Technical Specification: CERN Knowledge Explorer – Pioneering QoR Excellence in AI-Driven Scientific Discovery

Document Metadata

• Version: 1.0

• Date: July 17, 2025

• Authors: Ryan Lence

• Project Code: CKE-2025

• Status: Draft for Implementation

• Approval Chain: AI Systems Lead → Product Engineering Director → CTO

• Related Documents: QURE Framework Inspiration , Mistral Model Integration Guidelines (HF-INT-024)

Executive Summary

The CERN Knowledge Explorer (CKE) is an advanced, end-to-end AI platform designed to transform scientific PDFs into an interactive knowledge engine. This specification describes a high–Quality of Results (QoR) system focused on precision, multilingual fluency (including native Dutch support), and multimodal insights. By utilizing the unsloth/Mistral-Small-3.2-24B-Instruct-2506-GGUF model in 4-bit quantization, CKE delivers strong performance on commodity hardware such as the RTX 3090, maintaining VRAM usage at approximately 14 GB while enabling accurate retrieval-augmented generation (RAG), domain-specific adaptation, and a responsive chat interface.

The primary objective is to provide a system where each query yields reliable, high-fidelity scientific insights, with target metrics including over 90% recall, sub-5-second latency, and hallucination rates below 5%. This specification organizes the platform into modular components—complete with directory structures, interfaces, and implementation guidance—to streamline development and support efficient, scalable implementation.

Business Objectives and QoR Metrics

- Core Objectives:
    - Enable seamless exploration of CERN magazine content (extensible to any PDF source).
    - Achieve enterprise-grade QoR: High faithfulness (84%+ benchmark alignment), completeness, and multilingual accuracy.
    - Support Dutch as a first-class language for global inclusivity, targeting 80%+ cross-lingual precision.
- Key Performance Indicators (KPIs):
    - QoR Metrics: Answer Correctness (ROUGE-L >0.85), Faithfulness (self-critique score >0.9), Recall (Context Recall >0.9), Latency (<5s/query), Conciseness (response length < gold standard +20%).
    - System Metrics: VRAM Usage (<16GB on RTX 3090), Throughput (10+ queries/min), Scalability (handle 100k+ vectors).

Success Criteria: MVP deployment with 95% uptime; user satisfaction >4.5/5 in internal beta tests.

System Requirements

Functional Requirements

- Data Ingestion: Automated PDF download and metadata cataloging from configurable sources (e.g., CERN Courier).
- Vectorization: Multimodal embedding of text, images, and charts with multilingual support.
- Model Integration: 4-bit quantized Mistral model for RAG and fine-tuning, ensuring Dutch fluency.
- Fine-Tuning: Domain-specific adaptation on CERN data for enhanced QoR.
- User Interface: Web-based chat UI for interactive querying with visual outputs.
- Evaluation Harness: Built-in QoR monitoring and logging.

Non-Functional Requirements

- Performance: Optimized for RTX 3090; inference latency <2s/token.
- Security: Local-only deployment; no external API calls in MVP.
- Scalability: Modular design for future cloud migration (e.g., OpenAI / xAI API integration).
- Compatibility: Python 3.11+; libraries via Hugging Face, FAISS, Gradio.
- Multilingual Support: Native handling for Dutch/English; extensible to 20+ languages per Mistral benchmarks.
- Hardware Constraints: 4-bit quantization mandatory; target VRAM <14GB.

High-Level Architecture

CKE adopts a layered, microservices-inspired architecture for modularity and QoR assurance:
- Ingestion Layer: PDF downloader and parser.
- Processing Layer: Vector embedder and database.
- Core AI Layer: Quantized Mistral model with RAG pipeline.
- Fine-Tuning Layer: PEFT/LoRA/DoRA trainer.
- Presentation Layer: Gradio-based UI.
- QoR Layer: Metrics collector and dashboard.

Data Flow:
1. PDFs → Parser → Chunks (Text/Images).
2. Chunks → Embeddings → Vector DB.
3. Query → Retrieval → Augmented Prompt → Model Generation.
4. Output → UI with QoR Scores.

Tech Stack:
- Languages/Frameworks: Python, Hugging Face Transformers, unsloth (for efficient fine-tuning), FAISS (vector DB), Gradio (UI).
- Model: unsloth/Mistral-Small-3.2-24B-Instruct-2506-GGUF (4-bit Q4_K_M).
- Dependencies: Managed via requirements.txt; no runtime installs.

Implementation Phases and Cursor-Optimized Design

To streamline development in Cursor, we've structured each phase as independent modules with suggested file structures, class interfaces, and pseudocode skeletons (no actual code per guidelines).

Phase 1: PDF Download Module

Refer to directory structures, key components, and Cursor optimization prompts as outlined in the specification for implementation details.

Phase 2: Vectorization Module

Refer to directory structures, key components, and Cursor optimization prompts as outlined in the specification for implementation details.

Phase 3: Model Setup Module

Refer to directory structures, key components, and Cursor optimization prompts as outlined in the specification for implementation details.

Phase 4: Fine-Tuning Module

Refer to directory structures, key components, and Cursor optimization prompts as outlined in the specification for implementation details.

Phase 5: Chat UI Module

Refer to directory structures, key components, and Cursor optimization prompts as outlined in the specification for implementation details.

QoR Evaluation and Monitoring

Module: src/qor/
Classes: QoRMonitor (methods: compute_metrics(gold: str, pred: str) -> Dict – ROUGE, faithfulness via model self-critique).
Integration: Embed in all phases; dashboard in UI.

Development Guidelines for Cursor

Best Practices: Use Cursor for module-by-module implementation; start with skeletons (e.g., 'Define class with methods as per spec').
Testing: Unit tests per module (pytest); e2e QoR benchmarks.
Version Control: Git branches per phase.
Risks/Mitigations: VRAM overflow → Strict 4-bit enforcement; Dutch inaccuracies → Bilingual datasets.