# Kompile:

## E2E Building Blocks for RAG Infra

by Adam Gibson

presented on Apr 15 2026

# AI Agents.
# What are they for?
# How is RAG related?

# THE THREE MAIN USE CASES FOR AI AGENTS

## 1. AGENTIC ACTIONS & CODING
(Act on Your Behalf)

Agents execute complex tasks, reducing operational latency to affect on copires, and aramitation man multi-step actions.

Focuses on coding, full-stack tasks, and multi-step actions.


**OPTIMIZED WORKFLOW EXECUTION**
Breaking down and executing complex multi-step plans.


**CODING & DEV SUPPORT**
Code generation, debugging, refactoring, and integration tasks.

**AGENT-ENVIRONMENT INTERACTION**
Managing interfaces with specific protocols and environments.

**TASK-SPECIFIC ACTIONS**
From email orchestration (triating/responding) to CRM data management.

## 2. IDEATION & CONTENT GENERATION
(Roleplay & Brainstorming)

Agents synthesize content and conceptualize ideas across all communication formats, in brainwrize.

Focuses on creative and strategic problem-solving.


**IDEATION & CONCEPTUALIZATION**
Exploring new product features, creative problem-solving, and innovative ideas.

**ROLEPLAY & INTERACTIVE SCRIPTING**
Simulation design, training content generation, and character-driven interactions.

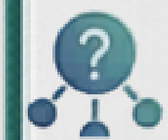

**CREATIVE CONTENT SYNTHESIS**
Generating compelling business documentation, pitch decks, and technical reports.

**MULTIMEDIA ASSET SYNTHESIS**
Video, audio, text-to-speech, and asset editing from natural language prompts.

## 3. INFORMATIONAL
(Research & Data Synthesis)

Agents handle and process large amounts of data to enable provide clear, grounded answers, to anes contact data.
Focuses on research,
Focuses on question-answering,
and sorting information.


**DEEP RESEARCH & SYNTHESIS**
Synthesizing information from diverse enterprise and external knowledge bases.



**QUESTION ANSWERING & GROUNDING**
Providing precise context and grounded answers.


**INFORMATION SORTING & CLASSIFICATION**
Managing and prioritizing large data repositories.

**CROSS-DOCUMENT ANALYSIS**
Identifying consensus and removing conflicts across multiple sources.

# WHAT IS RAG?

# WHAT IS RETRIEVAL-AUGMENTED GENERTION (RAG)?

## 1. KNOWLEDGE RETRIEVAL


**Aggregating Search Results.**
Collect relevant real-time data from diverse sources (e.g., search engines, company docs).

## 2. CONTEXT AUGMENTATION

**AGENT'S PROMPT**

How are agent architectures evolving?

**Context**

AI Agent

**AGENT'S PROMPT**

**Pasting Context into Prompt.**
Enriching the agent's base prompt with specific, relevant facts and documents.

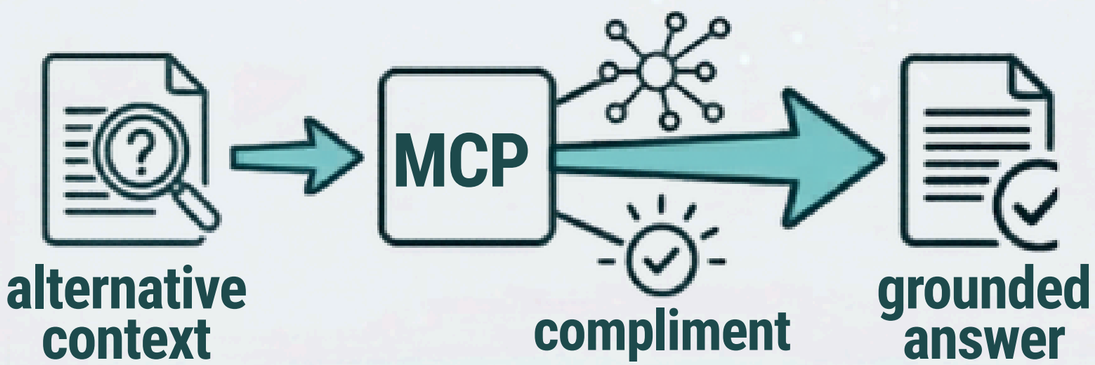
## 3. INFORMED RESPONSE

**INFORMED RESPONSE**

- Based on the retrieved context, current evolutions in agent architectures include...
- tool-use agents,
- re-planning loops, adtitops, and
- self-correction.

**Generating Grounded Answers.**
Producing factual and up-to-date responses grounded in the retrieved information.

# MCP:
# Both an Alternative and Compliment to a RAG System.



# MCP VS RAG: WHEN *NOT* TO USE RAG

## MCP (EXTERNAL DATA ACCESS)

### OPTIMIZED PRECISION & EFFICIENCY


**DIRECT TOOL CALLS:**
Agents pinpoint exact external data points via clean APIs.

**CHEAPER DATA RET RETRIEVAL:**
No large-scale embedding creation or vector database.

**BETTER TOOL INTEGRATION:**
Agents interact native with standard tool interfaces.


**UP-TO-DATE INFORMATION:**
Immediate access to real-time, dynamic data sources.

## KEY DIFFERENCES

**MCP**: Precise, Sparse, Structured Data via Tool Calls

**RAG**: Context, Neural Fusion via Vector Embedding

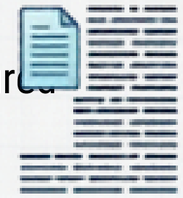
**MCP**: Lower Latency, Real-Time Access

**RAG**: Document & Context Focus

## RAG (KNOWLEDGE GROUNDING)

### KNOWLEDGE GROUNDING: USE CASES

**USE RAG FOR:**
Massive, dense, un-structured knowledge bases (internal wikis, manuals).



**USE RAG FOR:**
Context fusion across complex document topologies.

**RAG** IS NOT ALWAYS THE ANSWER:
Complexity and cost increase for precise, sparse, structured data retrieval.


**1. Precise Sparse Data Access**
MCP's advantages pinpoint exact external data points via clean API seHls.

**2. Contextual Document Recall**
RAG's strength of Context fusion & document topologies inference.

**3. System Performance & Cost**
Latency, lower latency, Real-time time, efficiency, sparse, used for retribution.

**Sources:** Mahntenagunmmannschaft, 1. Juli. 20
2. Eldeckors 2023, itürs-b
Cited. Saponen, et al. Impanco's: 20

# Quick RAG Architecture Overview

# RAG Operations

## READ (QUERY) SIDE

- Formulate specific search query
- Rank & retrieve multi-source results
- Form an accurate, contextual LLM response
- Summarize and ground generated answers

## WRITE (INGESTION) SIDE

- Index and process incoming raw data
- Create relationships for Graph RAG
- Store metadata & relationship summaries
- Maintain and update knowledge base

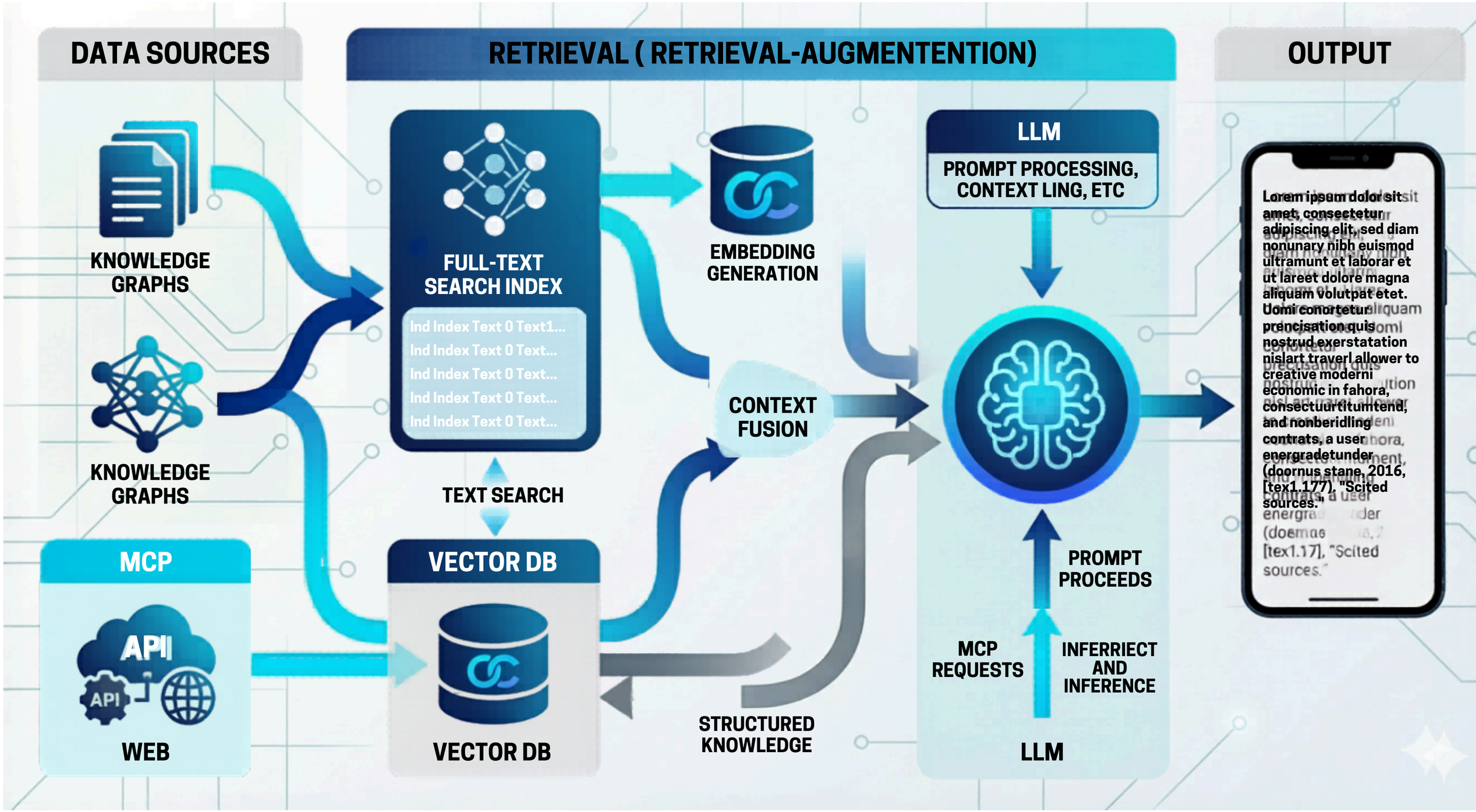
# Phases of RAG (Query & Ingestion)

## READ (QUERY) SIDE

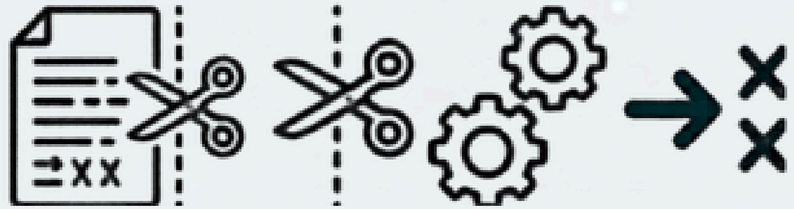
- Look up relevant entries with semantic search
- Look up relevant entities with graph search
- Rerank retrieved results
- Generate contextual response

## WRITE (INGESTION) SIDE

- Chunking: split raw data into manageable parts
- Indexing/Vector Insertion & Creation
- Entity Creation & Extraction
- Graph Relation Creation & Extraction


DATA SOURCES
RETRIEVAL ( RETRIEVAL-AUGMENTENTION)
OUTPUT
KNOWLEDGE GRAPHS
KNOWLEDGE GRAPHS
MCP
API
API
WEB
FULL-TEXT SEARCH INDEX
Ind Index Text 0 Text1...
Ind Index Text 0 Text...
Ind Index Text 0 Text...
Ind Index Text 0 Text...
Ind Index Text 0 Text...
TEXT SEARCH
VECTOR DB
VECTOR DB
EMBEDDING GENERATION
CONTEXT FUSION
STRUCTURED KNOWLEDGE
LLM
PROMPT PROCESSING, CONTEXT LING, ETC
PROMPT PROCEEDS
MCP REQUESTS
INFERRIECT AND INFERENCE
LLM
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonunary nibh euismod ultramunt et laborar et ut lareet dolore magna aliquam volutpat etet. Uomi conortetur prencisation quis nostrud exerstatation nislart traverl allower to creative moderni economic in fahora, consectuurtitumtend, and nonberidling contrats, a user energradetunder (doornus stane, 2016, [tex1.177). "Scited sources."

# Phase Highlight: Tokenization/Chunking



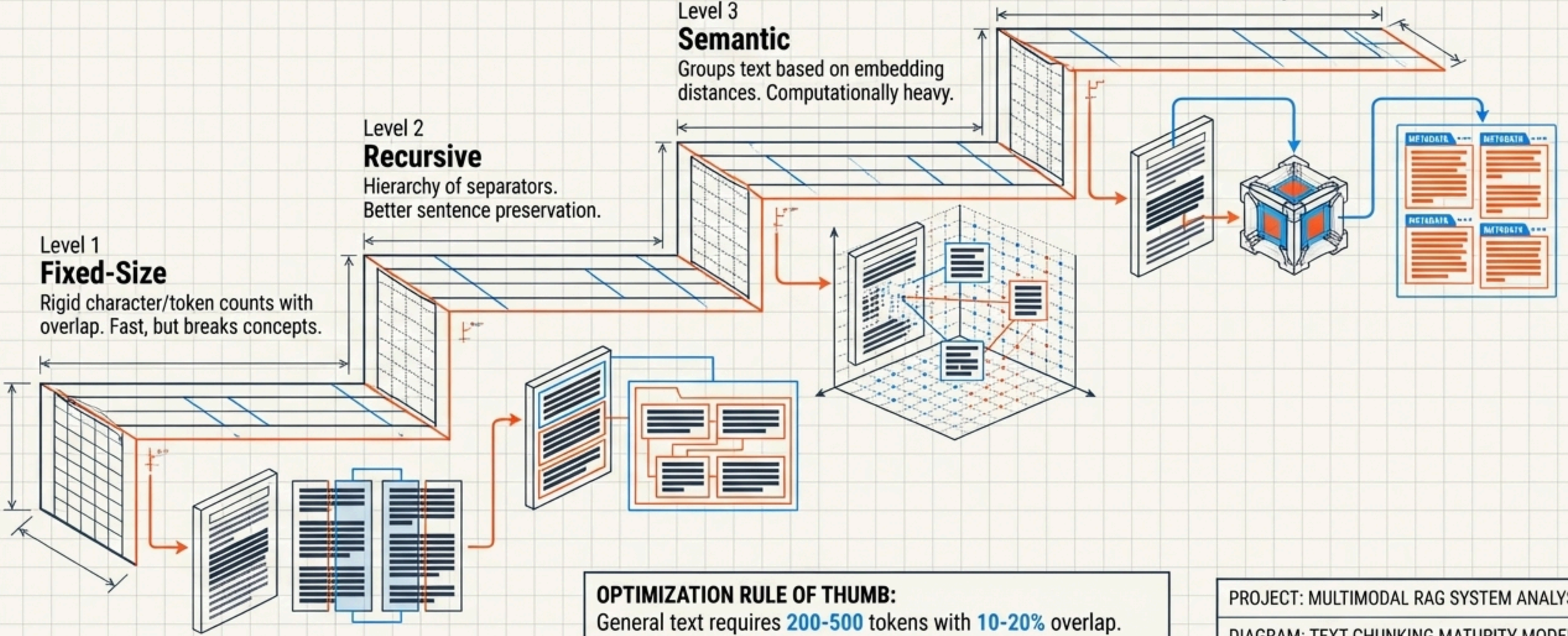
# TEXT CHUNKING MATURITY MODEL

Level 1
**Fixed-Size**
Rigid character/token counts with overlap. Fast, but breaks concepts.

Level 2
**Recursive**
Hierarchy of separators.
Better sentence preservation.

Level 3
**Semantic**
Groups text based on embedding distances. Computationally heavy.

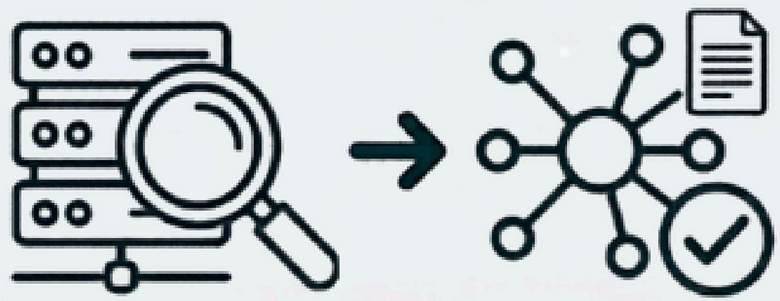
Level 4
**Context-Enriched / AI-Driven**
LLM detects breakpoints and injects metadata.



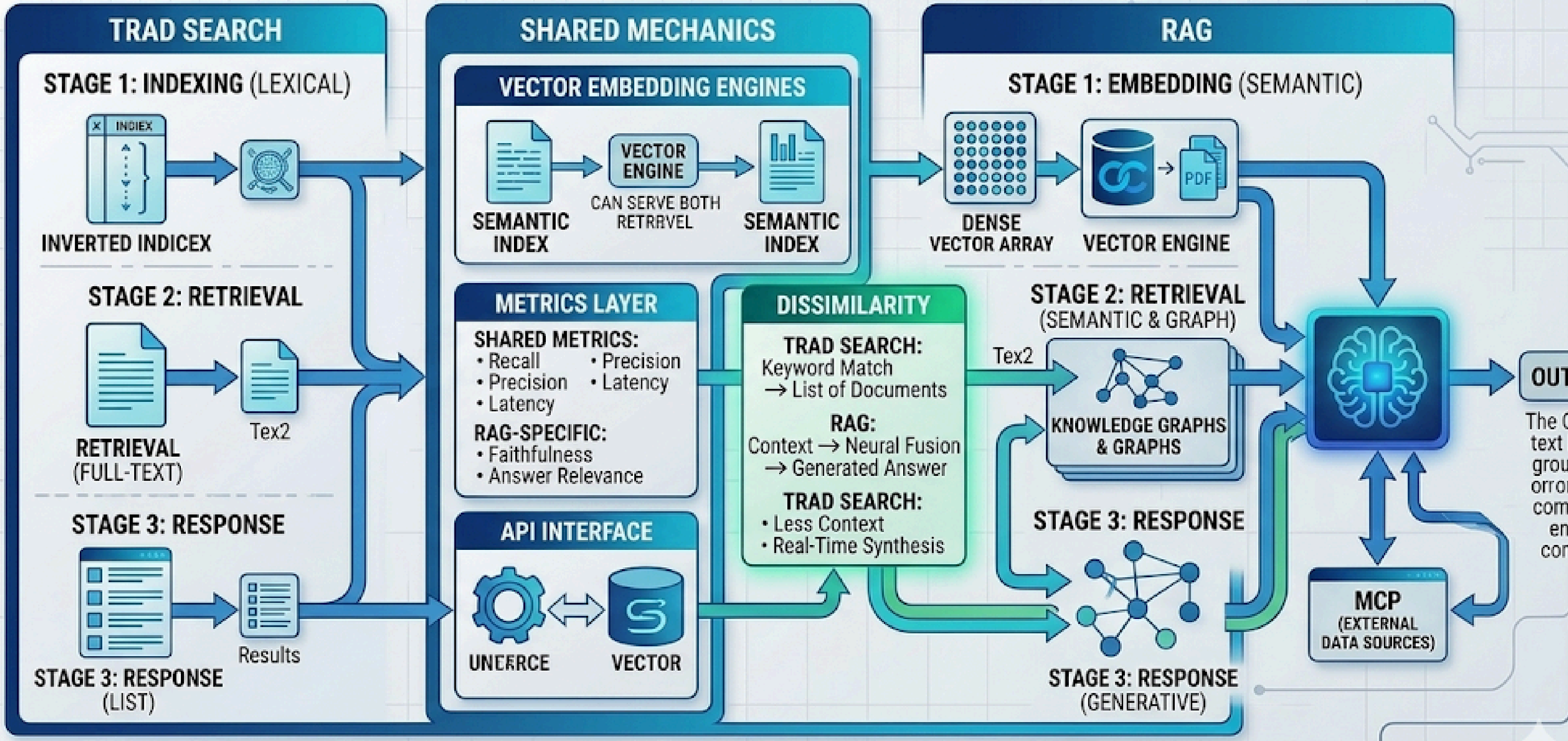
**OPTIMIZATION RULE OF THUMB:**
General text requires 200-500 tokens with 10-20% overlap.
Code/Technical requires 100-200 tokens with 15-25% overlap.

| | |
|---|---|
| PROJECT: MULTIMODAL RAG SYSTEM ANALYSIS | |
| DIAGRAM: TEXT CHUNKING MATURITY MODEL | |
| SCALE: 1:100 | DATE: OCT 26, 2024 |

# Phase Highlight: Retrieval



# TRADITIONAL SEARCH vs RAG: ARCHITECTURAL COMPARISON




**RETRIEVIAL:** Indexing text, structure, and ONTOLOGY delatioships, graph, and attachment relations.

**CONTEXT FUSION:** Assembling in ɑcument, anttachment ɾontext.

**LLM INFERENCE:** Forming response grounded in ɾ complete TOPOLOGY, and lead grouplette results.

**Sources:** Mahitennagument et al. 2021. 2. Eldeokors 2023, ItirS blue. Cited: Soporgon, et al. Innapoca0; blue.

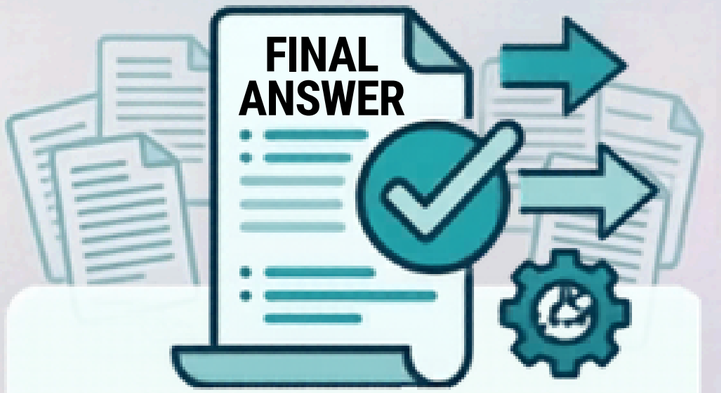
# RERANKING PIPELINE (NORMAL & GRAPH)


## 1. INITIAL RETRIEVAL (NORMAL & GRAPH)

- RAG queries both standard and graph-based data sources.
- Returns raw potential results (e.g., text snippets OR nodes & edges).
- Initial set size: *n* candidates.


## 2. DYNAMIC RERANKING ENGINE

- **Cross-Encoder Analysis:** Precise, context-aware query-document scoring.
- **Metadata-Rich Scoring:** Incorporates entity types, timestamps, and relationship weights.
- **Hybrid Fusion:** Combines scores from multiple specialized models.




## 3. REFINED RESULTS DELIVERY

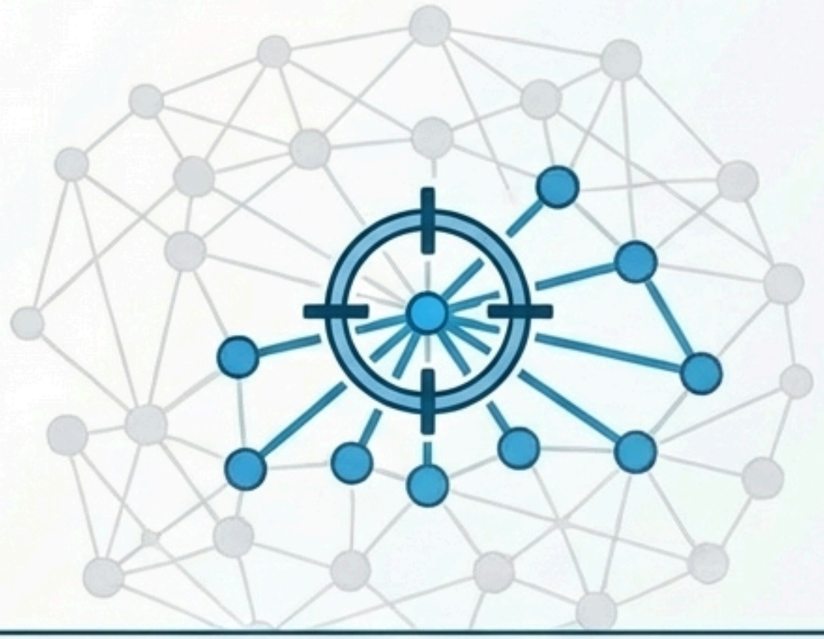
- **Top-k Final Answers:** Prioritizes the most relevant answers, relationship summaries (Graph), or text summaries (Normal).
- **Complete Contextual Grounding:** Ensures answer accuracy and reduces hallucinations.
- **Direct Actionable Insights:** Ready for end-user synthesis further processing.

# Graph RAG: Sorting out relationships between entities.

# Retrieval Modalities: Navigating the Topology

## Local Search

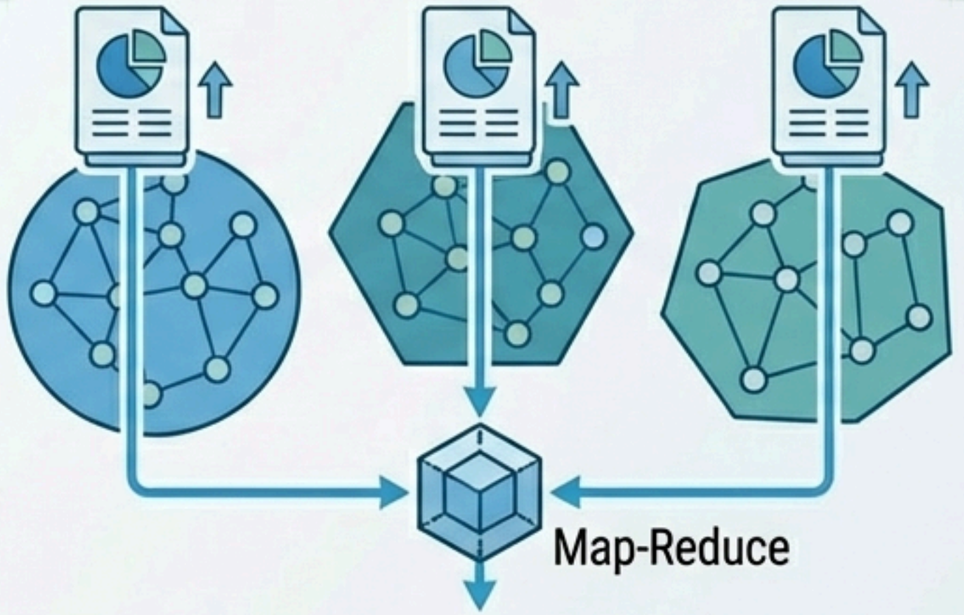
Entity-Centric Traversal



Best for targeted, specific questions requiring high relational depth (e.g., What are the dependencies of the auth module?).

## Global Search

Dataset-Wide Sensemaking



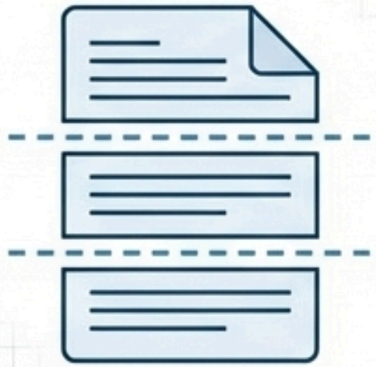

Best for abstract, thematic questions across the corpus (e.g., What are the main risk factors across the entire supply chain?).

**Emerging Modality:** DRIFT Search (Dynamic Reasoning and Inference with Flexible Traversal) balances both methods dynamically.

# Computing the Graph: The Indexing Engine

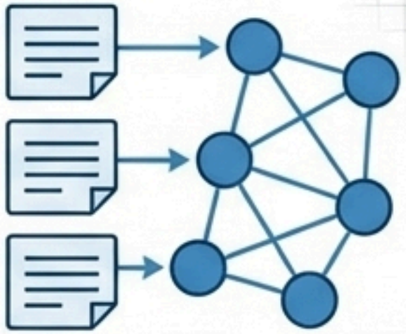
### 1. Text Segmentation

Documents shatter into precisely sized TextUnits (1200 tokens).



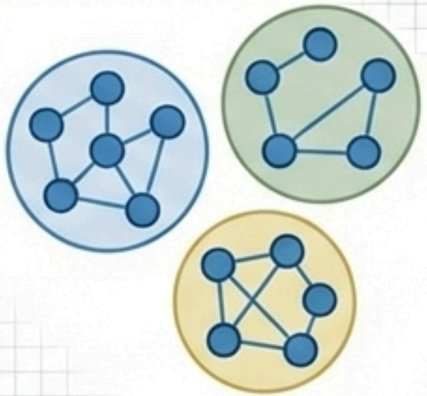
### 2. Entity & Relation Extraction

LLMs extract nodes (people, concepts) and edges (relationships).



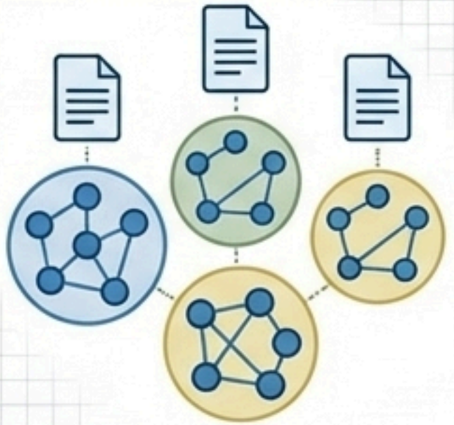
### 3. Community Detection (Leiden Algorithm)

Nodes are mathematically clustered into hierarchical groups.



### 4. Hierarchical Summarization

Generation of bottom-up Community Reports.



Graph construction costs can be 10–100x higher than standard RAG indexing, necessitating careful architectural choices.

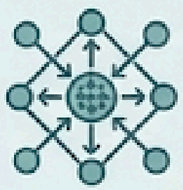
# Kompile
## AN Overview

# Kompile: Advanced RAG System - Comprehensive Provider Landscape

## 1. Vector Stores Role
- Indexing & Search

Chroma · Lucene · pgvector

- **Primary:** kompile-vectorstore-anserini (Lucene-backed HNSW)
- **Alternative:** kompile-vectorstore-pgvector (PostgreSQL pgvector extension)
- **Self-Hosted:** Chroma, Vespa · Pinecone
- **Enterprise-Managed:** Pinecone, Milvus, Qdrant · qdrant

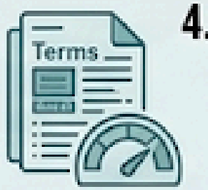


## 2. Embeddings Role
- Dense Encoding

- **HuggingFace Native (Rust-based):** tokenizers-rust/ libtokenizers
- **SameDiff Encoders** (bge-ev1.5, arctic, arctic-embed, etc.)
- **Proprietary APIs:** OpenAI, Anthropic, Google Gemini (via Spring AI)
- **Local Subprocess:** Sentence-Transformers (Python wraper)

## 3. LLM Providers Role
- Generation & Completion

- **OpenAI:** GPT-4o, GPT-3.5-turbo
- **Anthropic:** Claude 3.5 Sonnet, Opus
- **Anthropic:** Clarude 3.5 Sonnet, Opus
- **Google:** Gemini Pro, Gemini Flash
- **Meta:** Llama 3 (hosted via HF Inference API/Samediff-Rag)
- **Cohere:** command-r-v1 · cohere
- **Open-source (SameDiff):** Kompile-hosted GGUF/HF/Local via SameDiff DSP compiler

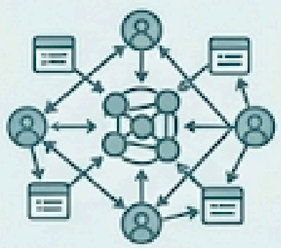



## 4. Search Engines Role
- Sparse & Hybrid IR

Anserini · Elastic

- **Primary: Anserini (Lucene IR Fork):** BM25 sparse + SameDiff dense hybrid
- **Enterprise:** ElasticSearch (Elastic IR/Vectors)
- **Search APIs:** Google Search, Bing Search (via Tools/Plugins)
- **Web Collections:** Wikipedia, Parquet, CommonCrawl, BEIR datasets

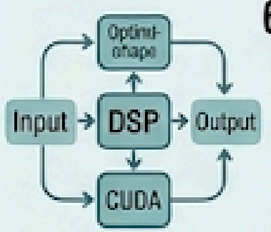
## INFRASTRUCTURE SERVICE LAYER & INTEGRATION (Spring AI)



## 5. Graph Stores Role - Structured RAG

neo4j · DGraph

- **Enterprise Graph:** Neo4j (via Kompile Neo4j graph tool)
- **Self-Hosted Graph:** JanusGraph, DGraph
- **Enterprise-Managed Graph:** Amazon Neptune




## 6. Model Execution Role - Core SameDiff DSP

- **Conversion & Optimisation:** ONNX/TensorFlow -> SameDiff (.sdz)
- GGML/GGUF Import & Metadata Extraction
- PEFT (LoRA), RLHF, DPO, KTO Alignment
- Triton/CUDA Fusion, Dead Code Elimination, Triton GPU Compilation
- OpenAI-Compatible API & MCP Server

GGUF · HF · NVIDIA

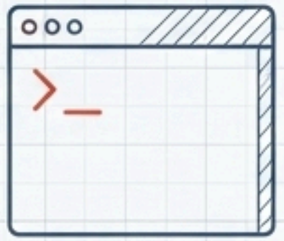
SameDiff Compiler Flow

## KNOWLEDGE BASE & PROVIDER CONNECTOR LAYER

A comprehensive, provider-aware AI/ML platform combining CLI tooling, RAG application framework, model lifecycle management, and a flexible execution framework, all designed for extensibility and universal integration with leading AI providers through plugin architecture and Spring AI connection

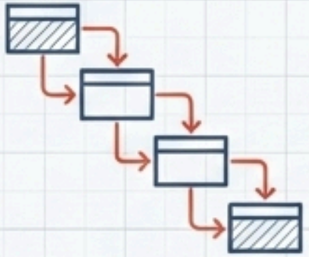
# End-to-End Operation: From Command to Control

## 1. Build



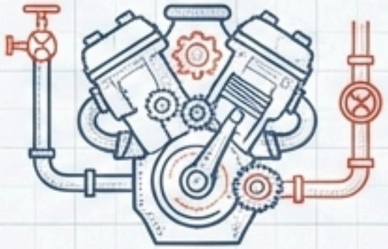
`kompile build app --preset=full`

## 2. Ingest



`kompile ingest /docs`

## 3. Execute



ReAct Agent retrieves via Anserini and generates via SameDiff.

## 4. Evaluate

Visualizing RAGAS evaluation benchmark scores.

Kompile provides a completely self-contained, offline-capable AI lifecycle. Build in the terminal, scale via native subprocesses, and rigorously evaluate in the browser.

# **CLI Chat:** The Multi-Agent REPL

## Multi-Agent System

Seamlessly switch between Claude, Codex, and Gemini within the same execution context.

kompile shell :: multi-agent repl

[Architect] ▾ Claude ▾

```
$ [Architect] kompile chat --agent=Claude
> Claude initialized. How can I help you structure your system today?

$ [User] Analyze the project structure and list all TODOs.
> Running bash command to find TODOs...
  $ grep -r "TODO" src/
  src/main/java/com/example/App.java: // TODO: Implement error handling here
  src/main/java/com/example/Service.java: // TODO: Refactor database connection
  src/test/java/com/example/AppTest.java: // TODO: Add more test cases

> Found 3 TODOs. Would you like me to prioritize them based on the
  architecture?
```

## Role Engine

Set agent personas to Architect, Devops, or Reviewer.

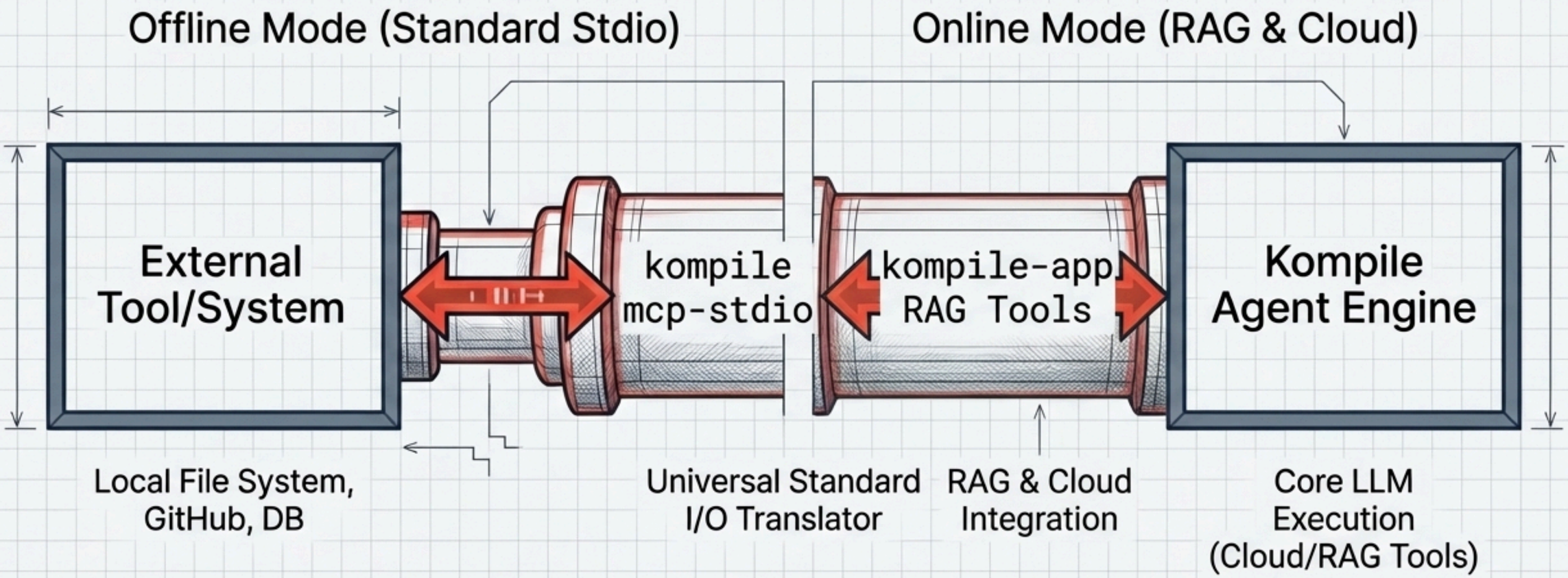
## Tool Integration

Native tool execution for Bash, web search, file edit, and memory management.

### Session Persistence

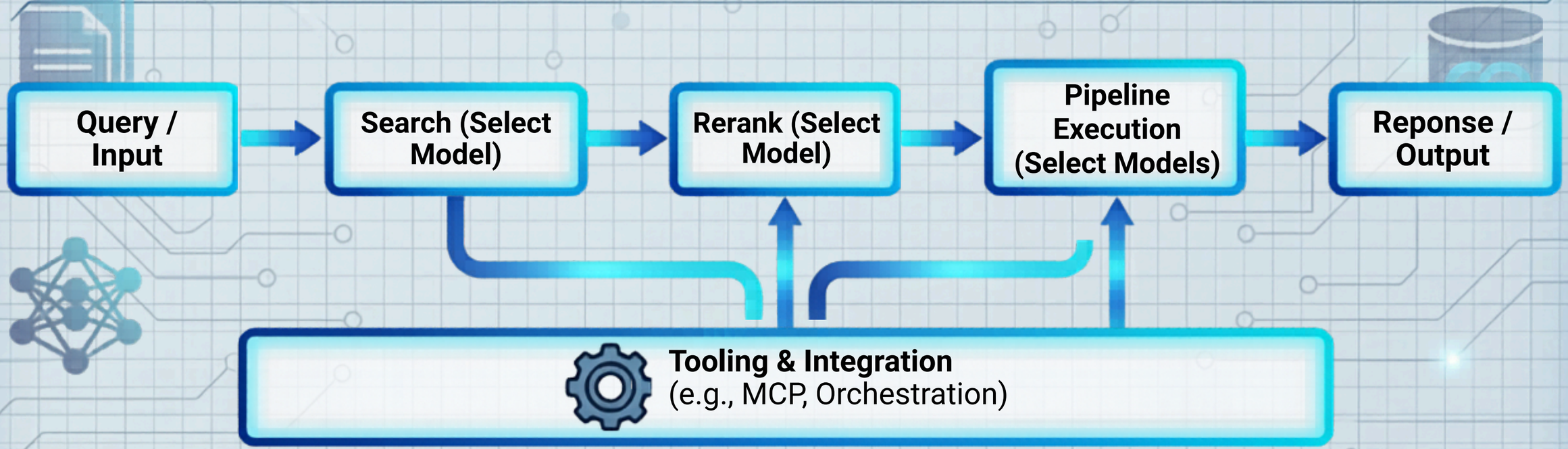
Use `kompile resume` for a multi-tab TUI to browse history, or `kompile session merge` to combine context across different agent architectures.

# Standardized Tooling via Dual MCP




Choose between offline '`mcp-stdio`' for local tool access or online '`kompile-app` RAG' for cloud-integrated tools, decoupling logic from the core agent.

# Model-First Approach: Full Control at Every Step

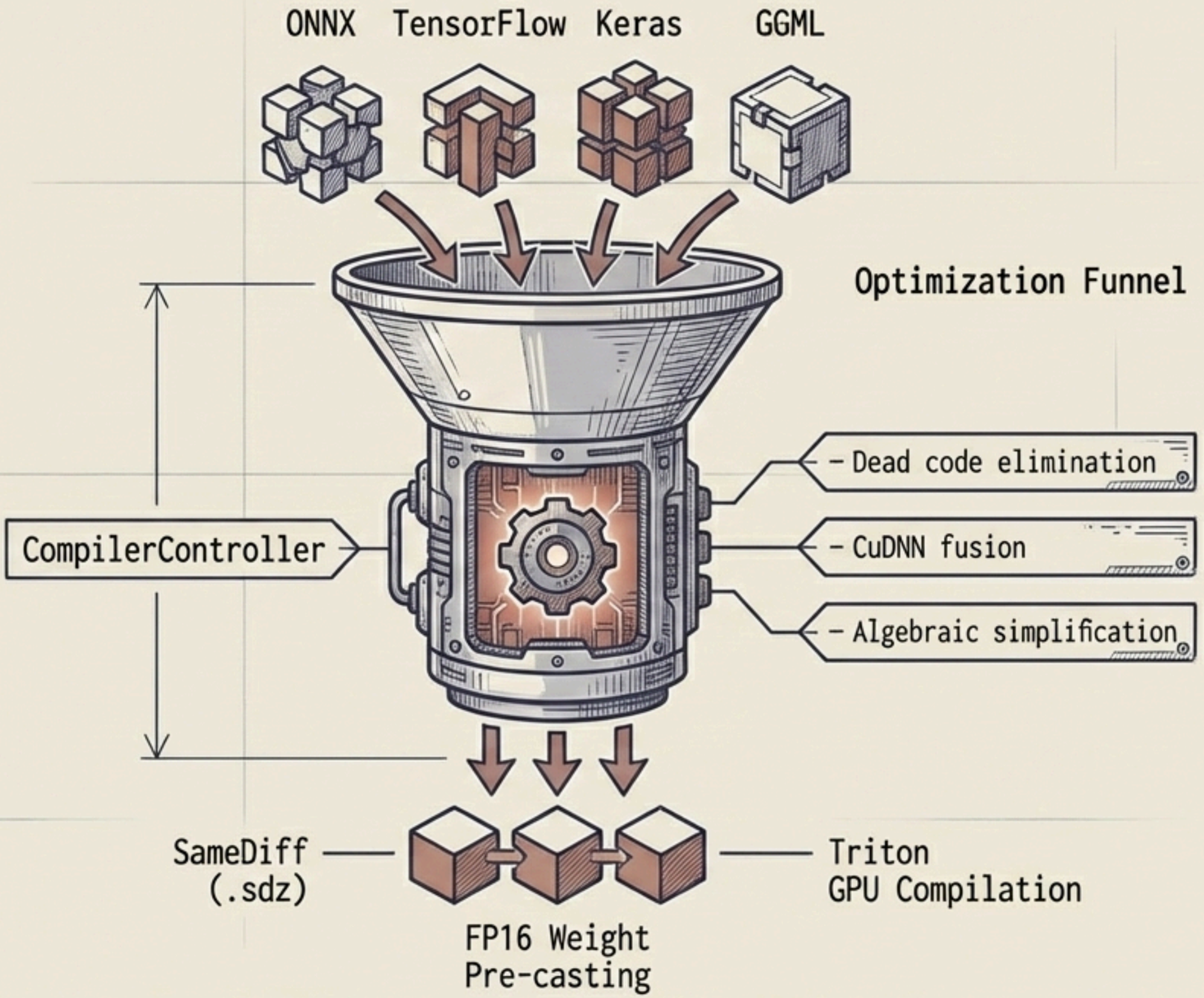



- **Granular Control:** Select and configure specific models for each execution stage (search, rerank, pipelines, generation).
- **Mix & Match:** Combine different models freely to optimize performance for specific tasks.
- **Seamless Integration:** Leverage MCP and other tooling for unified orchestration across all models and steps.

# Under the Hood: Maximum Inference Optimization

Swapping to local models only works if they run efficiently.

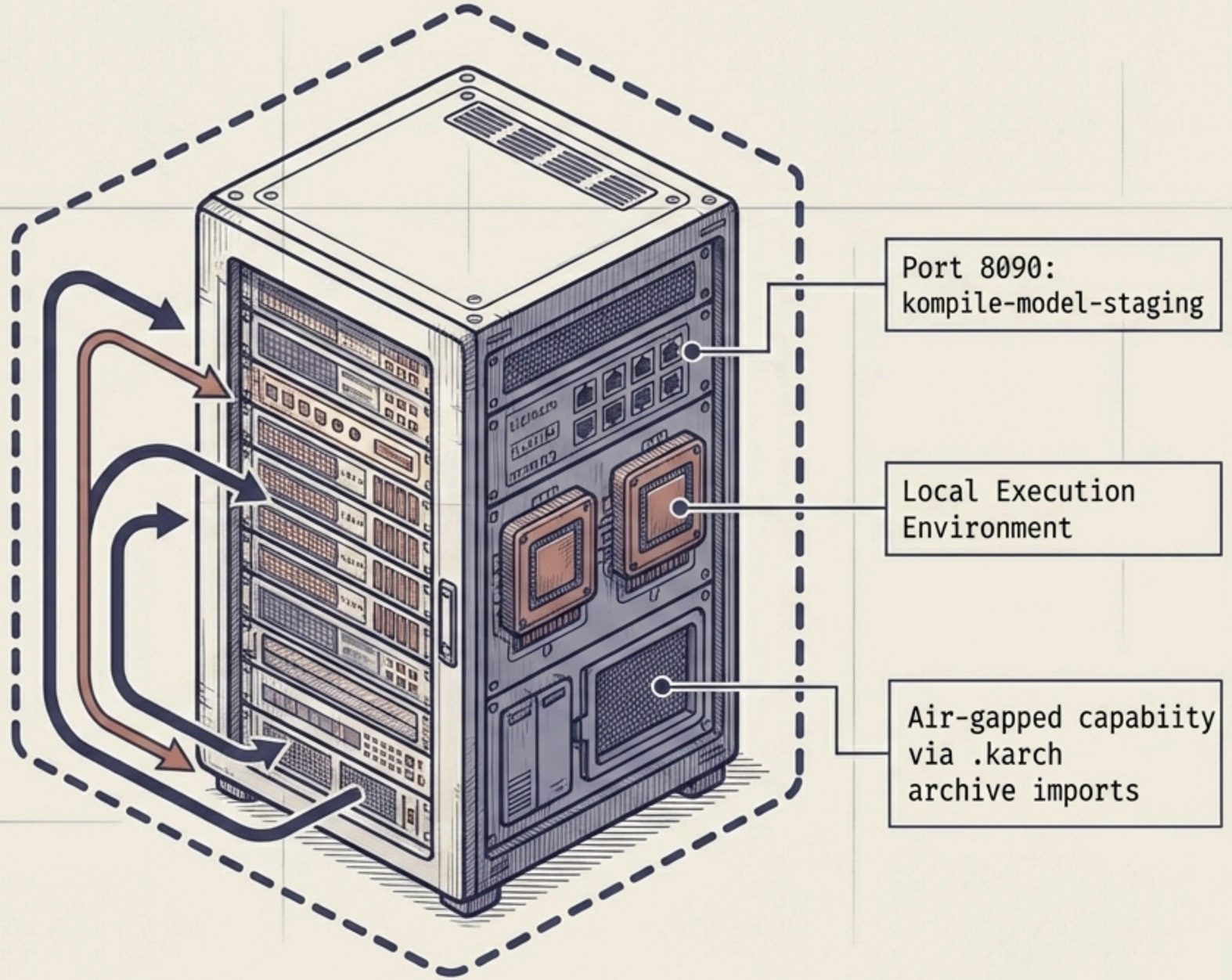
Kompile natively compiles fragmented raw models into optimized computation graphs, fusing operations and pre-casting weights to maximize performance on your specific hardware.



# The Model Control Engine

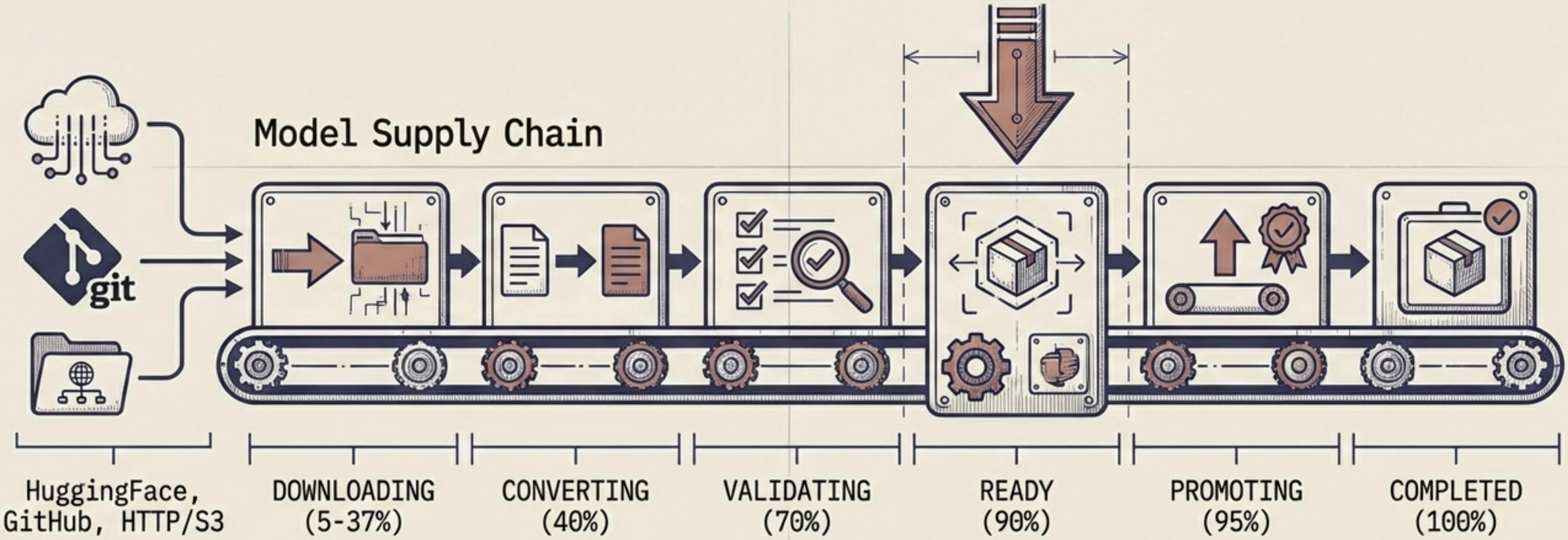
True sovereignty requires pulling models out of third-party clouds and into your own environment.

The Model Staging Server is a standalone Spring Boot service that manages the complete ML lifecycle entirely within your network—even operating fully air-gapped.



# The Automated Model Lifecycle

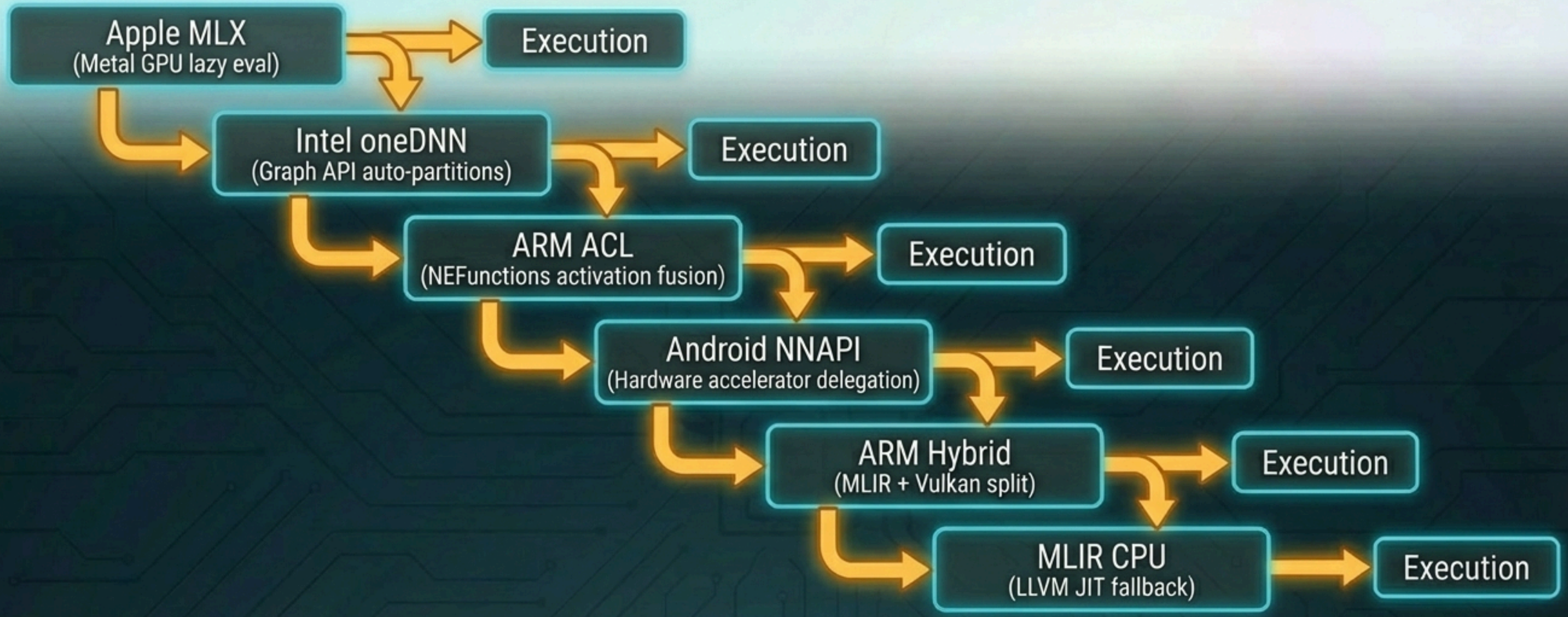
The `StagingController` orchestrates the exact state of proprietary or open-source models, completely abstracting the complexities of model acquisition and deployment.



# Model Execution

# CPU and Mobile Backend Cascade

On non-CUDA platforms, segments are intercepted by a deterministic priority cascade. Each backend applies platform-native fusion (e.g., MLX lazy evaluation on Apple Silicon, oneDNN graph partitioning on x86) using the exact same shape-key caching model as the GPU.



# Structural Matrix: GPU vs. CPU Compilation Models

| Dimension | GPU (Triton) | CPU Backends |
| --- | --- | --- |
| Fusion Granularity | Typed Sections (19 categories) | Whole-segment or backend-library decided |
| Code Generation | MLIR -> TTIR -> PTX | MLIR -> LLVM JIT or Native Library Call |
| Kernel Shape | Parallel Blocks / Grid | Sequential loops with SIMD vectorization |
| Memory Model | tt.ptr (Global memory) / SSA registers | memref (Main memory) / L1 cache |

# Inference Execution Models: The Industry Landscape

DSP occupies the optimal middle ground: fully compiled unlike llama.cpp, but far more deterministic and predictable than torch.compile's recompilation machinery.

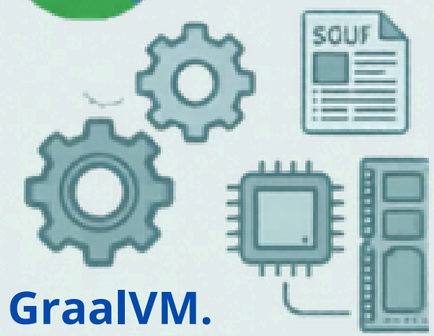
| Dimension | DSP | llama.cpp | torch.compile |
|---|---|---|---|
| Graph Construction | Compiled once, slot-indexed | Rebuilt every token | Captured via bytecode hijacking |
| Dynamic Shapes | Shape key hash recompilation | DAG rebuild per shape | SymInt symbolic ranges |
| Fusion Strategy | Section categories to JIT Triton | Strictly handwritten kernels | Automatic fusion scheduler |
| CUDA Graphs | Per-segment capture | Whole-graph, batch=1 only | Tree structures with shared pools |

# Compile on 2 levels. Your Applications customized. Your models optimized.

## 1. FULL AOT BINARY GENERATION & EXECUTION

**- Native Image & Native Infrastructure**

GraalVM.

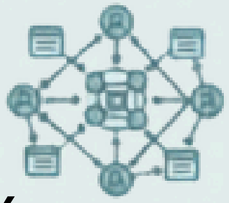



- **AOT Generation:** GraalVM Native Image compiler
- **Applications:** All RAG applications, web servers (Spring Boot), microservices (Quarkus)
- **Model Support:** AOT-comiled models (SameDiff DSP on JVM)
- **Performance:** Instant startup, minimal memory footprint
- **Infra Integration:** AOT-aware databases, vector stores (e.g., Lucene on JVM, Neo4j, pgvector via JVM driver)

## 2. MODEL COMPILATION PIPELINE & INFERENCE

**- SameDiff DSP & OptimizedRuntime**


ONNX



Sol

GGUF/HF

- **Conversion & Optimisation:** ONNX/SensorFlow
- **SameDiff (sdz),** FP16 weight casting, GGF metadata extraction
- **Training & Fine-tuning:** PEFT (LoRA), alignment (RLHF, DPO, KTO), evaluation
- **DSP Execution:** Cuda, Triton GPU compilation, dead code elimination, algebraic simplification
- **Airgap Support:** Model archiving (Karch) and promotion to **PRODUCTION**

**INFRASTRUCTURE SERVICE LAYER & INTEGRATION (SPRING AI)**

## 3. AOT-Aware Data Infrastructure

Lucene

neo4j

- HNSW Dense Vectors (Lucene on JV)
- AOT-aware pgvector SQL
- Neo4j Graph (via Native JVM tools)

neo4j

DGraph

## 4. AOT-Aware Model Tool Stack

GGUF/HF

ND4J

- tokenizers-rust/ libtokenizers
- Sentence-Transformers (Python subprocess)
- ND4J Workspace Cleanup & interruption handling

**KNOWLEDGE BASE & PROVIDER CONNECTOR LAYER**

A comprehensive, performance-aware AI/ML platform combining AOT-based execution and optimized model compilation through plugin architecture and Spring AI connection, all designed for universal integration with leading AI providers.

# Thank you!

adam@getkompile.com

Follow and Connect:

https://linkedin.com/in/agibsonccc

https:// github.com/deeplearning4j/deeplearning4j

https:// github.com/GetKompile