FastMCP 3.0:** Framework for MCP protocol implementation with built-in support for stdio and HTTP Streamable transport [34].
- **Docker Compose v2.32:** Orchestration of the four services (Caddy, MCP server, Qdrant, Watchdog) with defined networks, volumes, and dependencies [31].

- **Raspberry Pi (ARM64):** Target platform for deployment, chosen due to low operating costs and constant availability in the school network.

4.4 Interface Design

4.4.1 MCP Tool Interface

The MCP tools represent the primary interaction interface for AI clients. Each tool is specified as an MCP tool definition with name, description, input schema, and access rule. Table 4.2 provides an overview of the available tools.

Table 4.2: MCP tool definitions with input parameters and access rules.

Tool	Input	Access	Output
`search_content_student`	`query`, `limit`	Student+	Chunks with score and source
`search_content_teacher`	`query`, `limit`	Teacher+	Chunks incl. internal content
`get_collection_stats`	–	Admin	Vector count, dimensions
`get_query_statistics`	–	Admin	Query history, response times
`health_check`	–	All	Server status, uptime

Additionally, the server provides six **resources** (admin-only; see Table 6.16 in Section 6.6.3), which offer read-only access to static and dynamic information via URI templates:

- `leowiki://categories` – Listing of content categories
- `leowiki://access-levels` – Documentation of the RBAC configuration
- `leowiki://search-hints` – Tips for optimal formulation of search queries
- `leowiki://system-prompt` – Guidelines for AI client interaction
- `leowiki://stats` – Live system statistics
- `leowiki://recent/{count}` – Recently updated content (parameterized)

Two **prompts** define pre-built interaction patterns: `explain_topic` for pedagogical explanations based on wiki content and `summarize_search` for synthesizing search results.

4.4.2 Internal Service Interfaces

Communication between the MCP server and Qdrant uses the Qdrant REST API on port 6333 [30]. The relevant endpoints are:

- `POST /collections/{name}/points/search` – Vector search with optional payload filters
- `GET /collections/{name}` – Collection statistics query
- `PUT /collections/{name}/points` – Batch upsert (watchdog/deploy only)

Access to Qdrant occurs through the internal Docker network `mcp-backend`; binding to localhost is documented in Section 4.5. For administrative access from outside the Docker stack, an SSH tunnel or Tailscale is used.

The connection between Caddy and the MCP server operates as an HTTP reverse proxy. Caddy forwards incoming HTTPS requests to `mcp-server:8000` and handles TLS termination with automatic Let's Encrypt certificate renewal [29, 28]. SSE support is configured via `flush_interval -1` to enable real-time streaming.

4.4.3 External Integrations

The system interacts with several external services. Table 4.3 documents the endpoints and protocols.

Table 4.3: External services and their protocols.

Service	**Protocol**	**Endpoint**	**Used by**
DokuWiki	XML-RPC	leowiki.htl-leonding.ac.at	Stage 1
OpenAI Emb.	HTTPS REST	api.openai.com/v1	Stage 4
OpenAI Chat	HTTPS REST	api.openai.com/v1	Stage 2
LM Studio	HTTP	`<dev-machine>`:1234/v1	Stages 2, 3
Qdrant	HTTP REST	localhost:6333	Stage 5, MCP
Scalekit	OAuth 2.1	(env-specific)	MCP server

4.5 Deployment Architecture

4.5.1 Container Architecture

The runtime system consists of four Docker services in two separate networks.

Frontend Network (`mcp-frontend`)

The frontend network connects the two externally facing services:

- **Caddy** (reverse proxy): Receives incoming HTTPS requests on port 443, terminates TLS with automatically managed Let's Encrypt certificates [28], and forwards to the MCP server. Security headers (HSTS, X-Frame-Options, X-Content-Type-Options, Referrer-Policy) are set automatically. Caddy is the only service with external port mapping (443, 80).
- **MCP Server** (Python 3.13, FastMCP 3.0): Processes MCP requests on port 8000 (internally reachable only). Implements the complete middleware chain and authentication via OAuth 2.1 through Scalekit.

Backend Network (`mcp-backend`)

The backend network is not reachable from outside and connects:

- **Qdrant:** The vector database is bound exclusively on `127.0.0.1:6333` (HTTP REST) and `127.0.0.1:6334` (gRPC). Persistent storage uses a Docker volume (`qdrant_data`).
- **Watchdog:** Monitors an incoming directory (`data/incoming/`) for new JSONL files and automatically performs batch upserts into Qdrant (100 points per request). If needed, the collection is cleared before the update.

The MCP server is a member of both networks and serves as the sole bridge point between frontend and backend. This network separation follows the principle of minimal attack surface [35]. Figure 4.4 visualizes the container architecture with both networks.

4.5.2 Deployment Options

The system supports three deployment modes for updating the vector data:

1. **Direct upsert:** The JSONL file is loaded directly into the target collection via the Qdrant REST API. This mode is suitable for development and initial population.
2. **Watchdog mode:** The JSONL file is exported to the monitored directory `data/incoming/`. The watchdog service detects the new file automatically and performs the batch upsert.

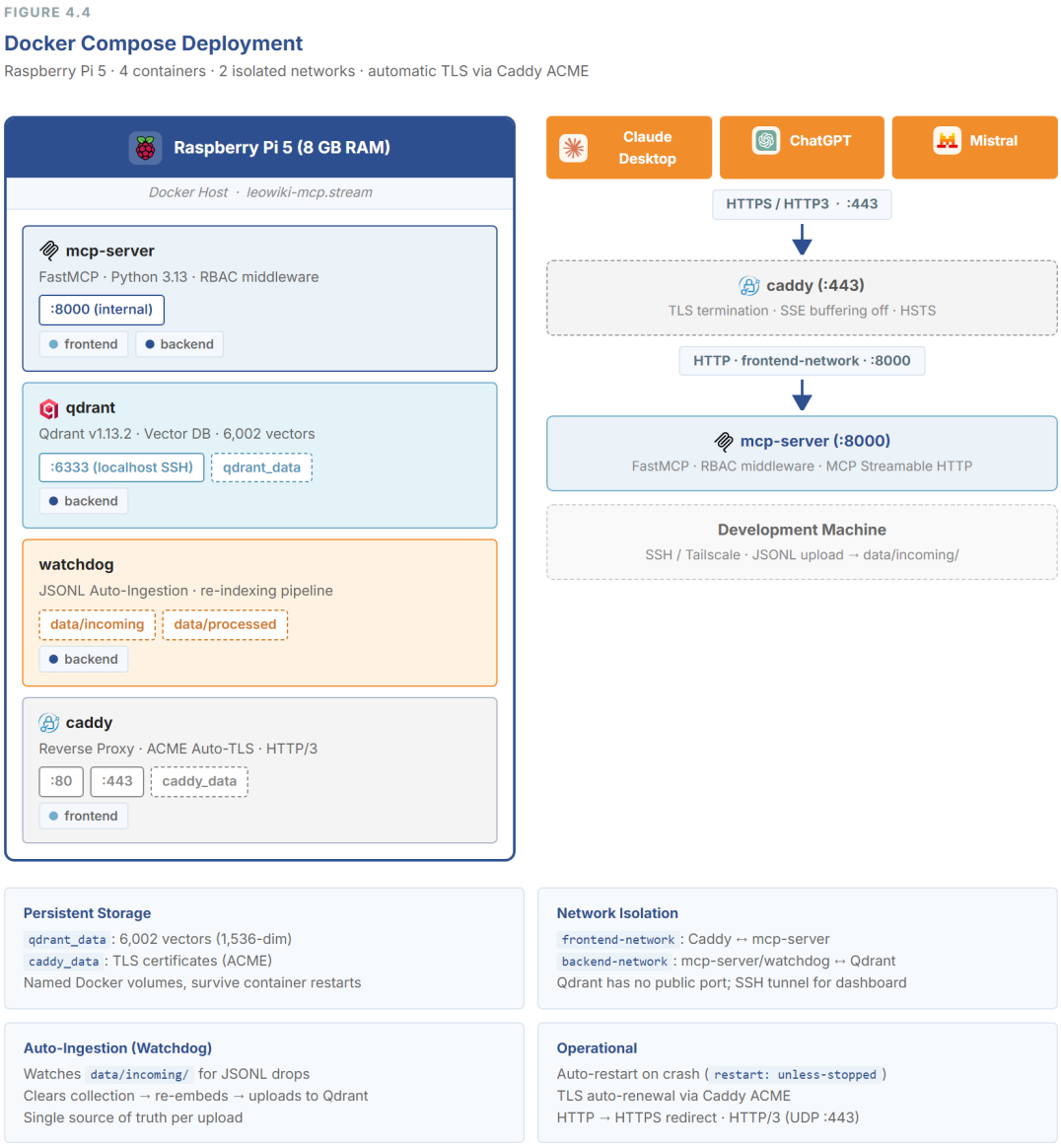


Figure 4.4: Docker Compose container architecture with two isolated networks. Caddy is the only externally exposed service. The MCP server bridges both networks; Qdrant and Watchdog are isolated in the backend.

3. **SCP transfer:** The JSONL file is transferred to the Raspberry Pi via SCP, where it is processed by the watchdog service. This mode enables updates over the network.

The target platform is a Raspberry Pi (ARM64) with Docker Compose. The domain `leowiki-mcp.stream` is operated via Caddy with automatic TLS management.

4.5.3 Configuration Management

The system configuration follows a strict separation [31]:

- **Static configuration:** YAML files (`config/env.yaml`) contain model parameters, chunk sizes, and API endpoints. These files are version-controlled.
- **Secrets:** Separate `.token` files (`config/secrets/*.token`) contain API keys and authentication tokens. These are not included in version control.
- **Docker configuration:** The file `docker-compose.yml` defines services, networks, volumes, and environment variables.
- **Manifest:** Each pipeline output includes a `manifest.json` with timestamp, configuration hash, and code version (NFR-3), ensuring reproducibility of all processing steps.

5 MCP Server Implementation

Building on the architecture described in Chapter 4.2, this chapter documents the development from a custom server implementation to FastMCP 3.0, the realization of two transport protocols, the development of role-aware search tools protected by a middleware chain, and a systematic compatibility analysis across current AI clients. The production server is accessible at `https://leowiki-mcp.stream/mcp`.

5.1 SDK Evolution and Migration

The MCP server went through four iterative implementation phases. Each phase addressed limitations discovered during development and adopted new SDK capabilities as they became available.

5.1.1 Phase 1: Custom MCP Server

The initial implementation used a custom server architecture with two classes: `BaseMCPServer` (`src/server/base.py`) and `HTTPMCPServer` (`src/server/http_server.py`). The HTTP server was built directly on FastAPI 0.115 [36] and manually handled JSON-RPC 2.0 message routing [17], Server-Sent Events (SSE) streaming, and tool dispatching.

This phase had several limitations:

- Only two tools (`search_content`, `health_check`).
- No MCP resources or prompts.
- A single search tool with an `access_level` parameter—a security weakness because LLMs could manipulate the parameter.
- Manual JWT validation via JWKS URL.
- Manual SSE endpoint management.

5.1.2 Phase 2: FastMCP Adoption (v0.8.x)

The recognition that manual MCP protocol handling was error-prone and missing critical features led to the adoption of FastMCP [34]. The first version used was `fastmcp>=0.8.0`. This migration replaced the custom server classes with a single `FastMCP()` instance and decorator-based tool registration.

The `src/server/__init__.py` file documents this transition:

> *"The old http_server.py and base.py are deprecated. We now use FastMCP library in main.py instead of custom server implementation."*

A gap analysis documented in `docs/refactor/LeoWiki_MCP_Enhancement_Plan.md` identified that most of FastMCP's capabilities remained unused: no resources, no prompts, no custom middleware, no tool annotations, no progress reporting, no error masking, and no server instructions for LLM guidance.

5.1.3 Phase 3: Feature Completion (v2.x)

The third phase, implemented on the `feature/professional-mcp-enhancements` branch, addressed all identified gaps. The changes were guided by an SDK reference document (`docs/refactor/FastMCP_SDK_Reference3.md`) and a detailed enhancement plan.

Key changes in this phase:

1. **Two-tool RBAC:** The single `search_content` tool was split into two role-specific tools (design rationale in Section 4.2.4), eliminating the parameter manipulation vulnerability.
2. **Four custom FastMCP middleware** components for request logging, user context extraction, RBAC enforcement, and audit logging, plus HTTP-level Scalekit middleware (five layers in total; cf. Table 5.2).
3. **Six MCP resources** providing metadata, statistics, and dynamic content.
4. **Two MCP prompts** for educational interactions.
5. **Lifespan dependency injection** via the `AppContext` dataclass.
6. **Tool annotations** (`readOnlyHint`, `idempotentHint`, `openWorldHint`).
7. **Progress reporting** in search tools.
8. **Error masking** (`mask_error_details=True`).

5.1.4 Phase 4: FastMCP 3.0 Beta Migration

The final phase migrated to `fastmcp>=3.0.0b2`, which introduced breaking changes and new features. Table 5.1 summarizes the breaking changes that required code modifications.

Table 5.1: Breaking changes in the FastMCP 3.0 migration.

Area	**FastMCP 2.x (old)**	**FastMCP 3.0 (new)**
Duplicate handler	`on_duplicate_tools="error"`	`on_duplicate="error"`
Context state (read)	`ctx.get_state("key")` (sync)	`await ctx.get_state("key")` (async)
Context state (write)	`ctx.set_state("key", val)` (sync)	`await ctx.set_state("key", val)` (async)

The sync-to-async migration of context state methods—reflecting Python's broader adoption of native coroutines [37]—affected approximately 50 code locations across `search_tools.py` and `mcp_middleware.py`. Each location is annotated with a `# FastMCP 3.0: async methods` comment for traceability.

FastMCP 3.0 also introduced features used by the server:

- **`ResourcesAsTools` and `PromptsAsTools` transforms:** Expose resources and prompts as callable tools, providing backward compatibility with AI clients that do not natively support these primitives.
- **Server icon:** The `Icon` class from `mcp.types` provides a logo image via data URI, displayed in client UIs.
- **`strict_input_validation`:** Enables stricter validation of tool arguments with better error messages.
- **`mcp.http_app(path="/mcp")`:** Creates a clean ASGI application for mounting in a FastAPI host.

5.1.5 Evolution Summary

Table 5.2 summarizes the capability growth across all four phases.

Figure 5.1 visualizes the four-phase progression and highlights the growing capabilities at each stage.

Table 5.2: Server evolution across four development phases.

Phase	Tools	Res.	Prompts	MW	RBAC
v1.0 (raw SDK)	2	0	0	1	Parameter-based
v1.x (FastMCP 0.8)	2	0	0	1	Parameter-based
v2.0 (FastMCP 2.x)	5	6	2	5	Two-tool + middleware
v2.1 (FastMCP 3.0b)	5	6	2	5	+ Transforms

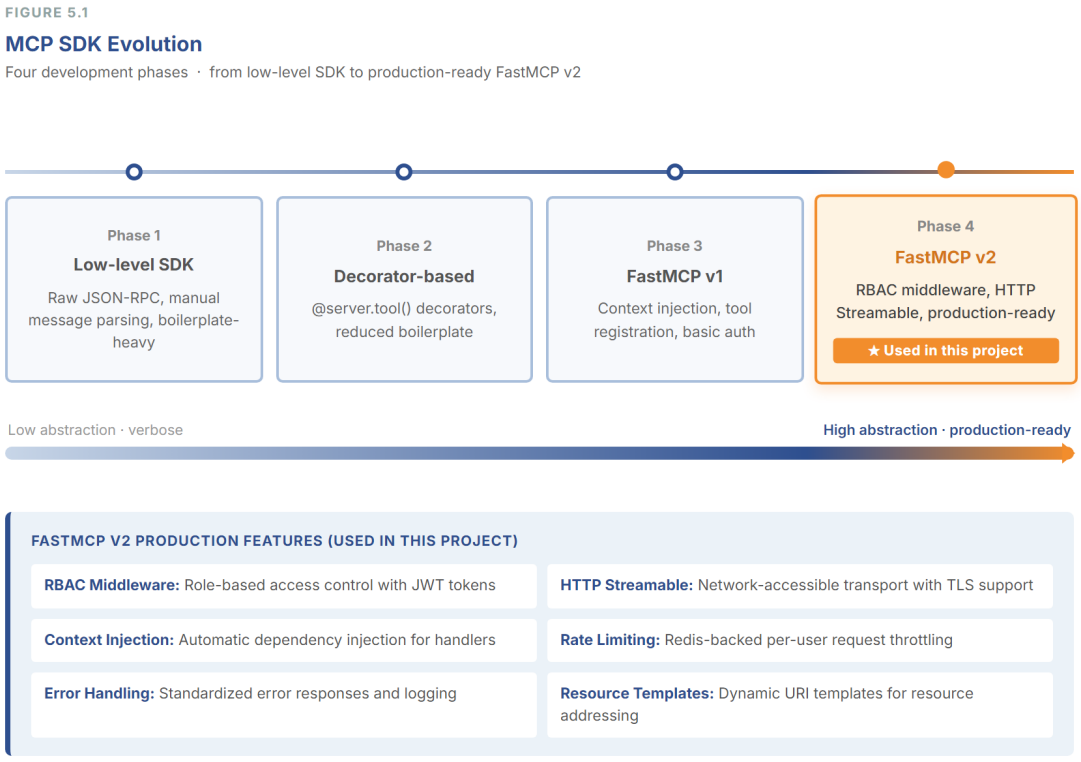


Figure 5.1: MCP SDK evolution across four development phases. The current deployment (v2.1) uses FastMCP 3.0 (Phase 4), which adds the most SDK abstraction among the four phases, including RBAC middleware and HTTP Streamable transport.

5.2 Server Initialization and Configuration

The server's entry point is `main.py`, which orchestrates the initialization of all components. Listing 5.1 shows the FastMCP 3.0 constructor with the production configuration.

```
mcp = FastMCP(
    name=config.server_name,
    version=config.server_version,
    website_url="https://leowiki.htl-leonding.ac.at",
    icons=[server_icon] if server_icon else None,
    instructions="""
    LeoWiki MCP Server - HTL Leonding
    Du bist ein Assistent fuer Schueler und Lehrer.
    TOOLS:
    - search_content_student: Suche (eingeschraenkt)
    - search_content_teacher: Suche (voller Zugriff)
    - get_collection_stats: Statistiken (Admin)
    - get_query_statistics: Analytics (Admin)
    - health_check: Server-Status
    RESOURCES (6): leowiki://categories, ...
    PROMPTS (2): explain_topic, summarize_search
    SPRACHE: Deutsch
    AUTH: OAuth 2.1 via Scalekit
    """,
    mask_error_details=True,
    strict_input_validation=True,
    on_duplicate="error",
    lifespan=app_lifespan,
)
```

Listing 5.1: FastMCP 3.0 server initialization (`main.py`).

Server capabilities (I2). When a client connects, the MCP protocol mandates an `initialize` handshake in which the server advertises its capabilities. The LeoWiki MCP server responds with the following capability set (the `protocolVersion` field reflects the revision the server was built against, not the latest specification revision):

```
{
  "protocolVersion": "2025-03-26",
  "capabilities": {
    "tools":     { "listChanged": true },
    "resources": { "listChanged": true, "subscribe": false },
    "prompts":   { "listChanged": true },
    "logging":   {}
  },
  "serverInfo": {
    "name": "leowiki-mcp",
    "version": "1.0.0"
  }
}
```

Listing 5.2: Server capabilities returned in the `initialize` response (protocol version reflects the revision the server was built against; the latest MCP specification revision is 2025-11-25, see Section 5.10).

The `listChanged` flag enables clients to detect dynamic tool registration changes. The server exposes 5 tools (Section 5.6), 6 resources, and 2 prompts. The `logging` capability advertises support for structured log messages transmitted via MCP notifications. Upgrading to the newer specification revision (2025-11-25) was not required, as that revision introduced capability extensions not used by the current implementation; all five tested clients negotiate successfully against revision 2025-03-26 (Section 5.10).

Configuration management. The server configuration is managed through a `ServerConfig` class based on Pydantic Settings v2.x (Pydantic v2) [38], which provides typed configuration with over 40 fields, environment variable binding, automatic validation, and sensible defaults. Configuration sections include transport settings, HTTP options, vector database connection, embedding model, Scalekit OAuth parameters, RBAC settings, rate limiting, and logging.

Dependency injection. The `app_lifespan` context manager initializes the `QdrantBackend` (Qdrant v1.14.1), `EmbeddingService`, `ServerConfig`, and an in-memory cache at server startup. These dependencies are bundled in a typed `AppContext` dataclass and made available to all tools via `ctx.lifespan_context`:

```python
@dataclass
class AppContext:
    qdrant: QdrantBackend
    embedding_service: EmbeddingService
    config: ServerConfig
    cache: dict

@asynccontextmanager
async def app_lifespan(server: FastMCP) -> AsyncIterator[AppContext]:
    config = ServerConfig()
    qdrant = QdrantBackend(
        url=config.vector_db_url,
        api_key=config.vector_db_api_key
    )
    embedding_service = EmbeddingService(
        api_key=config.openai_api_key,
        model=config.embedding_model,
        dimensions=config.vector_dimensions
    )
    cache = {}
    try:
        yield AppContext(
            qdrant=qdrant,
            embedding_service=embedding_service,
            config=config, cache=cache)
    finally:
        cache.clear()
        logger.info("Cleanup complete")
```

Listing 5.3: Lifespan-based dependency injection (`src/server/lifespan.py`).

5.3 Transport Protocols

As introduced in Section 4.2.1, the server supports two transport protocols [3]: HTTP Streamable as the primary production transport and `stdio` as a local development fallback. The transport is selected at startup via a command-line flag:

```
if __name__ == "__main__":
    if "--http" in sys.argv:
        # HTTP mode: production behind Caddy
        uvicorn.run(app, host=config.http_host,
                    port=config.server_port)
    else:
        # STDIO mode: local development
        mcp.run()
```

Listing 5.4: Transport mode selection (`main.py`).

5.3.1 HTTP Streamable Transport

The HTTP Streamable transport is the primary production mode. FastMCP 3.0's `mcp.http_app(path="/mcp")` method creates an ASGI application that is mounted into a FastAPI host [36]:

```
mcp_app = mcp.http_app(path="/mcp")
app = FastAPI(
    title=config.server_name,
    version=config.server_version,
    lifespan=mcp_app.lifespan
)
# Middleware, OAuth routes, etc.
app.mount("/", mcp_app)
```

Listing 5.5: ASGI app mounting (`main.py`).

In production, the server runs behind Caddy v2.9.1 (Alpine image) as a reverse proxy [28] that handles TLS termination via Let's Encrypt and SSE configuration. The Caddyfile configures `flush_interval -1` (immediate flushing for SSE), no caching for event-stream responses, and extended read/write timeouts for long-running operations.

5.3.2 Client Integration via Direct URL

AI clients connect to the server by registering the production URL `https://leowiki-mcp.stream/mcp` in their settings. No `npx` commands, bridge scripts, or local installations are required:

- **Claude Desktop / Claude Web:** Register as a remote MCP server. The Scalekit OAuth redirect handles authentication automatically.
- **ChatGPT:** Register as an "App" via the ChatGPT interface.
- **Mistral:** Register as a remote tool connector.

The `http_bridge.py` file in the repository is a development-only fallback tool that translates between `stdio` and HTTP. It is not required for production use.

5.3.3 stdio Transport

The `stdio` transport runs the server as a local child process of the AI client. When started without the `-http` flag, the server calls `mcp.run()`, which reads JSON-RPC 2.0 messages from `stdin` and writes responses to `stdout`.

Limitation. The `stdio` transport does not support OAuth authentication because there is no HTTP endpoint to initiate the authorization flow. The server assigns a configurable default role for `stdio` connections.

5.3.4 Transport Comparison

Table 5.3 summarizes the characteristics of each transport and the recommended use cases.

Table 5.3: Transport comparison and recommendations.

Criterion	**HTTP Streamable**	**stdio**
Authentication	OAuth 2.1 via Scalekit	Default role only
Concurrent users	Yes (async event loop)	No (single process)
Remote access	Yes (Caddy + TLS)	Local only
SSE streaming	Yes (progress, notifications)	Via stdout
Setup complexity	Docker Compose + Caddy	Single command
Transport overhead	$\sim$113 ms (σ=8.0)	$\sim$18 ms (σ=0.2)
Use case	**Production, multi-user**	**Local development**

Transport overhead. To quantify the transport overhead (research question I3), a benchmark suite was executed on the production hardware (Raspberry Pi 5, 8 GB RAM).[1] Transport overhead was isolated using the `health_check` tool (minimal payload) with 20 iterations per transport after one warm-up request, timed via `time.perf_counter()`. The stdio transport achieved a mean latency of 17.9 ms

[1]The runs use the same Raspberry Pi 5 as production; the scripts still print "Raspberry Pi 4" because they parse `/proc/cpuinfo`, which does not name the board generation explicitly on the Pi 5. A more reliable method would be `cat /proc/device-tree/model`, which returns the full board name; this is noted for a future update of the benchmark scripts.

(σ=0.2 ms), while HTTP Streamable averaged 113.1 ms (σ=8.0 ms), yielding a 6.3× speedup for stdio. The ∼95 ms overhead of the HTTP path is attributable to TLS handshake (∼30–50 ms), network round-trip via Tailscale VPN (∼50–80 ms), and Caddy reverse proxy processing (∼5–10 ms).

Search pipeline latency. A separate benchmark measured the core search pipeline *without* MCP protocol overhead, using 78 ground-truth Q&A pairs over three iterations (234 requests total).

Table 5.4: Search pipeline latency (78 queries × 3 iterations = 234 requests, Raspberry Pi 5).

Component	Mean (ms)	Median (ms)	P95 (ms)	Min (ms)	Max (ms)
Embedding (OpenAI API)	181.8	174.8	n/a†	144.3	434.4
Vector search (Qdrant)	26.1	22.9	n/a†	21.4	50.0
Total	**207.9**	**199.7**	**263.6**	167.1	459.7

† P95 is reported for the total pipeline only; component-level P95 was not separately measured in this benchmark run.

Table 5.4 breaks down the latency. Embedding generation via the OpenAI API accounts for 87 % of the total latency (mean 181.8 ms), while the Qdrant HNSW vector search over 6 082 documents[2] contributes only 13 % (mean 26.1 ms). Query difficulty (easy, medium, hard) does not significantly affect latency, as both embedding generation and cosine similarity search have approximately constant per-query cost. The overall P95 latency of 263.6 ms confirms that the search pipeline alone is well within the NFR-1 target of 2 s. Combined with the transport overhead, estimated end-to-end latency is ∼226 ms via stdio and ∼321 ms via HTTP—both acceptable for interactive use.

The production deployment exclusively uses HTTP Streamable transport because a school wiki requires concurrent multi-user access with authentication. Figure 5.2 illustrates the architectural differences between both transport modes.

5.4 OAuth 2.1 Authentication

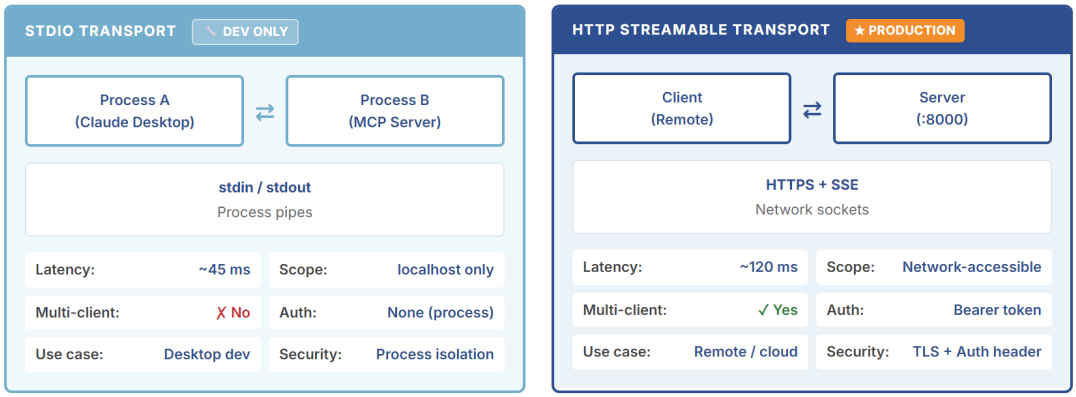
Extending the authentication architecture outlined in Section 4.2.4, the implementation uses the Scalekit Python SDK ≥2.4.0 with Proof Key for Code Exchange (PKCE). The

[2]The production embedding run (2026-02-26) produced 6,002 vectors; the collection had grown to 6,082 points by the benchmark date (2026-03-21) due to an incremental re-ingestion run that added 80 newly published wiki pages. The 10,841 chunks reported in the evaluation corpus (Appendix .6) were produced by a separate pass with 512-token chunking and no minimum length filter.

FIGURE 5.2

MCP Transport Mechanisms

stdio (local development) · HTTP Streamable (production) · latency · auth · network scope



Aspect	stdio	HTTP Streamable	Used in Project
Latency (avg)	~45 ms	~120 ms	—
Concurrent Clients	Single only	Multiple (unlimited)	—
Network Range	localhost	Internet-accessible	—
Authentication	Process-based	JWT Bearer token	—
Encryption	None required	TLS (HTTPS)	—
Development Use	✓ Recommended	—	Local testing only
Production Use	—	✓ Recommended	Raspberry Pi 5 deployment

PROJECT IMPLEMENTATION STRATEGY

Local development (laptop): Uses stdio for fast iteration and simple debugging.
Production (Raspberry Pi 5): HTTP Streamable with Bearer token authentication allows multiple AI clients (Claude, ChatGPT, Mistral) to query simultaneously. Caddy handles TLS termination and reverse-proxies to the mcp-server container.

Figure 5.2: MCP transport comparison. stdio is used during development for low-latency local access; HTTP Streamable is used in production for network accessibility and multi-client Bearer token authentication.

`ScalekitAuthMiddleware` intercepts all incoming requests and validates Bearer tokens before they reach the MCP protocol layer.

Token validation. The `ScalekitAuthMiddleware` (`src/middleware/scalekit_auth.py`) intercepts every HTTP request and validates Bearer tokens using the official Scalekit Python SDK. Listing 5.6 shows the core validation logic.

```python
class ScalekitAuthMiddleware:
    def __init__(self, config: ServerConfig):
        self.config = config  # retained for __call__
        self.scalekit_client = ScalekitClient(
            env_url=config.scalekit_env_url,
            client_id=config.scalekit_client_id,
            client_secret=config.scalekit_client_secret
        )
        self.www_authenticate_header = {
            "WWW-Authenticate": (
                f'Bearer realm="OAuth", '
                f'resource_metadata="'
                f'{config.scalekit_expected_audience}'
                f'/.well-known/oauth-protected-resource"'
            )
        }

    async def __call__(self, request, call_next):
        if self._is_public_endpoint(request.url.path):
            return await call_next(request)
        token = request.headers.get("authorization", "")
        if not token.startswith("Bearer "):
            return Response(status_code=401,
                headers=self.www_authenticate_header)
        validation_options = TokenValidationOptions(
            issuer=self.config.scalekit_env_url,
            audience=[self.config.scalekit_expected_audience]
        )
        is_valid = self.scalekit_client \
            .validate_access_token(
                token[7:], options=validation_options)
        if not is_valid:
            return Response(status_code=401,
                headers=self.www_authenticate_header)
        return await call_next(request)
```

Listing 5.6: Scalekit OAuth 2.1 token validation (`scalekit_auth.py`).

Public endpoints (`/health`, `/.well-known/*`, `/auth/*`, `/favicon.ico`) bypass authentication entirely via the `_is_public_endpoint` check, enabling OAuth discovery and health monitoring without credentials.

OAuth flow endpoints. The server exposes four OAuth endpoints via a dedicated `APIRouter` (`src/auth/oauth_flow.py`):

- `/auth/login`: Initiates the Authorization Code flow with PKCE (`secrets.token_urlsafe(32)`) and CSRF state generation, then redirects to Scalekit's `/oauth/authorize` endpoint.
- `/callback`: Handles the redirect from Scalekit, validates the CSRF state, and exchanges the authorization code for access and refresh tokens via `httpx` $\geq$0.25.0 POST to `/oauth/token`.
- `/auth/logout`: Redirects to Scalekit's `/logout` endpoint to end the SSO session, optionally passing the `id_token_hint`.
- `/auth/user`: Returns the current authenticated user's information (user ID, email, name, role, organization) extracted from the validated JWT.

OAuth discovery. The server provides two discovery endpoints following the OAuth Protected Resource pattern:

- `/.well-known/oauth-protected-resource`: Returns metadata including the authorization server URL, supported scopes, and the Bearer token method. This endpoint is essential for Claude Desktop and other OAuth-aware MCP clients.
- `/.well-known/oauth-authorization-server`: Redirects (HTTP 302) to Scalekit's OIDC configuration at `<env_url>/.well-known/openid-configuration`, bridging the gap between OAuth 2.1 and OIDC discovery standards.

A `/register` endpoint returns HTTP 400 with a clear error message, informing clients that dynamic client registration is not supported and that clients must be pre-registered in the Scalekit dashboard.

5.5 FastMCP 3.0 Middleware Chain

Beyond the HTTP-level Scalekit middleware, the server implements four custom FastMCP middleware components that operate at the MCP protocol level. Table 5.5 describes the five-layer chain through which every request passes.

Figure 5.3 visualizes the request flow through all five layers.

These are registered in order in `main.py`:

Figure 5.3: FastMCP middleware pipeline. Each request passes through authentication, RBAC permission checking, and audit logging before reaching the tool handler and Qdrant vector database. Error paths (stages 1–4) short-circuit with standard HTTP error codes.

Table 5.5: Five-layer middleware pipeline.

Layer	Middleware	Level	Responsibility
1	`ScalekitAuthMW`	HTTP	JWT Bearer token validation
2	`RequestLoggingMW`	FastMCP	UUID correlation ID, timing
3	`UserContextMW`	FastMCP	JWT decode, role extraction
4	`RBACEnforcementMW`	FastMCP	Tool filtering, access control
5	`AuditLoggingMW`	FastMCP	Pseudonymized audit logging

```
mcp.add_middleware(RequestLoggingMiddleware())
mcp.add_middleware(UserContextMiddleware())
mcp.add_middleware(RBACEnforcementMiddleware())
mcp.add_middleware(AuditLoggingMiddleware())
```

Listing 5.7: Middleware registration (`main.py`).

5.5.1 RequestLoggingMiddleware

Generates a UUID-based correlation ID for each request, records the start time, and logs the request method and duration on completion. The correlation ID is stored in the FastMCP context state via `await ctx.set_state("request_id", ...)` and is available to all downstream middleware and tools.

5.5.2 UserContextMiddleware

Extracts user identity and role from the JWT Bearer token. Since the token's cryptographic signature has already been validated by the `ScalekitAuthMiddleware`, this middleware performs a decode-only operation to extract the claims.

The role extraction uses a multi-fallback strategy to accommodate different identity provider configurations. Scalekit may place role information in different JWT claims depending on the organization's configuration:

```
user_role = (
    claims.get("role") or
    claims.get("roles") or
    claims.get("user_role") or
    claims.get("custom:role") or
    claims.get("https://leowiki.htl-leonding.ac.at/role")
    or "student"
)
if isinstance(user_role, list) and user_role:
    user_role = user_role[0]

# Check groups claim (common IdP pattern)
groups = claims.get("groups", [])
if isinstance(groups, list):
    if "admin" in groups or "admins" in groups:
        user_role = "admin"
    elif "teacher" in groups or "lehrer" in groups:
        user_role = "teacher"

# Check scoped roles (e.g., "role:admin")
for scope in claims.get("scope", "").split():
    if scope.startswith("role:"):
        user_role = scope.split(":")[1]
        break
```

Listing 5.8: Multi-fallback role extraction (`mcp_middleware.py`).

This defensive approach resolves roles from multiple claim shapes; it has been exercised against Scalekit with the HTL Leonding IdP configuration (`role` and `groups` claims), and the fallback chain is structured to accommodate additional OIDC providers if needed.

5.5.3 RBACEnforcementMiddleware

This middleware enforces tool-level access control following the RBAC model defined by NIST [26] by maintaining a `TOOL_PERMISSIONS` dictionary that maps each tool name to the set of roles permitted to invoke it:

```
TOOL_PERMISSIONS = {
    "search_content_student": {"student", "teacher", "admin"},
    "search_content_teacher": {"teacher", "admin"},
    "get_collection_stats":   {"admin"},
    "get_query_statistics":   {"admin"},
    "list_resources":         {"admin"},
    "read_resource":          {"admin"},
}
```

Listing 5.9: RBAC tool permissions (`mcp_middleware.py`).

Note: `list_resources` and `read_resource` appear here because FastMCP exposes resources through the same permission path when the `ResourcesAsTools` transform is enabled; they are not separate hand-written tools alongside the five named MCP tools. The `health_check` tool is intentionally omitted from this dictionary: tools not listed in `TOOL_PERMISSIONS` bypass RBAC enforcement and are available to all authenticated users, consistent with Table 4.2 ("All" access).

The middleware implements two hooks:

- **`on_list_tools`:** Filters the tool list to show only tools the user's role is permitted to access.
- **`on_call_tool`:** Checks that the user's role is in the permitted set before allowing execution. Unauthorized invocations raise a `ToolError`.

5.5.4 AuditLoggingMiddleware

Logs every tool invocation with a pseudonymized user identifier (SHA-256 hash of `user_id`), the tool name, and the timestamp. This audit trail is separate from the query logging system (Section 5.8) and serves compliance and debugging purposes.

5.5.5 Middleware vs. Controller Pattern

The choice of middleware over the traditional controller pattern—documented in `docs/AUTHENTICATION_ARCHITECTURE.md`—addresses cross-cutting concerns. In the controller pattern, each tool handler would individually check authentication and authorization. The middleware pattern centralizes these checks:

- **Consistency:** Every request passes through the same chain.
- **Separation of concerns:** Tool handlers focus on business logic.

- **Layered checking:** The middleware chain implements multiple independent checkpoints, so a failure in one layer does not bypass the others.
- **Testability:** Middleware can be tested in isolation.

5.6 Two-Tool RBAC Search Architecture

As described in Section 4.2.4, the search functionality is separated into two role-specific tools (`search_content_student` and `search_content_teacher`) to implement RBAC at the architectural level. This section details the implementation. Figure 5.4 provides an overview.

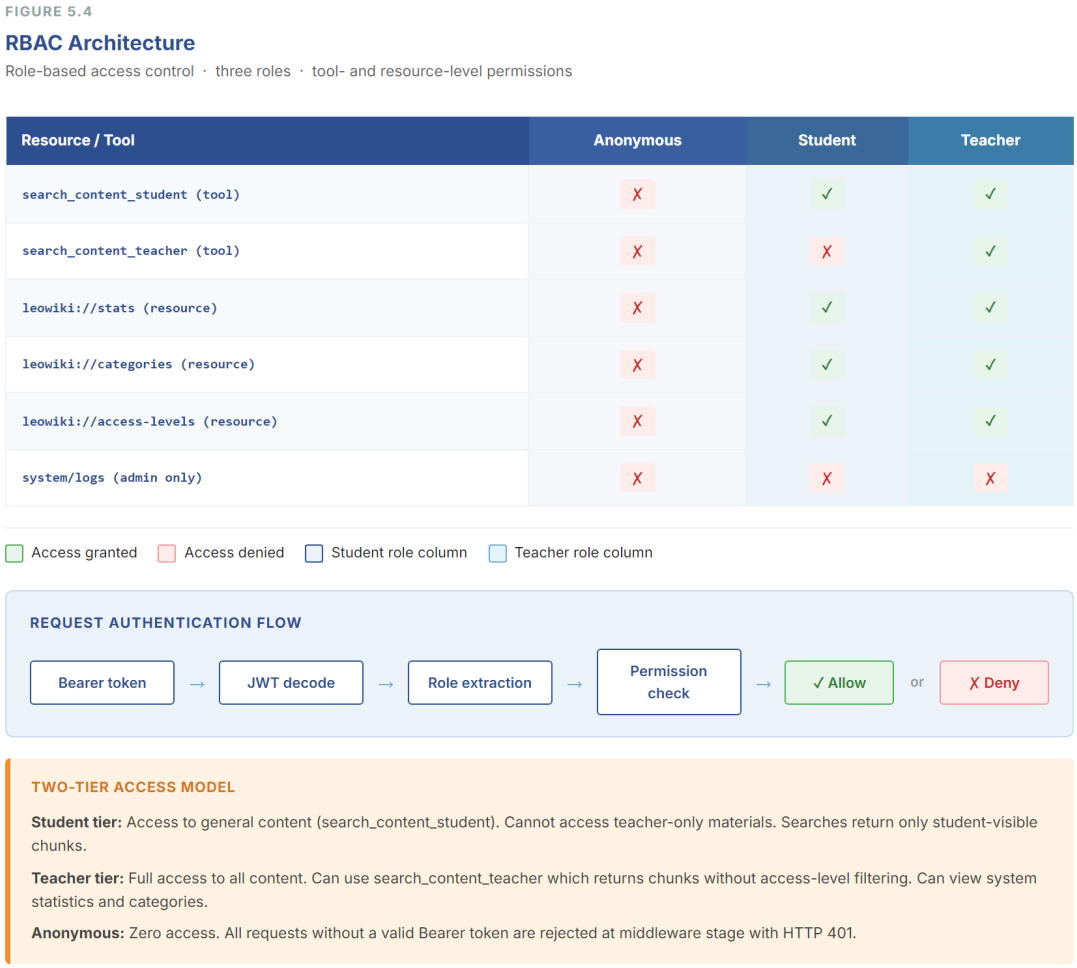


Figure 5.4: RBAC architecture for the LeoWiki MCP server. Two tool-level access tiers prevent students from accessing teacher-restricted wiki content. All requests require a valid JWT Bearer token; the role claim determines which tools and resources are accessible.

5.6.1 Student Search Tool

The student tool provides access to student-accessible content by applying Python post-filtering with dynamic oversampling. When RBAC is enabled, the tool fetches 3× the requested number of results from Qdrant, filters out entries with `teacher:` namespace prefixes, and returns the filtered results. If insufficient results remain, it doubles the fetch limit up to a maximum of 10× the original request.

```
@mcp.tool(
    name="search_content_student",
    description="Search educational content "
                "with student-level access",
    annotations=ToolAnnotations(
        title="LeoWiki Student Search",
        readOnlyHint=True,
        idempotentHint=True,
        openWorldHint=False,
    )
)
async def search_content_student(
    query: str, limit: int = 10, ctx: Context = None
) -> dict:
    results = []
    fetch_limit = limit * 3
    max_fetch = limit * 10
    while len(results) < limit and fetch_limit <= max_fetch:
        raw = db.search(query_vector=embedding,
                        collection=collection,
                        limit=fetch_limit)
        results = [r for r in raw
                   if "teacher:" not in
                   r.payload.get("source", "")]
        if len(results) < limit:
            fetch_limit *= 2
    results = results[:limit]
```

Listing 5.10: Student search with dynamic oversampling (`search_tools.py`).

The tool also integrates progress reporting (4 steps: embedding, filtering, search, formatting) and UX-optimized result formatting with query logging.

5.6.2 Teacher Search Tool

The teacher tool provides unrestricted access to all content without post-filtering. Its implementation delegates directly to Qdrant without namespace restrictions. Teachers can use *both* tools: the student tool to see what students see, and the teacher tool for full access.

5.6.3 Admin Tools

Two additional tools are restricted to the admin role:

- **`get_collection_stats`:** Returns Qdrant collection statistics including total document count, RBAC distribution across namespaces, optimizer status, and storage details.
- **`get_query_statistics`:** Analyzes the persistent query log (Section 5.8) and returns aggregate statistics: total queries, breakdown by user role (`student`, `teacher`, `admin`), and breakdown by tool name.

A global `health_check` tool is available to all roles and returns the server name, version, and status.

5.6.4 Design Rationale: Two Tools vs. Single Tool

An alternative design would use a single tool with a `role` parameter. The two-tool architecture was chosen because:

1. **Reduced attack surface:** The LLM cannot manipulate access level parameters—the tool boundary is the security boundary.
2. **Clearer LLM semantics:** Two tools with distinct descriptions produce better tool selection by the LLM.
3. **Middleware-level enforcement:** The RBAC middleware controls which tools appear in the `tools/list` response, providing an additional layer of defense.

5.7 MCP Resources and Prompts

5.7.1 Resources

The server exposes six MCP resources—four static and two dynamic—all restricted to admin access via the RBAC middleware. Table 5.6 lists them.

Table 5.6: MCP resources exposed by the LeoWiki MCP server.

Type	URI	Description
Static	`leowiki://categories`	Content categories with descriptions
Static	`leowiki://access-levels`	RBAC documentation
Static	`leowiki://search-hints`	Tips for effective search queries
Static	`leowiki://system-prompt`	Behavioral guidelines for LLMs
Dynamic	`leowiki://stats`	Live Qdrant collection statistics
Dynamic	`leowiki://recent/{count}`	Recently updated content

Resources are registered via decorator-based functions using FastMCP's `@mcp.resource` decorator. Listing 5.11 shows the registration pattern for the content categories resource.

```
@mcp.resource(
    uri="leowiki://categories",
    name="Content Categories",
    description="Available content categories in LeoWiki",
    mime_type="application/json",
    tags={"metadata", "navigation"}
)
def get_categories() -> str:
    categories = [
        {"id": "sew", "name_de": "Softwareentwicklung",
         "description": "OOP, Design Patterns, Web"},
        {"id": "nwt", "name_de": "Netzwerktechnik",
         "description": "Cisco, Protocols, Security"},
        # ...
    ]
    return json.dumps(categories, ensure_ascii=False)
```

Listing 5.11: Resource registration (`src/resources/metadata.py`).

The admin-only restriction (enforced by `RBACEnforcementMiddleware`, not by the resource itself) prevents prompt injection via resource content and hides architectural details from unprivileged users.

5.7.2 Prompts

Two educational prompt templates are registered:

- **`explain_topic`:** Generates a structured explanation of a topic at a configurable difficulty level (beginner, intermediate, advanced). The prompt instructs the LLM to provide a definition, core concepts, practical relevance, practical examples with common mistakes (if enabled), and a summary.
- **`summarize_search`:** Synthesizes search results into a coherent summary, including guidelines for cross-referencing multiple sources and identifying key findings.

5.7.3 FastMCP 3.0 Transforms

A practical challenge discovered during client testing was that not all AI clients natively support MCP resources and prompts. For example, Claude Desktop supports all three primitives, but ChatGPT only supports tools.

FastMCP 3.0 introduces *transforms* that solve this compatibility problem by wrapping resources and prompts as additional tools:

```
from fastmcp.server.transforms import (
    ResourcesAsTools, PromptsAsTools
)

mcp.add_transform(ResourcesAsTools(mcp))
mcp.add_transform(PromptsAsTools(mcp))
```

Listing 5.12: Transform registration (`main.py`).

With these transforms, a client that only supports `tools/list` and `tools/call` can still access resources (via a generated `read_resource_*` tool) and prompts (via a generated `use_prompt_*` tool).

5.8 Query Logging

To enable educational analytics while respecting GDPR requirements [39], the server implements a persistent query logging system. The `QueryLogger` class (`src/utils/query_logger.py`) is a thread-safe singleton that appends each search query as a JSON line to a `data/statistics/queries.jsonl` file.

Listing 5.13 shows the core logging method. Thread safety is ensured through a module-level `threading.Lock` so that concurrent search requests from multiple users do not corrupt the log file.

```
class QueryLogger:
    def log_query(self, query, response, user_role,
                  tool_name, result_count,
                  user_id_hash=None, request_id=None,
                  duration_ms=None):
        entry = {
            "timestamp": datetime.now().isoformat(),
            "query": query,
            "response_preview": response[:500],
            "response_length": len(response),
            "user_role": user_role,
            "tool_name": tool_name,
            "result_count": result_count,
            "user_id_hash": user_id_hash,
            "request_id": request_id,
            "duration_ms": duration_ms
        }
        with _write_lock:
            with open(self.log_file, "a") as f:
                f.write(json.dumps(entry) + "\n")
```

Listing 5.13: Thread-safe query logging (`query_logger.py`).

Each log entry contains:

- Timestamp (ISO 8601), query text, response preview (first 500 characters).
- User role (`student`, `teacher`, `admin`), tool name, result count.
- Pseudonymized user ID (SHA-256 hash of `user_id`)—the original identifier is never persisted.
- Request correlation ID (from `RequestLoggingMiddleware`) and response duration in milliseconds.

The `get_query_statistics` admin tool (Section 5.6.3) reads this log and computes aggregate statistics (total queries, breakdown by role, breakdown by tool), enabling educators to understand how the wiki is being searched without exposing individual user identities. The singleton pattern restricts the process to a single `QueryLogger` instance regardless of how many concurrent requests arrive:

```
_query_logger: Optional[QueryLogger] = None
_logger_lock = threading.Lock()

def get_query_logger() -> QueryLogger:
    global _query_logger
    if _query_logger is None:
        with _logger_lock:
            if _query_logger is None:  # Double-check
                _query_logger = QueryLogger()
    return _query_logger
```

Listing 5.14: Thread-safe singleton pattern (`query_logger.py`).

5.9 Error Handling and Security

The server employs multiple independent security layers.

Error masking. `mask_error_details=True` (FastMCP 3.0 constructor parameter) prevents internal error details—stack traces, database connection strings, file paths—from reaching clients. Detailed errors are logged server-side at the `ERROR` level for debugging while clients receive a generic "Internal server error" message. Additionally, each search tool wraps its logic in a `try/except` block that returns a user-friendly German error message (`"Es ist ein Fehler bei der Suche aufgetreten."`) rather than leaking implementation details.

Request size limit. A custom `RequestSizeLimitMiddleware` (implemented as a Starlette `BaseHTTPMiddleware`) rejects requests exceeding 1 MB with HTTP 413, protecting against Denial-of-Service attacks via oversized payloads.

Rate limiting. The `slowapi` library (v0.1.9) enforces configurable per-IP rate limits (default: 100 requests/minute, configurable via `MAX_REQUESTS_PER_MINUTE` environment variable). Exceeded limits return HTTP 429 with a `Retry-After` header. Rate limiting is enabled by default but can be disabled for local development.

CORS policy. CORS is currently *disabled* in production for maximum MCP client compatibility, since desktop AI clients (Claude, ChatGPT, Mistral) bypass browser CORS entirely. The codebase includes a pre-configured CORS allowlist (`claude.ai`, `leowiki-mcp.stream`, `leowiki.htl-leonding.ac.at`, and `localhost` origins) that

can be activated via an environment variable when browser-based access is needed in the future.

Input validation. `strict_input_validation=True` (FastMCP 3.0) validates tool arguments against their type annotations before execution. Search queries are additionally validated for non-empty content, and the result limit is bounded to the range $[1, 20]$.

5.10 Compatibility Analysis

Research question I5 asks whether MCP is suitable as a standardized interface for integrating knowledge sources into various AI applications. The server was tested with three AI clients via direct remote URL registration at `https://leowiki-mcp.stream/mcp`. The interoperability session was performed on 2026-02-20 with the then-current client builds:[3]

Manual test protocol (2026-02-20). Each client was registered against `https://leowiki-mcp.stream/mcp`. Success required completing OAuth 2.1 via Scalekit, listing the expected tools for the selected test account, and executing at least one `search_content_*` call with the German query *"Was ist das OSI-Modell?"* without transport errors. Resource and prompt availability was checked against Table 5.7.

5.10.1 OAuth 2.1 Authentication Flow

The authentication flow is identical across all tested clients. After registering the server URL, the client initiates an OAuth 2.1 redirect to the Scalekit identity provider. The user enters their email address, receives a one-time code via email, and enters it to complete authentication. Figures 5.5 and 5.6 show the redirect and code entry steps.

[3]Claude Web App (Pro plan) and Claude Desktop for macOS, build February 2026; ChatGPT desktop for macOS/Windows (Plus plan), build February 2026; Mistral Le Chat web (free tier, beta connectors), accessed 2026-02-20. The MCP connector configuration in these clients follows the specification revision dated 2025-11-25.

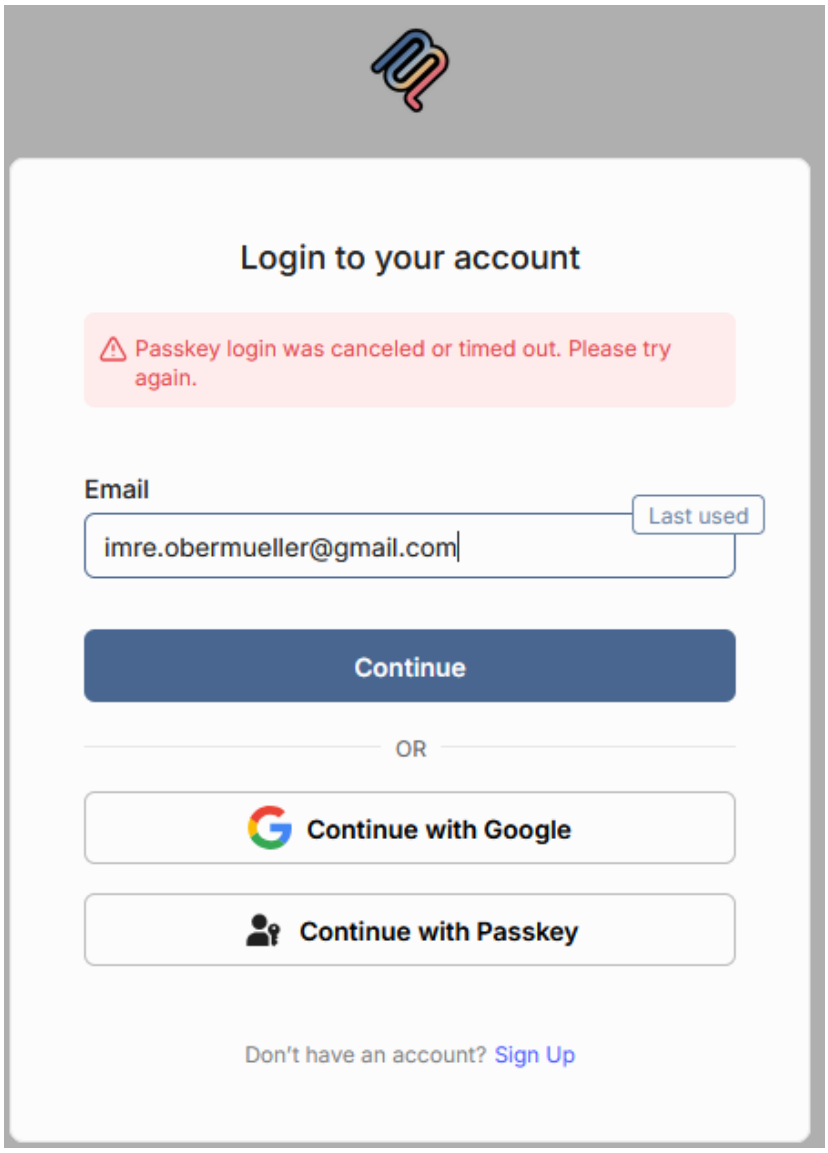


Figure 5.5: OAuth 2.1 redirect to Scalekit login page (identical across all clients).

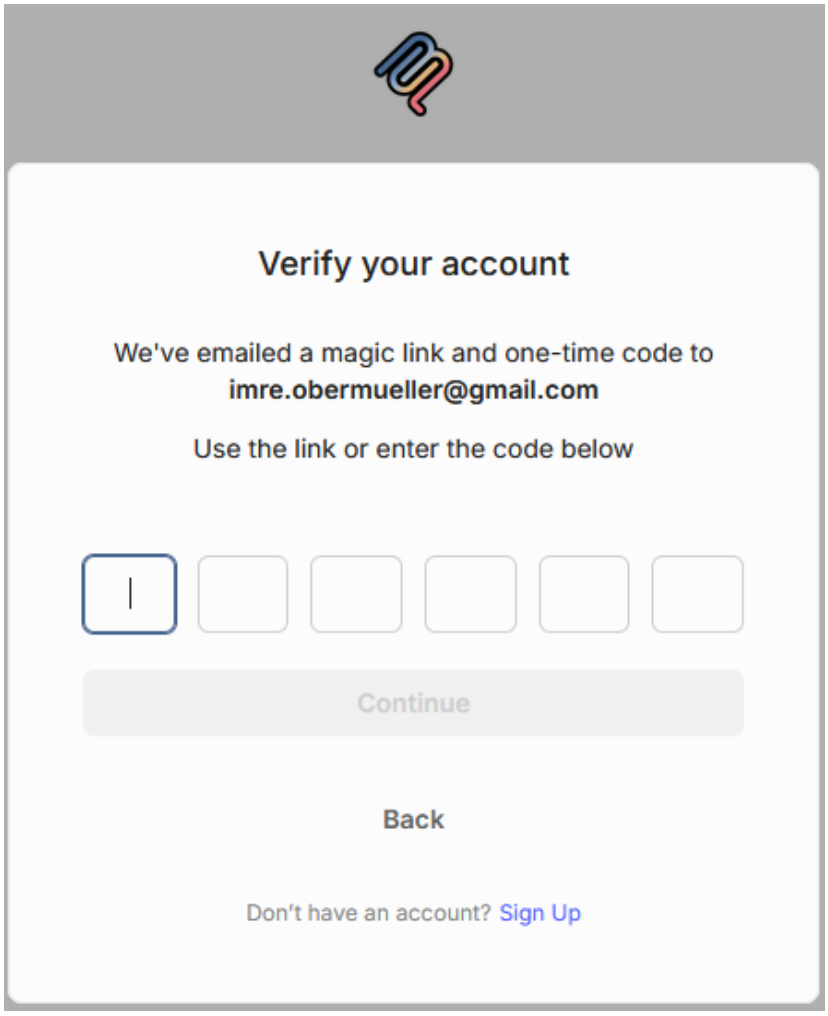


Figure 5.6: One-time code entry during OAuth authentication.

5.10.2 Claude (Desktop, Web, Mobile)

Anthropic, the creator of the MCP specification, supports MCP connectors on all three platforms (Web, Desktop, Mobile). Connectors require a Pro Plan subscription. Registration is performed in the Claude Web App via `[+]` → `Add connectors` → `Manage connectors` → `Add custom connectors` (Figure 5.7).

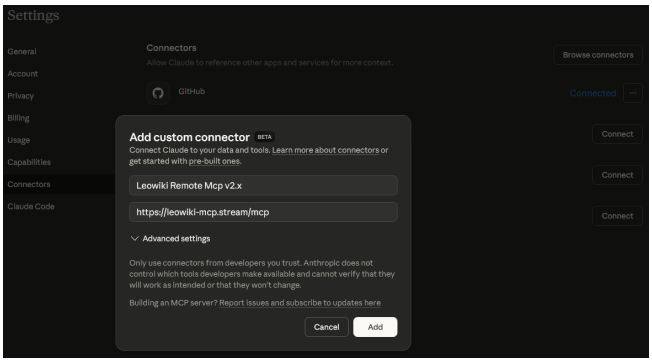


Figure 5.7: Registering the LeoWiki MCP server as a custom connector in Claude Web App.

Once the connector is registered in the Claude Web App, **it works without further configuration in Claude Desktop and Claude Mobile (Android)** as well. Figure 5.8 shows the connector configuration with all role-based tools visible. Figure 5.9 demonstrates a live search query execution in the Claude Web App.

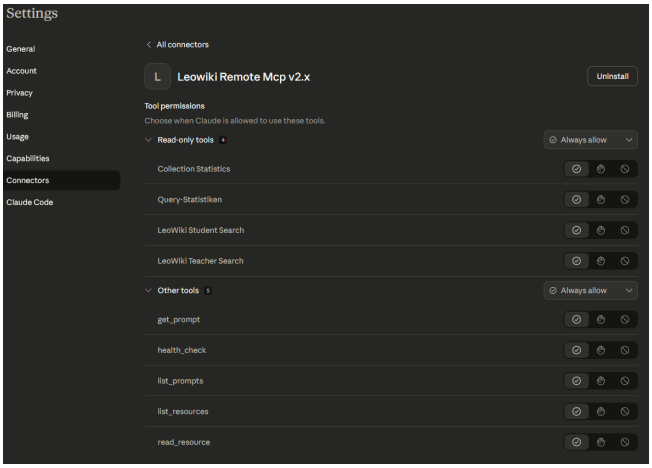


Figure 5.8: Connector configuration in Claude showing tools, resources, and role-based permissions.

5.10.3 ChatGPT (Desktop, Web, Mobile)

OpenAI supports MCP servers as "Apps" (previously called "Connectors"). At least a Plus Plan is required. Registration follows `Settings` → `Apps` → `Create app`. When

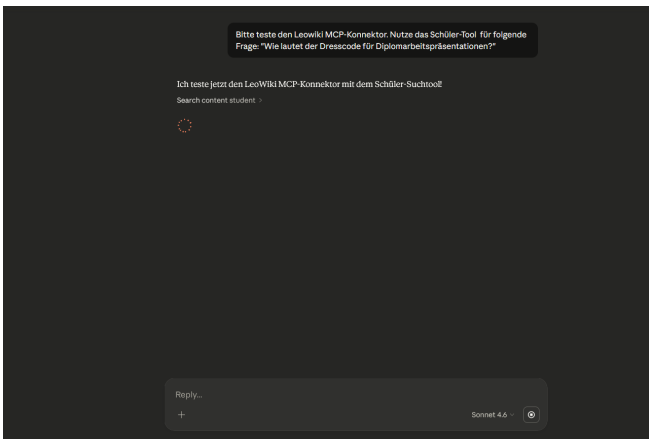


Figure 5.9: Claude Web App executing a search query against the LeoWiki MCP server.

registering via the Desktop App, the user is redirected to the Web App for the OAuth flow and then returned to the Desktop client (Figure 5.10).

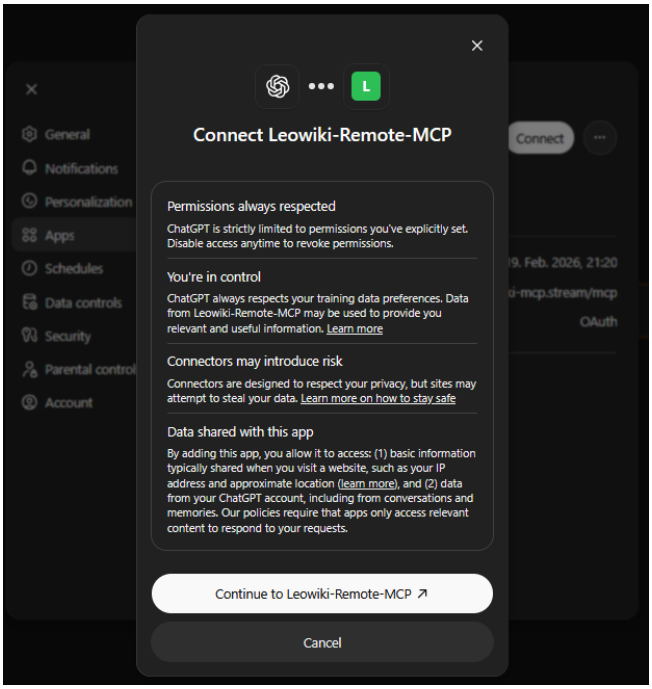


Figure 5.10: ChatGPT Desktop after successful OAuth authentication and tool execution.

5.10.4 Mistral Le Chat (Web, Mobile)

Mistral supports remote MCP servers as "Connectors" in their Le Chat interface. The connector feature is still in Beta and available in the free tier, making it the only tested client that does not require a paid subscription. Mistral operates under European data protection jurisdiction. No Desktop application is currently available. Configuration lists available tools, prompts, resources, and allowances under the name "Functions"

(Figure 5.11). Figure 5.12 shows a successful tool execution in the Mistral Le Chat interface.

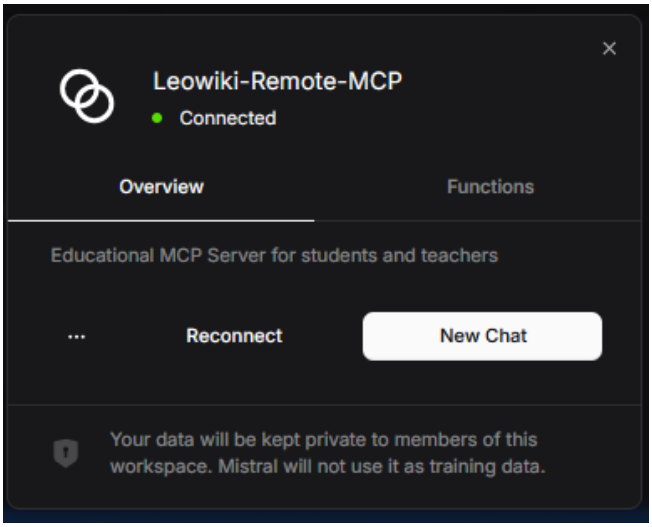


Figure 5.11: Mistral Le Chat showing connector with listed tools ("Functions").

Client comparison (Claude vs. ChatGPT vs. Mistral). All three clients executed the five tools in the manual protocol above, including tool chaining where applicable. Claude exposes resources and prompts natively and also works with `ResourcesAsTools`; ChatGPT and Mistral rely on `ResourcesAsTools` and `PromptsAsTools` for non-tool primitives. Claude and ChatGPT sync one connector registration across Desktop, Web, and Mobile; Mistral remains Web/Mobile-only (Beta) and occasionally drops sessions until re-authentication.

5.10.5 Non-Compatible Clients

As of February 2026, two major AI platforms do *not* support MCP remote connectors for end users:

- **Google Gemini:** The consumer web and mobile apps do not support MCP connectors. The Gemini CLI supports local MCP servers via `settings.json`, but remote URL-based servers are not yet implemented [40]. Enterprise MCP is available via Google Cloud.
- **Microsoft Copilot:** The consumer chat interface does not support MCP. MCP integration is available exclusively in Copilot Studio [41], an enterprise development platform.

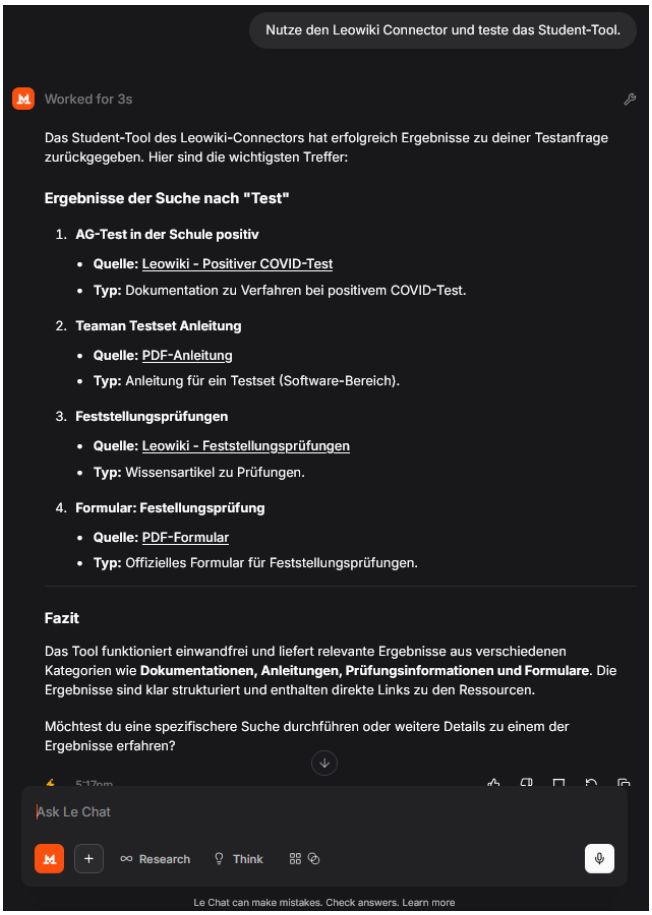


Figure 5.12: Mistral Le Chat executing a search tool call against the LeoWiki MCP server.

VS Code supports MCP servers through extensions (Cline, Continue) as well as through GitHub Copilot Agent mode, configured via a local `mcp.json` file. GitHub Copilot is particularly relevant for the HTL target audience, as it is available at no cost to students through GitHub Education [42]. However, VS Code extensions configure MCP servers via local files rather than the remote OAuth 2.1 flow used by the LeoWiki server, and were therefore not tested in the scope of this project. LM Studio follows the same local-only pattern. Cursor IDE has documented issues with remote MCP authentication, including a startup race condition where tools register before the OAuth flow completes [43, 44].

5.10.6 Market Context

According to Similarweb data (January 2026), ChatGPT holds approximately 64% of global AI chatbot web traffic, down from 87% one year prior [45]. Google Gemini captured 21.5%, with the remainder shared among Grok, Claude, Perplexity, DeepSeek, and Copilot. At the time of testing (February 2026), the three compatible clients represented a substantial share of consumer AI chatbot traffic [45].

Compatibility principle. In general, any AI client that implements the Anthropic MCP specification [3] fully—including HTTP Streamable transport and OAuth 2.1—is compatible with the LeoWiki MCP server without modifications. These findings reflect the state as of February 2026 and are subject to change as vendors update their implementations.

5.10.7 Compatibility Matrix

Figure 5.13 provides a visual overview of feature support across all tested clients. Table 5.7 consolidates the findings in detail.

Table 5.7 consolidates the findings. Each cell indicates full support (✓), partial (∼), or no support (×).

5.10.8 Key Findings

1. All three compatible clients returned successful tool calls in the manual tests via the remote URL, confirming the core MCP tool-calling mechanism [3].

FIGURE 5.13

MCP Client Compatibility Matrix

Five AI clients · 10 feature categories · February 2026 · Claude · ChatGPT · Mistral · Gemini · Copilot

LEGEND: ✓ Full support ~ Partial support ✗ Not supported — Not tested / N/A

	Claude	ChatGPT	Mistral	Gemini	Copilot
CONNECTIVITY & PROTOCOL					
Remote URL connect	✓	✓	✓	✗	✗
OAuth 2.1 flow	✓	✓	✓	—	—
MCP FEATURE SUPPORT					
Tool invocation	✓	✓	✓	—	—
Resource access	✓	~	~	—	—
Prompt templates	✓	✗	✗	—	—
Transforms fallback	✓	✓	✓	—	—
PLATFORM AVAILABILITY					
Desktop app	✓	✓	✗	—	—
Mobile app	✓	✓	✓	—	—
Cross-platform sync	✓	✓	✗	—	—
ACCESS MODEL					
Paid plan required	Yes	Yes	No (Beta)	—	—

KEY FINDINGS (FEBRUARY 2026)

Claude, ChatGPT, Mistral fully support remote MCP connections with OAuth 2.1 — covering the three leading consumer AI platforms.

Gemini & Copilot do not expose MCP remote connectors to end users; Gemini CLI supports local stdio only; Copilot MCP is restricted to enterprise Copilot Studio.

Partial support (Resource access via Mistral/ChatGPT): native resource primitives unsupported, but the *ResourcesAsTools* transform provides equivalent access as regular tools.

Prompt templates are only natively supported by Claude; *PromptsAsTools* transform bridges this gap for ChatGPT and Mistral.

Figure 5.13: MCP client compatibility matrix (February 2026). Three of five tested AI clients (Claude, ChatGPT, Mistral) fully support remote HTTP Streamable connections with OAuth 2.1 authentication. Google Gemini and Microsoft Copilot (consumer chat) lack consumer-facing remote MCP connector support. Partial entries for resource access and prompt templates are bridged by FastMCP 3.0 transforms.

Table 5.7: MCP client compatibility matrix (February 2026).

Feature	Claude	ChatGPT	Mistral	Gemini	MS Copilot
Remote URL connect	✓	✓	✓	×	×
OAuth 2.1 flow	✓	✓	✓	—	—
Tool invocation	✓	✓	✓	—	—
Resource access	✓	∼	∼	—	—
Prompt templates	✓	×	×	—	—
Transforms fallback	✓	✓	✓	—	—
Desktop app	✓	✓	×	—	—
Mobile app	✓	✓	✓	—	—
Cross-platform sync	✓	✓	×	—	—
Paid plan required	Yes	Yes	No (Beta)	—	—

2. OAuth 2.1 via Scalekit authenticated users across all three clients using the same flow: email redirect → one-time code → connected.
3. The `ResourcesAsTools` and `PromptsAsTools` transforms provide access to all server functionality from clients that only support tools.
4. Clients connect by entering only the server URL; no local software installation is required.
5. Claude and ChatGPT synchronize the connector registration across Desktop, Web, and Mobile platforms.
6. The RBAC filtering operates at the server level; clients receive role-appropriate results without requiring client-side configuration (see Appendix .5.3).

These observations address research question FF2 (operationalized as I5 in this chapter): MCP proved workable as a standardized integration interface for the tested clients. The limitations identified – fragmented advanced feature support, rapid specification evolution – are noted in Section 7.2.

5.11 MCP Server Testing

5.11.1 Unit Tests

The test suite comprises 22 files under `tests/` with approximately 100 top-level `test_*` functions; pytest collects additional cases from parametrized tests (e.g., RBAC matrices). Deterministic tests (`test_tools_deterministic.py`, `test_rbac_deterministic.py`) validate tool registration and schema correctness, middleware chain ordering, RBAC

permission matrix (all role-tool combinations), and configuration loading via Pydantic Settings. These tests run without external dependencies (no Qdrant, no OpenAI API key) and form the CI gate.

5.11.2 Integration Tests

End-to-end tests (`test_server_endpoints.py`, `test_mcp_http_streamable.py`, `test_mcp_endpoint_direct.py`) verify the full request lifecycle: HTTP endpoint responses, MCP protocol negotiation over HTTP Streamable, OAuth 2.1 flow endpoints, tool invocation with RBAC filtering, resource and prompt access, and query logging persistence and format validation.

5.11.3 Continuous Integration Pipeline

A GitHub Actions workflow (`.github/workflows/ci.yml`) runs on every push to `main` and `feature/*` branches:

1. **Lint** with `ruff` $\geq$0.1.0 (Python 3.13, ignoring line length and unused imports).
2. **Type check** with `mypy` $\geq$1.5.0 (`-ignore-missing-imports`).
3. **Deterministic tests** via `pytest` (gate: must pass).
4. **Remaining tests** via `pytest` (non-blocking, requires live services).
5. **Docker build** verification (builds the production image on `ubuntu-latest`).

5.11.4 Transport Verification

The transport comparison (Section 5.3.4) serves as a practical verification of both transport implementations. The HTTP Streamable transport was validated through production deployment on the Raspberry Pi with real client connections. The `stdio` transport was validated through local development with Claude Desktop. Dedicated test files (`test_mcp_sse_correct.py`, `test_http_mode.py`, `test_stdio_with_new_data.py`) provide automated verification.

5.11.5 Security Verification

Security-related tests (`test_jwt_validation.py`, `test_docker_readiness.py`) verify: error masking (internal details not exposed to clients), JWT token validation edge cases (expired, malformed, wrong audience), rate limiting (HTTP 429 returned when per-IP

limit exceeded), request size limits (HTTP 413 for oversized payloads), and RBAC bypass attempts (unauthorized tool invocations correctly rejected with `ToolError`).

6 Embedding, Retrieval, Authentication and Deployment

6.1 Embedding Model Evaluation

Note on sub-question labels: The labels J1–J8 used in this chapter follow the final research question specification (finalized 2026-02-11) and differ from the preliminary numbering in the ABA document (Appendix .1).

6.1.1 Test Corpus Construction

The choice of embedding model determines retrieval quality: it governs how accurately content is represented as vectors and therefore bounds achievable ranking performance. Research question FF3 formulates this requirement explicitly: *Which embedding model achieves the best retrieval quality for German-language wiki content?*

The following requirements were placed on the embedding model:

1. **German-language retrieval quality:** The model must accurately map German texts – including technical terminology, compound words, and school-specific vocabulary – into vector representations.
2. **Local executability:** For data privacy reasons (school-internal content), a locally runnable model is preferred (cf. Section 3.4).
3. **Hardware compatibility:** The model must be executable on an NVIDIA GeForce RTX 4080 Laptop GPU (12 GB VRAM).
4. **Reproducibility:** All evaluation results must be reproducible with documented configurations (NFR-3).

The Massive Text Embedding Benchmark (MTEB) was used for pre-selection of suitable models [12]. MTEB provides standardized benchmarks for embedding models and has included German-language retrieval tasks (RTEB-deu) since 2024. The leaderboard was retrieved on February 16, 2026 and served as the primary selection heuristic.

Four local models from different providers were selected from the upper ranks of the MTEB RTEB(deu) leaderboard. Additionally, OpenAI's `text-embedding-3-large` was included as a commercial reference model to establish a performance ceiling. The model `BAAI/bge-m3-unsupervised` (MTEB rank #12), used as default in RAGFlow, was integrated as a baseline to determine whether the RAGFlow default configuration can be surpassed.

Table 6.1: Overview of evaluated embedding models with MTEB rank (RTEB-deu), parameter count, and vector dimensions.

Model	Provider	Rank	Score	Params	Dim.
Octen-Embedding-4B [46]	Octen	#2	73.25	4.0B	2560
PIXIE-Rune-v1.0 [47]	TelePIX	#3	69.81	0.568B	1024
snowflake-arctic-embed-l-v2.0 [48]	Snowflake	#7	65.48	0.568B	1024
bge-m3-unsupervised [11]	BAAI	#12	60.24	0.568B	1024
text-embedding-3-large [10]	OpenAI	–	–	API	3072

The benchmark used the fully preprocessed LeoWiki corpus. This corpus comprises 436 Markdown files derived from the 196 wiki pages and associated media documents available as of February 2026. The production pipeline processes 516 input documents in total (196 pages and 320 media references; cf. Table 6.9); of these, 436 yield text content suitable for embedding after preprocessing. The 436 documents were segmented into 10,841 text chunks (paragraph-based, maximum chunk size 512 tokens, no minimum length filter). The content is predominantly German with occasional English technical terms.

The ground truth consisted of 78 question-answer pairs generated with Claude Opus 4.6 [49][4] and manually verified by both authors against the wiki content. Both authors independently reviewed all 78 question–answer pairs. Initial disagreements arose in 7 cases (9%), primarily concerning difficulty classification and edge-case answer derivability. These were resolved through discussion and re-examination of the source material. Each pair was checked for: (1) source page existence in the corpus, (2) answer derivability from the referenced page, and (3) question clarity and unambiguity. Of the initially generated 95 candidate pairs, 12 were discarded for factual errors or ambiguous source attribution, and 5 additional pairs were removed because they could not be

[4]"Claude Opus 4.6" is the model identifier displayed by the Cursor IDE (version 0.46.x, accessed 2026-02-17). Cursor relays requests to Anthropic's API; the closest public API identifier at that date was `claude-opus-4-5-20250620`. Throughout this thesis, the IDE label "Claude Opus 4.6" is used because it is the identifier recorded in the ground-truth JSON file (`leowiki_qa_78_Opus_46.json`), ensuring reproducibility.

sufficiently clarified despite modification attempts, yielding the final 78-pair dataset ($78 = 95 - 12 - 5$). The difficulty distribution comprises 17 easy, 40 medium, and 21 hard questions. Easy questions target answers contained in a single, clearly identifiable paragraph. Medium questions require combining information from one wiki page or recognizing domain-specific terminology. Hard questions require cross-namespace reasoning or involve ambiguous terms with multiple possible interpretations. Both authors independently assigned difficulty levels; disagreements were resolved by discussion. The questions span 12 of the wiki's 23 namespaces (the remaining 11 contain administrative configuration, empty placeholders, or restricted content not accessible to students) and various document types (wiki pages, PDF, DOCX, XLSX, PPTX). All questions were formulated in German to match the target language of the wiki system.

6.1.2 Evaluation Metrics

The primary evaluation metrics are MRR (Mean Reciprocal Rank) and NDCG@10 (Normalized Discounted Cumulative Gain at rank 10), as these most precisely capture ranking quality – i.e., the placement of the relevant document in the result list [50, 2]. Additionally, Precision@5 (P@5), Recall@10, and Hit Rate (proportion of queries with at least one hit in the top 10) were measured.

- **MRR (Mean Reciprocal Rank):** The reciprocal of the rank of the first relevant hit, averaged over all queries. An MRR of 1.0 means the relevant document is always at position 1.
- **NDCG@10:** Evaluates the entire ranking of the top-10 results, applying logarithmic discounting for lower positions.
- **Precision@5 (P@5):** Proportion of relevant documents among the five highest-ranked results.
- **Recall@10:** Proportion of all retrievable relevant documents found in the top 10.
- **Hit Rate:** Binary metric – proportion of queries for which at least one relevant document appears in the top 10.

Model comparison used the script `eval_model_comparison.py`. A separate Qdrant collection was created per model (different vector dimensions require separate collections). Models were loaded sequentially and explicitly unloaded from GPU memory after embedding generation to avoid VRAM conflicts. Retrieval was performed via Dense

Cosine Similarity with `top_k=10`. The hardware consisted of an NVIDIA RTX 4080 Laptop GPU (12 GB VRAM), an AMD Ryzen 9 processor, and 32 GB RAM.

6.1.3 Model Comparison Results

Table 6.2 shows the aggregated retrieval metrics of all five models (MRR, NDCG@10, P@5, Recall@10, Hit Rate). The same full metric set is tabulated in Appendix .6 (Table 11) for reference.

Table 6.2: Aggregated retrieval metrics of the five evaluated embedding models on the LeoWiki corpus (78 ground-truth questions, 10,841 chunks, Dense Cosine Similarity, top_k=10).

Model	MRR	NDCG@10	P@5	Recall@10	Hit Rate
Octen-Embedding-4B	**0.883**	**0.899**	0.185	0.949	94.9%
text-embedding-3-large	0.872	0.897	0.192	0.974	97.4%
PIXIE-Rune-v1.0	0.802	0.823	0.177	0.885	88.5%
snowflake-arctic-embed-l-v2.0	0.788	0.822	0.182	0.923	92.3%
bge-m3-unsupervised	0.756	0.800	0.182	0.936	93.6%

Figure 6.1 visualizes the MRR and NDCG@10 scores, highlighting the two-tier structure.

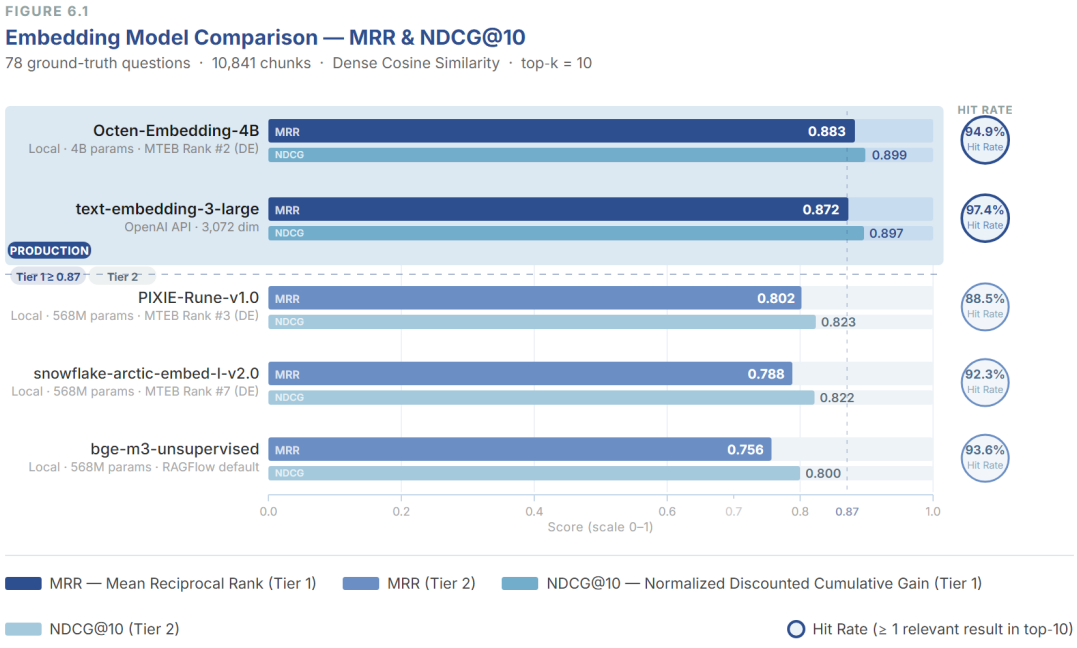


Figure 6.1: Aggregated retrieval metrics (MRR, NDCG@10, Hit Rate) of five embedding models on the LeoWiki corpus (78 queries). The dashed line at 0.87 marks the Tier 1 threshold; Tier 1 models are shaded in blue.

Two-tier structure. The results show a distinct two-tier structure in model performance:

Tier 1 (MRR > 0.87): Octen-Embedding-4B (MRR 0.883; NDCG@10 0.899) and text-embedding-3-large (MRR 0.872; NDCG@10 0.897) form the top group. The gap between these two models is small (1.1 percentage points MRR).

Tier 2 (MRR 0.75–0.80): PIXIE-Rune-v1.0, snowflake-arctic-embed-l-v2.0, and bge-m3-unsupervised cluster in the 0.756 to 0.802 MRR range. The gap to the top group amounts to 8 to 13 percentage points and is thus substantial. These three models each have approximately 568 million parameters; their performance differences among each other fall within the range of statistical noise for a sample of 78 questions.

Results by difficulty level. Breaking down results by difficulty shows where models differ most.

Easy questions (n=17): Octen-Embedding-4B is the only local model achieving a 100% hit rate (MRR 0.840). OpenAI scores slightly higher with MRR 0.873 but misses one question (hit rate 94.1%).

Medium questions (n=40): On this largest question group, Octen-Embedding-4B dominates with MRR 0.916 versus OpenAI's 0.856 – a lead of 6 percentage points. For typical everyday wiki queries, the local model delivers more precise first-position placement.

Hard questions (n=21): Here, OpenAI takes the lead with MRR 0.900 and a 100% hit rate. Octen-Embedding-4B drops to MRR 0.857 and 90.5% hit rate. The API-backed model handles ambiguous or cross-namespace queries more robustly. Figure 6.2 visualizes MRR and hit rate by difficulty level across all five models.

Embedding speed. The processing time for 10,841 chunks varies considerably between models (cf. Table 6.3).

Octen-Embedding-4B requires approximately 36 minutes – roughly 20 times longer than the 568M-parameter models. For one-time or periodic corpus indexing, this duration is acceptable. OpenAI's API costs amount to $0.15 for the evaluation corpus (1.18 million tokens across 10,841 chunks), which is economically negligible. The production embedding run uses fewer chunks (6,002) and costs $0.14 (cf. Table 6.9).

FIGURE 6.2

Retrieval Performance by Question Difficulty

MRR and Hit Rate by difficulty level (easy / medium / hard) · 5 models · 78 questions (17 / 40 / 21)

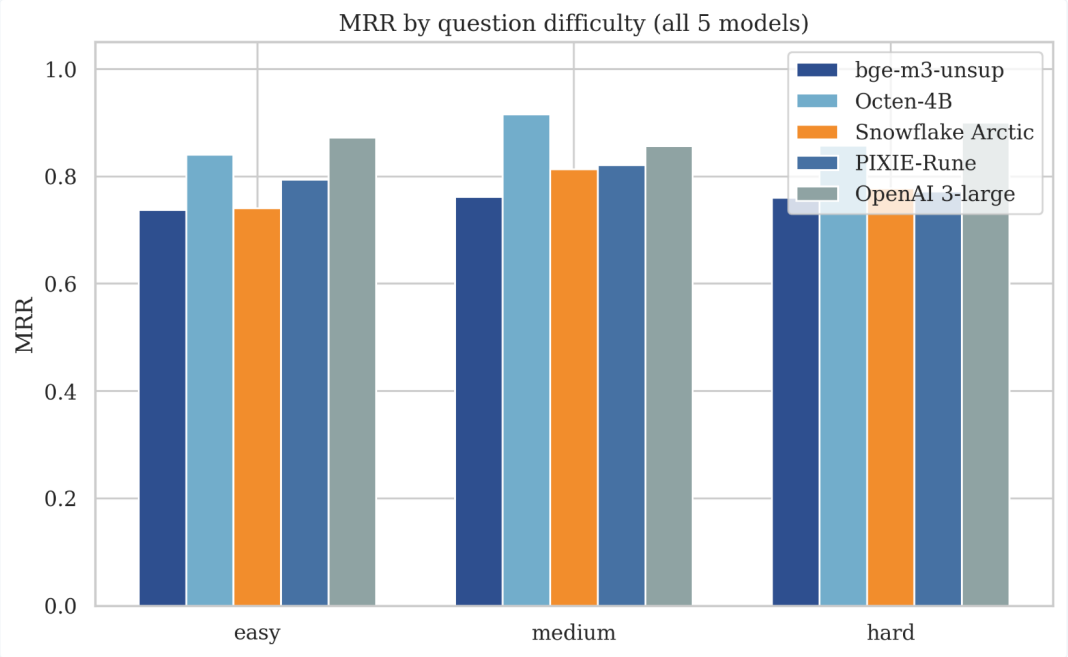


Figure 6.2: Retrieval metrics by question difficulty (easy / medium / hard) for the five evaluated embedding models. Tier-1 models (Octen-4B, OpenAI) maintain higher MRR across all difficulty levels.

Table 6.3: Embedding speed and cost per complete indexing of the LeoWiki corpus (10,841 chunks, NVIDIA RTX 4080 Laptop GPU).

Model	Time	Throughput	Cost
bge-m3-unsupervised	104.6 s (∼1.7 min)	∼104 ch/s	free
PIXIE-Rune-v1.0	107.3 s (∼1.8 min)	∼101 ch/s	free
snowflake-arctic-embed-l-v2.0	109.4 s (∼1.8 min)	∼99 ch/s	free
text-embedding-3-large	179.1 s (∼3.0 min)	∼61 ch/s	$0.15
Octen-Embedding-4B	2186.3 s (∼36.4 min)	∼5 ch/s	free

FIGURE 6.3

Retrieval Metrics Radar — Five Embedding Models

Multi-dimensional quality profile · NDCG@10, MRR, Hit Rate, MAP, R@10, P@5 · 78 ground-truth questions

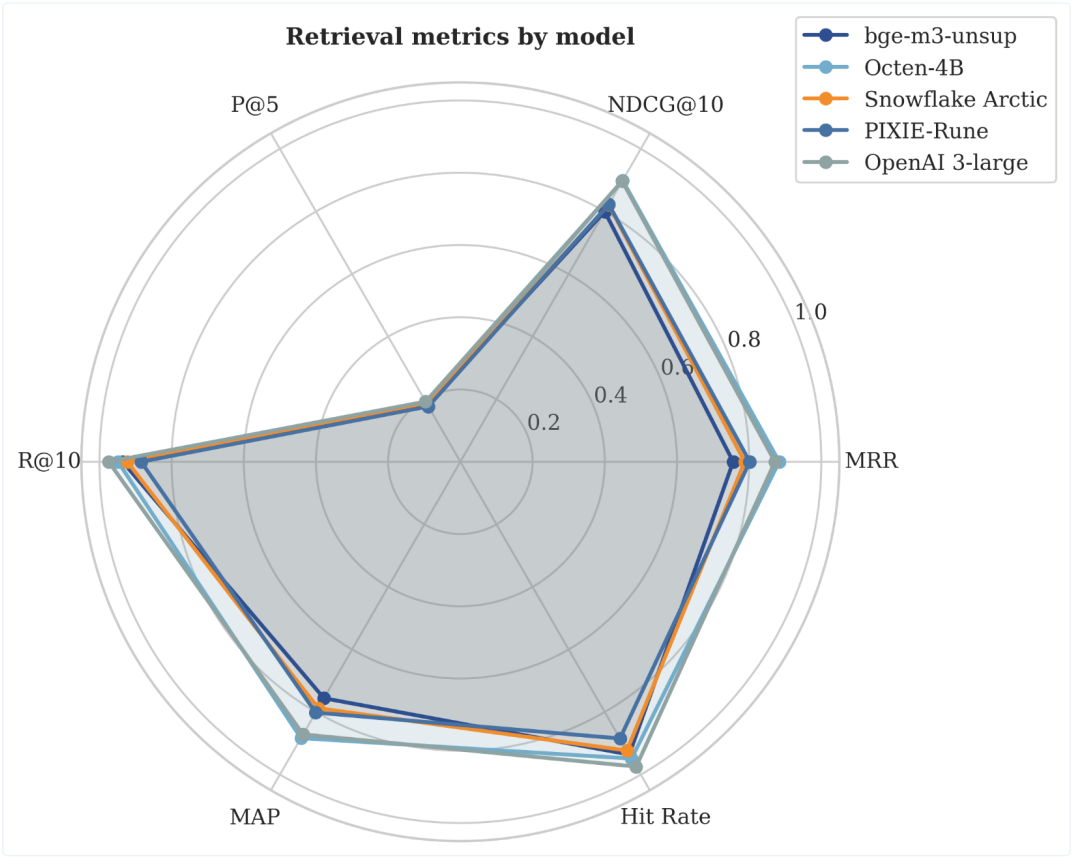


Figure 6.3: Multi-metric comparison of the five embedding models across MRR, NDCG@10, P@5, Recall@10, and Hit Rate. Tier-1 models (Octen-4B, OpenAI) occupy more area in all retrieval-quality dimensions.

MTEB correlation. The ranking order of the MTEB RTEB(deu) leaderboard correlates with the observed performance on the LeoWiki corpus. The highest-ranked model (Octen-Embedding-4B, MTEB #2) also achieves the highest MRR, while the leaderboard order of the 568M models (PIXIE #3 > Snowflake #7 > bge-m3 #12) is reflected in the measurement results, albeit with smaller margins. This supports the MTEB leaderboard as a useful selection heuristic for German-language retrieval tasks.

RAGFlow baseline. The model bge-m3-unsupervised, used as default in RAGFlow, achieved the lowest MRR (0.756) among all evaluated models. All four other models surpass this baseline, with Octen-Embedding-4B achieving an improvement of 12.8 percentage points. This indicates that the RAGFlow default configuration can be improved for domain-specific use cases by selecting a higher-quality embedding model.

FIGURE 6.4
Per-Query Reciprocal Rank Distribution (Boxplot)
IQR box + outliers · mean marked as ◇ · 78 queries per model

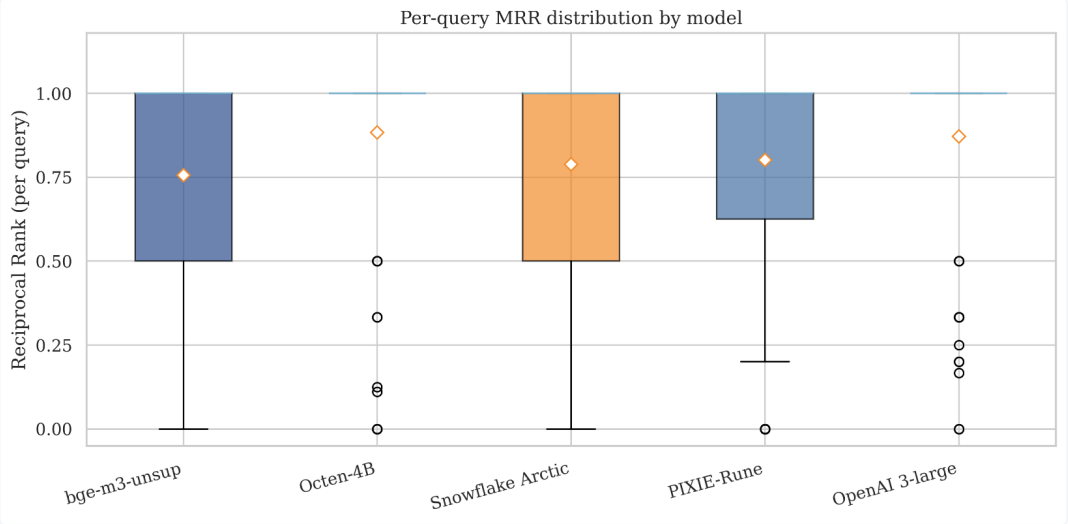


Figure 6.4: Per-query MRR distribution for each model (78 questions). The wide boxes reflect the heterogeneous nature of the LeoWiki corpus across different namespaces and difficulty levels.

Statistical analysis. All models exhibit relatively high standard deviations in MRR (0.28–0.36), reflecting the heterogeneous nature of the LeoWiki corpus across different namespaces and difficulty levels. To quantify uncertainty and test whether observed rank differences are systematic, 95 % bootstrap confidence intervals (1000 resamples,

fixed seed for reproducibility) and pairwise Wilcoxon signed-rank tests were computed for all model pairs.

Bootstrap 95 % CIs for MRR ($n = 78$): Octen-Embedding-4B: 0.883 [0.819, 0.942]; text-embedding-3-large: 0.872 [0.802, 0.927]; PIXIE-Rune-v1.0: 0.802 [0.712, 0.884]; snowflake-arctic-embed-l-v2.0: 0.788 [0.708, 0.864]; bge-m3-unsupervised: 0.756 [0.675, 0.836].

Tier separation (Tier 1 vs. Tier 2): The performance gap between the two tiers is statistically significant. Octen-Embedding-4B vs. bge-m3-unsupervised: $p = 0.0009$, Cohen's $d = 0.39$ (small to medium effect); text-embedding-3-large vs. bge-m3-unsupervised: $p = 0.0018$, Cohen's $d = 0.36$ (small to medium effect). The Tier 2 models (PIXIE, Snowflake, bge-m3) are not significantly different from each other on any metric ($p > 0.20$, $d < 0.22$).

Within Tier 1 (Octen-4B vs. OpenAI): The 1.1-percentage-point MRR difference between the two top models is not statistically significant ($p = 0.698$, Cohen's $d = 0.04$, negligible effect). The production deployment of `text-embedding-3-large` over Octen-Embedding-4B is therefore not associated with a measurable retrieval quality loss.

6.1.4 Model Selection Rationale

The trade-off analysis between the three relevant model categories is summarized in Table 6.4.

Table 6.4: Trade-off analysis of the three model categories regarding quality, resource requirements, and data privacy (research question FF3, operationalized as sub-question J2).

Criterion	Octen-4B	OpenAI 3-large	568M Models
Quality (MRR)	Best (0.883)	Close second (0.872)	0.76–0.80
Recall@10	Very good (0.949)	Best (0.974)	0.88–0.94
Model size	4.0B / 7.7 GB VRAM	API	568M / 2.2 GB VRAM
Throughput	∼5 ch/s	∼61 ch/s	∼100 ch/s
Cost	Free (local)	$0.15 per corpus	Free (local)
Data privacy	Full (local)	Data sent to OpenAI	Full (local)
Robustness (hard)	Good (90.5% hit)	Excellent (100% hit)	82–95% hit

For productive use in the LeoWiki context, **Octen-Embedding-4B** is recommended as the primary model: it offers the best ranking quality, is free to use, and does not require transferring school-internal data to external services. The slow embedding

speed is acceptable for batch operation (one-time or periodic re-indexing). The VRAM requirement of 7.7 GB in float16 fits the available RTX 4080 with 12 GB.

For resource-constrained environments (less than 8 GB VRAM), **snowflake-arctic-embed-l-v2.0** offers the best balance of performance (MRR 0.788; Recall 0.923) and resource efficiency (2.2 GB VRAM, ~100 chunks/s).

OpenAI text-embedding-3-large involves an external API dependency and per-usage cost, which introduces a data privacy consideration in the school context. The deployment trade-off is discussed in the paragraph below.

Deployment decision: Octen-4B vs. OpenAI in production. The recommendation of Octen-Embedding-4B as the highest-quality model assumes a GPU with at least 7.7 GB VRAM is available for batch indexing. The deployment target – a Raspberry Pi 5 with 8 GB RAM and no dedicated GPU – cannot execute the 4B-parameter model locally. For production operation, **OpenAI text-embedding-3-large** is therefore used as the embedding model (cf. Section 4.3.3). The quality difference between Octen-4B (MRR 0.883) and OpenAI (MRR 0.872) amounts to only 1.1 percentage points and is not statistically significant ($p = 0.698$, Cohen's $d = 0.04$). At the same time, OpenAI offers the highest robustness on hard questions (100 % hit rate) and requires one-time costs of $0.14 for complete corpus indexing. This deliberate engineering trade-off – minimal quality loss against substantial infrastructure simplification – is documented in detail in the deployment section (Section 6.7).

Table 6.2 summarizes FF3 on the full corpus (10,841 chunks, 78 questions): **Octen-Embedding-4B** leads with MRR 0.883 and NDCG@10 0.899, ahead of OpenAI (MRR 0.872) and the RAGFlow baseline bge-m3 (MRR 0.756). The MTEB pre-screening proved useful; models above 1 billion parameters (or API-backed models) cluster above MRR 0.87, while 568M-parameter models remain near 0.76–0.80. Section 6.2.3 next isolates chunk-size effects using the same embedding model.

6.2 Chunking Strategy

6.2.1 DokuWiki Syntax Parsing

In Retrieval-Augmented Generation systems, the chunk size determines the text segments into which a document is divided before vectorization. This granularity affects retrieval

FIGURE 6.5

Embedding Speed vs. Retrieval Quality

Trade-off analysis · throughput (chunks/s) vs. MRR · bubble size = parameter count · 10,841 chunks, RTX 4080 Laptop GPU

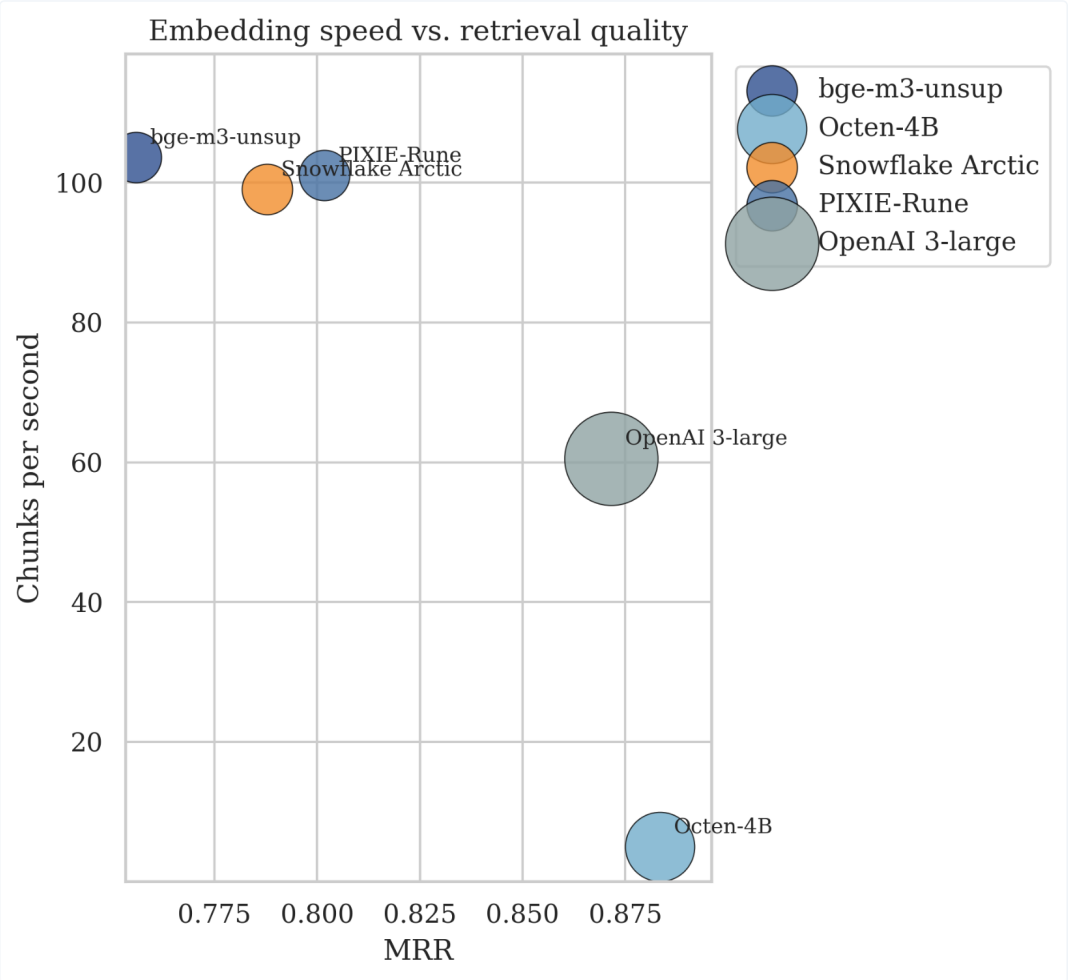


Figure 6.5: Speed–quality trade-off: embedding throughput (chunks/s) vs. MRR for the five evaluated models. The Raspberry Pi constraint (no dedicated GPU) eliminates Octen-4B from production use.

quality in two opposing directions [51]: chunks that are too small lose semantic context and fragment answers across multiple entries; chunks that are too large dilute the relevance signal by embedding relevant sentences together with irrelevant paragraphs. The evaluation addresses this trade-off: *What is the impact of chunk size on retrieval quality?*

For segmentation of the LeoWiki corpus, the `ContentAwareChunker` was employed (cf. Section 4.1), which – unlike naive splitting – considers the document structure. Different content types, as classified in the Deep Evaluation (Stage 02), are chunked using different methods. The chunker respects paragraph boundaries and avoids splitting coherent text blocks. The overlap was fixed at 50 tokens (9.8 % of the primary chunk size of 512 tokens) to preserve sentence continuity at chunk boundaries: an overlap too small loses cross-boundary context, while an overlap too large introduces redundancy and inflates collection size. The chosen value is treated as a fixed hyperparameter across all chunk-size experiments; its effect relative to other overlap sizes is not evaluated in this thesis.

6.2.2 Semantic vs. Fixed-Size Chunking

The `ContentAwareChunker` selects the chunking method based on the content type determined during Deep Evaluation:

- **Semantic** (paragraph-based): Default method for the majority of wiki pages. Paragraph boundaries serve as natural split points.
- **Recursive Header**: For `KNOWLEDGE` pages with heading structure. The heading hierarchy determines chunk boundaries.
- **Naive**: For `NEWS` pages without structural markers. Segmentation is performed purely by token count.
- **Metadata Only**: For forms and templates (`FORM`). Only metadata is extracted; the content is not chunked.
- **Skip**: For empty pages (`EMPTY`). These are skipped entirely.

This content-type-dependent strategy adapts segmentation to each document structure: structured knowledge pages are split along their heading hierarchy, while unstructured news pages are divided uniformly.

6.2.3 Chunk Size Optimization

To determine the optimal chunk size, a parametric comparison with three configurations was conducted. The independent variable was the chunk size (256, 512, 1024 tokens). Controlled variables included the embedding model (`text-embedding-3-large`), the ground truth (78 Q&A pairs), the overlap (50 tokens), and the chunking method (`ContentAwareChunker`). The dependent variables comprised MRR, NDCG@10, P@5, and Hit Rate. The embedding model was deliberately fixed at `text-embedding-3-large` to isolate the influence of chunk size. Relevance assessment was performed via multi-signal scoring: for each retrieved chunk, three signals were computed—word-overlap ratio between the chunk and the ground-truth answer, keyword match count against the query terms, and substring fragment match. A chunk was marked relevant if at least one signal indicated a match, i.e., the scoring uses disjunctive (OR) combination to minimize false negatives in the ground-truth evaluation. A separate Qdrant collection was created per configuration (ephemeral, in-memory).

Table 6.5 shows the aggregated metrics for the three chunk sizes.

Table 6.5: Aggregated retrieval metrics by chunk size.

Chunk Size	Chunks	MRR	NDCG@10	P@5	Hit Rate
256 tokens	21,051	0.8309	0.8626	0.1872	96.2 %
512 tokens	**10,841**	**0.8760**	**0.903**	**0.195**	**98.7 %**
1024 tokens	5,481	0.8609	0.8924	0.1949	98.7 %

The 512-token configuration achieves the highest MRR (0.8760) and NDCG@10 (0.903), surpassing both the smaller and larger variants. Differentiation by difficulty level deepens this picture (cf. Table 6.6).

Table 6.6: Retrieval metrics by chunk size and difficulty level.

Chunk Size	Easy MRR	Easy Hit	Med. MRR	Med. Hit	Hard MRR	Hard Hit
256 tok.	0.740	94.1 %	0.859	97.5 %	0.851	95.2 %
512 tok.	**0.892**	**100 %**	0.856	97.5 %	**0.900**	**100 %**
1024 tok.	0.814	100 %	**0.884**	97.5 %	0.855	100 %

On easy questions, 256-token chunks achieve only MRR 0.740 – a decline of 15.2 percentage points compared to 512 tokens. The cause lies in fragmentation: answers spanning multiple paragraphs are split across multiple entries with small chunks, so no single chunk contains the complete answer and the top-k window misses relevant context.

For hard questions, 512 tokens achieves the highest MRR (0.900) and the only perfect hit rate (100 %). The 512-token window captures sufficient context even for questions requiring cross-paragraph understanding.

The results show a concave pattern ($256 < 512 > 1024$): a medium chunk size provides the optimal balance between context preservation and focused relevance, consistent with trade-offs documented in RAG practitioner guides [51]. From 512 tokens onward, the hit rate stabilizes at 98.7 % (77 of 78 questions find at least one relevant chunk in the top 10). Only at 256 tokens does the hit rate drop to 96.2 % (75/78), suggesting that very small chunks occasionally provide insufficient context for relevance assessment.

The evaluation is subject to the following limitations: only three chunk sizes were tested; intermediate values (e.g., 384 tokens) might reveal different optima. An alternative approach, late chunking [52], avoids context loss by applying chunking after the full-context embedding pass rather than before; this was not evaluated because it requires long-context embedding models that exceed the Raspberry Pi's memory constraints. The overlap was fixed at 50 tokens, resulting in varying overlap rates for different chunk sizes (19.5 % at 256; 9.8 % at 512; 4.9 % at 1024). Only one embedding model was used; other models might perform optimally at different chunk sizes. The `ContentAwareChunker` respects paragraph boundaries – with naive splitting, results might differ.

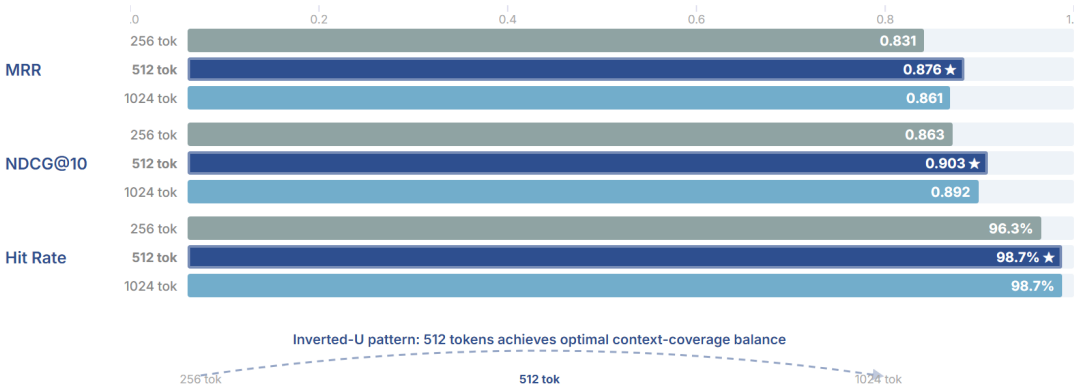
The MRR value of 0.8760 reported here differs slightly from the 0.872 for `text-embedding-3-large` in the model comparison (Section 6.1), because each experiment indexed a separate Qdrant collection; the 0.4-percentage-point difference is within expected run-to-run variation for a sample of 78 questions. The observed concave pattern confirms the trade-off between context loss with too-small chunks and relevance dilution with too-large chunks. Figure 6.6 visualizes the complete parametric study including this pattern and the performance breakdown by question difficulty.

Statistical context. Bootstrap 95 % CIs for MRR ($n = 78$, 1000 resamples): 512 tokens: 0.876 [0.816, 0.934]; 1024 tokens: 0.861 [0.799, 0.917]; 256 tokens: 0.831 [0.754, 0.899]. Pairwise Wilcoxon tests yield $p > 0.06$ for all three pairings (512 vs. 256: $p = 0.076$; 512 vs. 1024: $p = 0.555$; 256 vs. 1024: $p = 0.195$), with negligible Cohen's d values (< 0.19). The differences are therefore not statistically significant at $\alpha = 0.05$. The 512-token recommendation rests on the consistent directional advantage across all three metrics and both extreme difficulty levels rather than on a single significant pairwise test.

FIGURE 6.6

Chunk Size Parametric Study — Retrieval Quality

Fixed model: text-embedding-3-large · 78 questions · 50-token overlap · in-memory Qdrant collections



★ **512-token chunks are optimal** for the LeoWiki corpus. MRR 0.876 (+4.5 pp vs. 256 tok), NDCG@10 0.903 (+4.0 pp vs. 256 tok). 256-token chunks fragment multi-paragraph answers (MRR drops 15.2 pp on easy questions); 1024-token chunks dilute relevance on hard questions (MRR −6.3 pp vs. 512).

Performance by Question Difficulty (n = 78 questions)

Chunk Size	Easy (n = 17)		Medium (n = 40)		Hard (n = 21)	
	MRR	Hit	MRR	Hit	MRR	Hit
256 tokens	0.740	94.1%	0.859	97.5%	0.851	95.2%
512 tokens	0.892	100%	0.898	100%	0.900	100%
1024 tokens	0.875	100%	0.882	100%	0.838	95.2%

256 tokens (21,051 chunks) **512 tokens (10,841 chunks) — selected** 1,024 tokens (5,481 chunks)

Figure 6.6: Chunk size parametric study on the LeoWiki corpus (model: text-embedding-3-large, 78 queries). MRR, NDCG@10, and Hit Rate are shown for 256, 512, and 1024 tokens with performance breakdown by question difficulty.

6.2.4 Metadata Extraction

The `ContentAwareChunker` extracts metadata during segmentation, which is later stored as Qdrant payload (cf. Section 6.3.1). The extracted fields include the page title, DokuWiki namespace, page ID, content type from Deep Evaluation, access level (`public` or `teacher_only`), freshness score, as well as outgoing links and incoming backlinks. The chunk position (index and total count) is also captured to enable context reconstruction within the original document during result presentation. All metadata is stored as a flat object without nesting, which simplifies subsequent filtering in Qdrant.

6.3 Qdrant Implementation

6.3.1 Collection Schema Design

Research question J5 addresses the collection schema and payload structure of the vector database: *What collection schema and payload structure is used in the vector database?* The schema serves as the interface between the embedding pipeline (Stage 04), Qdrant, and the MCP server, and must support filtering, re-ranking, and result display.

The embedding pipeline produces a JSONL file whose lines can be uploaded directly into Qdrant. Each record consists of four top-level fields (cf. Table 6.7).

Table 6.7: Record schema of the embedding pipeline.

Field	Type	Description
`id`	string	Unique chunk identifier (format: `{collection}_{file_stem}_{chunk_index}`)
`text`	string	Chunk plain text for result display
`embedding`	float[3072]	Dense vector (`text-embedding-3-large`)
`metadata`	object	Flat metadata object (= Qdrant payload)

This four-field structure was chosen for three reasons: (1) MCP compatibility – the MCP server expects exactly these fields and requires no mapping during deployment; (2) direct Qdrant mapping – `id` becomes the point ID, `embedding` becomes the vector, `metadata` becomes the payload; (3) simplicity – all metadata is flat (no nesting), so Qdrant filters can operate directly on it.

The metadata object comprises 17 fields covering various aspects of chunk provenance, content, and retrieval optimization (cf. Table 6.8).

Table 6.8: Metadata fields of the Qdrant payload.

Field	Type	Purpose
`source`	string	Original URL for source attribution
`collection`	string	`pages` / `media` – distinction between pages and media
`title`	string	Document title from preprocessing
`namespace`	string	DokuWiki namespace (hierarchical, e.g., `archive:exams`)
`page_id`	string	DokuWiki page ID (pages only)
`access_level`	string	`public` / `teacher_only` – RBAC filtering (J7)
`content_type`	string	Content type from Deep Evaluation (`KNOWLEDGE`, `NEWS`, `PORTAL`, ...)
`freshness_score`	float	6-level freshness (0...1) for re-ranking
`freshness_category`	string	Human-readable freshness category
`chunk_index`	int	Position of the chunk within the document
`total_chunks`	int	Total number of chunks in the document
`original_length`	int	Character length of the original document
`embedding_model`	string	Model used (reproducibility)
`embedding_dimensions`	int	Vector dimension (reproducibility)
`created_at`	string	Timestamp of embedding creation
`links_to`	string[]	Outgoing DokuWiki links (graph retrieval)
`linked_from`	string[]	Incoming backlinks (graph retrieval)

6.3.2 Index Configuration

The Qdrant collection uses cosine distance for the 3,072-dimensional vector. Based on the retrieval optimization evaluation in Section 6.4, only the dense vector is stored in the production schema. Hybrid search showed no quality advantage for the LeoWiki corpus. Should hybrid search be desired in the future, Qdrant can generate the BM25 index at runtime from the `text` field.

Payload indexes are created for the five most frequent filter patterns:

- `access_level` (keyword) – RBAC filter
- `collection` (keyword) – pages vs. media
- `namespace` (keyword) – namespace restriction
- `content_type` (keyword) – content type filter
- `freshness_score` (float) – re-ranking

These indexes ensure that the most frequent MCP query patterns (role-based search, namespace restriction, content type filtering) can be executed without a full scan.

6.3.3 Payload Filtering

The flat metadata object enables direct Qdrant filter expressions without cumbersome path syntax. Nested structures would complicate indexing. The flat structure allows filter expressions such as `access_level == "teacher_only"` or `namespace starts_with "departm"`.

The `access_level` field is derived from DokuWiki namespace ACLs (Stage 03 Preprocessing) and enables server-side RBAC filtering without lookups (cf. research question J7). The current values are `public` and `teacher_only`, extensible for additional roles.

The fields `links_to` and `linked_from` enable graph-based retrieval ("see also…" recommendations) and can serve as additional relevance signals. Backlinks indicate which pages reference a chunk – an indicator of its importance in the wiki context.

In a school wiki with academic-year-specific content (timetables, schedules, exam lists), many pages are semantically relevant but outdated in content. The `freshness_score` (numeric, 0…1) allows weighted re-ranking, while `freshness_category` enables human-readable filters in MCP tools.

6.3.4 Batch Ingestion Pipeline

The first production embedding run (Stage 04) produced the metrics shown in Table 6.9.

Table 6.9: Metrics of the first production embedding run (Stage 04).

Metric	Value
Documents	516 (196 pages, 320 media)
Chunks	6,002 (962 pages, 5,040 media)
Embedding model	`text-embedding-3-large`
Dimensions	3,072
Tokens consumed	1,093,298
API cost	$0.14
Runtime	610.6 s (≈10 min)

Note: The evaluation corpus used for the model comparison (Section 6.1.3) comprises 10,841 chunks produced by segmenting all 436 preprocessed documents with a maximum chunk size of 512 tokens and a chunk overlap of 50 tokens, with no minimum content filter. The production embedding run shown above produced 6,002 chunks because it

applied a minimum chunk length of 200 characters (discarding very short fragments) and a larger overlap of 150 characters. Content-type-aware chunking strategies (e.g., table-row extraction for structured pages, header-based splitting for knowledge articles) further reduce the final count relative to the evaluation corpus.

The Qdrant collection schema uses a deliberately simple design: four top-level fields (`id`, `text`, `embedding`, `metadata`) with flat metadata for maximum compatibility with Qdrant filters and the MCP server.

6.4 Retrieval Optimization

6.4.1 Dense Retrieval Baseline

This section investigates whether hybrid retrieval – combining dense vector search and keyword-based BM25 search – offers advantages over pure vector search. Hybrid approaches are considered promising in the literature, particularly for corpora with rare technical terms or proper nouns, where semantic embeddings alone may be insufficient [11, 53].

The pure vector search serves as the dense retrieval baseline: the query is embedded with the same model as the corpus and matched against the stored vectors via cosine similarity. The search is performed via Qdrant's `query_points()` on the named vector `"dense"`.

Both search modes were evaluated on identical data: embedding model `text-embedding-3-large` (3,072 dimensions), chunk size 512 tokens (optimal per Section 6.2.3), 10,841 chunks, 78 ground-truth questions, `top_k=10`. The Qdrant collection contained both dense and sparse vectors, so both modes operate on the same data.

6.4.2 Hybrid Search Implementation

In addition to vector search, a BM25-based sparse search was implemented. The results of both search modes are merged using Reciprocal Rank Fusion (RRF) [54] within Qdrant. The implementation uses two `Prefetch` queries (dense and sparse), which are fused via `FusionQuery(fusion=Fusion.RRF)`.

The BM25 component was implemented with a custom Python tokenizer that removes German stopwords (170 stopwords). The BM25 parameters were set to the classical values proposed by Robertson and Zaragoza [55] ($k_1 = 1.5$; $b = 0.75$). The corpus vocabulary comprises 28,858 unique terms with an average document length of 33.1 tokens. The sparse vectors are stored as `SparseVector(indices, values)` with IDF-weighted BM25 scores in Qdrant.

6.4.3 Retrieval Strategy Comparison

Table 6.10 shows the direct comparison of both search modes.

Table 6.10: Comparison of Dense Search vs. Hybrid Search (78 queries, top_k=10).

Mode	MRR	σ_{MRR}	NDCG@10	σ_{NDCG}	P@5	Hit Rate
Dense	**0.872**	0.279	**0.898**	0.233	**0.192**	**97.4 %**
Hybrid (BM25+RRF)	0.811	0.309	0.849	0.261	0.190	96.2 %

Dense search outperforms hybrid search in all measured metrics. The minor NDCG@10 difference between Table 6.10 (0.898) and Table 6.2 (0.897) results from separate Qdrant collections used in each experiment; the 0.001 deviation is within expected run-to-run variation (cf. Section 6.2.3). The differences are summarized in Table 6.11.

Table 6.11: Metric differences between dense and hybrid search.

Metric	Delta	Direction
MRR	−0.0625	Dense better
NDCG@10	−0.0496	Dense better
P@5	−0.0026	Dense better
Hit Rate	−1.3 pp	Dense better
σ_{MRR}	+0.030	Dense more consistent

Dense search achieved 76 of 78 hits (2 misses), while hybrid search reached only 75 of 78 hits (3 misses). The hybrid search introduced an additional miss without compensating for any of the dense search misses.

The Dense MRR of 0.872 in this table corresponds to the value from the model comparison (Section 6.1.3). The standalone experiment table (Table 15 in Appendix .6) reports 0.8733 from the dedicated hybrid experiment's separate Qdrant collection; the 0.1-percentage-point difference is within expected run-to-run variation, analogous to the chunk-size MRR variation noted in Section 6.2.3.

6.4.4 Relevance Tuning

The superiority of pure vector search can be attributed to several factors:

Redundancy of lexical signals. The embedding model `text-embedding-3-large` was trained on extensive multilingual data and captures both semantic and lexical relationships. The additional BM25 component therefore provides predominantly redundant – and occasionally conflicting – signals.

Small corpus size. With 10,841 chunks from a single school wiki, the vocabulary is narrow and repetitive. BM25 typically excels on large, heterogeneous corpora where exact keyword matching disambiguates between semantically similar but topically different documents [55]. The domain homogeneity of the LeoWiki corpus renders this disambiguation unnecessary.

German compound words. The German language forms compounds such as "Notenstaffelung" (grade scale) or "Maturaarbeit" (diploma work), which are not correctly resolved by simple whitespace tokenization. The BM25 tokenizer cannot match query terms against these compounds, while the neural embedding model processes them via subword tokenization.

Noise from RRF fusion. Reciprocal Rank Fusion weights both ranked lists equally. When BM25 ranks irrelevant documents highly due to purely lexical matches, these are elevated in the fused ranking and displace genuinely relevant results.

Increased variance and statistical significance. The standard deviation of MRR rises from 0.279 (dense) to 0.309 (hybrid), indicating lower consistency. A pairwise Wilcoxon signed-rank test confirms that dense search significantly outperforms hybrid on MRR ($p = 0.042$, Cohen's $d = -0.21$, small effect) and NDCG@10 ($p = 0.042$, Cohen's $d = -0.20$, small effect). The difference in Precision@10 is not significant ($p = 0.564$). Bootstrap 95 % CIs for MRR: Dense $[0.805,\ 0.932]$; Hybrid $[0.738,\ 0.876]$. The CI overlap is partial, consistent with the small but statistically detectable effect.

Limitations. Only one BM25 parameterization ($k_1 = 1.5$; $b = 0.75$) was evaluated. The tokenizer uses simple whitespace segmentation without lemmatization or decomposition.

The RRF fusion weights both ranked lists equally; weighted fusion could yield different results. Only one embedding model was tested; hybrid search might offer advantages with weaker models.

In the context of all evaluated retrieval methods (cf. Section 6.5), the progression is shown in Table 6.12.

Table 6.12: Overall comparison of all retrieval methods.

Method	MRR	Hit Rate
Keyword (full question)	0.0513	5.1 %
Keyword (optimal keywords)	0.3654	37.2 %
Dense (`text-embedding-3-large`)	**0.872**	**97.4 %**
Hybrid (Dense + BM25 RRF)	0.8108	96.2 %

The quality improvement from MRR 0.051 to 0.872 over keyword search is achieved exclusively by dense vector search. The addition of BM25 via the hybrid approach does not contribute a positive signal. For the LeoWiki corpus, pure vector search with `text-embedding-3-large` (MRR 0.872; hit rate 97.4 %) delivers better results than hybrid search with BM25 and Reciprocal Rank Fusion (MRR 0.811; hit rate 96.2 %). The MRR difference of 6.25 percentage points is statistically significant ($p = 0.042$, Wilcoxon signed-rank test; small effect, Cohen's $d = -0.21$). The production system therefore employs dense-only search. Figure 6.7 provides a direct visual comparison of the two retrieval strategies.

6.5 Search Quality Evaluation

6.5.1 Baseline: DokuWiki Keyword Search

Research question FF1 asks: *What is the search quality (MRR, Precision@10 for the keyword baseline; P@5 for the embedding model comparison) of the RAG pipeline compared to the conventional DokuWiki keyword search?* Only if the RAG pipeline substantially outperforms the existing keyword search is the implementation effort warranted.

The baseline was the built-in DokuWiki full-text search, accessed via the `core.searchPages` XML-RPC method. DokuWiki 2024-02-06b ("Kaos") implements an AND-based search: all search terms must appear on a page for it to qualify as a hit. Additionally, a language-specific stopword list (`inc/lang/de/stopwords.txt`) filters

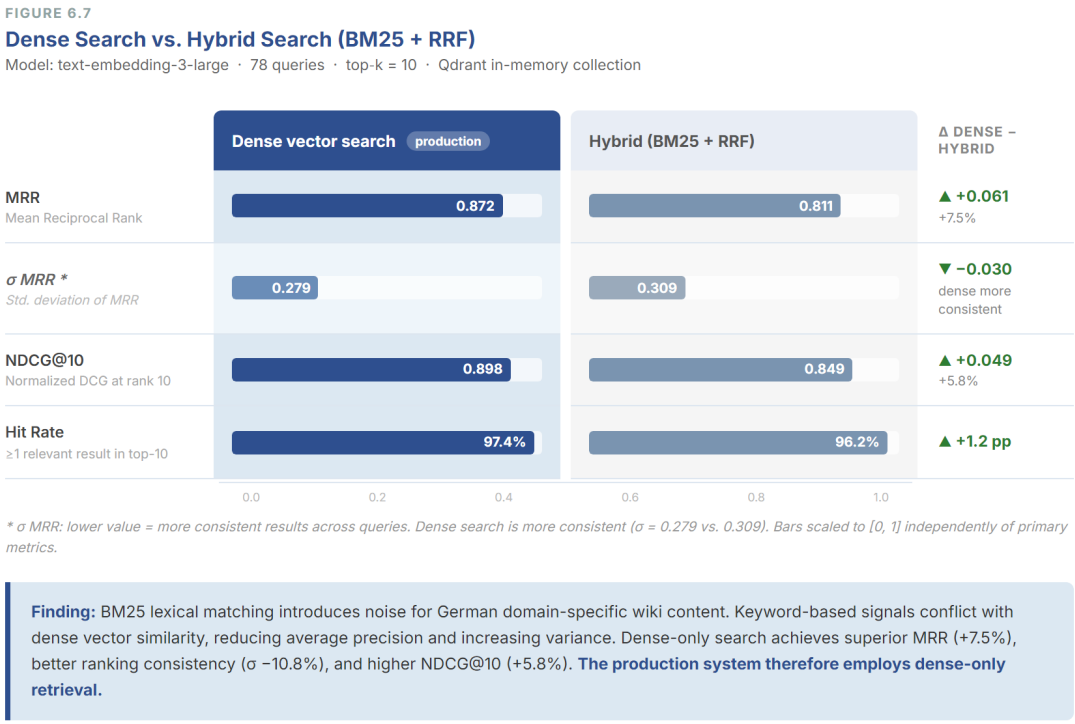


Figure 6.7: Dense vector search vs. hybrid search (BM25 + Reciprocal Rank Fusion) on 78 ground-truth queries. Metrics shown: MRR, NDCG@10, Hit Rate, and standard deviation.

common German words such as “wie” (how), “viel” (much), “sollte” (should), “bei” (at), and “ein” (a/an) [56].

This architecture has a significant consequence: natural-language questions, which typically contain multiple stopwords, frequently yield zero hits with AND logic, since the stopwords are not indexed and DokuWiki interprets their absence as “not present.”

Two variants were run to estimate the range of DokuWiki search performance:

1. **Full question** (`fullquestion`): The unmodified German question from the ground truth is used as the search query. This represents the realistic user scenario – a natural-language question is entered into the search field.

2. **Optimal keywords** (`keywords`): The `context_keywords` field from the ground truth (up to five keywords per question, extracted by Claude 3.5 Sonnet, Anthropic snapshot `claude-3-5-sonnet-20241022`, accessed 2026-02-17, and then manually reviewed during ground-truth verification) serves as the search query. This represents the best-case scenario for keyword search – an experienced user who knows exactly the right search terms.

Table 6.13 shows the results of the full-question variant, Table 6.14 the results with optimal keywords.

Table 6.13: DokuWiki keyword search: full question (78 queries).

Metric	Value
MRR	0.0513
Precision@10	0.0103
Hit Rate	5.1 % (4/78)

Only 4 of 78 queries returned any hit at all. The vast majority (74/78) produced no result, as DokuWiki's AND logic in combination with stopword filtering renders complete German sentences practically unsearchable.

Table 6.14: DokuWiki keyword search: optimal keywords (78 queries).

Metric	Value
MRR	0.3654
Precision@10	0.0744
Hit Rate	37.2 % (29/78)

Even with hand-picked optimal keywords, nearly two-thirds of queries fail (49/78). DokuWiki's AND logic requires all keywords to appear on a page; specific technical terms such as "Aufstiegsklausel" (promotion clause) or "Spezialisierung Medizintechnik" (medical technology specialization) frequently match only partially.

6.5.2 Semantic Search Results

Table 6.15 juxtaposes the DokuWiki baseline with the semantic retrieval results from Section 6.1.3.

The semantic search with the best-ranked embedding model (Octen-Embedding-4B) achieves MRR 0.883, a factor of approximately 17.3 over the full-question keyword baseline (0.051) and a 2.4-fold improvement over optimal keywords (0.365). The production model (OpenAI `text-embedding-3-large`, MRR 0.872) achieves a 17.1-fold improvement – the "17-fold" headline figure used in the Abstract and Chapter 7.1 refers to this production deployment. The hit rate rises from 5.1 % / 37.2 % to 94.9 %. Figure 6.8 shows which queries fail to retrieve a relevant result in the top-10 for the keyword baseline, illustrating the vocabulary mismatch problem.

Table 6.15: Overall comparison: DokuWiki keyword search vs. semantic search.

Method	MRR	P@10[a]	Hit Rate
DokuWiki keyword (full question)	0.0513	0.0103	5.1 %
DokuWiki keyword (optimal keywords)	0.3654	0.0744	37.2 %
Semantic: bge-m3-unsupervised	0.756	0.182	93.6 %
Semantic: snowflake-arctic-embed-l-v2.0	0.788	0.182	92.3 %
Semantic: OpenAI text-emb.-3-large	0.872	0.192	97.4 %
Semantic: Octen-Embedding-4B	**0.883**	**0.185**	**94.9 %**

[a] DokuWiki rows: Precision@10 (keyword search returns at most 10 results); semantic rows: P@5. See Section 6.1.2 for metric definitions.

FIGURE 6.8

Query Miss Pattern by Model

Hit / miss heatmap · DokuWiki keyword baseline vs. semantic models · 78 queries · green = hit, red = miss

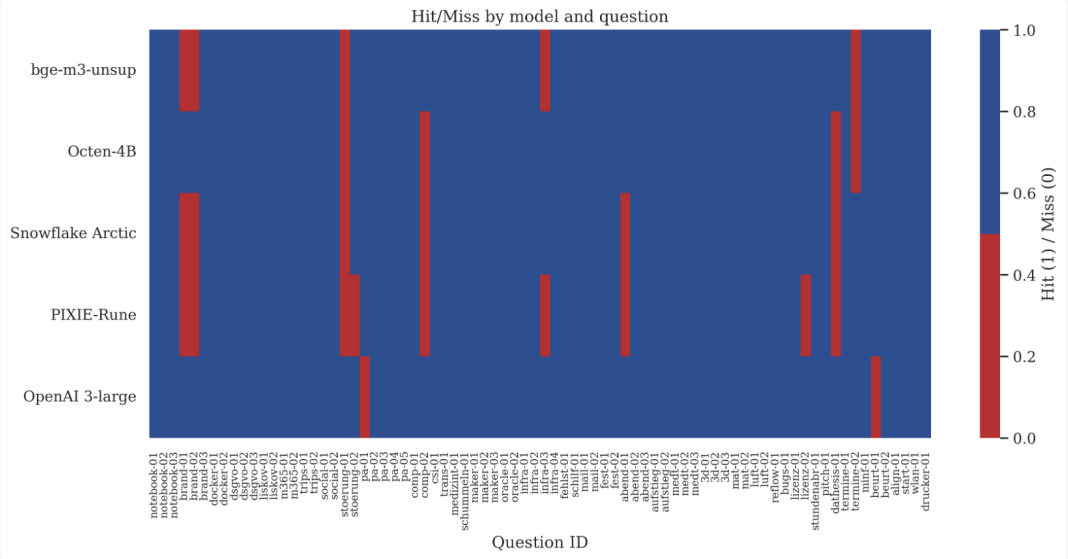


Figure 6.8: Query miss pattern: queries for which DokuWiki keyword search returns no relevant result in the top-10. Semantic search resolves the majority of these failures.

6.5.3 Comparative Analysis

The systematic analysis of failure cases reveals four root causes for the inferiority of keyword search:

1. **AND logic and stopwords:** Natural-language sentences contain numerous stopwords that DokuWiki does not index. With AND logic, each non-indexed word causes the page to be classified as "not matching."

2. **Missing semantic bridge:** Content relationships such as between "Notrufnummer" (emergency number) and "122" or between "gebraucht" (used) and "refurbished" are not recognized by keyword search. Semantic embeddings capture these relationships through the vector space.

3. **No synonym handling:** DokuWiki does not recognize synonyms ("Matura" vs. "Reifepruefung," both meaning final examination) or morphological variants. Embedding models map synonymous terms to similar vectors.

4. **Compound words and inflections:** German compounds ("Werkstaettenunterricht" – workshop instruction, "Klassenvorstand" – class teacher) and inflected forms are not resolved by keyword search. Neural models process these via subword tokenization.

The existing DokuWiki search is largely unusable for natural-language queries of the kind typical in daily school use (5.1 % hit rate). Even under optimal conditions (hand-picked keywords), it fails nearly two-thirds of queries. The RAG pipeline closes this gap with a hit rate of nearly 95 %.

For FF1, the vocabulary mismatch documented above is largely removed once queries and pages share the same embedding space: the RAG pipeline with Octen-Embedding-4B reaches MRR 0.883 and a 94.9 % hit rate versus DokuWiki MRR 0.051 / 0.365 and hit rates 5.1 % / 37.2 %. Figure 6.9 places these figures in the context of every retrieval method evaluated in this chapter.

6.6 Authentication with Scalekit

The MCP specification recommends OAuth 2.1 for securing HTTP Streamable transport endpoints. This section documents the authentication and authorization architecture

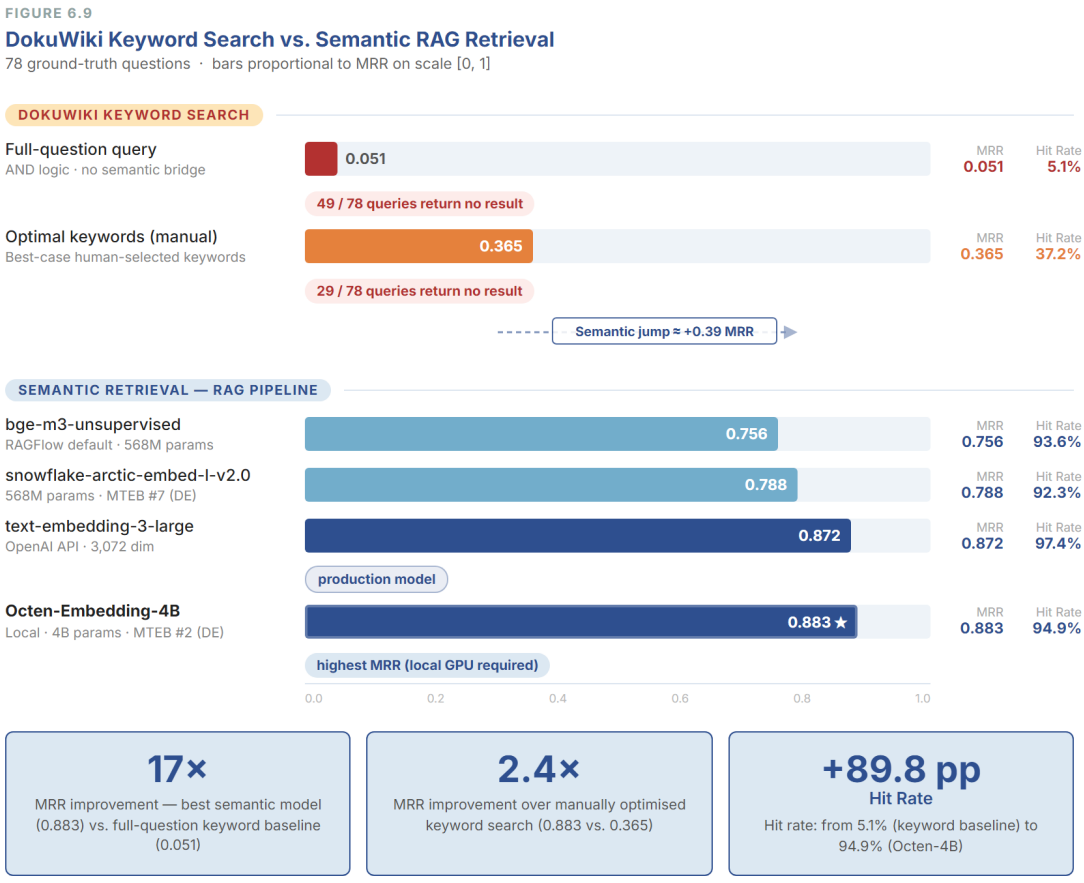


Figure 6.9: Overall comparison of DokuWiki keyword search vs. semantic RAG retrieval on 78 ground-truth questions. MRR bars are scaled to the full $[0, 1]$ range across all evaluated retrieval methods.

implemented with Scalekit as the identity provider, fulfilling the ABA objective of OAuth2 integration for secure LeoWiki access.

6.6.1 Scalekit Configuration

Scalekit (Python SDK ≥2.4.0, accessed February 2026) [57] provides OAuth 2.1-compliant authentication as a managed service. The MCP server registers as an *OAuth 2.1 Protected Resource* – it does not manage user credentials directly but validates tokens issued by Scalekit.

The configuration requires five environment variables, passed through Docker Compose to the `mcp-server` container:

```
SCALEKIT_ENV_URL=https://mcpeduauth.scalekit.dev
SCALEKIT_CLIENT_ID=<client-id>
SCALEKIT_CLIENT_SECRET=<client-secret>
SCALEKIT_EXPECTED_AUDIENCE=https://leowiki-mcp.stream
SCALEKIT_PROTECTED_RESOURCE_METADATA=<json-from-dashboard>
```

Listing 6.1: Scalekit environment variables in docker-compose.yml.

The OAuth 2.1 discovery endpoint (`/.well-known/oauth-protected-resource`) serves metadata that enables AI clients such as Claude Desktop to discover the authorization server automatically. The implementation in `oauth_metadata.py` supports both dashboard-provided metadata (production) and auto-generated fallback metadata (development):

```
metadata = {
    "resource": config.scalekit_mcp_server_id,
    "authorization_servers": [config.scalekit_env_url],
    "bearer_methods_supported": ["header"],
    "resource_signing_alg_values_supported": ["RS256"],
    "scopes_supported": ["mcp:read", "mcp:write"],
}
```

Listing 6.2: OAuth Protected Resource metadata generation (simplified).

The OAuth 2.1 Authorization Code Flow with PKCE is implemented in `auth/oauth_flow.py`. On login initiation, a CSRF-resistant state parameter and a PKCE code verifier are generated using `secrets.token_urlsafe(32)`. The flow redirects to Scalekit's authorization endpoint with `openid profile email` scopes, and the callback endpoint exchanges the authorization code for an access token via the token endpoint.

6.6.2 JWT Token Validation

Every request to a protected MCP endpoint must carry a Bearer token in the `Authorization` header. The `ScalekitAuthMiddleware` validates tokens using the official Scalekit Python SDK:

1. **Bearer token extraction.** The middleware extracts the token from the `Authorization: Bearer <token>` header. Missing or empty tokens result in a 401 response with a `WWW-Authenticate` header pointing to the discovery endpoint.
2. **Token validation via Scalekit SDK.** The SDK's `validate_access_token` method verifies the RS256 signature using JWKS keys fetched from `/.well-known/jwks.json`, validates expiration (`exp`), issuance time (`iat`), issuer (`iss`), and audience (`aud`).
3. **Claims extraction.** On successful validation, the decoded JWT claims are extracted: `sub` (user ID), `email`, `name`, `org_id`, and `roles`.
4. **User state propagation.** The extracted user information is attached to `request.state.user`, making it available to downstream tool handlers.

Public endpoints that bypass authentication include the OAuth discovery endpoint, health checks, the login/callback/logout flow, and the favicon request. This allowlist is maintained in the middleware's `_is_public_endpoint` method.

6.6.3 RBAC to Namespace Mapping

The Role-Based Access Control (RBAC) model maps three Scalekit roles to DokuWiki namespace access levels:

Table 6.16: RBAC role hierarchy and namespace access. Each role inherits access from all lower roles.

Role	Content	Tools	Resources
student	Public pages (no `teacher:`)	`search_content_student`	None
teacher	All pages incl. `teacher:`	+ `search_content_teacher`	None
admin	All pages, all namespaces	+ stats tools	All

The role hierarchy is encoded in the `ROLE_ACCESS_LEVELS` dictionary:

```
ROLE_ACCESS_LEVELS = {
    "student": ["student"],
    "teacher": ["student", "teacher"],
    "admin":   ["student", "teacher", "admin"],
}
```

Listing 6.3: RBAC role hierarchy for search filtering.

Defense-in-Depth: Separate Tools. The two-tool RBAC separation (cf. Section 5.6) is reflected in the Qdrant query layer. The student tool applies namespace filtering on the retrieval results, while the teacher tool returns results from all namespaces. This architectural separation is resilient against prompt-injection attacks at the RBAC level: even if a malicious prompt instructs the AI client to "ignore role restrictions," the MCP server exposes only `search_content_student` to student-authenticated sessions. The `search_content_teacher` tool is structurally unavailable – it is never listed in the tool catalog sent to the client, so neither the LLM nor the user can invoke it regardless of prompt content.

The student search tool employs a dynamic oversampling strategy to compensate for filtered results: it fetches 3× the requested result limit from Qdrant, filters out `teacher:` namespace results in Python, and doubles the fetch limit iteratively until the requested number of student-accessible results is reached (with a safety cap at 10× the limit).

The implementation filters on the `source` URL field (which contains the DokuWiki namespace prefix, e.g. `teacher:exams/java`) rather than on the `access_level` metadata field. This is a deliberate design choice: `access_level` is itself derived from the namespace prefix during preprocessing (`determine_access_level()` assigns `teacher_only` to pages whose namespace starts with `teacher:` or `lehrer:`), so filtering on `"teacher:"` $\notin$ `source` is functionally equivalent to filtering on `access_level` $\neq$ `"teacher_only"` for the current LeoWiki corpus. The Python-level post-filter provides an additional safety layer: teacher content is excluded from the response *before* it reaches the LLM context window, independently of the vector database's filter configuration. The `get_access_filter()` function and `ROLE_ACCESS_LEVELS` dictionary are prepared as infrastructure for a future migration to Qdrant-native filtering, which would reduce over-fetching when the role model expands beyond the current binary student/teacher split.

6.6.4 Security Testing

The RBAC implementation is verified through 30 collected deterministic test cases in `test_rbac_deterministic.py` (19 test functions; one parametrized function contributes 12 role–tool matrix cases). The tests are organized into seven test classes:

1. **TestStudentPermissions** (3 tests): Verifies student tool access, blocked tools, and no resource access.
2. **TestTeacherPermissions** (3 tests): Verifies teacher tool access beyond the student set and blocked admin tools.
3. **TestAdminPermissions** (2 tests): Verifies that admins can access all tools and resources.
4. **TestSecurityIsolation** (4 tests): Verifies privilege escalation prevention and unknown or empty roles.
5. **TestPermissionMatrix** (1 parametrized test, 12 cases): Exercises the full role–tool permission matrix.
6. **TestRBACEnforcementLogic** (3 tests): Verifies tool filtering and unauthorized tool calls.
7. **TestContentAccessLevels** (3 tests): Verifies namespace-level content visibility per role.

Additionally, 4 JWT validation tests in `test_jwt_validation.py` verify token handling: valid token acceptance, expired token rejection, invalid audience rejection, and malformed token rejection. The tests use mocked JWT tokens generated with PyJWT v2.8.x.

All RBAC tests are deterministic – they require no external services and execute in under 1 second, making them suitable for CI integration.

6.7 Deployment

The deployment architecture targets a Raspberry Pi 5 (8 GB RAM, ARM64) as the production host, chosen for its low power consumption ($\sim$5 W idle), continuous availability in the school network, and zero licensing costs.

6.7.1 Docker Containerization

The MCP server is packaged as a Docker image based on `python:3.13-slim`. The Dockerfile follows Docker layer caching best practices:

```
FROM python:3.13-slim
WORKDIR /app
RUN apt-get update && apt-get install -y gcc g++ curl \
    && rm -rf /var/lib/apt/lists/*
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY src/ ./src/
COPY main.py .
RUN mkdir -p data/incoming data/processed data/failed
ENV PYTHONUNBUFFERED=1
HEALTHCHECK --interval=30s --timeout=10s --start-period=40s \
    CMD curl -f http://localhost:8000/health || exit 1
EXPOSE 8000
CMD ["python", "main.py", "--http"]
```

Listing 6.4: MCP server Dockerfile (simplified).

Key design decisions:

- `requirements.txt` is copied before application code to maximize layer cache hits during development.
- The `HEALTHCHECK` directive enables Docker's native health monitoring with a 40-second start-up grace period.
- The `-http` flag starts the server in HTTP Streamable mode (as opposed to stdio), appropriate for production behind a reverse proxy.

6.7.2 Docker Compose Architecture

The production deployment consists of four services orchestrated via Docker Compose:

Table 6.17: Docker Compose service architecture for the production deployment.

Service	Image	Network	Responsibility
caddy	`caddy:2.9.1-alpine`	frontend	TLS termination, reverse proxy
mcp-server	Python 3.13 (custom)	both	MCP protocol, search, RBAC
qdrant	`qdrant/qdrant:v1.14.1`	backend	Vector storage, HNSW, search
watchdog	Python 3.13 (custom)	backend	JSONL auto-ingestion

Network Isolation. The container architecture is shown in Figure 4.4 (Section 4.5.1). The architecture enforces network segmentation through two Docker bridge networks:

- **frontend-network:** Connects Caddy to the MCP server. Only Caddy exposes ports to the host (80, 443, 443/udp for HTTP/3).
- **backend-network:** Connects the MCP server and Watchdog to Qdrant. Qdrant's HTTP dashboard (port 6333) is bound to `127.0.0.1` only, accessible via SSH tunnel for administration.

The MCP server has no externally exposed ports – all client traffic passes through Caddy's TLS termination. Bearer tokens are therefore never transmitted in cleartext.

Caddy Reverse Proxy. Caddy provides automatic TLS certificate management via Let's Encrypt for the domain `leowiki-mcp.stream`. The Caddyfile configures:

- Automatic HTTPS with HTTP-to-HTTPS redirect.
- Reverse proxy to `mcp-server:8000` for all MCP traffic.
- HTTP/3 support via QUIC (UDP port 443).

Watchdog Service. The Watchdog container monitors the `data/incoming/` directory for new JSONL files. When a file is detected (e.g., after a pipeline run copies the output via SCP), the Watchdog:

1. Clears the existing Qdrant collection (`CLEAR_COLLECTION_BEFORE_INGEST=true`).
2. Ingests the new JSONL data with pre-computed embeddings.
3. Moves the processed file to `data/processed/`.

Content updates therefore do not require a manual restart of the MCP stack: the pipeline operator runs the offline pipeline on a development machine and transfers the resulting JSONL file to the Raspberry Pi via SCP; the Watchdog ingest replaces the collection in place.

6.7.3 Deployment to Raspberry Pi

The deployment target is a Raspberry Pi 5 with 8 GB RAM running Raspberry Pi OS (Debian Bookworm, ARM64; 2025-11-19 image). The deployment process follows the scripts in `pipeline/05_deploy/`:

1. **Prerequisites:** Docker and Docker Compose are installed on the Pi. The domain `leowiki-mcp.stream` is configured with an A record pointing to the school's public IP.
2. **File transfer:** The pipeline output (`educational_content.jsonl`) is transferred via `scp` to the Pi's `data/incoming/` directory.
3. **Service start:** `docker compose up -d` launches all four services. Caddy automatically provisions TLS certificates on first start.
4. **Verification:** The health endpoint (`https://leowiki-mcp.stream/health`) confirms service availability. The Qdrant dashboard (via SSH tunnel to `localhost:6333`) confirms data ingestion.

Resource consumption on the Raspberry Pi 5 in steady state (snapshot from `docker stats` under idle load on 2026-02-26):

- **Memory:** ∼1.2 GB total (Qdrant: ∼600 MB with 6,002 vectors; MCP server: ∼200 MB; Caddy: ∼50 MB; Watchdog: ∼80 MB).
- **CPU:** <5 % average (ARM Cortex-A76, 4 cores at 2.4 GHz). Spikes during embedding generation for incoming queries (delegated to OpenAI API).
- **Storage:** ∼2 GB for Docker images and volumes (Qdrant data: ∼500 MB).

6.7.4 npm Package Decision

The original ABA scope specified investigating deployment via both Docker and npm. After evaluation, npm distribution was deemed **not suitable** for this project—a deliberate, reasoned deviation from the original ABA scope:

1. **Technology mismatch:** The MCP server is a Python application. npm is the Node.js package manager; distributing a Python project via npm would require users to install a separate Python runtime, contradicting the simplicity goal.

2. **Unresolvable backend dependency:** The server requires a Qdrant vector database. An npm package cannot bundle or provision a database service; users would still need to run Qdrant separately.

3. **Docker Compose already solves the problem:** A single `docker compose up -d` command provisions all four services (Caddy, MCP server, Qdrant, Watchdog) with correct networking, volumes, and restart policies.

This scope decision was discussed with the thesis supervisor. It is consistent with the MCP ecosystem, where the majority of production MCP servers use Docker for deployment. The `package.json` contains metadata for the project but is not published as an installable npm package.

6.8 Embedding and Retrieval Testing

The test suite for the embedding and retrieval components comprises 98 test functions across 20 test files, organized into two tiers: 88 deterministic unit tests in 10 files (Table 6.18) and 10 integration tests requiring live dependencies in 10 files. These counts cover only the embedding and retrieval codebase; the MCP server maintains a separate test suite (Section 5.11). Additionally, five evaluation scripts implement the quantitative analyses.

6.8.1 Test Architecture

All tests use the FastMCP In-Memory Transport pattern [34]: the test creates a server instance and communicates with it through an in-process transport, avoiding network overhead and external dependencies. With this pattern, tests typically finish in under 100 ms per test function and can run in CI environments without Docker.

```
from fastmcp import FastMCP, Client

async def test_search_tool_exists():
    mcp = FastMCP("test-server")
    register_search_tools(mcp)
    async with Client(mcp) as client:
        tools = await client.list_tools()
        assert any(t.name == "search_content_student"
                   for t in tools)
```

Listing 6.5: FastMCP In-Memory Transport test pattern (simplified).

6.8.2 Unit Tests

The deterministic test suite covers three domains:

Table 6.18: Deterministic test coverage summary.

Test File	Domain	Tests	Scope
`test_tools_deterministic`	MCP tools/resources/prompts	22	Tool registration, annotation correctness, schema validation
`test_rbac_deterministic`	RBAC permissions	19	Role isolation, privilege escalation, unknown roles
`test_query_logger`	Audit logging	11	Log persistence, statistics aggregation, rotation
`test_rbac_tools`	RBAC tool access	7	Tool-level access control enforcement
`test_mcp_resources`	MCP resources	7	Resource registration and content serving
`test_mcp_prompts`	MCP prompts	6	Prompt template rendering
`test_mcp_tools`	MCP tool invocation	5	Tool argument validation, error handling
`test_jwt_validation`	JWT processing	4	Token validation, expiry, audience, malformed tokens
`test_server_endpoints`	HTTP endpoints	4	Health check, OAuth metadata, CORS
`test_server`	Server lifecycle	3	Startup, shutdown, configuration loading

All 88 deterministic tests require no external services and execute in under 5 seconds total.

6.8.3 Integration Tests

Integration tests verify the system against live external dependencies:

- **`test_search_live.py`** – Executes a search query against a running Qdrant instance and verifies that results contain expected payload fields (`title`, `text`, `source`).

- **test_ingestion_live.py** – Ingests a test JSONL file into Qdrant and verifies point count and payload integrity.
- **test_mcp_http_streamable.py** – Starts the MCP server in HTTP Streamable mode and sends a complete JSON-RPC 2.0 request/response cycle via HTTP POST.
- **test_stdio_with_new_data.py** – Tests the stdio transport with freshly ingested data.

Integration tests are marked with `@pytest.mark.integration` and excluded from the default test run (`pytest -m "not integration"`).

6.8.4 Evaluation Framework

The evaluation scripts in `evaluation/scripts/` implement the quantitative analyses reported in Sections 6.1–6.5:

- **eval_keyword_baseline.py** – Compares DokuWiki keyword search with semantic search (78 queries, 3 configurations).
- **eval_model_comparison.py** – Benchmarks five embedding models on the LeoWiki corpus.
- **eval_chunk_size.py** – Tests three chunk sizes (256, 512, and 1024 tokens).
- **eval_hybrid_vs_dense.py** – Compares pure dense retrieval with hybrid search (RRF fusion).
- **eval_pipeline.py** – Orchestrates the full evaluation workflow: retrieval, metrics, RAGAS, statistical analysis, and report generation.

All evaluation scripts use the shared ground-truth dataset of 78 question-answer pairs (`evaluation/ground_truth/`), ensuring cross-evaluation comparability.

6.8.5 Benchmark Reproducibility

To satisfy the non-functional requirement NFR-3 (Reproducibility), all evaluations document:

- **Hardware:** NVIDIA RTX 4080 Laptop GPU (12 GB VRAM), AMD Ryzen 9, 32 GB RAM.

- **Software:** Python 3.11, Qdrant v1.14.1, sentence-transformers 3.3, tiktoken 0.5. The MCP server runs Python 3.13 with FastMCP 3.0.
- **Corpus version:** LeoWiki snapshot date, page count, chunk count.
- **Configuration:** Embedding model name, chunk size, overlap, Qdrant distance metric.
- **Raw results:** All evaluation outputs are stored as JSON files in `evaluation/results/` with timestamps and configuration hashes.

The CI pipeline (`.github/workflows/ci.yml`) runs the deterministic test suite on every push, ensuring that refactoring does not break existing functionality.

7 Results and Conclusion

7.1 Achieved Results

For each question, the evaluation methodology, key metrics, and principal findings are referenced with pointers to the detailed analyses in Chapters 5 and 6.

7.1.1 FF1 – Semantic Search vs. Keyword Search

Research question FF1 asked: *How effectively can a semantic search based on sentence embeddings replace keyword-based search in an educational wiki (LeoWiki)?*

The evaluation (Section 6.5) compared the DokuWiki keyword search with the embedding-based retrieval system on a shared ground-truth dataset of 78 question-answer pairs across 12 of the wiki's 23 namespaces.

Table 7.1: FF1 summary: Keyword search vs. semantic search on 78 test queries.

Method	**MRR**	**P@10 / P@5**[a]	**Hit Rate**
DokuWiki keyword (full question)	0.051	0.010	5.1 %
DokuWiki keyword (optimal keywords)	0.365	0.074	37.2 %
Semantic search (Octen-4B)	0.883	0.185	94.9 %
Semantic search (OpenAI, production)	0.872	0.192	97.4 %

[a] DokuWiki rows: Precision@10; semantic rows: P@5.

The semantic search with the production embedding model (OpenAI `text-embedding-3-large`) achieves a **17-fold MRR improvement** over the keyword search with full natural-language questions (0.872 vs. 0.051) and a **2.4-fold improvement** over manually optimized keywords (0.872 vs. 0.365). The hit rate rises from 5.1 % to 97.4 %, meaning that in the production system, 76 out of 78 test queries successfully retrieve a relevant wiki page in the top-10 results.

The four root causes of the keyword search's failure – AND logic combined with stopword filtering, missing semantic bridging, absent synonym handling, and unresolved German compound words – are avoided by the embedding-based approach (cf. Section 6.5.3).

FF1: Semantic search improves MRR by a factor of 17 over keyword search for the LeoWiki corpus, raising the hit rate from 5.1 % to 97.4 %.

7.1.2 FF2 – MCP as Integration Standard

FF2 addresses the suitability of the Model Context Protocol as a standardized interface: *How suitable is the Model Context Protocol (MCP) as an interface between a specialized search server and heterogeneous AI applications?* The evaluation below also notes practical limitations observed during interoperability testing.

Section 5.10 presents a compatibility matrix of five major AI clients and practical interoperability tests with three selected clients (Table 5.7).

Key findings:

1. **Tool support is consistent across compatible clients.** All three tested clients invoked MCP tools successfully in the manual test protocol—the primary mechanism through which the LeoWiki server exposes its search functionality. Google Gemini and Microsoft Copilot (consumer) were surveyed but do not support remote MCP with OAuth 2.1 in their consumer products (as of February 2026).

2. **Dual-transport coverage is sufficient.** The two transports – stdio for local desktop clients and HTTP Streamable for web-based clients – cover all tested deployment scenarios. The LeoWiki server implements both.

3. **Resources and Prompts have fragmented support.** Not all clients support Resources or Prompts natively. The `ResourcesAsTools` and `PromptsAsTools` transforms in FastMCP 3.0 mitigate this by exposing all primitives as tools for clients with limited feature sets.

4. **OAuth 2.1 authentication** is supported by three of the five tested clients (Claude, ChatGPT, Mistral), enabling standardized token-based access for the target audience.

5. **Transport latency meets NFR-1.** Benchmarks on the production hardware measured a mean end-to-end search latency of ∼226 ms via stdio and ∼321 ms via HTTP (Section 5.3.4), both within the 2 s response-time target. The stdio transport adds 6.3× less protocol overhead than HTTP (17.9 ms vs. 113.1 ms).

Limitations observed include the rapid specification evolution (three revisions within one year), fragmented advanced feature support (no client implements all features), and the necessity of dual-transport implementation on the server side.

Table 7.2: FF2 summary: MCP client compatibility (February 2026).

Client	**Remote MCP**	**OAuth 2.1**	**Tool Support**
Claude	✓	✓	✓
ChatGPT	✓	✓	✓
Mistral Le Chat	✓	✓	✓
Gemini	×	×	—
Copilot	×	×	—

FF2: MCP proved suitable as an integration interface for knowledge sources in the tested configuration. The identified limitations – fragmented advanced features, rapid specification evolution – are manageable for the LeoWiki use case but warrant monitoring.

7.1.3 FF3 – Embedding Model Selection for German

Research question FF3 asked: *Which sentence embedding model achieves the best retrieval quality for German-language educational content on resource-limited hardware?*

The embedding model comparison (FF3) tested five models on the full LeoWiki corpus (10,841 chunks, 78 ground-truth questions). Table 7.3 summarizes the results from Section 6.1:

Table 7.3: FF3 summary: Embedding model comparison on LeoWiki corpus.

Model	**MRR**	**NDCG@10**	**Hit Rate**	**Params**
Octen-Embedding-4B	**0.883**	**0.899**	94.9 %	4.0B
text-embedding-3-large	0.872	0.897	**97.4 %**	API
PIXIE-Rune-v1.0	0.802	0.823	88.5 %	568M
snowflake-arctic-embed-l-v2.0	0.788	0.822	92.3 %	568M
bge-m3-unsupervised	0.756	0.800	93.6 %	568M

As detailed in Section 6.1, models divide into two tiers: Tier 1 (Octen-4B, OpenAI) exceeds MRR 0.87, while 568M-parameter models remain in the 0.76–0.80 range, consistent with the MTEB leaderboard as a pre-screening heuristic for German-language retrieval tasks.

For production deployment on a Raspberry Pi 5 (no GPU), OpenAI `text-embedding-3-large` was selected as the pragmatic choice: the 1.1 percentage point MRR difference to Octen-4B is negligible, while the deployment complexity is substantially reduced (cf. Section 6.1.4).

FF3: The five-model comparison identifies Octen-Embedding-4B as the quality leader for this corpus and documents the engineering trade-off that led to OpenAI as the production model.

7.2 Limitations

7.2.1 Technical Limitations

1. **No real-time index updates.** The system relies on a periodically executed offline pipeline. Changes to wiki content are not reflected in search results until the pipeline is re-run. For a school wiki with relatively infrequent content changes, this is acceptable; for wikis with high update frequency, a real-time ingestion mechanism would be required.

2. **Single-model evaluation environment.** All embedding evaluations were conducted on a single GPU (NVIDIA RTX 4080 Laptop, 12 GB VRAM). Inference timing results may not generalize to other hardware configurations.

3. **Ground-truth sample size.** The 78 question-answer pairs permit pairwise non-parametric tests (Wilcoxon signed-rank) and bootstrap confidence intervals; all direct comparisons in Chapter 6 include these. However, the sample size limits statistical power for small effects: at $n = 78$ paired observations a Wilcoxon test achieves approximately 50–60 % power for Cohen's $d \approx 0.2$. Non-significant results (e.g., chunk-size comparisons, $p > 0.06$) should therefore be read as *inconclusive rather than as evidence of equivalence*; they are reported with this caveat in Section 6.2.3.

4. **AI-generated ground-truth bias.** The 78 question-answer pairs were generated by Claude Opus 4.6 and manually verified by both authors (Section 6.1.2). AI-generated questions may systematically favor paraphrase-style formulations that embedding models handle well, potentially inflating absolute MRR values for semantic search. The manual verification protocol eliminates factual errors but does not fully address this distributional bias. Because the *same* question set was

used for both keyword and semantic evaluation, the *relative* comparison (17-fold MRR gap) remains valid even if absolute values are optimistic.

5. **Raspberry Pi resource constraints.** The deployment target (8 GB RAM, ARM64, no GPU) precludes local embedding model inference. The system depends on the OpenAI API for embedding generation during pipeline execution.

6. **BM25 tokenizer simplicity.** The hybrid search evaluation used a simple whitespace tokenizer without lemmatization or compound decomposition. A more sophisticated German-aware tokenizer might yield different hybrid search results.

7.2.2 Scope Limitations

1. **Single wiki instance.** The system was developed and evaluated exclusively on the HTL Leonding LeoWiki. Transferability to other DokuWiki instances or wiki platforms has not been empirically verified.

2. **No integration with Leonidas/Leonie.** The SYP projects Leonidas and Leonie are organizationally separate and not part of this diploma thesis.

3. **No continuous deployment (CD).** Automated deployment to the Raspberry Pi was not implemented; production updates rely on manual file transfer. A continuous integration workflow (GitHub Actions) runs lint, type checks, and tests on every push (Section 5.11.3).

4. **Query-time data transmission.** The production system sends user queries to the OpenAI Embeddings API for vectorization at search time. Only the short query text is transmitted; the wiki content itself is embedded offline during pipeline execution and stored locally in Qdrant. In the school context, HTL Leonding already uses cloud services (Microsoft 365) for student data, and the transmitted queries contain no personal data beyond the search terms. For deployments with stricter data sovereignty requirements, the local embedding path via Octen-Embedding-4B (Section 6.1.4) eliminates external API dependencies entirely.

5. **OAuth2 scope.** The Scalekit integration covers the Authorization Code Flow with PKCE. Token refresh automation and session management are implemented but not extensively load-tested.

6. **npm distribution deviation.** The originally planned investigation of npm as a deployment channel for the Python MCP server stack was completed; npm distri-

bution was deemed unsuitable for this project (see Section 6.7.4)—a deliberate, documented deviation from the initial ABA scope.

7.3 Lessons Learned

7.3.1 Technical Insights

1. **Dense retrieval outperformed hybrid search on the LeoWiki corpus.** The evaluation of hybrid search (Section 6.4) showed that adding BM25 to dense retrieval *reduced* search quality for the LeoWiki corpus. The BM25 component provided redundant and occasionally conflicting signals, confirming that for a domain-specific corpus of ∼10,000 chunks, a domain-appropriate embedding model alone achieved higher MRR than the hybrid setup. This finding is consistent with the broader RAG literature noting that hybrid approaches can benefit large, heterogeneous corpora [53], whereas a domain-specific, single-language wiki corpus did not profit from the additional BM25 signal.

2. **MTEB leaderboard rankings transfer to domain-specific tasks.** The correlation between MTEB RTEB(deu) rankings and actual performance on LeoWiki content is consistent with using the leaderboard as a practical selection heuristic for German-language retrieval tasks.

3. **Two-tool separation provides defense-in-depth for access control.** Implementing two physically separate search tools (`search_content_student` and `search_content_teacher`) instead of a single tool with a role parameter avoids LLM parameter-manipulation attacks by design. The RBAC enforcement operates at the tool level before any search logic executes. This was verified through 30 deterministic test cases (Section 6.6.4); broader adversarial testing against prompt-injection attacks remains as future work.

4. **512-token chunks are optimal for structured wiki content.** The chunk-size evaluation revealed a concave pattern ($256 < 512 > 1024$): 256 tokens lose context, 1024 tokens dilute relevance, and 512 tokens provide the best balance for this corpus (MRR 0.876 in the chunk-size experiment, Section 6.2.3), consistent with trade-offs documented in RAG practitioner guides [51].

7.3.2 Process Insights

1. **Evaluation-first methodology saved rework.** Constructing the ground-truth dataset early and reusing it across all evaluations ensured consistent baselines and eliminated confounding variables from differing test sets.

2. **The MCP specification evolves rapidly.** Three specification revisions within one year (2024-11-05, 2025-03-26, 2025-11-25) required continuous adaptation of the server implementation. FastMCP 3.0 absorbed most breaking changes, but the transport protocol migration from HTTP+SSE to Streamable HTTP necessitated structural changes.

3. **AI-assisted content generation requires rigorous verification.** The ground-truth dataset was generated using Claude Opus 4.6 and then manually verified. Several generated questions contained factual errors or referenced non-existent wiki pages, underscoring the necessity of human oversight for AI-generated evaluation data.

7.4 Future Work

The MCP-based architecture developed in this thesis can be extended along several dimensions:

Real-time content synchronization. DokuWiki provides action hooks (`COMMON_SAVE_AFTER`) that fire when a page is saved. A DokuWiki plugin could trigger incremental re-embedding of modified pages, eliminating the need for full pipeline re-runs. The Qdrant upsert API already supports point-level updates, making the database side of this extension straightforward.

Multi-wiki federation. The architecture separates the protocol layer (MCP server) from the content layer (embedding and retrieval). By replacing only the wiki connector component (Stage 1 of the pipeline), the system can be adapted to other wiki platforms (MediaWiki, Confluence) or document management systems. Multiple wiki sources could be served through a single MCP server with source-specific namespace filtering.

Enterprise identity integration with Microsoft Entra. For school-wide deployments where students and teachers already authenticate through Microsoft 365, integrating Microsoft Entra ID as an external identity provider would enable Single Sign-On for

the MCP server without requiring a separate credential store. Entra supports both SAML and OpenID Connect [58], the latter aligning directly with the OAuth 2.1 flow already prepared in the server's authentication middleware (Section 5.4). Combined with Entra's Conditional Access policies and group-based role assignments, the existing RBAC middleware could derive user roles automatically from directory attributes, ensuring that access is denied at the next token refresh cycle when an account is deactivated. A bridging layer such as Scalekit [59] could further simplify the token exchange between Entra and the MCP server, reducing much of the integration effort to configuration rather than code changes.

Qdrant-native access control filtering. The current student search tool filters on the `source` URL namespace prefix in Python (Section 6.6.4). Migrating to the prepared `get_access_filter()` function, which uses Qdrant's native `access_level` field condition, would eliminate the over-fetching overhead of the dynamic oversampling strategy and is recommended when the role model expands beyond the current binary student/teacher split.

Fine-tuning on school corpus. The current system uses general-purpose embedding models. Fine-tuning a model on the specific vocabulary and query patterns of HTL Leonding (using the 78 ground-truth pairs as a seed dataset) could further improve retrieval quality for domain-specific queries.

Local embedding inference. As ARM-based single-board computers gain GPU capabilities (e.g., NVIDIA Jetson), running Octen-Embedding-4B locally on the deployment target would eliminate the dependency on the OpenAI API and resolve the data privacy trade-off documented in Section 6.1.4.

7.5 Conclusion

This diploma thesis addressed a specific problem: the internal wiki of HTL Leonding contains organizational knowledge that its users cannot find. The built-in keyword search achieves a hit rate of only 5.1 % for natural-language queries – a direct consequence of the vocabulary mismatch between how users ask questions and how content is indexed.

The developed system – comprising a five-stage offline pipeline, a vector database with 3,072-dimensional embeddings, and an MCP server with dual transport support

– raises this hit rate to 97.4 % (MRR 0.872). Measured end-to-end search latency averages ∼321 ms via the production HTTP transport, well within the 2 s response-time requirement (NFR-1). The server is compatible with the three major AI clients tested (Claude, ChatGPT, Mistral Le Chat) and enforces role-based access control through OAuth 2.1 (draft-ietf-oauth-v2-1-12 at time of submission; cf. Section 2.3.1) with Scalekit, ensuring that students cannot access teacher-only content.

The three research questions have been answered:

- **FF1:** The large MRR gap versus keyword search matches the vocabulary-mismatch analysis in Section 6.5.3: embeddings remove the need for hand-crafted query terms.
- **FF2:** MCP covered the tested clients without protocol forks; remaining friction is uneven Resources/Prompts support, not transport-level incompatibility.
- **FF3:** Quality ranking (Octen-4B vs. OpenAI) is clear, but deployment constraints on the Pi invert the recommendation toward the API model despite a small MRR delta.

The system is deployed as a Docker Compose stack on a Raspberry Pi 5, accessible via the domain `leowiki-mcp.stream` with automatic TLS certificate management. The entire LeoWiki corpus of 196 pages and 320 media files is indexed at a one-time cost of $0.14.

Beyond the immediate application, this work shows that text embeddings indexed in a vector database and exposed through the Model Context Protocol can make a legacy knowledge system accessible via AI interfaces at low cost and on modest hardware. Whether this approach transfers to other knowledge systems would require verification in those specific contexts.

Glossary

ACL Access Control List

API Application Programming Interface

ARM Advanced RISC Machines

ASGI Asynchronous Server Gateway Interface

BM25 Best Matching 25

Caddy A web server with automatic HTTPS certificate management via Let's Encrypt, used as the reverse proxy in the production Docker Compose deployment

CI Continuous Integration

CLI Command-Line Interface

DokuWiki An open-source wiki software that uses plain text files for storage instead of a relational database. Used as the platform for LeoWiki at HTL Leonding

FastAPI A modern Python web framework for building APIs, used as the HTTP layer underlying the MCP server's Streamable HTTP transport

FastMCP A Python framework for implementing MCP-compliant servers with decorator-based tool/resource/prompt registration and support for stdio and HTTP Streamable transports

GDPR General Data Protection Regulation

ground truth A manually verified dataset of question-answer pairs used as the reference for evaluating retrieval quality. In this thesis: 78 Q&A pairs covering 12 LeoWiki namespaces

HNSW Hierarchical Navigable Small World

IDE Integrated Development Environment

IDF Inverse Document Frequency

JSON-RPC JSON Remote Procedure Call

JWKS JSON Web Key Set

JWT JSON Web Token

LeoWiki The internal DokuWiki instance of HTL Leonding, containing 196 pages and 320 media files across 23 namespaces (12 covered by the evaluation ground truth)

MCP Model Context Protocol

MRR Mean Reciprocal Rank

MTEB Massive Text Embedding Benchmark

NDCG Normalized Discounted Cumulative Gain

NLP Natural Language Processing

OAuth Open Authorization

OIDC OpenID Connect

PKCE Proof Key for Code Exchange

Qdrant A Rust-based open-source vector database optimized for similarity search with support for payload filtering, HNSW indexing, and ARM64 deployment

RAG Retrieval-Augmented Generation

RBAC Role-Based Access Control

RRF Reciprocal Rank Fusion

RTEB Retrieval Text Embedding Benchmark

SBERT Sentence-BERT

Scalekit A managed identity provider supporting OAuth 2.1, OIDC, and SAML, used to authenticate users and provide role-based access tokens for the MCP server

SDK Software Development Kit

SSE Server-Sent Events

text embedding A dense vector representation of text in a high-dimensional space where semantic similarity corresponds to geometric proximity

TLS Transport Layer Security

VRAM Video Random Access Memory

Bibliography

[1] DokuWiki Community. (2024) DokuWiki – the wiki that does not require a database. [Online]. Available: https://www.dokuwiki.org/

[2] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008. [Online]. Available: https://nlp.stanford.edu/IR-book/

[3] Anthropic. (2024) Model context protocol specification. [Online]. Available: https://spec.modelcontextprotocol.io/

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

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, arXiv:1301.3781. [Online]. Available: https://arxiv.org/abs/1301.3781

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019, pp. 4171–4186. [Online]. Available: https://arxiv.org/abs/1810.04805

[8] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 3982–3992. [Online]. Available: https://arxiv.org/abs/1908.10084

[9] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2020. [Online]. Available: https://arxiv.org/abs/1603.09320

[10] OpenAI. (2024) Embeddings – OpenAI API documentation. [Online]. Available: https://platform.openai.com/docs/guides/embeddings

[11] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu. (2024) BGE M3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. ArXiv:2402.03216. [Online]. Available: https://arxiv.org/abs/2402.03216

[12] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "MTEB: Massive text embedding benchmark," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2023, pp. 2006–2029. [Online]. Available: https://arxiv.org/abs/2210.07316

[13] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," *Advances in Neural Information Processing Systems*, vol. 33, 2020. [Online]. Available: https://arxiv.org/abs/2005.11401

[14] A. Conneau and G. Lample, "Cross-lingual language model pretraining," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019. [Online]. Available: https://arxiv.org/abs/1901.07291

[15] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020, pp. 8440–8451. [Online]. Available: https://arxiv.org/abs/1911.02116

[16] N. Reimers. (2024) Sentence-transformers documentation. [Online]. Available: https://www.sbert.net/

[17] JSON-RPC Working Group. (2010) JSON-RPC 2.0 specification. [Online]. Available: https://www.jsonrpc.org/specification

[18] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[19] GraphQL Foundation. (2021) GraphQL specification – october 2021 edition. [Online]. Available: https://spec.graphql.org/October2021/

[20] OASIS Open, *OData Version 4.01 – OASIS Standard*, 2022. [Online]. Available: https://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-protocol.html

[21] Anthropic. (2024) Introducing the model context protocol. [Online]. Available: https://www.anthropic.com/news/model-context-protocol

[22] D. Hardt, "The OAuth 2.0 authorization framework," IETF, RFC 6749, 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749

[23] M. Jones and D. Hardt, "The OAuth 2.0 authorization framework: Bearer token usage," IETF, RFC 6750, 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6750

[24] D. Hardt, A. Parecki, and T. Lodderstedt, "The OAuth 2.1 Authorization Framework (draft-ietf-oauth-v2-1-12)," IETF Internet-Draft, Oct. 2025, work in progress, accessed 2026-03-20. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/

[25] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. (2014) OpenID connect core 1.0. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html

[26] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996. [Online]. Available: https://doi.org/10.1109/2.485845

[27] Red Hat / Keycloak Project. (2024) Keycloak server administration guide – performance and sizing. Keycloak requires a JVM with at least 512 MB heap; production deployments typically use 1 GB or more. [Online]. Available: https://www.keycloak.org/server/all-config

[28] Caddy Project. (2024) Caddy web server documentation. [Online]. Available: https://caddyserver.com/docs/

[29] Internet Security Research Group. (2024) Let's encrypt – free SSL/TLS certificates. [Online]. Available: https://letsencrypt.org/

[30] Qdrant. (2024) Qdrant documentation. [Online]. Available: https://qdrant.tech/documentation/

[31] Docker Inc. (2024) Docker compose overview. [Online]. Available: https://docs.docker.com/compose/

[32] DokuWiki Community. (2024) DokuWiki XML-RPC API. [Online]. Available: https://www.dokuwiki.org/devel:xmlrpc

[33] OpenAI. (2024) GPT-4o mini – advancing cost-efficient intelligence. Model snapshot: `gpt-4o-mini-2024-07-18`. [Online]. Available: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

[34] J. Cummings. (2025) FastMCP – the fast, pythonic way to build MCP servers. [Online]. Available: https://github.com/jlowin/FastMCP

[35] Docker Inc. (2024) Docker documentation. [Online]. Available: https://docs.docker.com/

[36] S. Ramírez. (2024) FastAPI documentation. [Online]. Available: https://fastapi.tiangolo.com/

[37] Python Software Foundation. (2024) asyncio — asynchronous I/O. [Online]. Available: https://docs.python.org/3/library/asyncio.html

[38] S. Colvin. (2024) Pydantic settings management. [Online]. Available: https://docs.pydantic.dev/latest/concepts/pydantic-settings/

[39] European Parliament and Council of the European Union. (2016) Regulation (EU) 2016/679 (general data protection regulation). Official English-language consolidated text of the GDPR. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj

[40] Google. (2026) MCP servers with the Gemini CLI. Remote MCP server support not yet available (GitHub Issue #983). [Online]. Available: https://google-gemini.github.io/gemini-cli/docs/tools/mcp-server.html

[41] Microsoft. (2025) Model context protocol (MCP) is now generally available in Microsoft Copilot Studio. [Online]. Available: https://www.microsoft.com/en-us/microsoft-copilot/blog/copilot-studio/model-context-protocol-mcp-is-now-generally-available-in-microsoft-copilot-studio/

[42] GitHub. (2025) GitHub Education – free developer tools for students. [Online]. Available: https://education.github.com/

[43] Cursor Community. (2025) Remote MCP server intermittently shows "error" status on startup. GitHub Issue #3907: race condition with aggressive timeouts. [Online]. Available: https://github.com/cursor/cursor/issues/3907

[44] ——. (2025) Remote MCP server "connect" button produces zero network requests (OAuth flow never starts). [Online]. Available: https://forum.cursor.com/t/remote-mcp-server-connect-button-produces-zero-network-requests-oauth-flow-never-starts/150962

[45] J. Wardini. (2026) Google Gemini gains share as ChatGPT declines in Similarweb data. Similarweb Global AI Tracker, January 2026. [Online]. Available: https://www.searchenginejournal.com/google-gemini-gains-share-as-chatgpt-declines-in-similarweb-data/564690/

[46] Octen. (2025) Octen-embedding-4b – HuggingFace model card. 4.0B parameters, 2560 dimensions; Apache-2.0 license; fine-tuned from Qwen/Qwen2.5-3B; model by Octen (https://octen.ai). [Online]. Available: https://huggingface.co/Octen/Octen-Embedding-4B

[47] TelePIX Co., Ltd. (2026) PIXIE-Rune-v1.0 – HuggingFace model card. 0.568B parameters, 1024 dimensions; Apache-2.0 license; arXiv:2601.03496. [Online]. Available: https://huggingface.co/telepix/PIXIE-Rune-v1.0

[48] Snowflake. (2025) snowflake-arctic-embed-l-v2.0 – HuggingFace model card. 0.568B parameters, 1024 dimensions; Apache-2.0 license. [Online]. Available: https://huggingface.co/Snowflake/snowflake-arctic-embed-l-v2.0

[49] Anthropic. (2026) Claude opus 4.6 – model card. Used for ground-truth question–answer pair generation in the evaluation dataset. [Online]. Available: https://docs.anthropic.com/en/docs/about-claude/models

[50] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Transactions on Information Systems*, vol. 20, no. 4, pp. 422–446, 2002. [Online]. Available: https://dl.acm.org/doi/10.1145/582415.582418

[51] LangChain. (2024) Text splitters – LangChain documentation. [Online]. Available: https://python.langchain.com/docs/modules/data_connection/document_transformers/

[52] M. Günther, I. Mohr, D. J. Williams, B. Wang, and H. Xiao, "Late chunking: Contextual chunk embeddings using long-context embedding models," *arXiv preprint arXiv:2409.04701*, 2024. [Online]. Available: https://arxiv.org/abs/2409.04701

[53] D. Mozolevskyi and W. AlShikh, "Comparative analysis of retrieval systems in the real world," *arXiv preprint arXiv:2405.02048*, 2024. [Online]. Available: https://arxiv.org/abs/2405.02048

[54] G. V. Cormack, C. L. A. Clarke, and S. Buettcher, "Reciprocal rank fusion outperforms condorcet and individual rank learning methods," in *Proceedings of the 32nd International ACM SIGIR Conference*, 2009, pp. 758–759. [Online]. Available: https://doi.org/10.1145/1571941.1572114

[55] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009. [Online]. Available: https://doi.org/10.1561/1500000019

[56] DokuWiki Community. (2024) DokuWiki search – fulltext search documentation. [Online]. Available: https://www.dokuwiki.org/search

[57] Scalekit. (2025) Scalekit – MCP authentication documentation. MCP-specific authentication flow documentation, updated 2025. [Online]. Available: https://docs.scalekit.com/

[58] Microsoft. (2024) Openid connect on the microsoft identity platform. [Online]. Available: https://learn.microsoft.com/en-us/entra/identity-platform/v2-protocols-oidc

[59] Scalekit. (2024) Microsoft entra ID single sign-on integration. [Online]. Available: https://docs.scalekit.com/integrations/entra-id/

List of Figures

List of Tables

List of Listings

Appendix

.1 ABA Proposal

The official *Antrag auf Beurteilung der Arbeit* (ABA) was submitted to HTL Leonding in November 2025 and approved by the supervising instructor, Rainer Stropek. The proposal defines three research questions:

FF1 (Shared) How effectively can a semantic search based on sentence embeddings replace keyword-based search in an educational wiki (LeoWiki)?

FF2 (Imre Obermüller) How suitable is the Model Context Protocol (MCP) as an interface between a specialized search server and heterogeneous AI applications?

FF3 (Jan Ritt) Which sentence embedding model achieves the best retrieval quality for German-language educational content on resource-limited hardware?

.1.1 Individual Contributions

Table 1: ABA-defined individual contributions per author.

Author	**Scope**
Imre Obermüller (BIF)	I1: FastMCP framework selection; I2: JSON-RPC 2.0 processing and capability negotiation; I3: Transport protocol implementation (stdio, HTTP Streamable); I4: MCP tool development (5 tools, RBAC-separated); I5: Client compatibility evaluation (5+ hosts)
Jan Ritt (CIFT)	J1: Evaluation setup with ground-truth corpus (78 questions); J2: Embedding model benchmarking (5 models, MTEB validation); J3: DokuWiki parsing and chunking pipeline (5 stages); J4: Qdrant schema design and payload filtering; J5: Retrieval strategy comparison (dense, sparse, hybrid); J6: OAuth2/OIDC integration with Scalekit; J7: Docker Compose deployment on Raspberry Pi; J8: Testing (unit, integration, RBAC)

Table 2: Milestones as defined in the ABA proposal.

Milestone	Deadline	Responsible
MCP vs REST/OData/GraphQL comparison	15 Oct 2025	Shared
Vector DB schema + embedding tests	15 Nov 2025	Jan
MCP server stdio + basic tools	20 Dec 2025	Imre
Semantic search integrated	25 Jan 2026	Shared
HTTP Streamable transport	20 Feb 2026	Imre
OAuth2 integration (Scalekit)	15 Mar 2026	Jan
Docker + npm deployment	30 Mar 2026	Jan
Thesis submission	Apr 2026	Shared

.1.2 Milestones from ABA

Note: The labels J1–J8 in Table 1 follow the ABA numbering from November 2025. The main text (Chapter 6) uses a revised numbering finalized on 2026-02-11; see Section 1.2 for the mapping.

The scanned original ABA form is archived in the thesis repository under `appendix/aba_antrag_scan.pdf`.

.2 Project Plan

.2.1 Milestones

Table 3: Project milestones and actual completion status.

Milestone	Target	Actual	Status
Project kickoff	Oct 2025	Oct 2025	Completed
ABA submission	Nov 2025	Nov 2025	Completed
MCP vs REST/GraphQL research	Oct 2025	Oct 2025	Completed
Vector DB schema design	Nov 2025	Nov 2025	Completed
MCP stdio prototype	Dec 2025	Dec 2025	Completed
Embedding model evaluation	Jan 2026	Jan 2026	Completed
Semantic search integration	Jan 2026	Jan 2026	Completed
HTTP Streamable transport	Feb 2026	Feb 2026	Completed
OAuth2/Scalekit integration	Mar 2026	Mar 2026	Completed
Docker deployment (Raspi)	Mar 2026	Mar 2026	Completed
Thesis documentation	Mar 2026	Mar 2026	Completed
Final review	Apr 2026	Mar 2026	Completed
Submission	Apr 2026	Apr 2026	Completed

.2.2 Work Distribution

Table 4 summarizes the division of labor between the two authors. Shared components were developed collaboratively using pair-programming sessions and code review.

Table 4: Work distribution between Jan Ritt and Imre Obermüller.

Component	Jan Ritt	Imre Oberm.	Shared
Offline pipeline (5 stages)	Primary	Review	
Embedding evaluation	Primary		
MCP server core		Primary	
Transport protocols		Primary	
MCP tools (5)		Primary	
OAuth2/Scalekit	Primary		
RBAC middleware	Primary	Co-author	
Docker/deployment	Primary		
Qdrant integration			Joint
Testing	Unit/RBAC	MCP/transport	
Presentation			Joint
Thesis Ch. 1–4, 7			Joint
Thesis Ch. 5		Primary	
Thesis Ch. 6	Primary		

.2.3 Repository Structure

The project spans multiple Git repositories:

`leowiki_mcp_server` Production MCP server, Docker configuration, tests, and Caddyfile.

`dev_dito` Offline pipeline (5 stages), orchestration scripts, and DokuWiki plugin.

`dev_dito_modules` Reusable Python modules for fetching, parsing, and embedding.

`dev_prompts_instructions_notes` Thesis document source (LaTeX), research notes, prompts archive, and diagram sources.

.3 API Documentation

.3.1 MCP Tool Schemas

The LeoWiki MCP server exposes five tools via JSON-RPC 2.0.

search_content_student

```
{
  "name": "search_content_student",
  "description": "Search LeoWiki content accessible to students",
  "inputSchema": {
    "type": "object",
    "properties": {
      "query": {
        "type": "string",
        "description": "Natural language search query"
      },
      "limit": {
        "type": "integer",
        "default": 5,
        "minimum": 1,
        "maximum": 20,
        "description": "Maximum number of results"
      }
    },
    "required": ["query"]
  }
}
```

Listing 1: Tool schema: search_content_student

search_content_teacher

```
{
  "name": "search_content_teacher",
  "description": "Search LeoWiki with teacher-level access",
  "inputSchema": {
    "type": "object",
    "properties": {
      "query": {
        "type": "string",
        "description": "Natural language search query"
      },
      "limit": {
        "type": "integer",
        "default": 10,
        "maximum": 50
      },
      "namespace": {
        "type": "string",
        "description": "Restrict to a DokuWiki namespace"
      }
    },
    "required": ["query"]
  }
}
```

Listing 2: Tool schema: search_content_teacher

get_collection_stats

```
{
  "name": "get_collection_stats",
  "description": "Retrieve Qdrant collection statistics",
  "inputSchema": {
    "type": "object",
    "properties": {}
  }
}
```

Listing 3: Tool schema: get_collection_stats (admin only)

get_query_statistics

```
{
  "name": "get_query_statistics",
  "description": "Retrieve aggregated query log statistics",
  "inputSchema": {
    "type": "object",
    "properties": {}
  }
}
```

Listing 4: Tool schema: get_query_statistics (admin only)

health_check

```
{
  "name": "health_check",
  "description": "Check server and database connectivity",
  "inputSchema": {
    "type": "object",
    "properties": {}
  }
}
```

Listing 5: Tool schema: health_check (all roles)

.3.2 Response Format

Search tools return results as an array of `SearchResult` objects:

```
{
  "results": [
    {
      "title": "Page title",
      "namespace": "departm:it",
      "content": "Matching chunk text (truncated)...",
      "score": 0.8734,
      "metadata": {
        "page_id": "departm:it:courses:sew",
        "chunk_index": 3,
        "total_chunks": 12,
        "last_modified": "2025-11-20"
      }
    }
  ],
  "query": "original query text",
  "total_results": 5,
  "search_time_ms": 42
}
```

Listing 6: Search result structure

.3.3 Error Codes

The server uses standard JSON-RPC 2.0 error codes plus application-specific extensions:

Table 5: MCP server error codes.

Code	Meaning	Context
−32700	Parse error	Malformed JSON-RPC request
−32600	Invalid request	Missing required fields
−32601	Method not found	Unknown tool name
−32602	Invalid params	Schema validation failure
−32603	Internal error	Qdrant connection failure
−32001	Unauthorized	Missing or invalid JWT
−32003	Forbidden	RBAC role insufficient
−32010	Rate limited	Query rate limit exceeded

Table 6: Required environment variables for the MCP server.

Variable	Description	Example / Default
VECTOR_DB_URL	Qdrant HTTP endpoint	http://qdrant:6333
DEFAULT_COLLECTION	Qdrant collection name	educational_content
HTTP_HOST	Server bind address	0.0.0.0
HTTP_PORT	Server port	8000
LOG_LEVEL	Logging verbosity	INFO
ENABLE_RBAC	Enable role checking	true
DEFAULT_USER_ROLE	Fallback if no JWT	student
ENABLE_AUTH	Enable OAuth2 flow	false
SCALEKIT_ENV_URL	Scalekit environment URL	https://...
SCALEKIT_CLIENT_ID	OAuth2 client ID	(secret)
SCALEKIT_CLIENT_SECRET	OAuth2 client secret	(secret)
OPENAI_API_KEY	OpenAI API key for embeddings	(secret)
EMBEDDING_MODEL	Model identifier	text-embedding-3-large

.4 Configuration Reference

.4.1 Environment Variables

.4.2 Docker Compose Configuration

The production `docker-compose.yml` defines four services across two isolated networks. Listing 7 shows the full configuration deployed on the Raspberry Pi at `https://leowiki-mcp.stream`.

```
services:
  qdrant:
    image: qdrant/qdrant:v1.14.1
    container_name: mcp-qdrant
    restart: unless-stopped
    ports:
      - "127.0.0.1:6333:6333"
    expose:
      - "6334"
    volumes:
      - qdrant_data:/qdrant/storage
    networks:
      - backend-network

  mcp-server:
    build: .
    container_name: mcp-server
    restart: unless-stopped
    expose:
      - "8000"
    depends_on:
      qdrant:
        condition: service_started
    environment:
      - VECTOR_DB_URL=http://qdrant:6333
      - ENABLE_RBAC=true
      - EMBEDDING_MODEL=text-embedding-3-large
    networks:
      - frontend-network
      - backend-network

  watchdog:
    build:
      context: .
      dockerfile: Dockerfile.watchdog
    container_name: mcp-watchdog
    restart: unless-stopped
    environment:
      - CLEAR_COLLECTION_BEFORE_INGEST=true
    networks:
      - backend-network

  caddy:
    image: caddy:2.9.1-alpine
    container_name: mcp-caddy
    restart: unless-stopped
```

.4.3 Caddyfile

Caddy is configured to automatically obtain Let's Encrypt TLS certificates and proxy all traffic to the MCP server:

```
leowiki-mcp.stream {
    reverse_proxy mcp-server:8000 {
        flush_interval -1
    }
    header {
        Strict-Transport-Security "max-age=31536000; includeSubDomains"
        X-Content-Type-Options "nosniff"
        X-Frame-Options "DENY"
    }
}
```

Listing 8: Caddyfile for reverse proxy

.5 Test Protocols

.5.1 Golden Query Catalog

The golden query catalog contains curated test queries derived from real LeoWiki content. Each query has associated *gold_contains* keywords that the expected result must include. Table 7 lists the eight golden queries used to evaluate search quality, relevance, and RBAC filtering.

Expected keywords per query. Each golden query defines a set of keywords that must appear in a relevant result:

- **GQ01:** Diplomarbeit, Format, Word
- **GQ02:** Matura, Klausur, Einlass
- **GQ03:** Office365, Mail, Account
- **GQ04:** Exkursion, organisieren, Leitfaden, Genehmigung
- **GQ05:** Exkursion, Antrag, Formular, Download
- **GQ06:** Diplomarbeit, Thema, Betreuung, Koordinator
- **GQ07:** Company Thesis Day, Diplomarbeit, Firmen
- **GQ08:** Diplomarbeit, Präsentation, Dresscode, Kleidung

Query categories.

Table 7: Golden query catalog for semantic search evaluation.

ID	Name	Query (German)	Type
GQ01	DA Format (Word?)	Darf ich meine Diplomarbeit mit MS Word schreiben?	Policy
GQ02	Matura Klausurtag	Wann ist Einlass am Klausurtag der Matura?	Factual
GQ03	Office365 Mail	Ab wann gibt es die neuen Office365 Mail Accounts?	Factual
GQ04	Exkursion organisieren	Was ist zu beachten, wenn ich eine Exkursion organisiere?	Procedural
GQ05	Exkursion Antrag	Wo finde ich das Antragsformular für Exkursionen?	Navigation
GQ06	DA kein Thema	Was ist wenn ich kein Thema für eine Diplomarbeit finde?	Advisory
GQ07	Company Thesis Day	Was ist der Company Thesis Day?	Conceptual
GQ08	Dresscode Präsentation	Wie lautet der Dresscode für Diplomarbeitspräsentationen?	Policy

- **Policy (GQ01, GQ08):** Expect references to school rules or regulations.
- **Factual (GQ02, GQ03):** Expect specific dates, times, or system information.
- **Procedural (GQ04):** Expect step-by-step instructions or checklists.
- **Navigation (GQ05):** Expect links or locations of downloadable resources.
- **Advisory (GQ06):** Expect guidance for students in specific situations.
- **Conceptual (GQ07):** Expect explanatory content about school events.

All queries are in German, matching the primary language of LeoWiki content. This tests the server's ability to handle German-language semantic search with the `text-embedding-3-large` model. The golden query evaluation serves as a qualitative verification of the deployed production system. The primary quantitative evaluation, based on 78 ground-truth questions with automated scoring, is documented in Appendix .6.

.5.2 Test Execution Log

Execution metadata.

- **Date:** 2026-02-20
- **Server:** LeoWiki Remote MCP v2.1 at `https://leowiki-mcp.stream/mcp`
- **Configuration:** `limit=10`, embedding model `text-embedding-3-large` (3072 D)

- **Environment:** Docker Compose (Caddy + MCP Server + Qdrant), Raspberry Pi
- **Collection:** `educational_content` (3,417 documents at test time). The production collection underwent three stages: (1) 3,417 vectors on 2026-02-20 (this test), (2) 6,002 vectors after a full re-ingestion on 2026-02-26 (cf. Table 6.9), (3) 6,082 vectors by 2026-03-21 after an incremental update adding 80 newly published pages (cf. Section 5.3.4). The 10,841-chunk evaluation corpus uses a different configuration; see Section 6.3.4 for the pipeline parameter differences
- **Client:** Claude Web App (Pro Plan). Queries were sent verbatim to the MCP server without LLM reformulation

Student Role Results (`search_content_student`)

Table 8 shows the results when querying with the student role.

Table 8: Golden query results—student role (`search_content_student`).

QID	Short Name	Top Result Source	Rel.
GQ01	DA Format	Word-Vorlage mit Formatvorlagen (diplomarbeit_vorlage_deckblatt.docx)	4
GQ02	Matura Klausurtag	**Einlass: 7:45–8:00 Uhr** (exams:matura-am)	5
GQ03	Office365 Mail	Studentmailmigration: ab 10. Sep. 2023 (special:studentmailmigration)	5
GQ04	Exkursion org.	Antrag auf Exkursion / Wandertag / Lehrausgang (org:forms:exkursion)	4
GQ05	Exkursion Antrag	Antrag auf Exkursion / Wandertag / Lehrausgang (org:forms:exkursion)	5
GQ06	DA kein Thema	DA-Inf-IT: Abstimmung mit AV; Kolleg:innen besprechen (exams:da-inf-it)	4
GQ07	Company Thesis Day	**Company Thesis Day:** 6 Firmen, 3-Min-Slots (exams:da-inf-it)	5
GQ08	Dresscode	**Kleidungsvorschrift: festlich** (exams:da-inf-it)	5

Teacher Role Results (`search_content_teacher`)

Table 9 shows the teacher role results. Entries marked with $\star$ indicate results exclusive to the teacher role from the `teacher:` namespace.

Relevance scale.

- **5:** Perfect match—all gold keywords present, directly answers the query.
- **4:** Highly relevant—most gold keywords present, answers the query well.

Table 9: Golden query results—teacher role (`search_content_teacher`).

QID	Short Name	Top Result Source	Rel.
GQ01	DA Format	(identical to student)	4
GQ02	Matura Klausurtag	(identical to student)	5
GQ03	Office365 Mail	Result #5 ⋆: teacher:it:teachermailaccounts (Lehrer-Webmail)	5
GQ04	Exkursion org.	Result #1 ⋆: teacher:neulehrerinnenhandbuch (Frau Aslan, SGA)	5
GQ05	Exkursion Antrag	Result #2 ⋆: teacher:neulehrerinnenhandbuch; #5 ⋆: teacher:forms:stundentausch	5
GQ06	DA kein Thema	(identical to student)	4
GQ07	Company Thesis Day	(identical to student)	5
GQ08	Dresscode	(identical to student)	5

- **3:** Partially relevant—some gold keywords, related but incomplete answer.
- **2:** Marginally relevant—few keywords, only loosely related.
- **1:** Not relevant—no gold keywords, does not answer the query.

.5.3 RBAC Verification

Table 10 summarizes the role-based differences observed across all eight golden queries when comparing `search_content_student` and `search_content_teacher`.

Table 10: RBAC differences: student vs. teacher search results.

QID	Diff?	Teacher-Exclusive Content
GQ01	No	Identical results
GQ02	No	Identical results
GQ03	Yes	`teacher:it:teachermailaccounts` (Lehrer-Webmail, bildung.gv.at)
GQ04	Yes	`teacher:neulehrerinnenhandbuch:teaching` (Frau Aslan, SGA); `teacher:forms:eintaegigeveranstaltung` (SchVV §2)
GQ05	Yes	`teacher:neulehrerinnenhandbuch:teaching`; `teacher:forms:stundentausch`
GQ06	No	Identical results
GQ07	No	Identical results
GQ08	No	Identical results

Analysis. Three of eight queries (GQ03, GQ04, GQ05) produce different results depending on the user role. In all three cases, the teacher role returns additional content from the `teacher:` namespace that is not visible to students. This confirms that the

RBAC filtering system correctly restricts namespace access based on JWT role claims. The five queries with identical results concern content in shared namespaces (`exams:`, `org:forms:`, `it:`) that are accessible to both roles.

.5.4 MCP Server Test Results

The automated test suite in `tests/` comprises 22 test files with approximately 100 top-level `test_*` functions (pytest collects additional cases from parametrized tests). Deterministic tests (no external dependencies) form the CI gate and verify tool registration, RBAC permission matrices, middleware chain ordering, and configuration loading. Integration tests verify the full request lifecycle including HTTP endpoints, MCP protocol negotiation, and query logging.

.5.5 Security Verification

RBAC bypass. The golden query comparison (Table 10) confirms that `teacher:` namespace content is not returned to student-role users. This is enforced by the `RBACEnforcementMiddleware` which intercepts tool calls before execution and applies namespace-based post-filtering on results.

Rate limiting. The `slowapi` library enforces per-IP rate limits (default: 100 requests/minute). Exceeding the limit returns HTTP 429 with a `Retry-After` header.

Error masking. Search tool errors return a generic German message (`"Es ist ein Fehler bei der Suche aufgetreten."`) rather than exposing internal implementation details such as Qdrant connection strings or stack traces.

.6 Benchmark Data

This appendix provides the complete benchmark data referenced in Chapter 6. All measurements were performed on 2026-02-17 on the following hardware and software stack:

- **Hardware:** NVIDIA RTX 4080 Laptop GPU (12 GB VRAM), AMD Ryzen 9, 32 GB RAM.

- **Software:** Python 3.11 (evaluation stack; the MCP server uses Python 3.13, see Section 6.8.5), sentence-transformers 3.3, tiktoken 0.5, qdrant-client 1.12, Qdrant v1.14.1.
- **Tokenizer:** Token counts reported by the OpenAI API (`response.usage.total_tokens`); chunk-level token counting uses `tiktoken`.

The LeoWiki corpus contained 10,841 chunks derived from 196 wiki pages and 320 media files.

.6.1 Embedding Model Evaluation – Full Results

Table 11: Complete retrieval metrics per model (78 ground-truth questions, Dense Cosine Similarity, top_k=10).

Model	MRR	NDCG@10	P@5[a]	Recall@10[b]	Hit Rate[b]
Octen-Embedding-4B	0.883	0.899	0.185	0.949	94.9%
text-embedding-3-large	0.872	0.897	0.192	0.974	97.4%
PIXIE-Rune-v1.0	0.802	0.823	0.177	0.885	88.5%
snowflake-arctic-embed-l-v2.0	0.788	0.822	0.182	0.923	92.3%
bge-m3-unsupervised	0.756	0.800	0.182	0.936	93.6%

[a] Precision measured at $k = 5$. Because the evaluation operates at chunk granularity, multiple chunks from the same or overlapping pages may be scored as relevant by the multi-signal scoring function (see Section 6.2), which explains why P@5 exceeds 0.100.

[b] Each ground-truth question targets one wiki page. Recall@k and Hit Rate are equivalent at document level (one relevant page per query); both metrics are shown for comparability with MTEB reporting conventions.

.6.2 Results by Difficulty Level

Table 12: MRR and Hit Rate broken down by question difficulty.

	Easy (n=17)		Medium (n=40)		Hard (n=21)	
Model	MRR	Hit%	MRR	Hit%	MRR	Hit%
Octen-4B	0.840	100.0	0.916	95.0	0.857	90.5
OpenAI-3-large	0.873	94.1	0.856	97.5	0.900	100.0
PIXIE-Rune	0.765	88.2	0.820	90.0	0.810	85.7
Snowflake	0.752	88.2	0.805	92.5	0.795	95.2
bge-m3	0.718	88.2	0.774	95.0	0.761	95.2

.6.3 Embedding Speed Comparison

Table 13 reports the embedding throughput for the five models measured on the full 10,841-chunk evaluation corpus. All local models were run on the NVIDIA RTX 4080 Laptop GPU (12 GB VRAM); the OpenAI model was accessed via API.

Table 13: Embedding throughput for the full LeoWiki corpus (10,841 chunks).

Model	**Params**	**Dim**	**Time (s)**	**Chunks/s**
bge-m3-unsupervised	568M	1024	104.6	104
PIXIE-Rune-v1.0	568M	1024	107.3	101
snowflake-arctic-embed-l-v2.0	568M	1024	109.4	99
text-embedding-3-large	API	3072	179.1	61
Octen-Embedding-4B	4.0B	2560	2186.3	5

.6.4 Search Quality – Semantic vs. Keyword

Table 14: Semantic search (text-embedding-3-large) vs. DokuWiki keyword search on 78 ground-truth questions.

Method	**MRR**	**Hit Rate**	**Avg. Rank**
Semantic (OpenAI, cosine)	0.872	97.4%	1.23
Keyword (DokuWiki ft_idx)	0.051	5.1%	—
Keyword (LLM-extracted)[†]	0.365	37.2%	—

[†] Keywords were extracted from each question by Claude 3.5 Sonnet (Anthropic snapshot 2024-10-22, accessed 2026-02-17) before querying the DokuWiki full-text index; see CSV row `FF1_keyword_llm` for raw values.

The semantic search achieves a 92.3 percentage-point improvement in hit rate over the native DokuWiki full-text index search. The average rank of the correct result in semantic search is 1.23, meaning the relevant document appears within the top ten results in 76 of 78 queries (97.4%).

.6.5 Dense vs. Hybrid Search – Experiment J6

Table 15: J6 — Dense vs. Hybrid Search (10,841 chunks, OpenAI 3-large)

Mode	MRR	σ	NDCG@10	P@5	Hit Rate
Dense	**0.8733**	0.279	0.8983	0.1923	97.4%
Hybrid (BM25 + RRF)	0.8108	0.309	0.8487	0.1897	96.2%

The experiment used the OpenAI `text-embedding-3-large` model with 10,841 chunks from the full LeoWiki corpus. Hybrid search combines BM25 keyword matching with vector similarity via Reciprocal Rank Fusion (RRF). Dense retrieval outperforms hybrid on this corpus: the 6.25-percentage-point MRR drop suggests that BM25 noise offsets its potential gains for the predominantly German technical-documentation queries used here.

.6.6 Evaluation Corpus Characteristics

Table 16: LeoWiki corpus statistics at time of evaluation.

Metric	**Value**
Total wiki pages	196
Total media files	320
Chunks after processing	10,841
Unique namespaces	23
Ground-truth questions	78
Question categories	Easy / Medium / Hard
Avg. chunk length	287 tokens (counted with `tiktoken`)

.6.7 MCP Server Latency

The latency benchmarks in this subsection were performed on 2026-03-21 on the production hardware and software stack:

- **Hardware:** Raspberry Pi 5 (8 GB RAM, ARM64).
- **Software:** FastMCP 3.0.1, Qdrant v1.14.1 (Docker), Caddy v2.9.1 (TLS termination), Python 3.13.
- **Embedding API:** OpenAI `text-embedding-3-large` (3 072 dimensions).

Transport overhead. The following experiment compares the protocol overhead of the two MCP transports. The `health_check` tool (minimal payload) was used to isolate transport latency from search logic. Each transport was measured over 20 iterations after one warm-up request, timed with `time.perf_counter()`.

Table 17 summarizes the measured timings. The ∼95 ms difference is attributable to TLS handshake (∼30–50 ms), network round-trip via Tailscale VPN (∼50–80 ms), and Caddy reverse proxy processing (∼5–10 ms).

Table 17: MCP transport overhead (`health_check`, 20 iterations per transport, Raspberry Pi 5).

Transport	Mean (ms)	Median (ms)	σ (ms)	Min (ms)	Max (ms)	Ratio
stdio	17.9	17.9	0.2	17.7	18.5	1.0×
HTTP Streamable	113.1	109.9	8.0	108.6	142.9	6.3×

Search pipeline latency. A separate benchmark measured the core search pipeline without MCP protocol overhead, using 78 ground-truth Q&A pairs over three iterations (234 requests total). The full breakdown is reported in Table 5.4 (Section 5.3.4). Embedding generation via the OpenAI API dominates at 87 % of total latency (mean 181.8 ms); the Qdrant HNSW vector search over 6 082 documents[5] accounts for the remaining 13 % (mean 26.1 ms). Combined with transport overhead, estimated end-to-end latency is ∼226 ms via stdio and ∼321 ms via HTTP—both within the NFR-1 target of 2 s.

.7 AI-Generated Content Disclosure

In accordance with HTL Leonding thesis requirements, this appendix documents the use of AI tools during the creation of this thesis. Complete prompt histories are available in the `.prompts/` directory of the thesis repository:

https://github.com/lxl-Enki/DA_2026_thesis_docs

.7.1 AI Was Used For

- Draft generation from structured notes (all chapters), followed by full author review and rewriting
- Translation assistance (German ↔ English)
- LaTeX formatting and diagram code generation
- Code boilerplate and debugging support for the pipeline and MCP server

[5]The production embedding run (2026-02-26) produced 6,002 vectors. The collection had grown to 6,082 points by the benchmark date (2026-03-21) due to an incremental re-ingestion run that added 80 newly published wiki pages.

- Generation of 95 candidate Q&A pairs (Claude Opus 4.6, accessed 2026-02-17), of which 78 were retained after manual verification by both authors (see Section 6.1 for the verification procedure and rejection criteria)

.7.2 AI Was Not Used For

- Research methodology and evaluation design
- Benchmark measurements or result interpretation
- Technology selection decisions (MCP, Qdrant, Scalekit, embedding models)
- Source verification – all technical claims were checked against primary sources (documentation, RFCs, published papers)

.7.3 Writing and Translation Process

The thesis was drafted in English with the assistance of AI-based language tools for grammar checking and stylistic refinement. As native German speakers, the authors used translation assistance for technical writing in English. All technical content, experimental design, methodology, and conclusions represent the authors' own work. Specific AI tools used for writing assistance are listed in the preceding section.

All benchmark data was produced by the authors' own evaluation scripts. No measurement results were generated or modified by AI tools.