Retrieval-Augmented Generation (RAG) and Vector Databases

Dr Yeo Wee Kiang

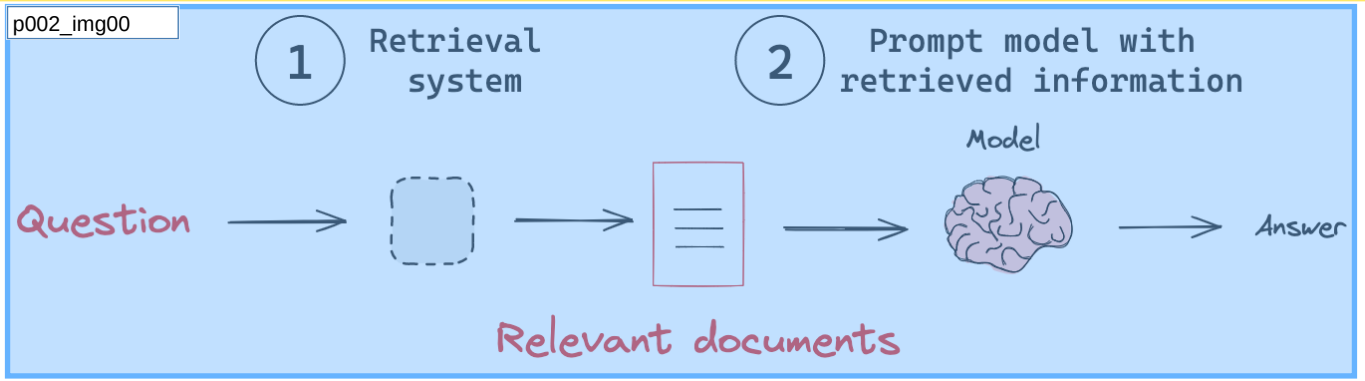
NUS | Computing
National University of Singapore

Introduction-scenarios

Introduction to Retrieval-Augmented Generation (RAG)

- **Combines retrieval and generation for enhanced responses**
  - Mixes internal (model) and external (document-based) memories
  - Achieves greater factual accuracy and specificity in output
- **Useful for knowledge-intensive NLP tasks**
  - Enhances question answering (QA), fact verification, and open-domain retrieval tasks
  - Supports generating contextually accurate answers in complex scenarios

p002_img00



Lewis-prohibited

A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems, 33, 9459-9474.

Main-response

Main steps in Retrieval-Augmented Generation (RAG)

No.	Step	Goal	Algorithms
1	Query Understanding & Embedding	Convert user query into a numerical vector representation.	• Embedding models: • OpenAI Embeddings, BERT, Sentence Transformers • Tokenisation & normalisation tools
2	Retrieval (Searching Stage)	Find top-k candidate documents quickly.	• Sparse Retrieval: BM25, TF-IDF • Dense Retrieval: ANN algorithms (HNSW, FAISS, ScaNN, PQ) • Graph Retrieval: Personalised PageRank, Graph Neural Networks (GNNs) • Hybrid Retrieval: Sparse + Dense combined
3	Scoring & Ranking	Rank documents by relevance for the query.	• Similarity Metrics: Cosine Similarity, Inner Product, Euclidean Distance • Re-Rankers: Cross-Encoder models (MonoBERT, ColBERT) • Fusion Methods: Reciprocal Rank Fusion (RRF), CombSUM
4	Context Fusion & Generation	Feed top-ranked documents to LLM for final response.	• LLMs: GPT, LLaMA, T5, FLAN-T5 • Context Fusion: Concatenation, attention-based methods • Adaptive Methods: ReAct, Agentic RAG, iterative retrieval

Why-audit

Why Retrieval-Augmented Generation (RAG) is useful

The problem with LLMs alone

- Frozen knowledge after the training cut-off
- Hallucinations when facts are missing
- No access to your private data by default
- Limited context window and rising token costs
- Fine-tuning is slow to update and hard to audit

Why-questions

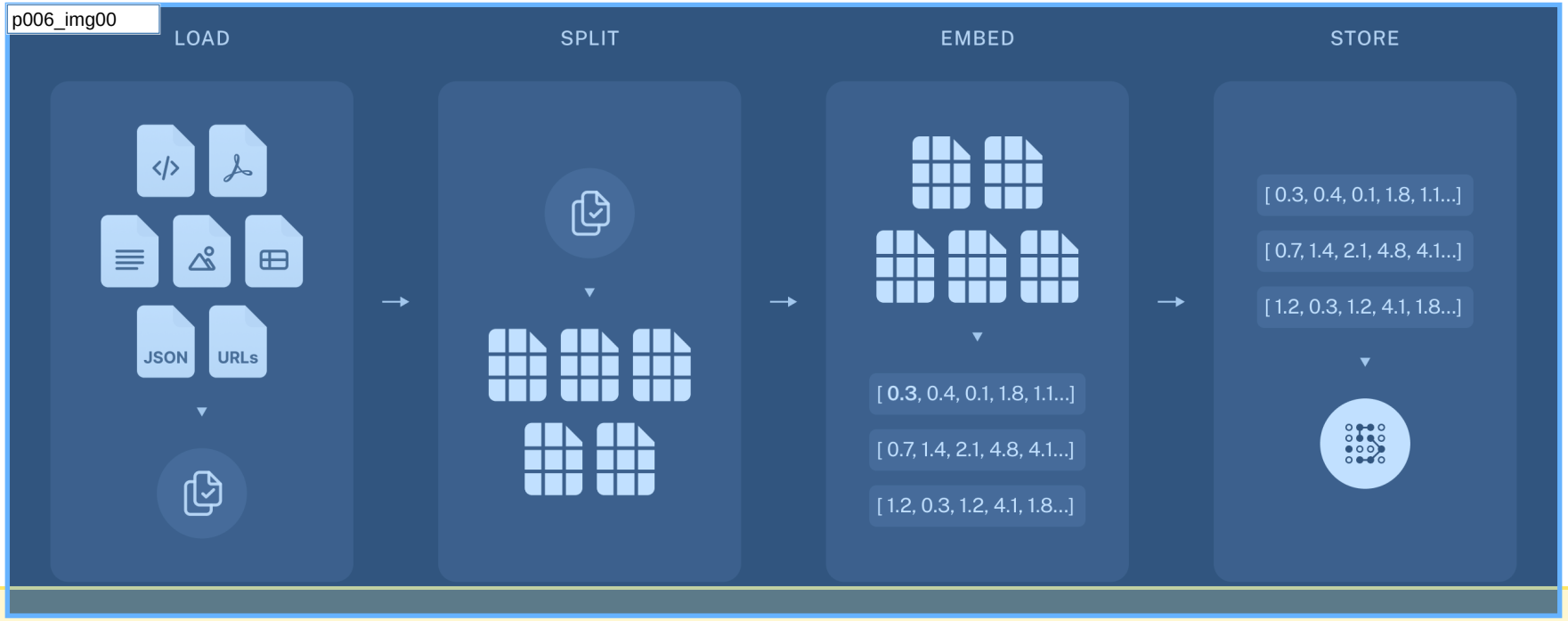
Why Retrieval-Augmented Generation (RAG) is useful

What RAG adds

- External, updatable memory: index PDFs, wikis, tickets, tables
- Targeted search: dense vectors, BM25, hybrid search
- Grounded answers with citations to retrieved passages
- Access controls and tenant isolation for data governance
- Iterative retrieval for multi-step questions

Introduction to Retrieval-Augmented Generation (RAG)

p006_img00



--prohibited

- *Load*: First, we need to load our data. This is done with Document Loaders.
- *Split*: Text splitters break large Documents into smaller chunks. This is useful both for indexing data and passing it into a model, as large chunks are harder to search over and won't fit in a model's finite context window.
- *Store*: We need somewhere to store and index our splits, so that they can be searched over later. This is often done using a VectorStore and Embeddings model.

https://python.langchain.com/docs/tutorials/rag/

Why-prohibited

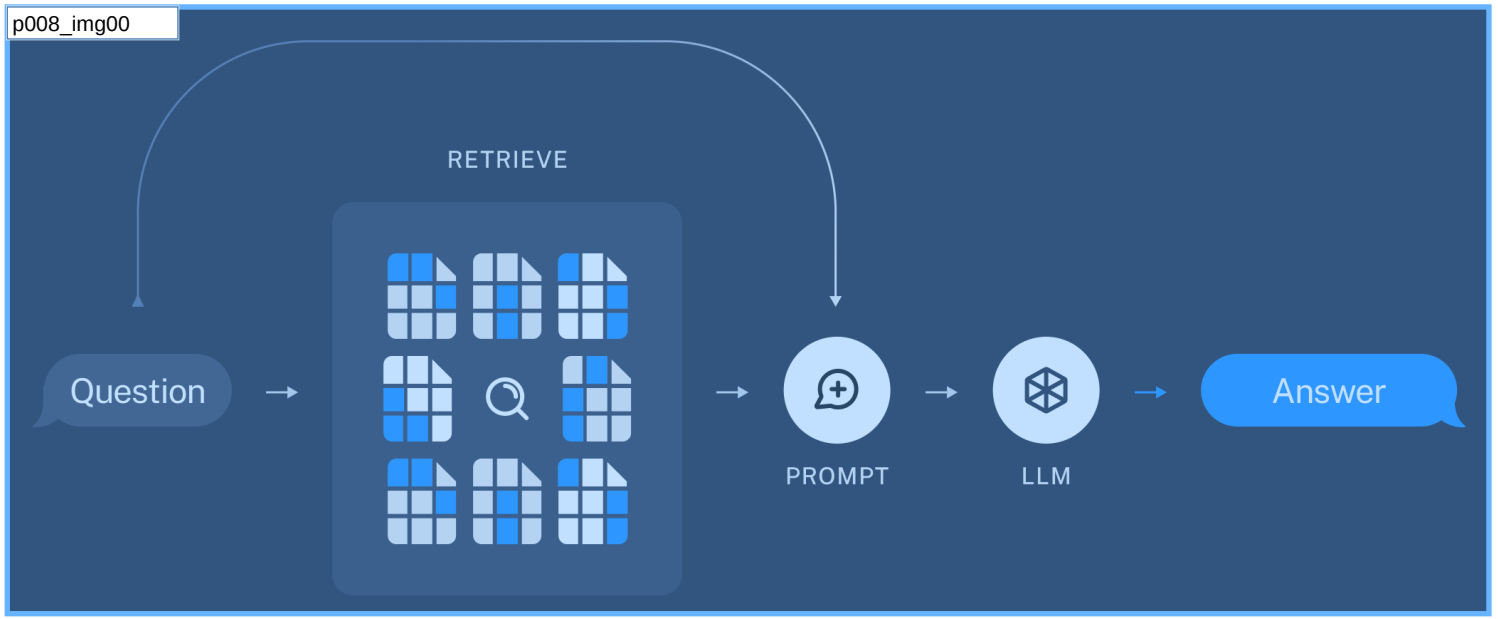
Why document indexing is necessary

- **Efficiency**
  - Enables fast retrieval of relevant documents from a large corpus.
- **Scalability**
  - Allows the system to handle massive amounts of data without significant performance loss.
- **Real-time Retrieval**
  - Makes it possible to search and retrieve documents quickly enough for applications requiring immediate responses
- **Fixed Component of RAG**
  - The document index is not updated during training; only the Query Encoder and BART generator are fine-tuned
- **Robust Representation for Various Tasks**
  - Initial indexing creates a general representation that supports diverse downstream applications

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems, 33, 9459-9474.

Introduction to Retrieval-Augmented Generation (RAG)

p008_img00



--prohibited

- **Retrieval and generation**
  - *Retrieve*: Given a user input, relevant splits are retrieved from storage using a Retriever.
  - *Generate*: A ChatModel / LLM produces an answer using a prompt that includes both the question with the retrieved data

https://python.langchain.com/docs/tutorials/rag/

Searching-efficiently

Searching vs Scoring vs Generation in RAG

Searching: Finding candidate documents

- What it does:
  - Quickly retrieves the top-k most likely documents from a large collection.
- How it works:
  - Uses retrieval algorithms like BM25, Dense Retrieval with ANN (HNSW, FAISS), Graph Retrieval.
- Goal:
  - Speed & Coverage : Find a small set of promising documents efficiently.

Searching-top

Searching vs Scoring vs Generation in RAG

Scoring: Ranking by relevance

- What it does:
  - Assigns a relevance score to each retrieved document relative to the query.
- How it works:
  - Uses similarity metrics (Cosine, Inner Product, Euclidean) and re-rankers (Cross-Encoders, RRF).
- Goal:
  - Ensure the most relevant documents appear at the top.

Searching-knowledge

Searching vs Scoring vs Generation in RAG

Generation: Producing the final answer

- What it does:
  - Feeds the top-ranked documents into a Large Language Model (LLM) to generate a context-aware answer.
- How it works:
  - Uses LLMs (GPT, LLaMA, T5) with context fusion methods like concatenation or attention.
- Goal:
  - Generate coherent, informative responses grounded in retrieved knowledge.

Chunking-system

Chunking in RAG

- Chunking is the process of breaking a large document into smaller parts called “chunks”.
- In RAG, chunking allows documents to be indexed and retrieved in smaller, meaningful segments.
  - Rather than feeding the entire document to the language model, only relevant chunks are retrieved and used to generate answers.
- This improves the efficiency, accuracy, and cost-effectiveness of the system.

Why-retrieval

Why do we need chunking?

- Large language models (LLMs) have a limited context window (e.g. 4,000 to 100,000 tokens).
  - If a document is too long, we cannot feed the whole thing into the LLM.
- By chunking, we ensure that only the most relevant parts are used within this limited window.
- Chunking also:
  - Reduces token cost by avoiding irrelevant information
  - Improves answer precision by narrowing focus
  - Speeds up processing and retrieval

Chunk-case

Chunk size and overlap

- Chunk size
  - Refers to how many tokens or words are in each chunk.
  - Must be small enough to fit within the LLM's context window, but large enough to retain meaning.
  - Common sizes: 100–500 tokens depending on use case.

--14

- Chunk overlap
  - Repeats a portion of the previous chunk in the next one.
  - Prevents loss of context at boundaries, especially useful for long sentences or narratives.
  - Typical overlaps: 10%–30% of the chunk size.

The-tokens

The role of the LLM context window

- The context window determines how many tokens the LLM can “see” at once.
  - Older LLMs (e.g. GPT-3.5) have 2k–4k token limits. Newer models (e.g. GPT-4-turbo) go up to 128k tokens or more.
- Your retrieved chunks plus the prompt must all fit within this window.
  - Chunking must be designed with this in mind, or important content may be truncated or dropped.
- You should also consider the embedding model’s limit when indexing chunks (often 512–8192 tokens).

Chunking-expensive

Chunking strategies

Strategy	Description
Fixed-length chunking	• Divides text into equal-sized segments by a fixed number of tokens or words. • It is simple and fast but may cut across sentences or paragraphs, harming coherence.
Sentence-based chunking	• Chunks follow sentence boundaries for better readability. • Keeps complete sentences intact, preserving grammatical and semantic structure, though chunk sizes can vary significantly.
Recursive / context-aware chunking	• Tries paragraphs first, then splits further if too long. • Attempts to split by larger structures such as paragraphs, then sentences, then tokens in a hierarchical way to respect natural text boundaries.
Semantic chunking	• Uses meaning to group similar sentences together • Uses sentence-level embeddings to cluster related content into meaningful chunks. This improves retrieval relevance but is more computationally expensive.

Chunking-context

Chunking strategies

Strategy	Description
Task-oriented/agentic chunking	• Splits by sections like “Problem”, “Solution”, “Summary”. • Segments documents based on functional sections aligning well with agent workflows and structured tasks.
Hierarchical chunking	• Tracks parent–child relationships between sections and subsections, enabling navigation and structured summarisation in documents with clear organisation.
Vision-guided chunking (multimodal)	• Ensures text and visuals such as diagrams remain together when chunking PDFs or slides, preserving multimodal context.

Tripathi-17

Tripathi, V., Odapally, T., Das, I., Allu, U., & Ahmed, B. (2025). Vision-Guided Chunking Is All You Need: Enhancing RAG with Multimodal Document Understanding. arXiv preprint arXiv:2506.16035.

Fixed-size-retrieval

Fixed-size chunking

Simple and predictable, fixed-size chunking fits well within model token limits and is easy to implement. It can improve retrieval efficiency by ensuring chunks aren't too large for processing. However, arbitrary boundaries risk splitting semantic units like sentences or phrases, potentially degrading context.

Steps & Example:

1. Set chunk_size = 20 words, overlap = 0.
2. Text: "Machine learning is a field of artificial intelligence focused on building systems that learn from data. It encompasses supervised, unsupervised, and reinforcement learning techniques."
3. Chunk 1 (words 1–20): "Machine learning is a field of artificial intelligence focused on building systems that learn from data. It encompasses supervised, unsupervised,"
4. Chunk 2 (words 21–40): "and reinforcement learning techniques."

(Note: The last chunk will contain the remaining words, which may be less than the chunk_size.)

5. Index both for retrieval.

Fixed-Size-retrieval

Fixed-Size with Overlap

Adding overlap helps preserve context across boundary chunks, reducing wrong retrieval when content spans chunks. The drawback is increased storage and compute from duplicated content.

Steps & Example:

1. Set chunk_size = 20 words, overlap = 5.
2. Text: “Machine learning is a field of artificial intelligence focused on building systems that learn from data. It encompasses supervised, unsupervised, and reinforcement learning techniques.”
3. Chunk 1 (words 1–20): “Machine learning is a field of artificial intelligence focused on building systems that learn from data. It encompasses supervised, unsupervised,”
4. Chunk 2 (words 16–35): “data. It encompasses supervised, unsupervised, and reinforcement learning techniques.”
5. Index both for retrieval.

Sentence-Bas-retrieval

Sentence-Based Chunking

Chunks respect natural linguistic boundaries, preserving meaning and coherence. But sentence lengths vary, leading to inconsistent chunk sizes and possible inefficiency.

Steps & Example:

1. Text: “Machine learning is a field of artificial intelligence focused on building systems that learn from data. It encompasses supervised, unsupervised, and reinforcement learning techniques.”
2. Split into sentences.
3. Chunk 1: “Machine learning is a field of artificial intelligence focused on building systems that learn from data.”
4. Chunk 2: “It encompasses supervised, unsupervised, and reinforcement learning techniques.”
5. Index both for retrieval.

Recursive-tokens

Recursive (Context-Aware) Chunking

This hierarchical chunking preserves semantic structure and context by intelligently merging text units (paragraphs, then sentences) to fit token limits. It's more complex and computationally demanding than fixed-size methods.

Steps & Example:

1. Set token limit = 30 tokens.
2. Text: “Large language models process vast amounts of text and code, learning complex linguistic patterns for various tasks. They can then generate human-like responses and creative content. This advanced AI technology is rapidly transforming many industries and applications globally.”
3. Recognize a text unit (e.g., a paragraph). If it's too long (exceeds the maximum chunk size), proceed to the next step.
4. Split into smaller semantic units (e.g., sentences).
   - Sentence A: "Large language models process vast amounts of text and code, learning complex linguistic patterns for various tasks." (~22 tokens)
   - Sentence B: "They can then generate human-like responses and creative content." (~11 tokens)
   - Sentence C: "This advanced AI technology is rapidly transforming many industries and applications globally." (~15 tokens)

The-21

The token counts provided for each sentence (~22, ~11, ~15) are estimates. The actual number of tokens generated from text depends entirely on the specific tokenizer used and its vocabulary. Different tokenizers will produce different token counts for the exact same text.

Recursive-tokens

Recursive (Context-Aware) Chunking

Steps & Example:

5. Merge these smaller units (e.g., sentences) sequentially to form chunks, ensuring each chunk's token count gets as close as possible to the specified token limit without exceeding it.

6. Forming Chunk 1:
   - Start with Sentence A (~22 tokens). Can we add Sentence B (~11 tokens)? 22+11=33 tokens. This exceeds the 30-token limit.
   - Therefore, Chunk 1 is just Sentence A.
   - Chunk 1 Text: "Large language models process vast amounts of text and code, learning complex linguistic patterns for various tasks." (~22 tokens)

7. Forming Chunk 2:
   - Start with Sentence B (~11 tokens). Can we add Sentence C (~15 tokens)? 11+15=26 tokens. This is within the 30-token limit.
   - Therefore, Chunk 2 is Sentence B + Sentence C.
   - Chunk 2 Text: "They can then generate human-like responses and creative content. This advanced AI technology is rapidly transforming many industries and applications globally." (~26 tokens)

Semantic-sentences

Semantic Chunking

By clustering semantically related sentences, this method yields chunks aligned with topics, improving retrieval relevance. But it adds embedding cost, similarity computation overhead, and requires tuning similarity thresholds.

Steps & Example:

1. Text: “The Amazon rainforest is the world's largest tropical rainforest, spanning nine countries. It is incredibly biodiverse, home to millions of species of plants and animals, many undiscovered. Deforestation for cattle ranching and agriculture remains a major threat to this vital ecosystem. Separately, recent breakthroughs in artificial intelligence are rapidly advancing natural language processing capabilities. These new models can understand and generate human language with unprecedented accuracy and fluency.”

2. Split text into sentences.

3. Embed each sentence (convert it into a numerical vector representation) and then compute semantic similarity between adjacent or logically related sentences.

Semantic-fluency

Semantic Chunking

Steps & Example:

4. Merge sentences that are semantically similar (e.g., exceeding a predefined similarity threshold) until a significant semantic shift occurs, forming cohesive chunks.

   Similarity Threshold for merging: 0.75):

   - Sentence 1 (S1): "The Amazon rainforest is the world's largest tropical rainforest, spanning nine countries."
   - Sentence 2 (S2): "It is incredibly biodiverse, home to millions of species of plants and animals, many undiscovered."
   - Sentence 3 (S3): "Deforestation for cattle ranching and agriculture remains a major threat to this vital ecosystem."
   - Sentence 4 (S4): "Separately, recent breakthroughs in artificial intelligence are rapidly advancing natural language processing capabilities."
   - Sentence 5 (S5): "These new models can understand and generate human language with unprecedented accuracy and fluency."

Semantic-processing

Semantic Chunking

Steps & Example:

5. Process (Illustrative Similarity Scores)
   - S1 & S2: Semantic Similarity = 0.88 (High). Merge. (Current Chunk: S1+S2)
   - Merged (S1+S2) & S3: Semantic Similarity = 0.78 (Still high). Merge. (Current Chunk: S1+S2+S3)
   - Merged (S1+S2+S3) & S4: Semantic Similarity = 0.15 (Very low : a topic shift). Do NOT merge.
   - S4 & S5: Semantic Similarity = 0.92 (High). Merge. (Current Chunk: S4+S5)
6. Resulting Chunks:
   - Chunk 1: "The Amazon rainforest is the world's largest tropical rainforest, spanning nine countries. It is incredibly biodiverse, home to millions of species of plants and animals, many undiscovered. Deforestation for cattle ranching and agriculture remains a major threat to this vital ecosystem.“
   - Chunk 2: "Separately, recent breakthroughs in artificial intelligence are rapidly advancing natural language processing capabilities. These new models can understand and generate human language with unprecedented accuracy and fluency."
7. Index each resulting cohesive chunk for information retrieval or further processing.

Agentic (Task-Oriented) Chunking

Agentic chunks fine-tune models for tasks like summarisation or QA by providing context-rich segments, enhancing reasoning. However, they demand manual structuring or schema inference, increasing complexity and domain dependence.

Steps & Example:

1. Text: "Reinforcement learning is an area of machine learning concerned with how intelligent agents ought to take actions in an environment to maximize the notion of cumulative reward. It differs significantly from supervised learning, which relies on labeled input-output pairs. A classic example is a computer program learning to play chess: the 'reward' is winning the game, and the 'actions' are making moves on the board. Another instance is training robots to navigate complex environments, where rewards are given for reaching destinations. However, limitations include the vast amounts of data often required and the difficulty in debugging due to their trial-and-error nature."

2. Identify and Label relevant sections within a document based on anticipated agent tasks or predefined content schemas (e.g., "Definition," "Key Features," "Examples," "Applications," "Limitations").

3. Extract content corresponding to each labeled section to form distinct, task-specific chunks.

Agentic-agent

Agentic (Task-Oriented) Chunking

Steps & Example:

4. Labelled and Extracted Chunks:
    - *Definition* Chunk: "Reinforcement learning is an area of machine learning concerned with how intelligent agents ought to take actions in an environment to maximize the notion of cumulative reward. It differs significantly from supervised learning, which relies on labeled input-output pairs."
    - *Examples* Chunk: "A classic example is a computer program learning to play chess: the 'reward' is winning the game, and the 'actions' are making moves on the board. Another instance is training robots to navigate complex environments, where rewards are given for reaching destinations."
    - *Limitations* Chunk: "However, limitations include the vast amounts of data often required and the difficulty in debugging due to their trial-and-error nature."
5. Index each resulting chunk with its corresponding task label for efficient retrieval and targeted processing by an agent.

Hierarchical-text

Hierarchical Chunking

This method maintains document hierarchy and context, aiding navigation, multi-level summarization, and logical flow. However, chunk sizes vary, and managing these hierarchical links and their associated metadata adds notable complexity.

Steps & Example:

1. Identify the inherent structural hierarchy of the document (e.g., chapters, sections, sub-sections, paragraphs).
2. Define "Parent" chunks based on higher-level structural units (e.g., a section or paragraph).
3. Define "Child" chunks as smaller, constituent units within the parent (e.g., individual sentences or sub-paragraphs).
4. Store metadata explicitly linking parent chunks to their children, preserving the document's original structure.
   - Original Document Snippet:

     ## Introduction to Large Language Models
     Large Language Models (LLMs) are advanced AI systems. They are designed to understand, generate, and manipulate human language based on vast datasets.

     ### Core Principles of Operation
     LLMs learn probabilistic relationships between words. This enables them to predict the next word in a sequence, forming coherent text.

Hierarchical-text

Hierarchical Chunking

Steps & Example:

5. Resulting Hierarchical Chunks:
   - Parent Chunk 1 (Section): "Introduction to Large Language Models"
     - Child Chunk 1.1 (Sentence): "Large Language Models (LLMs) are advanced AI systems."
     - Child Chunk 1.2 (Sentence): "They are designed to understand, generate, and manipulate human language based on vast datasets."
   - Parent Chunk 2 (Section): "Core Principles of Operation"
     - Child Chunk 2.1 (Sentence): "LLMs learn probabilistic relationships between words."
     - Child Chunk 2.2 (Sentence): "This enables them to predict the next word in a sequence, forming coherent text."

Vision-guide-limits

Vision-guided Chunking (Multimodal)

Essential for documents with integral visuals (e.g., PDFs), this method ensures visual-textual cohesion, giving LMMs complete context. However, it requires advanced LMMs and vision processing, increasing costs and complexity.

Steps & Example:

1. Analyze the document visually to identify and segment distinct zones containing text, images, tables, and other visual elements.

2. Identify visually and semantically relevant relationships between these zones (e.g., a paragraph explicitly referencing a diagram, an image positioned next to explanatory text, a caption belonging to a figure).

3. Form chunks that combine both textual and visual context based on these identified relationships, ensuring visual-textual cohesion.

4. Chunk 1 (Visually Cohesive):
   - Textual Content: "During the benchmark tests, system performance was closely monitored. As Figure 1 illustrates, CPU utilization reached its peak during the data processing phase, indicating a high computational load. This sustained usage was a key factor in the overall task completion time. Memory usage, though significant, remained within expected limits."

o-30

   - Visual Context: (Associated with "Figure 1: Global Temperature Anomaly") - .

     (This chunk combines the text that describes the data presented in Figure 1 with the image itself, forming a complete unit of information.)

5. Index these multimodal chunks to allow models to access rich, contextually cohesive information combining both visual and textual data.

Two-chunking

Two approaches to handling multimodal documents in RAG

- Multimodal documents like PDFs, slide decks, and reports often contain a mix of text, tables, images, and diagrams.
- There are two general strategies for making these documents searchable in a RAG pipeline:
  - Vision-guided chunking (Traditional Approach)
    - Provides structured, explainable retrieval at the chunk level.
  - Whole-page visual embedding
    - offers flexibility and layout preservation, without needing manual chunking.

Vision-guide-context

Vision-guided chunking (traditional approach)

- Segments documents into meaningful chunks that include both text and associated visuals.
- Process:
  1. Identify regions of interest (e.g. paragraph + diagram).
  2. Group related text and images into one chunk.
  3. Embed each chunk using a large multimodal model.
- Ideal when semantic relationships between text and visuals need to be preserved in context.

Whole-page visual embedding (ColPali example)

- Avoids traditional chunking by treating each page as a single image and retrieving information based on patch-level analysis rather than text chunks.
- Process:
  1. Converts each document page into a high-resolution image.
  2. Applies a Vision-Language Model (VLM) to divide the image into spatial patches (e.g. 16×16 grid regions).
  3. Each patch is embedded independently, capturing both visual and textual features.
  4. During retrieval, a user query is embedded and compared with all patch embeddings to identify relevant regions on the page (this is known as late interaction).
  5. The system highlights patches with high similarity scores, effectively retrieving information at the sub-page level.
- Ideal for scanned or visually dense documents.

Hugging Face Blog (2024): ColPali: Vision-Language Retrieval for Multimodal Documents https://huggingface.co/blog/manu/colpali

How to choose a chunking strategy

- For plain text:
  - Use fixed-size or sentence-based chunking for simplicity.
- For documents with structure:
  - Use recursive or task-oriented chunking to preserve hierarchy.
- For visual or scanned content:
  - Use vision-guided chunking to keep images and text together.
- Match chunk size to the task:
  - Factual retrieval: 64–128 tokens per chunk
  - Broader summaries: 512–1024 tokens
  - High-context reasoning: up to 2048 tokens (if supported)

There's no one-size-fits-all solution.

Strategies must be tuned based on:
- The type of content
- The retrieval method
- The model's token limits
- The application's end goal

Bhat, S. R., Rudat, M., Spiekermann, J., & Flores-Herr, N. (2025). Rethinking Chunk Size For Long-Document Retrieval: A Multi-Dataset Analysis. arXiv preprint arXiv:2505.21700.

Han, S. (2025, June 18). Finding the best chunking strategy for accurate AI responses. NVIDIA Developer Blog. Retrieved July 2, 2025

Trade-Offs-time

Trade-Offs

- Smaller chunks
  - More precise, faster, and cheaper
  - May lose broader context
- Larger chunks
  - Preserve more meaning and context
  - Risk exceeding model limits or including irrelevant information
- Overlapping chunks
  - Improve continuity, especially in narrative text
  - Increase storage and processing time

What-space

What is a Vector Database?

- A vector database stores information as high-dimensional vectors instead of rows and columns.
- Each vector is a list of numbers representing the meaning (semantics) of text, images, or other data.
- Example:
  - "Apple (fruit)" → [0.12, 0.95, …]
  - "Apple (company)" → [0.84, 0.34, …]
- Similar items have vectors close together in space.

Why-space

Why Vector Databases are needed

- Traditional databases rely on exact keyword matches.
- Semantic search needs to find items with similar meanings, not just similar words.
- Vector databases use mathematical similarity measures:
  - Cosine similarity → angle between vectors
  - Euclidean distance → distance in vector space

Core-documents

Core workflow of a Vector Database

- Data to Vectors
  - Use embeddings (e.g., OpenAI, BERT) to convert text/images to vectors.
- Indexing
  - Organise vectors for fast similarity search using algorithms like HNSW or FAISS.
- Querying
  - Convert query → vector → find nearest vectors → return matching documents.

Key-operations

Key components inside a Vector Database

- Vector Store:
  - Stores the vector embeddings.
- Indexing Engine:
  - Speeds up search using Approximate Nearest Neighbour (ANN) methods.
- Metadata Store:
  - Keeps additional info (e.g., titles, IDs) linked to each vector.
- Query Interface:
  - API for search, insert, delete operations.

Why-production

Why Vector Databases matter in RAG

- Store chunked documents as vectors for fast retrieval.
- Enable semantic search for context before generation.
- Support hybrid search: keyword + vector queries together.
- Essential for scalable RAG pipelines in production.

Popular-integrate

Popular Vector Databases

Database	Free Tier?	Special Features	Typical Use Case
FAISS	Yes	GPU support, fast ANN search	Local prototypes
Milvus	Yes (Community)	Scalable, cloud-native	Production search systems
Pinecone	Limited Free Tier	Fully managed, easy to integrate	Enterprise SaaS integrations
Qdrant	Yes	Open-source, hybrid search	Hybrid RAG pipelines
Weaviate	Yes	Built-in ML modules	Semantic + hybrid search
Chroma	Yes	Simple setup, easy LangChain integration	Lightweight RAG & rapid prototyping

Unauthorized-prototyping