Benutzerhandbuch: RAG-System Integration

1. Einführung

Das RAG-System (Retrieval-Augmented Generation) stellt eine moderne Lösung für intelligente Dokumentenverarbeitung und semantische Suche dar. Dieses Handbuch führt Sie durch alle Aspekte der Installation, Konfiguration und Nutzung des Systems.

RAG-Systeme kombinieren die Vorteile von vortrainierten Sprachmodellen mit der Möglichkeit, spezifische Dokumente und Datenquellen zu durchsuchen. Dies ermöglicht präzise und kontextuelle Antworten basierend auf Ihren eigenen Dokumenten.

Die wichtigsten Komponenten des Systems umfassen: Document Processing Pipeline, Embedding Generation, Vector Storage, Retrieval Engine und Query Processing. Jede Komponente ist modular aufgebaut und kann individuell konfiguriert werden.

2. Installation und Setup

Bevor Sie mit der Installation beginnen, stellen Sie sicher, dass Ihr System die Mindestanforderungen erfüllt: Python 3.8+, mindestens 8GB RAM, 50GB freier Speicherplatz.

Installieren Sie zunächst die erforderlichen Dependencies: pip install rag-system torch transformers sentence-transformers chromadb fastapi uvicorn. Für GPU-Unterstützung installieren Sie zusätzlich: pip install torch[cuda].

Erstellen Sie eine Konfigurationsdatei config.yaml mit Ihren spezifischen Einstellungen. Wichtige Parameter sind: chunk_size (empfohlen: 512), overlap_ratio (empfohlen: 0.1), embedding_model und vector_database_path.

Starten Sie das System mit: python -m rag_system start --config config.yaml. Das System wird automatisch auf Port 8000 gestartet und ist unter http://localhost:8000 erreichbar.

3. Dokumentenverarbeitung

Der erste Schritt bei der Nutzung des RAG-Systems ist das Hochladen und Verarbeiten Ihrer Dokumente. Das System unterstützt verschiedene Dateiformate: PDF, DOCX, TXT, HTML und Markdown.

Beim Upload werden Dokumente automatisch in kleinere Textabschnitte (Chunks) aufgeteilt. Die optimale Chunk-Größe liegt zwischen 300-800 Zeichen, um ein gutes Verhältnis zwischen Kontext und Präzision zu gewährleisten.

Für jedes Chunk wird ein Embedding-Vektor generiert, der die semantische Bedeutung des Textes repräsentiert. Diese Vektoren werden in einer Vektordatenbank gespeichert für schnelle Ähnlichkeitssuchen.

Metadaten wie Dokumententitel, Autor, Erstellungsdatum und Kategorien werden extrahiert und mit den Chunks verknüpft. Dies ermöglicht gefilterte Suchen nach spezifischen Kriterien.

4. Suche und Retrieval

Die semantische Suche erfolgt über die API-Endpoints oder die Web-Oberfläche. Geben Sie Ihre Frage in natürlicher Sprache ein - das System findet automatisch die relevantesten Dokumentenabschnitte.

Der Retrieval-Prozess umfasst mehrere Schritte: Query Embedding Generation, Vector Similarity Search, Reranking der Ergebnisse und Context Assembly für die finale Antwortgenerierung.

Sie können die Suchparameter anpassen: max_results (Anzahl der Ergebnisse), similarity_threshold (Mindestähnlichkeit), filters (Dokumentfilter) und rerank_enabled (Ergebnis-Reranking).

Für komplexe Anfragen nutzen Sie die erweiterte Suchsyntax mit Booleschen Operatoren, Phrasensuche und Feldspezifikationen. Beispiel: 'machine learning AND (neural networks OR deep learning)'

5. API-Integration

Das RAG-System bietet eine umfassende REST-API für die Integration in bestehende Anwendungen. Die API folgt OpenAPI 3.0 Standards und bietet automatische Dokumentation unter /docs.

Wichtige Endpoints: POST /documents/upload (Dokument hochladen), GET /documents (Dokumente auflisten), POST /search (Semantische Suche), DELETE /documents/{id} (Dokument löschen).

Für die Authentifizierung verwenden Sie API-Keys, die in der Konfiguration definiert werden. Jeder Request muss den Header 'X-API-Key' enthalten. Rate-Limiting ist standardmäßig auf 100 Requests pro Minute gesetzt.

Die API unterstützt sowohl synchrone als auch asynchrone Verarbeitung. Für große Dokumente verwenden Sie den asynchronen Upload-Endpoint, der eine Job-ID zurückgibt zur Statusabfrage.

6. Monitoring und Wartung

Das System bietet umfassende Monitoring-Features über das Dashboard unter /admin. Hier finden Sie Statistiken zu Dokumentenanzahl, Speicherverbrauch, Query-Performance und Systemauslastung.

Wichtige Metriken: Durchschnittliche Query-Latenz, Embedding-Generierung pro Minute, Cache-Hit-Rate und Speicherverbrauch der Vektordatenbank. Alerting ist für kritische Schwellwerte konfigurierbar.

Regelmäßige Wartungsaufgaben umfassen: Index-Optimierung (wöchentlich), Backup der Vektordatenbank (täglich), Log-Rotation (täglich) und Performance-Tuning basierend auf Usage-Patterns.

Für Backup und Disaster Recovery exportieren Sie regelmäßig die Vektordatenbank und Konfigurationsdateien. Der Export-Befehl: python -m rag_system export --output backup_folder

7. Troubleshooting

Bei Performance-Problemen prüfen Sie zuerst die Systemressourcen: CPU, RAM und Festplatten-I/O. Das RAG-System ist speicherintensiv - mindestens 8GB RAM werden empfohlen.

Häufige Probleme: Langsame Suchzeiten (Index-Rebuild erforderlich), hoher Speicherverbrauch (Chunk-Size reduzieren), schlechte Suchqualität (Embedding-Model wechseln).

Für Debugging aktivieren Sie das Debug-Logging: python -m rag_system start --log-level DEBUG. Logs werden in logs/rag_system.log gespeichert.

Bei Problemen mit der Dokumentenverarbeitung prüfen Sie die unterstützten Dateiformate und Größenbeschränkungen. Maximale Dateigröße: 100MB pro Dokument.