LLM Foundations & RAG Architecture

From Text Generation
to
Retrieval-Augmented Generation (RAG)

Part 1

What LLMs Actually Do

What is a Large Language Model?

An LLM is:

A probabilistic next-token prediction engine

It does **not**:

- Think
- Reason like humans
- Know facts

It predicts the most likely next word.

How LLMs Generate Text

Input:

```
Machine learning is
```

Model predicts:

- powerful
- transforming
- a
- the

Each word has a probability.

The highest probability token is chosen (depending on decoding settings).

LLM = Pattern Recognition at Scale

During training:

- Trained on massive text corpora
- Learns statistical patterns
- Learns structure of language
- Learns relationships between words

It compresses language patterns into billions of parameters.

Important Reality

LLMs do not store knowledge like databases.

They store:

- Statistical relationships
- Semantic associations
- Pattern likelihoods

They generate answers.
They do not retrieve facts.

Tokenization

Before text goes into an LLM:

Text → Tokens → Numbers

Example:

```
Machine learning is powerful
```

Might become:

```
[1234, 5821, 98, 4411]
```

A token is not always a word.
It can be:

- Part of a word
- A full word
- Punctuation

Why Tokenization Matters

Because:

- Cost depends on number of tokens
- Context window depends on tokens
- Truncation happens at token level
- Prompt engineering depends on token limits

Tokens are the real currency.

Context Window

The context window is:

The maximum number of tokens the model can see at once

Includes:

- System prompt
- User query
- Retrieved documents
- Model response

If exceeded:
Old tokens get truncated.

Why Context Window Is Critical for RAG

If you retrieve too many chunks:

- You exceed context window
- Important info gets cut
- Answers degrade

Retrieval must be controlled.

Hallucination

Hallucination = Model generates confident but incorrect information.

Why it happens:

- Model predicts probable text
- It fills gaps statistically
- It does not verify truth

Example:
Model invents citations.
Model fabricates facts.

Why LLM Alone Is Not Enough

Problems:

- No access to private data
- Knowledge cutoff
- Hallucination risk
- Cannot search databases
- Cannot access live information

Solution?

Why RAG Exists

RAG = Retrieval-Augmented Generation

Instead of asking LLM:

"Answer from your memory"

We say:

"Answer using these documents"

We inject retrieved knowledge into the prompt.

RAG Architecture Deep Dive

RAG Full Pipeline

User Query
→ Embed query
→ Search vector DB
→ Retrieve top-k chunks
→ Inject into prompt
→ LLM generates grounded answer

Step 1: Query Embedding

User question:

```
What are admission requirements?
```

We convert it into a vector.

Then compare it against stored document vectors.

Similarity search begins.

Step 2: Top-k Selection

We retrieve:

Top 3
Top 5
Top 10

Most similar chunks.

Trade-off:

- Too small → Missing context
- Too large → Noise + token overflow

Choosing k is critical.

Step 3: Metadata Filtering

Vector similarity alone is not enough.

We can filter by:

- Document type
- Year
- Department
- Category
- Author
- Language

Example:

Only search in:
"Policy documents from 2024"

Hybrid filtering improves precision.

Step 4: Prompt Construction

We build a structured prompt:

System:
"You are an assistant. Use only provided context."

Context:
[Chunk 1]
[Chunk 2]
[Chunk 3]

User:
"What are admission requirements?"

This is grounding.

Step 5: Query Rewriting

Sometimes user queries are vague.

Example:
"Tell me about fees"

Rewrite to:
"What are the tuition fees for undergraduate programs in 2025?"

Rewriting improves retrieval quality.

Methods:

- LLM-based rewriting
- Rule-based expansion
- Keyword expansion

Step 6: Re-ranking

After retrieving top-k:

We can:

- Re-score using a cross-encoder
- Use LLM to re-rank
- Combine keyword + vector scoring

Re-ranking improves precision before generation.

Full RAG System Architecture

Common RAG Failure Points

- Poor chunking
- Too large chunk size
- Bad top-k selection
- No metadata filtering
- Context window overflow
- Weak prompt structure
- High temperature

Key Insight

LLM = Generator
Vector DB = Retriever

RAG = Controlled Knowledge Injection

Example Scenario

Data Source:
College Handbook (PDF → Chunked → Embedded → Stored in Vector DB)

Step 1 – User Asks a Vague Question

User Query:

"Tell me about fees"

Problem:

- Which program?
- Domestic or international?
- Undergraduate or postgraduate?
- Tuition or other fees?
- Which year?

This is vague.

Step 2 – Query Rewriting (LLM Comes Into Play)

We use the LLM to clarify intent before retrieval.

System Prompt:

```
You are a query rewriting assistant.
Your task is to rewrite vague or ambiguous student questions into
clear, specific, and searchable academic queries.
Rules:
- Default to the current academic year unless specified.
- Default to domestic students unless specified.
- Assume undergraduate level unless stated otherwise.
- Do NOT answer the question.
- Output only the rewritten query.
- Do not add explanations.
```

User:
"Tell me about fees"

LLM Output (Rewritten Query)

"What are the tuition fees for undergraduate programs for domestic students in 2025?"

Now the query is structured and specific.

This improves retrieval quality.

Step 3 – Embedding & Vector Search

Rewritten Query → Embedding → Vector DB search

We retrieve Top-3 most similar chunks.

Example retrieved chunks:

Chunk 1:
"Undergraduate domestic tuition fees for 2025 are $8,500 per year..."

Chunk 2:
"Additional student services fee: $450 per semester..."

Chunk 3:
"International undergraduate tuition fees for 2025 are $14,000..."

Step 4 – Metadata Filtering

We filter:

- Program Level = Undergraduate
- Student Type = Domestic
- Year = 2025

Chunk 3 is removed (international).

Now only relevant chunks remain.

Step 5 – Final Answer Generation

Now the LLM generates the final answer.

Final System Prompt

"You are a university assistant.
Answer ONLY using the provided context.
If the answer is not in the context, say:
'The information is not available in the handbook.'
Do not add external knowledge.
Be concise and structured."

Prompt Sent to LLM

System:
(above system prompt)

Context:
"Undergraduate domestic tuition fees for 2025 are $8,500 per year.
Additional student services fee: $450 per semester."

User:
"What are the tuition fees for undergraduate programs for domestic students in 2025?"

LLM Final Output

"For 2025, undergraduate tuition fees for domestic students are $8,500 per year.
In addition, a student services fee of $450 per semester applies."

What Just Happened?

1. User asked vague question.
2. LLM rewrote the query.
3. Query embedded.
4. Vector search performed.
5. Metadata filtering applied.
6. Context injected into prompt.
7. LLM generated grounded answer.

Where LLM Was Used

1. Query Rewriting
2. Final Answer Generation

Vector DB handled retrieval.
LLM handled language + reasoning.

Why This Matters

Without RAG:
LLM might hallucinate outdated fees.

With RAG:
Answer is grounded in handbook data.

This is controlled generation.