Pre-Read Document

Context Engineering, System Architecture, and RAG

This document prepares participants for sessions covering Context Engineering, RAG (Retrieval-Augmented Generation), RAG Data Ingestion, and System Architecture. It introduces foundational concepts required for deeper architectural and implementation discussions.

---

1. Context Engineering

1.1 What is Context in Generative AI?

In Generative AI systems, *context* refers to all information provided to a language model at inference time to guide output generation.

Context may include:

- System instructions
- User query
- Conversation history
- Retrieved documents
- Structured data
- Tool outputs
- Persistent memory

Models generate responses based solely on what is available in the current context window.

1.2 What is Context Engineering?

Context Engineering is the structured design and assembly of information supplied to an LLM before generation.

It involves:

- Selecting relevant information
- Filtering irrelevant or noisy data
- Structuring inputs clearly
- Controlling ordering and formatting
- Managing token budgets

It is a system-level discipline rather than a prompt-level technique.

1.3 Context Engineering vs Prompt Engineering

Prompt Engineering	Context Engineering
Focuses on crafting instructions	Focuses on information pipeline design
Optimizes wording	Optimizes data selection and structure
Often static	Dynamic and data-driven
Single interaction focus	End-to-end architecture focus

Prompt Engineering improves how instructions are written.
Context Engineering determines what information the model receives and how it is structured.

1.4 Context Engineering in Agentic AI

In Agentic AI systems, models may:

- Retrieve external knowledge
- Call tools or APIs
- Maintain memory

- Plan multi-step reasoning

Context Engineering ensures:

- Relevant memory is available
- Tool outputs are formatted correctly
- Retrieved content is ranked and filtered
- State is preserved across steps

Without structured context assembly, agent reliability decreases significantly.

1.5 Types of Context

1. Instruction Context – System-level behavior and constraints
2. User Context – Current query
3. Conversational Context – Chat history
4. Knowledge Context – Retrieved external documents
5. Tool Context – Outputs from APIs or tools
6. Memory Context – Persistent state

---

2. Developing Text-Based Generative AI Applications

2.1 Standard Application Flow

1. User input
2. Context assembly
3. Optional retrieval
4. Model generation
5. Post-processing

6. Response delivery

2.2 Typical Use Cases

- Enterprise knowledge assistants
- Document question answering
- Customer support automation
- Internal search systems
- Summarization pipelines

---

3. System Architecture (End-to-End Solutioning)

3.1 High-Level Architecture

User → Frontend → Backend API →

Context Assembly →

Retrieval Layer →

LLM →

Post Processing → Response

3.2 Architectural Layers

1. Interface Layer – Web, mobile, or API interface
2. Application Layer – Business logic
3. Retrieval Layer – Vector database and search
4. Model Layer – LLM inference
5. Data Layer – Document storage

3.3 Key Architectural Considerations

- Latency management
- Token limits
- Cost optimization
- Observability and logging
- Security and access control
- Scalability

---

4. Retrieval-Augmented Generation (RAG)

4.1 What is RAG?

RAG combines information retrieval with generative modeling. Instead of relying solely on model training data, relevant documents are retrieved dynamically at runtime.

4.2 RAG Architecture Overview

User Query
↓
Embedding Model
↓
Vector Database Search
↓
Top-K Relevant Documents
↓
Context Assembly
↓

LLM

↓

Grounded Response

4.3 Core Components

Embeddings

Text is converted into numerical vector representations.

Vector Database

Stores embeddings and enables similarity search.

Examples include:

- FAISS
- Pinecone
- Weaviate
- Chroma

Retriever

Fetches the most relevant chunks using similarity metrics.

Generator

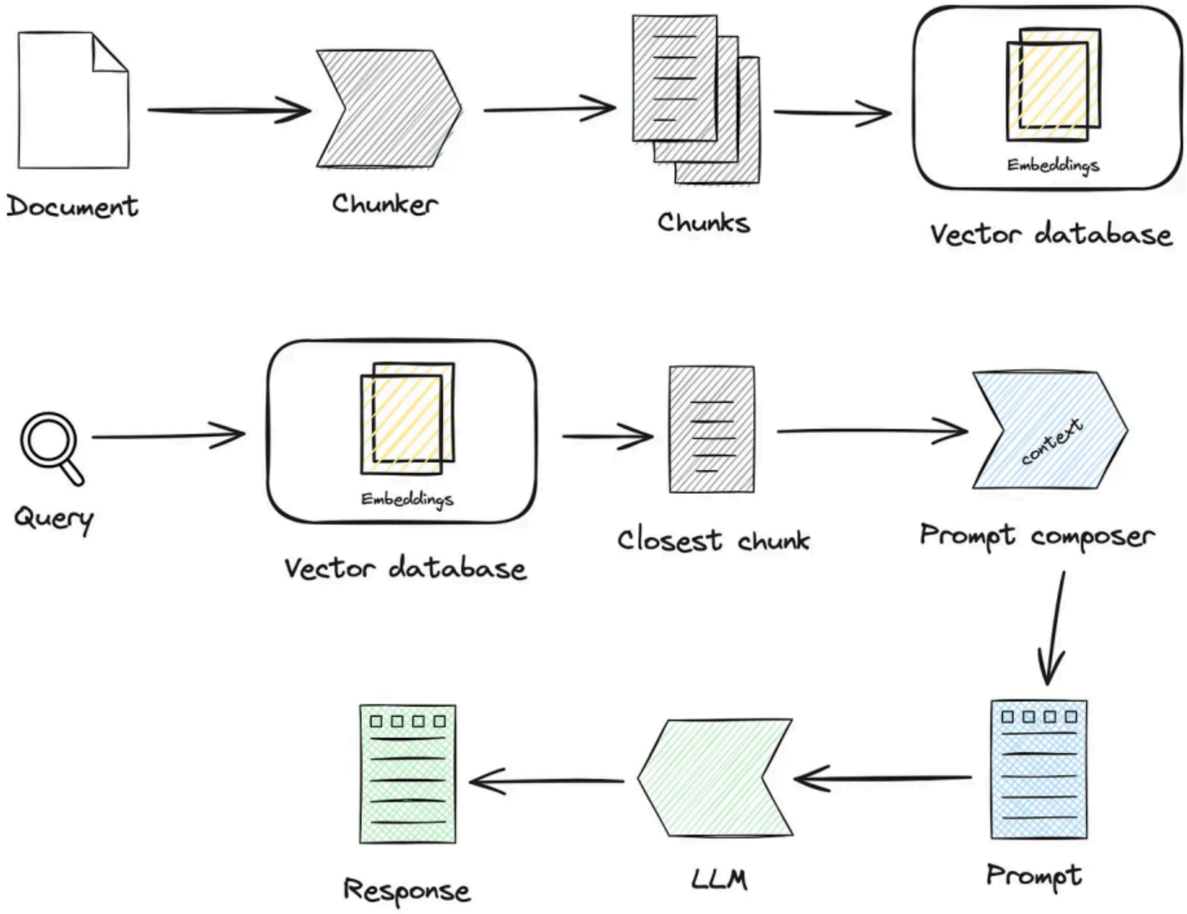
The LLM produces responses using retrieved context.

---

5. RAG Data Ingestion

Data ingestion quality directly affects retrieval accuracy and response reliability.

5.1 Document Ingestion Workflow



Step 1: Document Collection

Sources may include:

- PDFs
- Word documents
- Web pages
- Databases
- APIs

Step 2: Cleaning and Preprocessing

- Remove headers and footers
- Normalize formatting
- Remove irrelevant noise
- Convert to structured text

Step 3: Chunking

Chunking divides large documents into manageable segments suitable for embedding and retrieval.

Common Chunking Strategies

Strategy	Description	Suitable For
Fixed-size	Equal token splits	Simple documents
Overlapping	Sliding window approach	Preserving continuity
Semantic	Split by meaning or topic	Knowledge-heavy content
Section-based	Split by headings	Manuals and structured docs

Step 4: Embedding Generation

Each chunk is converted into a vector representation.

Step 5: Vector Storage

Embeddings are stored along with metadata such as:

- Source
- Section
- Timestamp
- Author

---

6. Context Assembly in RAG

Context Assembly combines:

- Top-k retrieved chunks
- System instructions
- Output constraints
- Formatting rules

The objective is to provide structured, relevant, and minimal information to the model.

---

7. Grounded Generation

Grounded generation ensures that responses are based strictly on retrieved documents.

Common techniques:

- Instructing the model to answer only from context
- Forcing citation references
- Returning “Not found in context” if applicable
- Lowering randomness during generation

---

8. Reducing Hallucinations

Strategies include:

1. Improving retrieval quality
2. Optimizing chunk size
3. Applying re-ranking

4. Adding metadata filters
5. Tightening instructions
6. Limiting temperature
7. Evaluating outputs systematically

---

9. RAG vs Fine-Tuning

RAG	Fine-Tuning
Dynamic knowledge updates	Static knowledge
External data retrieval	Knowledge embedded in weights
Easier to update	Requires retraining
Provides citations	Harder to trace sources

Use RAG when:

- Knowledge changes frequently
- Transparency and citations are required
- Large external document sets exist

Use Fine-Tuning when:

- Style or behavior needs adjustment
- Structured output patterns are required
- Domain-specific reasoning must be internalized

---

10. Retrieval Pipeline Design Decisions

Critical decisions include:

- Embedding model selection
- Chunk size and overlap
- Top-k value
- Re-ranking strategy
- Metadata filtering
- Hybrid search (keyword + vector)

---

11. Developing a RAG-Based Application

Basic implementation flow:

1. Upload documents
2. Preprocess and chunk
3. Generate embeddings
4. Store in vector database
5. Implement retrieval API
6. Add LLM generation layer
7. Build interface
8. Monitor performance