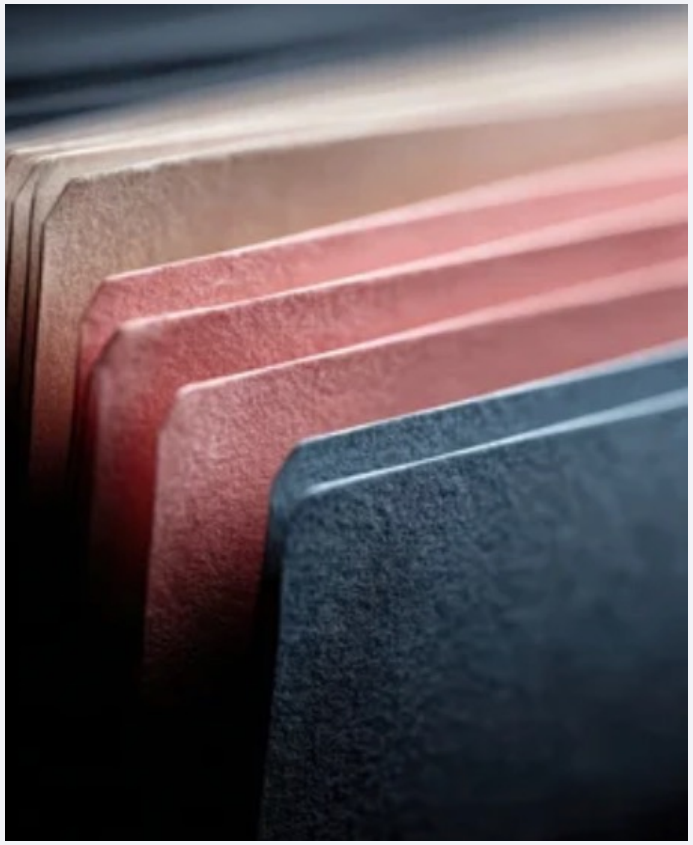
Retrieval-Augmented Generation (RAG):

Dr Maria Prokofieva

What we will cover

- **Why LLM-only systems fail** on factual and document-based tasks
- **What retrieval changes:** new capabilities + new failure modes
- **Basic RAG pipeline design:** chunking → embeddings → semantic search → context assembly → generation
- **Chunking + retrieval quality:** how they determine answer quality downstream
- **Grounded generation with traceable sources** across different backends
- **When basic RAG is insufficient** and the common upgrade paths
- **RAG for research workflows:** literature review, interview analysis, policy briefs



Why LLM-only Systems Fail

- LLMs generate text based on patterns, not verified facts
- They do not have access to private or proprietary documents
- They may produce plausible answers when information is missing
- They cannot reliably indicate uncertainty or source provenance

What retrieval enables

- **Grounding**: answers are constrained by retrieved passages from an external knowledge base.
- **Traceability**: you can attach sources (doc name + chunk id + page + quote span).
- **Coverage beyond context window**: store large corpora, pull only what's needed per question.
- **Updateability**: change the knowledge base without retraining the model.

RAG pipeline (functioning)

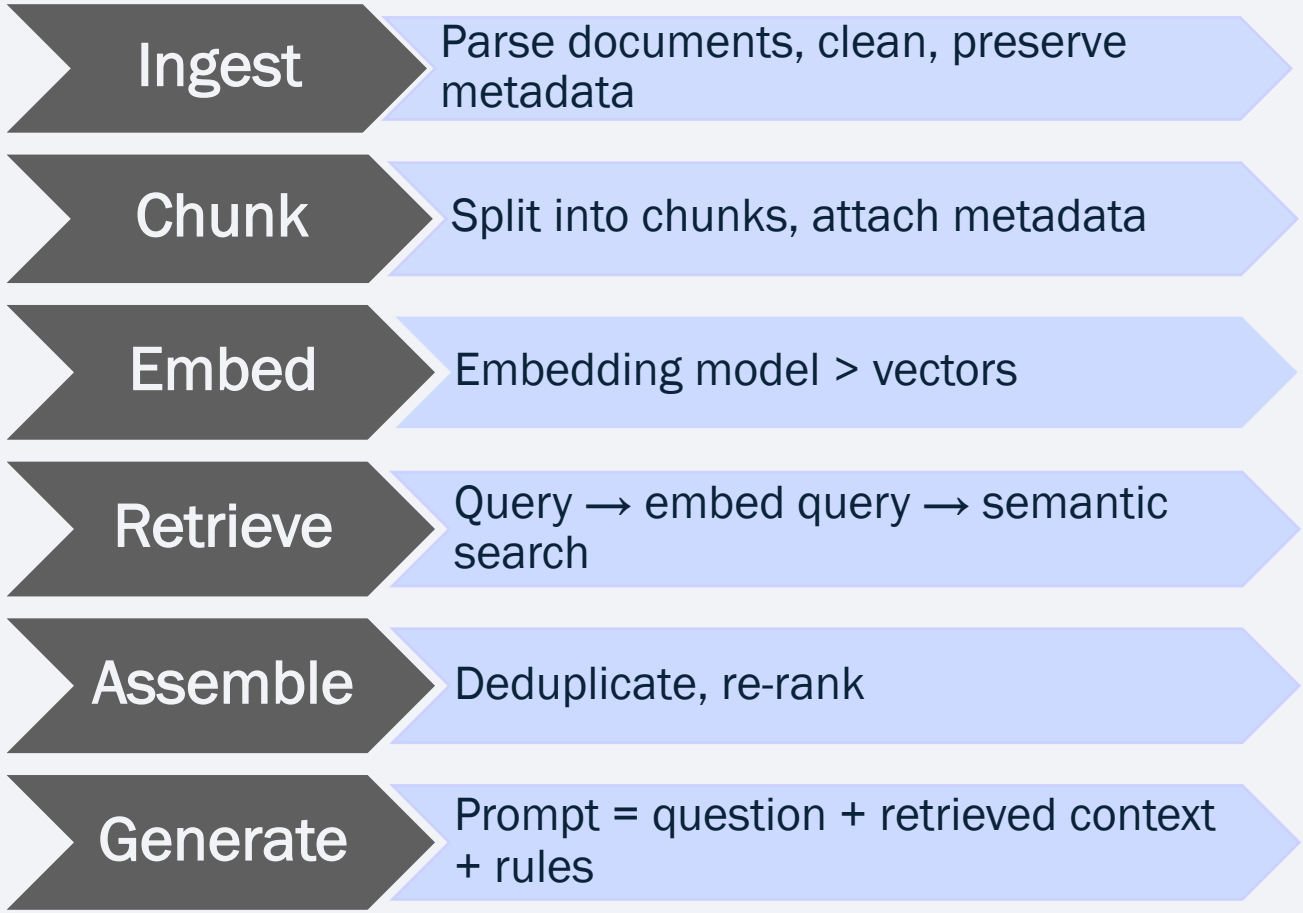
Retrieval: find the right evidence from a private corpus

Generation: produce an answer **grounded in that evidence**

Most failures in RAG are **retrieval failures**, not “model intelligence” failures.

Basic RAG pipeline

Design Components



Ingestion

Turning Messy Documents into Searchable Knowledge

Parsing (layout-aware)

- PDFs/Word/HTML

Cleaning

- Remove repeated text
- Remove irrelevant text

Metadata capture

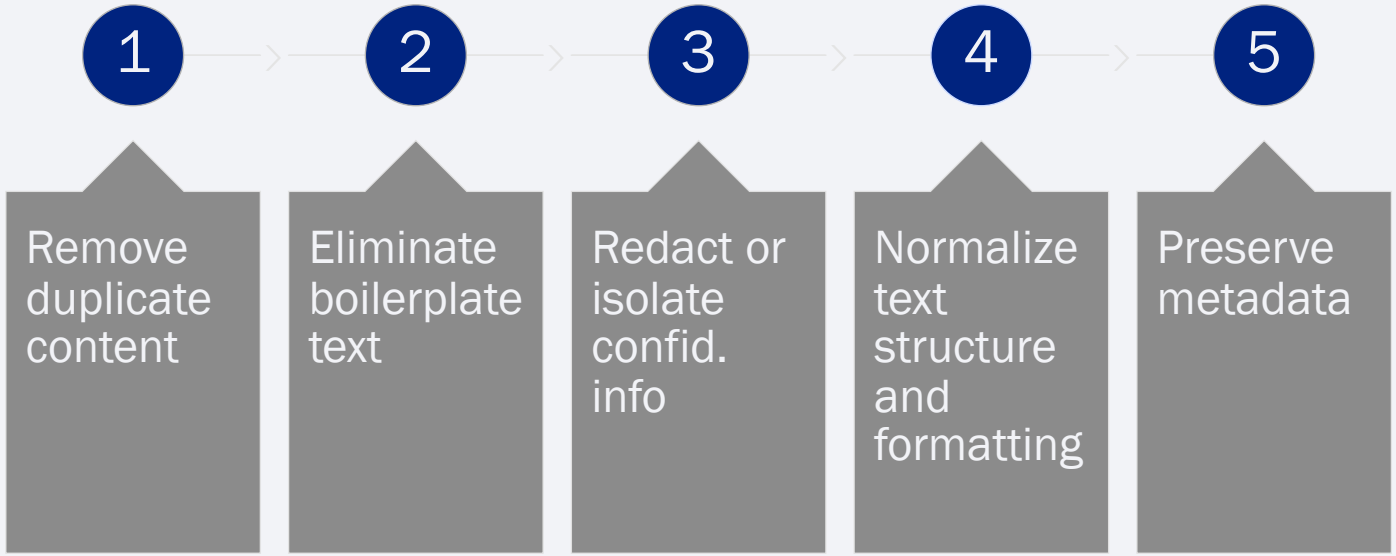
- Source IDs/dates/section names/page numbers

Document Parsing

1. Detect file type / encoding.
2. Extract raw text per type
3. Clean & normalize
4. Structure capture
5. Chunking
6. Metadata & provenance
7. Validate & log

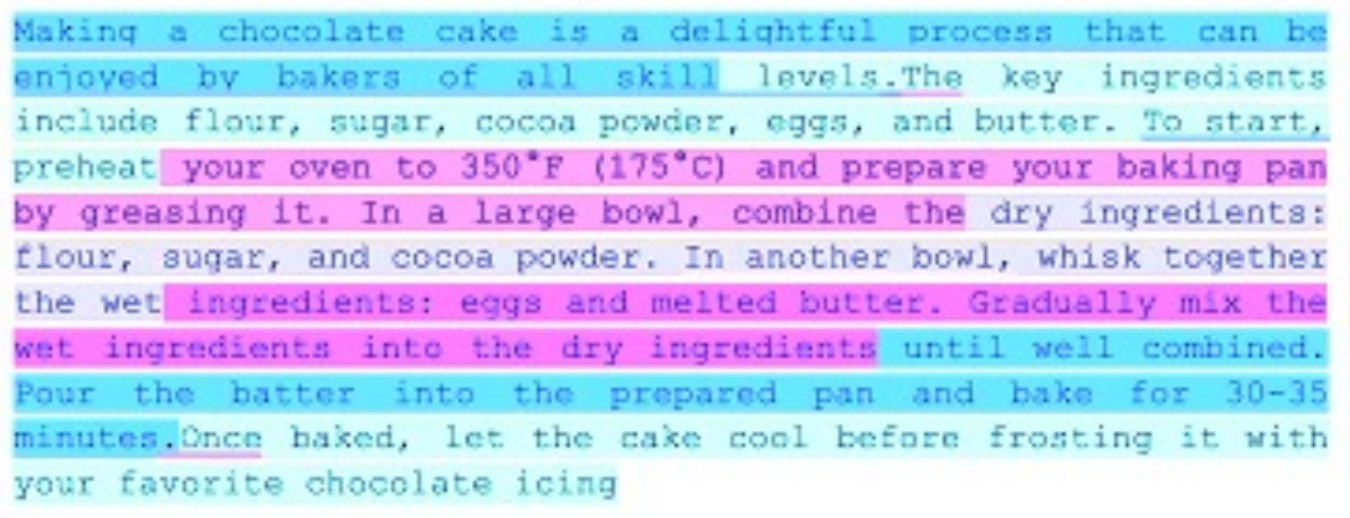
Cleaning

Preparing Documents for Retrieval



Chunking

- Documents are split into smaller parts
- These parts are what the system searches
- The model never sees the full document



Chunking Strategies

Simple splitting

Splits text into fixed-size pieces (by characters, tokens, or words) with no regard for content, structure, or meaning.

Structure-based splitting

Splits text using the document's natural formatting and hierarchy (headings, paragraphs, sections, lists, tables, etc.) to create logically coherent chunks.

Meaning-based splitting (semantic chunking)

Splits text at points where semantic similarity drops (using embeddings, sentence similarity, or topic shifts) so each chunk contains a complete, self-contained idea or topic.

Fixed-size Chunking

- Splits text into uniform token or character lengths
- Fast and simple to implement
- Ignores document structure and semantic boundaries
- Prone to fragmented or incomplete retrieval results
- **Use when:** rapid prototyping or low-structure text

Recursive Chunking

- Splits content hierarchically using document structure
- Preserves section and paragraph boundaries
- Provides stable retrieval with moderate preprocessing cost
- **Use when:** structured documents (reports, policies, papers)

Semantic Chunking

- Groups text by semantic coherence rather than size
- Improves retrieval relevance for topic-specific queries
- Higher computational and implementation cost
- **Use when:** dense analytical content or cross-sectional queries

Contextual Chunking (Chunk + Summary)

- Attaches a short contextual summary to each chunk
- Reduces ambiguity and context-poor retrieval results
- Increases preprocessing complexity and storage requirements
- **Use when:** chunks risk losing meaning outside their original context

Late / Page-level / Multimodal Chunking

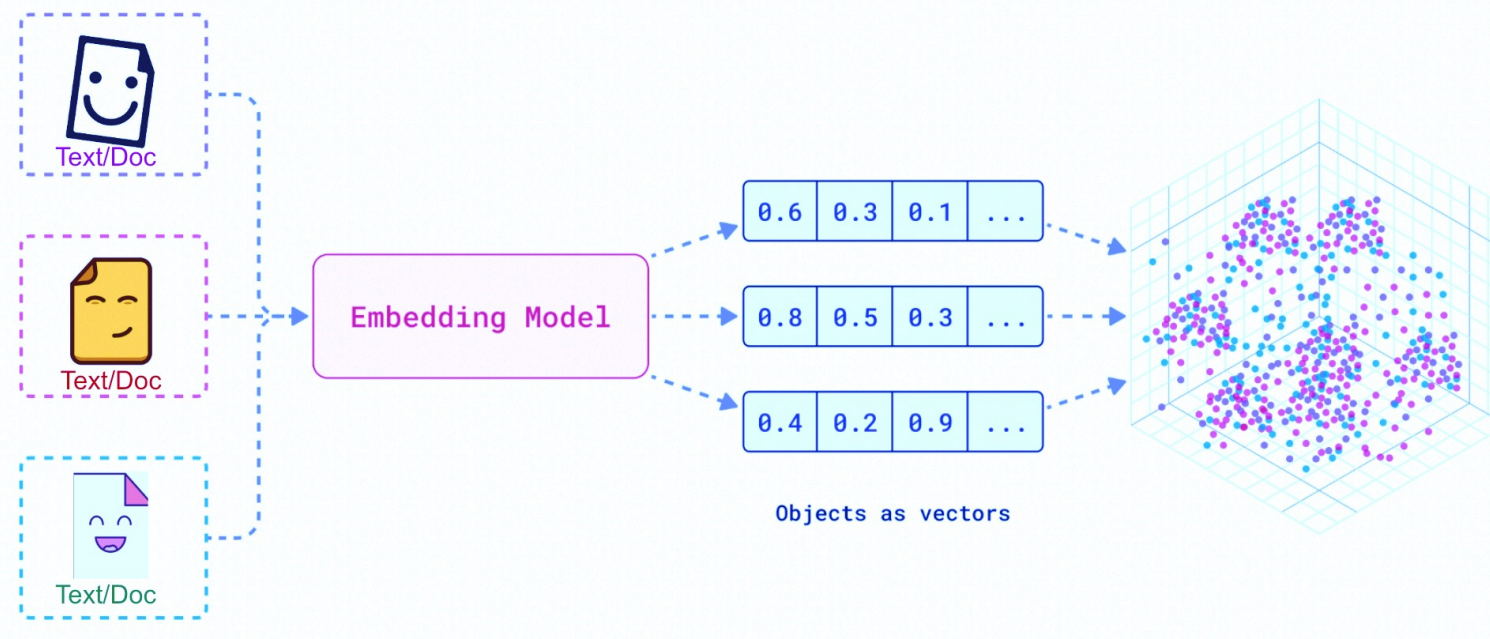
- Defers chunking until query time or preserves layout context
- Retains tables, figures, and spatial relationships
- Requires more complex retrieval and ranking logic
- **Use when:** layout or multimodal context affects interpretation

Chunking Strategy Comparison

Strategy	Retrieval Precision	Context Continuity	Complexity	Typical Use Case
Fixed-size	Low–Medium	Low	Low	Prototyping
Recursive	Medium–High	Medium	Medium	Default baseline
Semantic	High	Medium	High	Topic queries
Chunk + summary	High	High	High	Ambiguous text
Late / multimodal	High	High	Very High	Layout-sensitive

Embeddings

- Convert text into a form the system can compare
- Allow the system to find text that is *similar in meaning*
- Power search across large document collections



Retrieval and Ranking

- **Retrieval** selects candidate document chunks relevant to a query.

- Finds relevant parts of documents
- Narrows the search space
- Determines what information is available

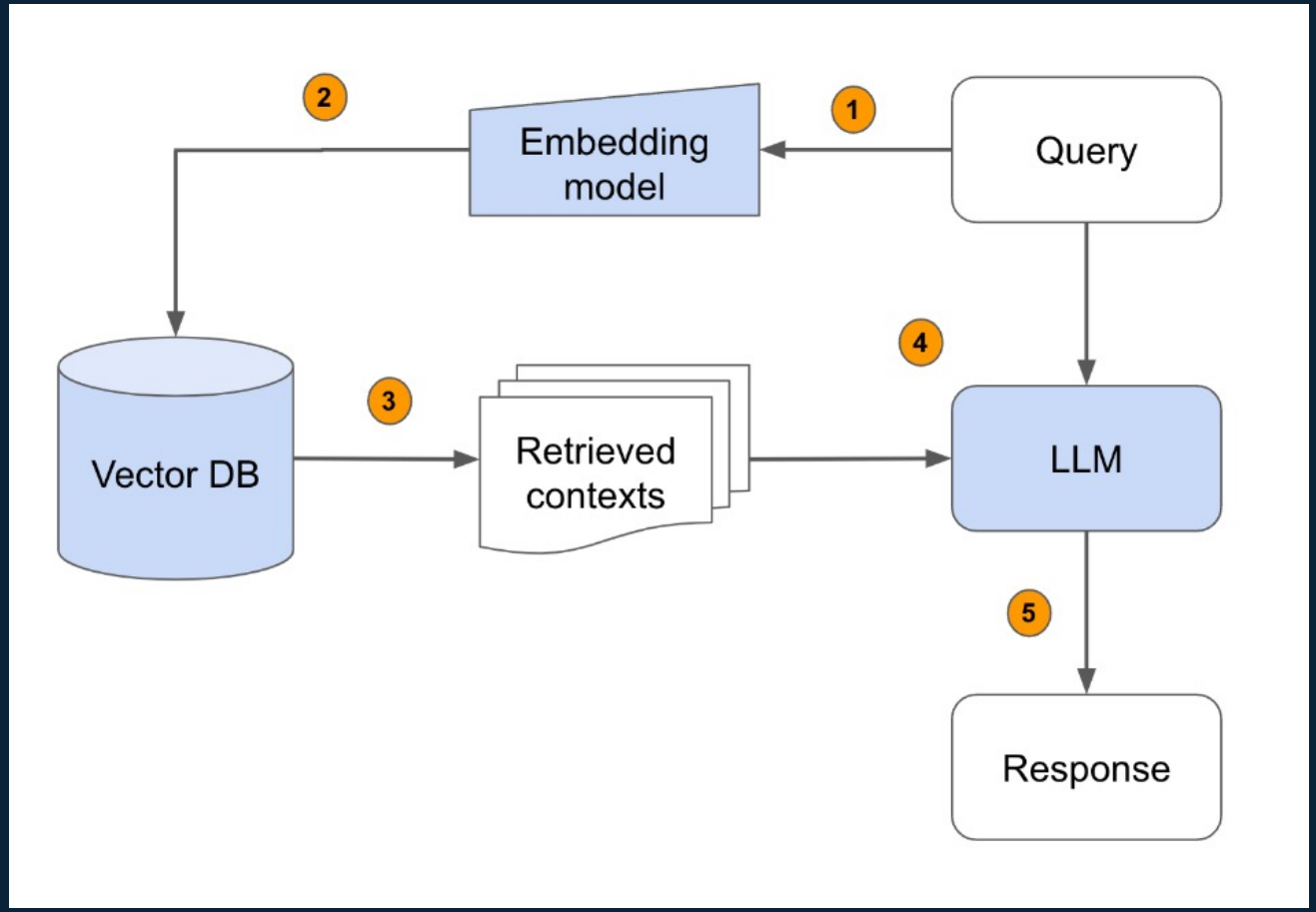
- **Ranking** orders retrieved candidates by relevance before generation.

- Orders retrieved information by relevance
- Prioritizes what the model sees first
- Affects answer quality directly

Basic Retrieval (Vector Search)

Key parameters:
- Similarity metric
- Top-k value
- Optional metadata filters

Embed	Embed the user query
Perform	Perform similarity search over chunk embeddings
Return	Return top-k nearest neighbors



Ranking and Re-Ranking

- Ranking improves precision by re-ordering retrieved candidates using:
- Cross-encoders
- LLM-based relevance scoring
- Heuristic or rule-based scoring

Hybrid Retrieval Strategies

Common hybrid approaches:

- Vector search + keyword (BM25)
- Vector search + structured filters
- Multi-query expansion

Multi-Stage Retrieval Pipelines

Typical multi-stage pipeline:

1. Broad retrieval (high recall)
2. Filtering and deduplication
3. Re-ranking (high precision)
4. Context assembly

Retrieval and Ranking Failure Modes

Relevant content not retrieved

Why:

Recall failure (embedding mismatch, bad chunking, query drift)

- Chunking redesign
- Embedding alignment
- Query expansion
- Recall tuning (top-k, hybrid search)

Correct content retrieved but ranked too low

Why:

Weak or misaligned ranking signals

- Two-stage ranking
- Relevance feature weighting
- Task-aware ranking
- Pre-ranking filters

Conflicting sources without resolution

Why:

No conflict reasoning layer

- Conflict detection
- Structured comparison
- Source hierarchy rules
- Temporal resolution

Over-retrieval exceeding context limits

Why:

Context overload

- Post-retrieval pruning
- Context compression
- Dynamic context budgeting
- Answer-first retrieval
- Multi-pass generation

Retrieval and Ranking Strategy Comparison

Approach	Recall	Precision	Cost	Complexity
Vector only	Medium	Low	Low	Low
Vector + filters	Medium	Medium	Low	Medium
Hybrid (vector+BM25)	High	Medium	Medium	Medium
Multi-stage + rerank	High	High	High	High

Assemble Context: what the model sees

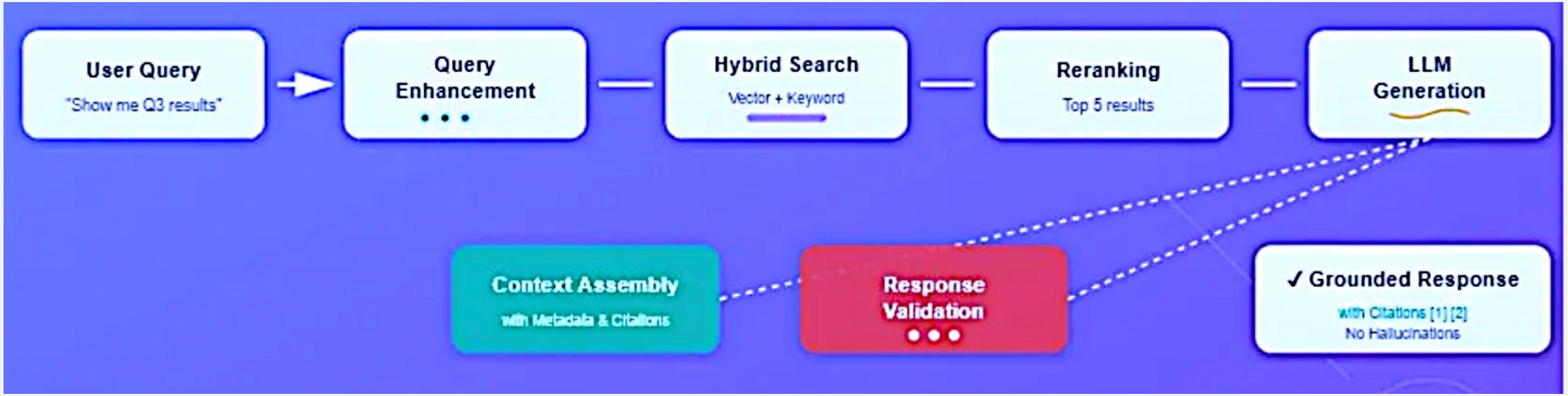
Context assembly is the process of selecting, structuring, and constraining retrieved content before generation.

- Operates after retrieval and ranking
- Determines what evidence the model can use
- Enforces context window and citation requirements

Common failures include:

- Relevant evidence excluded due to token limits
- Redundant chunks crowding out unique information
- Incorrect ordering breaking logical flow
- Lost or misaligned citations

Assemble Context, Generation and Grounding



Generation and Grounding: What the Model Is Allowed to Say

Generation and grounding produce an answer constrained by the assembled context.

- The model must use provided evidence only
- Output must be traceable to sources
- Uncertainty must be reported explicitly

Grounding is enforced through:

- Prompt constraints (“use sources only”)
- Citation requirements
- Two-step generation (extract → synthesize)
- Structured output formats

When Basic RAG Works

- Questions are simple and well defined
- Information is contained in one place
- Documents are stable and consistent
- No interpretation or reconciliation is required

Limitations of RAG

- Information is spread across multiple sources
- Sources are incomplete or inconsistent
- Questions require comparison or interpretation
- The system must handle uncertainty

What Typically Needs to Change

- Better selection of relevant information
- Better control over what is prioritised
- Clear handling of conflicting or missing evidence
- Explicit rules for acceptable answers

Takeaways

- Basic RAG is effective for simple, single-hop queries over stable documents
- Retrieval, ranking, context assembly, and grounding impose hard limits on system performance
- Most failures originate upstream of generation
- System upgrades should be selected based on observed failure modes
- Prompting and model choice cannot compensate for architectural limitations