CIKLUM

Databases for GenAI

Embeddings, Vector Databases & Production Patterns

October 2025

Speaker

Maksym Lypivskyi

Head of Cloud Platforms & AI Director

- 9 years at Ciklum driving large-scale cloud and software delivery initiatives
- 3 years specializing in AI
- Core interest: making AI systems reliable, production-ready, and business-impactful

Agenda

01 Foundations

02 Advanced RAG patterns

03 Production readiness

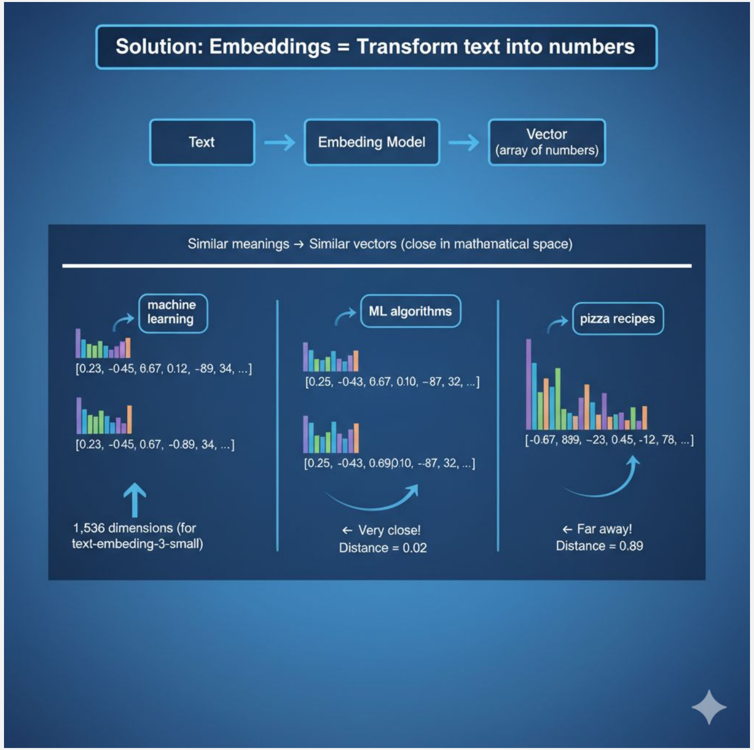
Foundations

What Are Embeddings?

The Problem: Computers don't understand text meaning

"machine learning algorithms"
"ML techniques"
"deep neural networks"

Are these similar? How do we compute that?



How Similarity Search Works

The Vector Search Process

1. INDEXING

Documents → Embedding Model → Store vectors

Doc 1: "Python tutorial" → [0.2, -0.4, 0.6, ...] → Store

Doc 2: "Java programming" → [0.3, -0.5, 0.5, ...] → Store

Doc 3: "Cooking recipes" → [-0.8, 0.9, -0.2, ...] → Store

2. SEARCHING

User Query: "learn Python" → [0.21, -0.42, 0.58, ...]

Compute distances to ALL stored vectors:

-Distance to Doc 1: 0.03 ✓ (very close!)

-Distance to Doc 2: 0.15 (somewhat close)

-Distance to Doc 3: 0.91 (far away)

Similarity Metrics

-Cosine similarity: Angle between vectors (most common)

-Euclidean distance: Straight-line distance

-Dot product: Vector multiplication

Embedding Quality Matters

CIKLUM

🏛️ EVERLAW (Legal Discovery - 1.4M documents)

87% accuracy

💬 MINDLID

82% **3 recall**

📧 INTERACTION CO (Email Assistant - 100 emails)

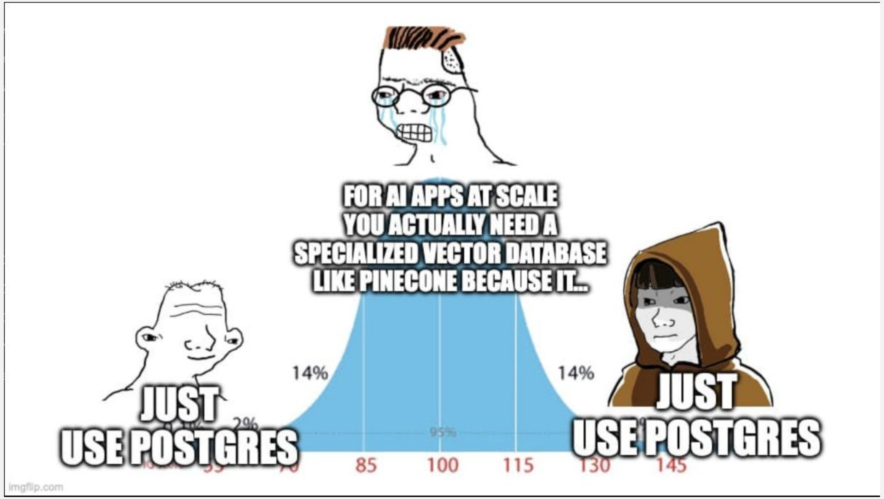
21.45s embedd

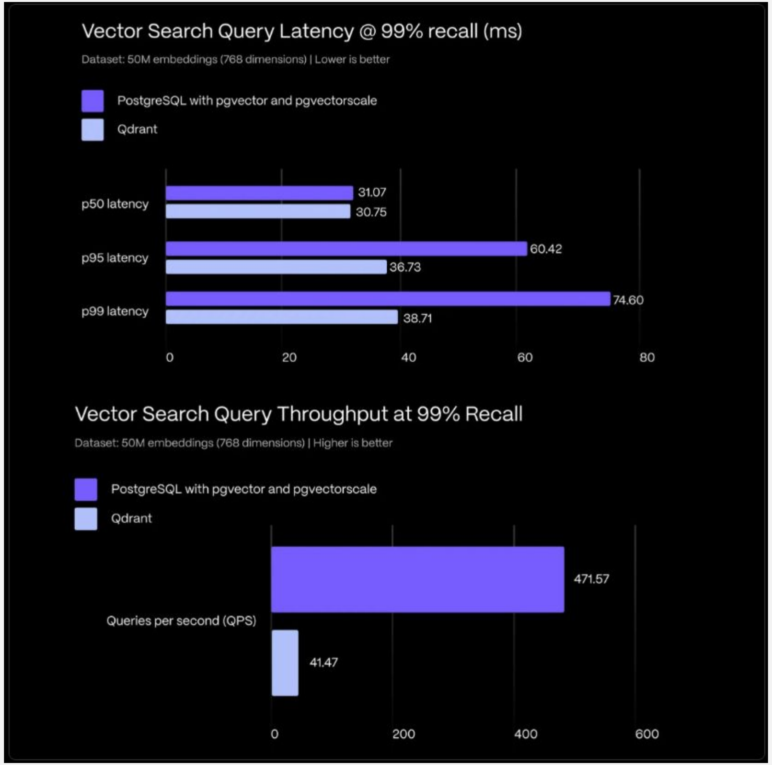
Model	Provider	MTEB	Dims	Price/1M
stella_en_1.5B_v5	NovaSearch	71.54	1024-8192	Free (OSS)
gemini-embedding-001	Google	68.32	3072	$0.15
text-embedding-3-large	OpenAI	64.6	3072	$0.13
text-embedding-3-small	OpenAI	62.3	1536	$0.02

https://huggingface.co/spaces/mteb/leaderboard

PostgreSQL Revolution

PostgreSQL and Pgvector: Now Faster Than Pinecone, 75% Cheaper, and 100% Open Source





From MVP to Scale

Mid Size High-Performance Vector Search

CIKLUM

Architecture & Performance

-Embedded database (runs in-process with your application)
-112 QPS at 10M vectors
-Python and JavaScript SDKs
-Open-source (Apache 2.0 license)

Memory Optimization:

-In-memory and persistent storage modes
-Optimized for datasets under 1M vectors
-Simple distance functions (cosine, L2, inner product)

Advanced Capabilities:

-Zero-configuration setup (pip install chromadb)
-Metadata filtering
-Multiple collection support

chroma

Architecture & Performance

-Rust-based for maximum speed
-626 QPS at 99.5% recall (1M vectors)
-Sub-3ms p95 latency
-Open-source (Apache 2.0 license)

Memory Optimization:

-Binary Quantization: 40x memory reduction - Scalar quantization
-Product quantization

Advanced Capabilities:

-Native hybrid search (dense + sparse vectors)
-Advanced payload filtering
-Distributed architecture for horizontal scaling
-RESTful and gRPC APIs

qdrant

Architecture & Performance

-Built for massive scale (billions of vectors)
-2,098 QPS at 100% recall (10M vectors)
-Sub-10ms latency on performance configurations
-Open-source (Apache 2.0 license)

Memory Optimization:

-Int8 compression: 75% memory savings
-RabitQ 1-bit quantization
-Multiple quantization strategies

Advanced Capabilities:

-GPU acceleration support
-Up to 100,000 collections per cluster
-Distributed architecture with query node separation
-Multiple distance metrics and index types (HNSW, IVF, DiskANN)

Milvus

Advanced RAG patterns

The Multi-Hop Reasoning Problem

CIKLUM

Complex Query:
"What are the regulatory implications of our Q3 marketing strategy for the European market?"

Traditional RAG fails

Retrieves isolated chunks:

- X -"Q3 marketing strategy increased social media spend"
- X -"European market regulations overview"
- X -"GDPR compliance requirements"

Problem: Can't connect the dots!
Marketing → Budget → Compliance → GDPR → Europe

This is where we need Graph RAG

GraphRAG Solution

CIKLUM

neo4j

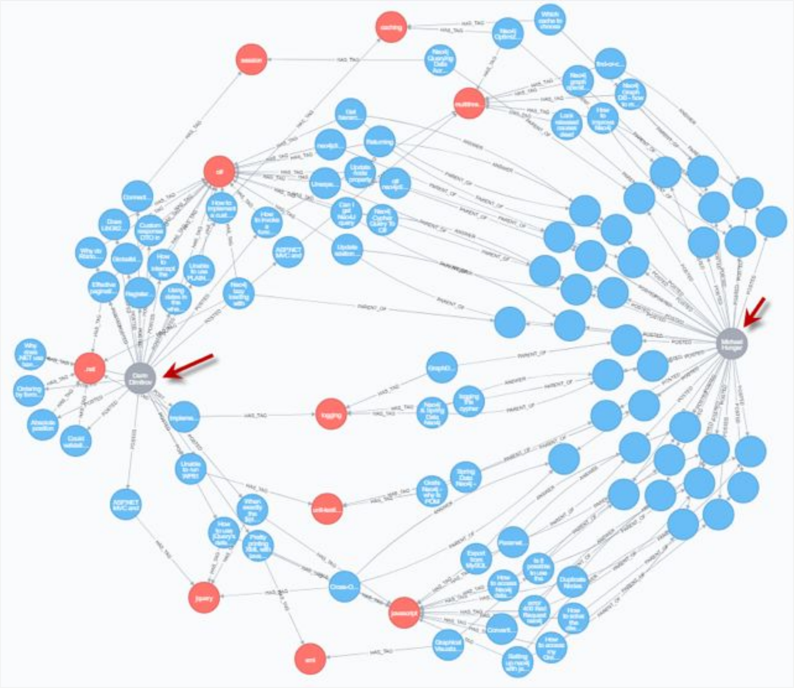
ArangoDB

FalkorDB

Amazon Neptune

MEM GRAPH

TigerGraph



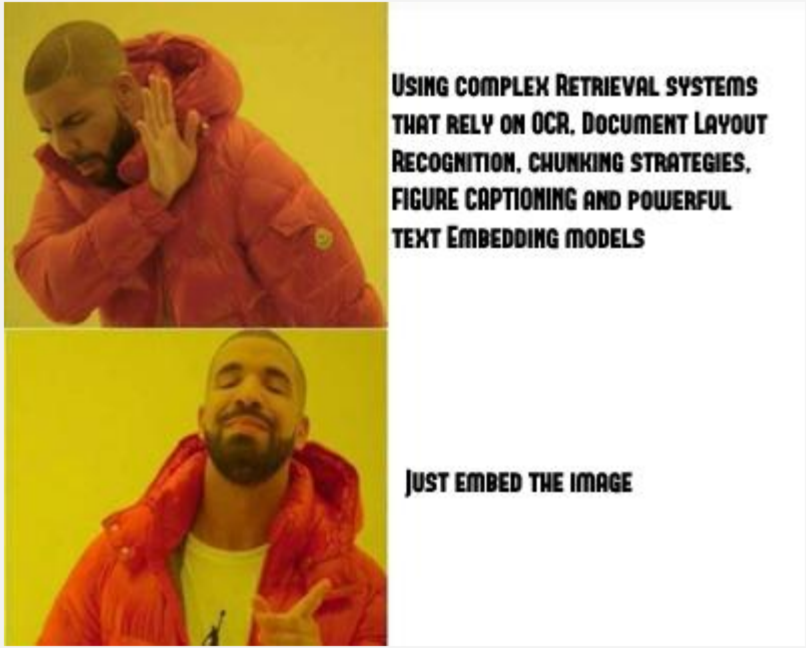
The PDF Processing Problem

CIKLUM

Traditional RAG Pipeline for PDFs

- ☒ OCR → Errors with complex layouts
- ☒ Layout detection → Misses table structures
- ☒ Figure captioning → Expensive specialized models
- ☒ Chunking → Loses visual context
- ☒ Embed text only → No visual information

Result: 40-60% information loss on visual documents

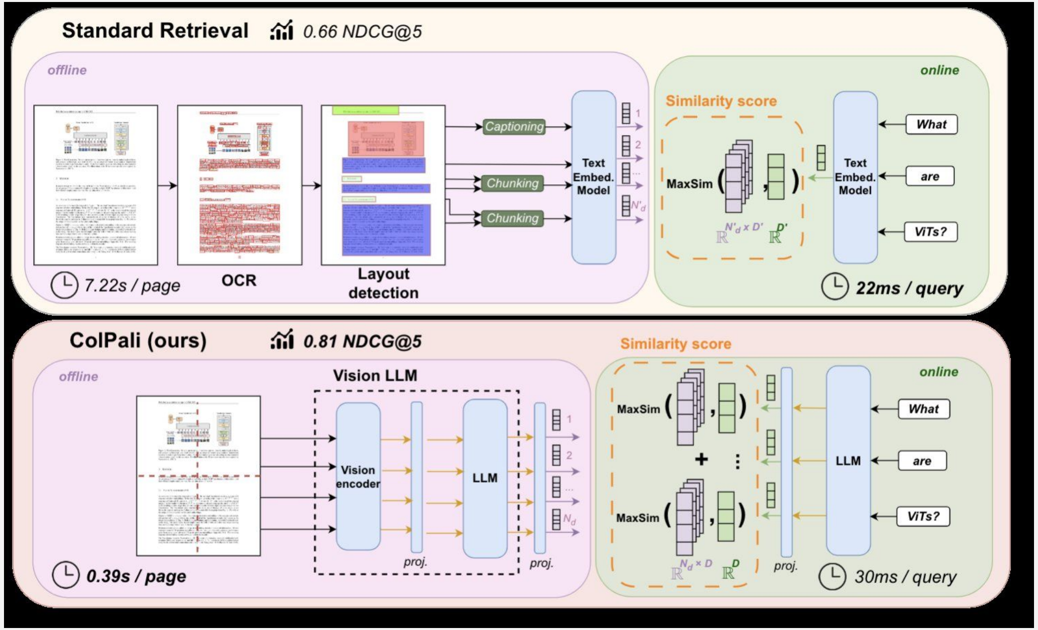


ColPali Revolution

Treat Pages as Images (No OCR!)

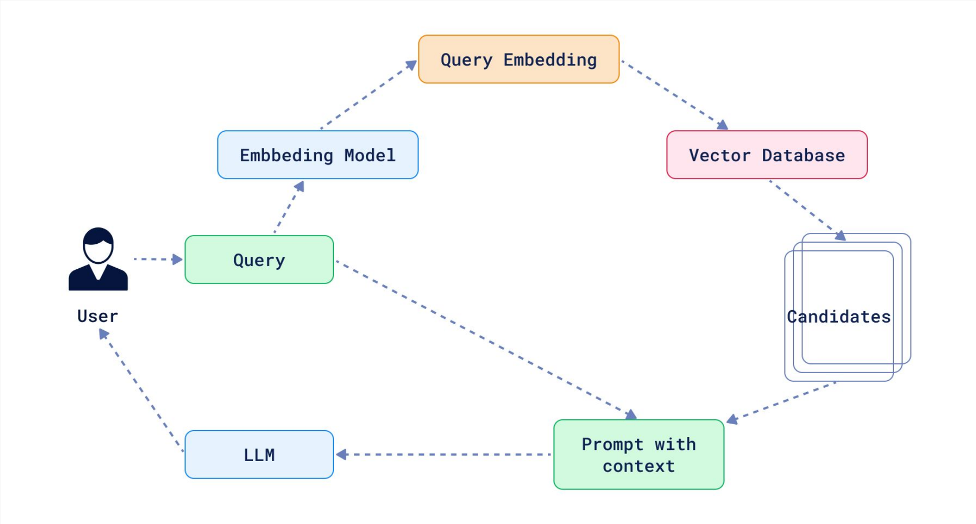
Architecture

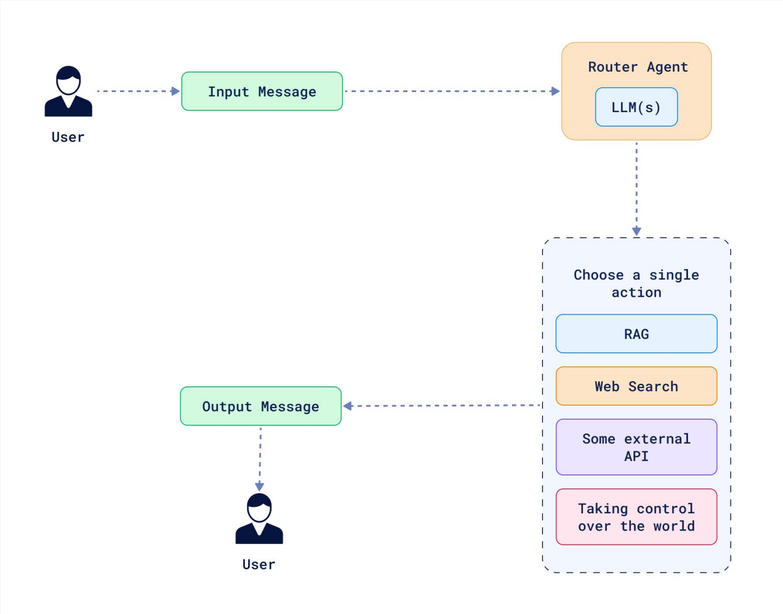
- PaliGemma-3B Vision Language Model
- 32×32 patches = 1,024 patch embeddings per page
- ColBERT late interaction matching



What Makes RAG "Agentic"

CIKLUM





Agentic Patterns

CIKLUM

Query Routing

User: "What's the latest on AI regulation?"

Router: "latest" detected → Route to Web Search ✓

User: "What's our Q3 marketing budget?"

Router: "our" = internal → Route to Vector DB ✓

Query Decomposition

Complex: "Compare our Q3 to industry and predict Q4"

Agent breaks down:

- "Our Q3 metrics" → Internal DB
- "Industry Q3 benchmarks" → Web Search
- "Q4 factors" → Internal DB

Then synthesizes all results

Self-Correction

1. Retrieval
2. Grade docs
3. Score < threshold?
4. Rewrite query
5. Retry
6. Grade again

✅ **Use When:** Queries span sources, complexity varies

❌ **Don't Use:** Latency-critical (<500ms SLA), simple retrieval

Hybrid Search + Reranking Power

Vector-only: Misses exact matches, acronyms
BM25-only: No semantic understanding

Microsoft Azure AI Search Benchmarks:

-Vector-only: 43.8 NDCG@3

-Hybrid: 48.4 NDCG@3 (+10.5%)

-Hybrid + Reranker: 60.1 NDCG@3 (+37.2%)

Anthropic Contextual Retrieval Study:

-Baseline failure rate: 5.7%

-Hybrid + Reranker: 1.9%

-67% reduction in retrieval failures

2 Stage Architecture Rerank:

Stage 1:

Similarity search → Top 100

Stage 2:

Re-ranking pipeline → Top 5-10

✅ **Use When:** Queries span sources, complexity varies

❌ **Don't Use:** Latency-critical (<500ms SLA), simple retrieval

Production readiness

Production Best Practices

CIKLUM

Do's

✅ 1. Use Hybrid Search (vector + BM25) as baseline

✅ 2. Implement metadata filtering (security-critical)

✅ filter = {"department": "finance", "access_level": user.role}

✅ 3. Monitor retrieval quality (recall@k, NDCG)

✅ Tools: TruLens, LangSmith, DeepEval

✅ 4. Keep embeddings fresh (re-index on doc changes)

✅ 5. Evaluate systematically (not "vibes-based")

Don'ts

❌ 1. Vector-only search (use hybrid)

❌ 2. Ignore access controls (data leakage = lawsuit)

❌ 3. Overload context window (keep <50%)

❌ 4. Skip evaluation frameworks (doesn't scale)

❌ 5. Neglect data freshness (stale = wrong answers)

CIKLUM

Thank you!