**Agentische RAG-Systeme: Evaluation führender GitHub-Repositories und Architekturempfehlungen**

**Die Zukunft der RAG-Systeme liegt in der intelligenten Kombination von Vektor-Suche, Graphen-basiertem Reasoning und agentischer Orchestrierung.** Microsoft GraphRAG, LightRAG und Graphiti definieren dabei den State-of-the-Art, während LangGraph, CrewAI und LlamaIndex die führenden Orchestrierungsframeworks darstellen. Entscheidend für den Projekterfolg: Die Wahl zwischen diesen Technologien hängt primär von Latenz-Anforderungen, Datentypen und Query-Komplexität ab. Der Model Context Protocol (MCP) hat sich nach nur 6 Monaten als universeller Standard etabliert – mit offizieller Unterstützung von OpenAI, Google und Microsoft – und bietet die ideale Basis für zukünftige Enterprise-Tool-Integrationen. Hybrid-Architekturen, die Vektor-Suche für semantische Ähnlichkeit mit Graph-Datenbanken für Multi-Hop-Reasoning kombinieren, erzielen in Produktionsumgebungen 40-60% bessere Relevanz-Scores als reine Vektor-basierte Ansätze.

**Agenten-Frameworks: Die Orchestrierungs-Schicht**

Die vier führenden Frameworks unterscheiden sich fundamental in Architektur-Paradigma und Anwendungsschwerpunkt, wobei alle produktionsreif und aktiv maintained sind.

**LangGraph** (github.com/langchain-ai/langgraph, 10K+ Stars) definiert Workflows als zustandsbasierte Graphen mit expliziten Knoten und Kanten. Diese Graph-Architektur ermöglicht präzise Kontrolle über komplexe Multi-Step-Workflows mit Zyklen, bedingten Verzweigungen und State-Persistence. LangGraph bietet die umfassendste Produktions-Infrastruktur: LangGraph Platform für managed deployments, LangGraph Studio für visuelles Debugging und native Durable Execution mit automatischem Resume bei Failures. Enterprise-Adoption bei Klarna, Replit und Uber belegt Produktionsreife. Die Implementierung erfordert jedoch steile Lernkurve und mehr Boilerplate-Code als Alternativen.

**CrewAI** (github.com/crewAIInc/crewAI, 39.1K Stars) verfolgt einen rollenbasierten Ansatz mit spezialisierten Agenten-Teams. Die einfachste Lernkurve aller Frameworks kombiniert mit exzellenter Performance – Benchmarks zeigen 5,76x schnellere Ausführung als LangGraph bei QA-Tasks. CrewAI erreichte in nur 8 Monaten 100.000+ zertifizierte Developer und bietet sowohl autonome Crews als auch kontrollierte Flows. Der unabhängige Tech-Stack (keine LangChain-Dependency) und die kommerzielle Enterprise-Plattform mit Control Plane machen CrewAI zur ersten Wahl für schnelle Produktiv-Deployments mit klaren Rollen-Definitionen.

**AutoGen** (github.com/microsoft/autogen, 25K+ Stars) implementiert ein konversationsbasiertes Multi-Agent-System mit asynchronem Message-Passing. Die v0.4-Rewrite transformierte AutoGen von einem Research-Tool zu einer production-ready Plattform mit Event-driven Architecture und Cross-Language-Support (Python + .NET). Microsoft-Backing garantiert langfristige Unterstützung. AutoGen eignet sich besonders für flexible Multi-Agent-Collaboration und Code-Execution-Workflows, erfordert aber mehr Infrastruktur-Setup als managed alternatives.

**LlamaIndex** (github.com/run-llama/llama\_index, 44.7K Stars) ist purpose-built für RAG mit der besten Daten-Integration im Markt: 300+ Connectoren zu APIs, Datenbanken, Cloud-Storages und Dokumenten-Formaten. Agentic RAG ist Core-Feature mit ReAct-Agents, Router-Agents und dem neuen Workflows-Framework für Event-driven Orchestration. LlamaCloud bietet managed RAG-Services mit LlamaParse für state-of-the-art Document Parsing (90+ Dateiformate). Die opinionated architecture beschleunigt RAG-Entwicklung erheblich, während die Modularität Flexibilität erhält.

Die **Architektur-Paradigmen** unterscheiden sich fundamental: LangGraph nutzt Directed Acyclic Graphs mit State-Management, CrewAI verfolgt Team-basierte Koordination, AutoGen implementiert Peer-to-Peer Message Exchange und LlamaIndex orchestriert Data-centric Query Routing. **RAG-spezifische Features** sind bei LlamaIndex am ausgereiftesten mit nativer Unterstützung für HyDE, Self-RAG, RAPTOR und agentic retrieval modes. LangGraph und CrewAI integrieren RAG primär über Tools und Memory-Module, während AutoGen framework-agnostische Integration ermöglicht.

**GraphRAG und Graphiti: Knowledge Graphs für Reasoning**

**Microsoft GraphRAG** (github.com/microsoft/graphrag, 23.8K+ Stars) revolutioniert RAG durch hierarchisches Community-basiertes Reasoning. Der Indexing-Prozess extrahiert Entities und Relationships via LLM, konstruiert Knowledge Graphs mit NetworkX/Neo4j und wendet Leiden-Clustering für Community Detection an. Die resultierende hierarchische Struktur ermöglicht sowohl Global Search (holistische Questions über Summaries) als auch Local Search (Entity-spezifische Queries) und DRIFT Search (kombiniert beide Ansätze). Benchmarks zeigen 86,31% Accuracy bei RobustQA versus 59-72% für reine Vektor-Methoden.

Die Stärke liegt in Complex Document Analysis mit Multi-Hop-Reasoning über große, zusammenhängende Dokument-Kollektionen. Azure Deployment Accelerator und Solution Accelerator ermöglichen One-Click-Production-Deployment. Die signifikanten Limitierungen: Extrem hohe Indexing-Kosten durch intensive LLM-Nutzung, statische Graph-Struktur erfordert vollständige Re-Indexierung bei Updates, keine native temporale Awareness. GraphRAG eignet sich für Read-heavy Workloads mit seltenen Updates und komplexen analytischen Queries.

**LightRAG** (github.com/HKUDS/LightRAG, 18K Stars) bietet eine schnellere, kostengünstigere Alternative mit dual-level retrieval (low-level entities + high-level topics) und inkrementellen Graph-Updates ohne Regenerierung. Die Architektur unterstützt multiple Graph-Backends (Neo4j, PostgreSQL AGE, Memgraph) und Vector Stores (Milvus, Qdrant, ChromaDB, Faiss). Vier Query-Modi adressieren verschiedene Use Cases: Local (entity-level), Global (topic-level), Hybrid (kombiniert) und Mix (Knowledge Graph + Vector). Web UI, API Server und Ollama-kompatibles Interface machen LightRAG besonders developer-friendly. Performance-Vergleiche zeigen schnelleres Indexing und niedrigere Token-Kosten als Microsoft GraphRAG bei vergleichbarer oder besserer Accuracy.

**Graphiti** (github.com/getzep/graphiti, 14K Stars) adressiert ein fundamentales Problem anderer Systeme: Temporal Awareness für agentisches Long-Term Memory. Die bi-temporale Datenstruktur trackt sowohl Valid Time (wann Events real stattfanden) als auch Transaction Time (wann Informationen ins System kamen) und ermöglicht Point-in-Time Queries und historische Rekonstruktion. Real-time inkrementelle Updates ohne Batch-Recomputation unterscheiden Graphiti von GraphRAG-Ansätzen.

Das Hybrid Retrieval System kombiniert semantische Embeddings, BM25 Keyword Search und Graph Traversal mit sub-100ms Latency. Die Architektur trennt Episodic Subgraph (rohe Conversation Data), Semantic Subgraph (extrahierte Entities und Facts) und Community Subgraph (High-Level Concepts). Native Integration mit Neo4j, FalkorDB, Amazon Neptune und MCP-Server-Support für Claude/Cursor positionieren Graphiti als führende Lösung für conversational AI mit evolving context. Benchmarks zeigen 94,8% Deep Memory Retrieval Accuracy (vs. MemGPT 93,4%) und 18,5% Verbesserung bei temporalen Reasoning Tasks.

**Integration mit Agent-Frameworks**: LlamaIndex bietet native GraphRAG-Implementierung via PropertyGraph und GraphRAGStore. LangChain integriert über Custom Retrievers und Neo4j-Tools. CrewAI nutzt Graph-Queries als Agent Tools. Die praktische Realität: **Graphiti eignet sich primär für Agent Memory, während Microsoft/LightRAG für Document Analysis optimiert sind**. Hybrid-Ansätze kombinieren oft beide – Graphiti für Session Memory, GraphRAG für statische Knowledge Bases.

**Vektor-Datenbanken und RAG-Orchestrierung**

**Qdrant** (github.com/qdrant/qdrant, 21K Stars) führt Performance-Benchmarks an: 3ms Latency bei 1M Embeddings, höchste RPS aller getesteten Systeme, 24x Kompression durch asymmetrische Quantisierung. Die Rust-basierte Architektur ermöglicht advanced Filtering während Graph-Traversal (nicht Post-Search) und RBAC mit OAuth2/OIDC. Native Integration mit LangGraph, CrewAI, AutoGen und LlamaIndex. Self-hosted oder Qdrant Cloud für managed service. Optimal für Performance-kritische Production RAG mit Kosten-Sensitivität.

**Weaviate** (github.com/weaviate/weaviate, 11K Stars) differenziert sich durch GraphQL-Interface für komplexe Queries und native RAG-Integration (Combined Retrieval + Generation in einer Query). Modulare AI ermöglicht Plug-in verschiedener Embedding-Models. Binary Quantization reduziert Memory um 32x. Multi-Tenancy mit Cold Storage Offloading skaliert für Enterprise. Elysia Framework bietet Open-Source Agentic RAG mit Decision-Tree Logic. Weaviate Cloud für managed deployments.

**Pinecone** (Proprietary, Managed Service) bietet die einfachste Operations-Experience mit serverless Auto-Scaling und Pinecone Assistant API für Out-of-the-box Agentic Workflows. JSON Output Mode, Custom Instructions und Context API für Tool Integration. Evaluation API misst Correctness und Completeness. Sub-10ms Latency bei Milliarden Embeddings. Namespace-Support für Multi-Tenancy. Optimal für Teams ohne DB-Expertise und Multi-Region Requirements. Trade-off: Vendor Lock-in und keine Self-Hosting Option.

**ChromaDB** (github.com/chroma-core/chroma, 12K Stars) fokussiert Developer Velocity mit pip install chromadb für instant local setup. MCP Server enables AI Agents direkte Collection-Verwaltung. WASM-powered Browser-Support für Client-side Vector Operations. Serverless Cloud Deployment mit Auto-Scaling. Limitierungen: Keine distributed Data Replication, nicht für Milliarden-Scale geeignet. Ideal für Rapid Prototyping und Cost-sensitive Projects.

**LangChain vs. LlamaIndex für RAG-Orchestrierung**: LangChain bietet breitestes Ecosystem (40+ Vector Stores, 160+ Document Loaders) und maximale Flexibilität durch LangGraph State Management. Produktions-Features über LangSmith (Tracing, Debugging) und LangServe (API Deployment). Conditional Retrieval, Query Decomposition und Ensemble Retrievers als Advanced Patterns. LlamaIndex überzeugt durch opinionated RAG-Patterns, die Development beschleunigen: Document-Level Agents, Router Agents, Auto-Routed Modes. LlamaCloud managed services mit LlamaParse (Best-in-Class Document Parsing) und LlamaExtract (Schema-based Extraction). Evaluation Modules für Correctness, Faithfulness, Relevance out-of-the-box.

**Empfehlung**: LangChain für Full-Stack Agentic Applications mit komplexen Multi-Tool Workflows. LlamaIndex für Data-centric RAG mit heterogenen Datenquellen und schneller Time-to-Production. Viele Teams kombinieren beide – LlamaIndex für Data Ingestion und Indexing, LangGraph für Workflow Orchestration.

**Model Context Protocol: Der Universal-Standard für Tool-Integration**

Der Model Context Protocol etablierte sich in nur 6 Monaten als **"USB-C für AI"** mit Industry-wide Adoption. OpenAIs offizielle MCP-Integration im März 2025 markierte den Inflection Point – gefolgt von Google DeepMind, Microsoft und 12+ Agent Frameworks. Die Juni 2025 Spec (2025-06-18) standardisiert OAuth 2.1, Streamable HTTP Transport und Structured Content mit JSON Schemas.

**MCP löst das N×M Problem**: Statt Custom Integrations für jede Kombination von N AI Applications und M Data Sources definiert MCP einen universellen Protokoll-Standard. Die **Client-Host-Server Architektur** trennt Host Applications (Claude, VS Code), MCP Clients (1:1 mit Servers) und MCP Servers (expose capabilities). JSON-RPC 2.0 als Base Protocol mit Transport über stdio (local processes) oder Streamable HTTP (remote services).

**Core Primitives** adressieren verschiedene Integration-Patterns: Tools (AI-executable Functions), Resources (Context und Data für Models), Prompts (Templated Messages), Sampling (Server-initiated agentic behaviors), Roots (Filesystem boundary inquiries) und Elicitation (Server-initiated User Information Requests).

**Framework-Integration Status**: LangChain bietet langchain-mcp-adapters Package mit MultiServerMCPClient für parallele Server-Connections und automatische Tool-Conversion. LlamaIndex integriert via BasicMCPClient und McpToolSpec. OpenAI Agents SDK implementiert First-Class Native Support mit HostedMCPTool, MCPServerStdio und MCPServerStreamableHttp. CrewAI bietet bi-direktionale Integration – MCPServerAdapter nutzt MCP-Servers als Crew Tools, während CrewAI Workflows als MCP-Servers exposable sind.

**Ecosystem Maturity**: 70.000+ GitHub Stars, 500+ Community-built MCP Servers, 9 offizielle SDKs (Python, TypeScript, Java, Kotlin, C#, Go, Ruby, PHP, Swift). Microsoft MCP Servers für Azure, Office 365, DevOps und Kubernetes. Neo4j Labs MCP Server für Cypher-Queries und Graph Algorithms. Composio MCP mit 100+ pre-built Tools und managed Authentication.

**Architektur-Empfehlungen für RAG + MCP**: MCP ersetzt nicht RAG, sondern komplementiert es. Optimales Pattern kombiniert RAG für Fast Semantic Lookup mit MCP für Structured Queries (Text-to-SQL) und Action-Taking (API Calls). LlamaIndex Blog "Does MCP Kill Vector Search?" empfiehlt **Hybrid Approach**: Vector RAG für Unstructured Data Retrieval, MCP Tools für Native Permissions und Structured Access. Query Router entscheidet basierend auf Intent: Simple Facts → Vector RAG, Relationship Queries → Graph Traversal, Temporal Queries → Graphiti, Complex Analysis → GraphRAG Communities.

**Future-Proofing Strategy**: Implementiere MCP Server Capability für deine Plattform via Official SDKs (Python/TypeScript recommended). Starte mit stdio für MVP, plane HTTP für Production. Implementiere OAuth 2.1 und User Consent Flows von Anfang an. Build Hybrid RAG+MCP Architecture mit Smart Routing. Wähle Framework basierend auf Use Case: LangChain für breites Ecosystem, LlamaIndex für RAG-first Approach, OpenAI SDK für OpenAI-centric Projects, mcp-agent für Pattern-based Agent Design.

**Complete Agentic RAG Repositories: Production-Ready Systeme**

**RAGFlow** (github.com/infiniflow/ragflow, 25K Stars) definiert den Standard für Enterprise-Grade Agentic RAG mit No-Code Interface. Version 0.8+ initiiert die "Agentic Era" mit Graph-based Task Orchestration und Multi-Agent Support. 22+ File Format Support, kombinierte Vector + Full-Text + Sparse Vector Search und Knowledge Graphs. Built-in Templates für Customer Service, HR Assistants und Document Analysis. MCP Server Support und Self-RAG Workflows mit Relevance Evaluation. Production Features umfassen Real-time Data Ingestion, Multi-Round Dialogue mit Context Management und Built-in Citations.

**LightRAG** kombiniert wie beschrieben GraphRAG mit Production-Ready Features: Web UI, API Server, Docker Deployment, Workspace Isolation. Multimodal Support via RAG-Anything Integration für Text, Images und Tables. Citation Functionality und Reranker Integration verbessern Accuracy. Active Development für EMNLP 2025 garantiert Langzeit-Support.

**Microsoft GraphRAG** mit Azure Deployment Accelerator und Solution Accelerator für One-Click Production Deployment. Comprehensive Documentation und Prompt Tuning Guide für Optimization. Cost-Optimization Techniques reduzieren Token-Nutzung auf 2-3% versus naive Approaches.

**RAGApp** (github.com/ragapp/ragapp, 3.8K Stars) liefert "Agentic RAG as simple as OpenAI GPTs" built on LlamaIndex. Docker-Deployment mit Admin UI (Port 8000), Chat UI und API. Multiple LLM Backends (OpenAI, Gemini, Ollama). One-Command Setup: docker run -p 8000:8000 ragapp/ragapp.

**Architektur-Patterns aus Production Examples**:

**Single-Agent Agentic RAG** (Router mit Tool Selection) eignet sich für Prototyping und Simple Use Cases. Ein Agent mit multiplen Tools (Vector DB, Web Search, Graph Queries) und Decision Logic. LangGraph Agentic RAG Tutorial und LlamaIndex Simple Agent als Referenz-Implementierungen.

**Multi-Agent Agentic RAG** (Specialized Agents mit Orchestration) adressiert Enterprise Applications. Coordinator Agent routet Queries zu spezialisierten Retrieval Agents (per Data Source), Analysis Agents (Process und Verify) und Generation Agent (Final Synthesis). CrewAI Multi-Agent Crews, AutoGen Conversational Groups, LangGraph Multi-Agent Graphs und Memgraph Multi-Agent RAG als Examples.

**Hybrid Vector-Graph Retrieval** (Parallel Retrieval + Fusion) erzielt 40-60% Relevance Improvements. Query → Embedding → Parallel zu Vector Similarity Search und Graph Traversal/Cypher Query → Context Fusion → LLM Generation. Implementierungen: HybridRAG (BlackRock/Nvidia Paper), LlamaIndex PropertyGraph mit Hybrid Mode, Neo4j + Qdrant Combinations.

**Memory-Enhanced Agentic RAG** kombiniert Short-term (Recent Conversation State), Long-term (RAG über Past Interactions), Semantic Memory (Facts), Episodic Memory (Past Experiences) und Procedural Memory (How-to Knowledge). CrewAI Memory System, LangGraph State Management, AutoGen Memory Protocol als Referenzen.

**Architektur-Empfehlungen und Best Practices**

**Hybrid Search Implementation** kombiniert Semantic Search (Vector) mit Keyword Search (BM25) via Reciprocal Rank Fusion. Weaviate und Qdrant bieten native Hybrid Search, Pinecone auf API-Level. Reranking mit Cross-Encoder Models (ms-marco-MiniLM) Post-Retrieval verbessert Precision. Query Decomposition separiert Structured (SQL/Graph) von Unstructured (Vector) Information Needs.

**Query Routing Patterns**: Classifier-Based Routing (Adaptive RAG) nutzt smaller LLM für Query Complexity Analysis und routet zu Direct Generation, Single-Step Retrieval oder Multi-Step Reasoning. Intent-Based Routing detektiert Factual Lookup → Graph, Semantic → Vector, Complex → Hybrid. Metadata-Enhanced Routing filtert Documents vor Retrieval nach Date, Source, Topic Tags.

**Memory Management Architekturen** folgen mehrschichtigen Patterns. **Short-Term Working Memory** für Current Conversation Context mit Summarization bei Context Window Limits. **Long-Term Persistent Memory** via Vector Databases, Knowledge Graphs oder SQL Stores mit RAG-based Retrieval. **Temporal Memory** (Graphiti) trackt Edge Validity und Transaction Time für Point-in-Time Queries. Redis Memory Management bietet Fast Read/Write, Built-in Eviction Strategies und Integration mit allen Major Frameworks. **A-MEM** (Agentic Memory) implementiert Zettelkasten-inspired Interconnected Knowledge Networks mit Dynamic Indexing.

**Trade-off Analyse Centralized vs. Distributed Coordination**: Centralized bietet Simple Implementation mit Single Point of Control, aber Bottleneck bei High Loads. Distributed erhöht Coordination Complexity, aber bessere Scalability und Resilience. Empfehlung: Starte centralized für MVP, migriere zu Distributed bei steigender Scale. Hybrid Approach: Centralized Orchestrator mit Distributed Execution.

**Synchronous vs. Asynchronous Retrieval**: Synchronous hat höhere Total Latency (Sum aller Operations) aber einfachere Implementation. Asynchronous reduziert Latency (Max statt Sum), erfordert aber Complex State Management. Use Synchronous für Dependent Operations (must complete in order), Asynchronous für Independent Multi-Source Retrievals.

**Single vs. Multi-Database Architecture**: Single Database (meist Vector) eignet sich für Simple Similarity Search mit Lower Operational Overhead. Multi-Database (Vector + Graph + SQL) optimiert verschiedene Query Types mit höherem Maintenance Overhead. Best Practice: Primary Vector DB für Semantic Search, Graph DB für Relationships, SQL für Structured Data, Unified API Layer abstrahiert Complexity. Example: Precina Health nutzt Qdrant + Memgraph für Real-time Insights mit Historical Context.

**Evaluation, Monitoring und Anti-Patterns**

**RAG-Specific Metrics**: Contextual Relevancy (Relevanz retrieved Context zu Query), Contextual Recall (vollständige Information), Contextual Precision (korrekte Ranking). Generation Metrics: Answer Relevancy, Faithfulness/Groundedness (Hallucination Detection), Citation Accuracy. Tools: RAGAS (Open-Source, Reference-Free mit LLM Judges), DeepEval (CI/CD Integration), Arize (Production Monitoring), TruLens (Domain-Specific).

**Agentic-Specific Metrics**: Tool Call Accuracy (korrekte Tool Selection und Arguments), Agent Goal Accuracy (User Goal Achievement), Topic Adherence (Within Predefined Domains), Multi-Turn Performance. Frameworks: Ragas Agentic Metrics, RagaAI Catalyst 8-Step Evaluation Framework, IBM watsonx.governance für Enterprise.

**Critical Anti-Patterns**: Over-Trusting the Model ohne Guardrails führt zu propagierten Hallucinations. Ignoring Metadata in Retrieval verschenkt Filtering-Potential. Static Workflows für Dynamic Problems erfordern Adaptive Routing wie Adaptive RAG. Poor Chunking Strategy (zu groß verliert Precision, zu klein verliert Context) benötigt Adaptive Chunking. Context Window Overload reduziert Model Performance – Reranking und Compression essentiell. Lack of Citations verhindert Verification – constrain Citations zu Retrieved Doc IDs. No Failure Modes führt zu Hallucinated Responses – implement "I don't know" Thresholds.

**Vergleichstabelle: Agentische Features**

| **Framework** | **Stars** | **Architektur** | **RAG Support** | **Production** | **Learning Curve** | **Best For** |
| --- | --- | --- | --- | --- | --- | --- |
| **LangGraph** | 10K+ | Graph State Machine | ⭐⭐⭐⭐ | ⭐⭐⭐⭐⭐ | Steep | Complex Workflows, Precise Control |
| **CrewAI** | 39.1K | Role-Based Teams | ⭐⭐⭐⭐ | ⭐⭐⭐⭐⭐ | Easy | Quick Development, Team Collaboration |
| **AutoGen** | 25K+ | Conversational Multi-Agent | ⭐⭐⭐ | ⭐⭐⭐⭐ | Medium | Research, Code Execution, Microsoft Ecosystem |
| **LlamaIndex** | 44.7K | Data-Centric Routing | ⭐⭐⭐⭐⭐ | ⭐⭐⭐⭐ | Medium | Document-Heavy RAG, Data Integration |

| **GraphRAG System** | **Stars** | **Temporal Support** | **Update Strategy** | **Cost** | **Best For** |
| --- | --- | --- | --- | --- | --- |
| **Microsoft GraphRAG** | 23.8K+ | Basic Timestamps | Static (Full Reindex) | High Indexing | Complex Analysis, Static Datasets |
| **LightRAG** | 18K | No | Incremental | Low | Fast Development, Cost-Efficiency |
| **Graphiti** | 14K | Native Bi-Temporal | Real-time Incremental | Moderate | Agent Memory, Conversational AI |

| **Vector DB** | **Stars** | **Performance** | **Deployment** | **Best For** |
| --- | --- | --- | --- | --- |
| **Qdrant** | 21K | 3ms @ 1M (Best) | Self-hosted / Cloud | Performance-Critical Production |
| **Weaviate** | 11K | Very Good | Self-hosted / Cloud | Hybrid Search, GraphQL Queries |
| **Pinecone** | Proprietary | Sub-10ms @ Billions | Managed Only | Zero-Ops, Multi-Region |
| **ChromaDB** | 12K | Good | Local / Serverless | Rapid Prototyping, Cost-Sensitive |

**Konkrete Empfehlungen für dein Projekt**

**Für dein agentisches RAG-System mit den vier Komponenten**:

**Ein Bild, das Text, Diagramm, Reihe, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.**

**Component 1 - Vector-Based Search**: Wähle **Qdrant** (Open-Source, höchste Performance, Advanced Filtering) oder **Pinecone** (Managed, Zero-Ops) basierend auf Ops-Ressourcen. Implementiere Hybrid Search (Vector + BM25) von Anfang an für bessere Precision.

**Component 2 - Microsoft GraphRAG**: Nutze **LightRAG** statt Microsoft GraphRAG für schnellere Iteration und niedrigere Kosten. Implementiere für statische Knowledge Bases mit seltenen Updates. Alternative: LlamaIndex PropertyGraph für bessere Framework-Integration.

**Component 3 - Graphiti für Long-Term Memory**: **Graphiti ist optimal für Agent Memory** mit Temporal Awareness und Real-time Updates. Integriere mit Neo4j Backend. Separiere Episodic (Raw Conversations) von Semantic Memory (Extracted Facts).

**Component 4 - MCP Integration**: Implementiere MCP Server via Python SDK. Starte mit stdio Transport für Local Tools, plane Streamable HTTP für Remote Services. Design Hybrid Architecture: RAG Tools für Semantic Search, MCP Tools für Structured Queries und Actions.

**Framework-Wahl**: **LangGraph für Orchestration** (präzise Workflow-Kontrolle, State Management, Production Features) kombiniert mit **LlamaIndex für Data Ingestion** (300+ Connectors, opinionated RAG patterns). Alternative: **CrewAI** wenn Development Speed prioritär ist.

**Architecture Pattern**: Implementiere **Hierarchical Multi-Agent Structure**: Coordinator Agent routet basierend auf Query Type zu Specialized Agents (Vector Search Agent, Graph Query Agent, Action Agent). Parallel Retrieval wo möglich für niedrigere Latency. LangGraph Send API für Concurrent Execution. Hybrid Vector-Graph Retrieval mit Reciprocal Rank Fusion.

**Memory Strategy**: Drei-schichtige Architektur: Redis für Short-Term Memory (fast reads), Qdrant für Semantic Long-Term Memory (vector similarity), Graphiti für Temporal Episodic Memory (relationship evolution). Implement Memory Decay via Time-based Expiration und Relevance Scoring.

**MCP Version 2 Vorbereitung**: Alle gewählten Frameworks (LangGraph, LlamaIndex, CrewAI) haben native MCP Support. Design Tool-First Services expecting AI consumption. Implement OAuth 2.1 Authentication von Start. Plan Modular MCP Server Architecture für Future Extensions.

**Monitoring Stack**: RAGAS für Reference-Free Evaluation, Arize Phoenix für Development Tracing, LangSmith (wenn LangChain/LangGraph) für Production Observability. Track Retrieval Quality, Tool Call Accuracy, Agent Goal Achievement und Cost per Query.

**Production Roadmap**: Starte mit Single-Agent Router + Hybrid Search für MVP (Week 1-2). Implementiere Multi-Agent Orchestration mit LangGraph (Week 3-4). Integriere Graphiti Memory (Week 5-6). Add GraphRAG für Complex Analysis (Week 7-8). Implement MCP Servers für Enterprise Tools (Week 9+). Kontinuierliche Evaluation und Iteration basierend auf Production Metrics.

Diese Architektur kombiniert Best-of-Breed Komponenten für Maximum Flexibility, Performance und Future-Proofing via MCP-Standard.