Multimodal RAG Strategies - Research & Implementation Report

Prepared for:
Architecture, Engineering & Construction (AEC) Project – Multimodal RAG Implementation

By
Priti Borse

Date
October 7, 2025

TABLE OF CONTENTS

1. What it is & how it works?

Multimodal Retrieval-Augmented Generation (RAG) extends traditional RAG by including multiple modalities such as text, images, diagrams, and tables. This is particularly useful in the construction domain, where critical information is often stored in blueprints, annotated site photos, scanned documents, and safety charts.

Think of it as searching both in a library of manuals (text) and a photo gallery of site inspections (images). Instead of relying only on text search, multimodal RAG ensures that images, layouts, and diagrams can also be retrieved and understood.

---

2. Problem Statement:

In the Architecture, Engineering, and Construction (AEC) domain, a large portion of critical project information exists in **visual formats** such as blueprints, site photographs, and inspection diagrams. Traditional **text-based RAG systems** are unable to interpret or retrieve insights from such non-text data effectively.

The aim is to design and evaluate a Multimodal RAG system capable of accurately processing, aligning and retrieving information from both textual and visual content found in construction documents such as PDFs, drawings, floor plans, reports and embedded images. The core challenge is to identify an optimal embedding strategy that can uniformly represent and integrate heterogeneous data OCR text, complex images, technical diagrams and structured layouts so that the system can correctly interpret and retrieve information regardless of modality. This requires solving key issues in modality alignment, cross modal semantic understanding, retrieval efficiency, embedding quality and scalability while ensuring cost effectiveness. The ultimate goal is to determine which embedding method or combination of methods best captures the relationships between text and images so that multimodal queries are answered accurately, consistently and with high semantic fidelity.

---

3. Three Main Strategies to process images in Multimodal RAG:

1. Embed all modalities into the same vector space:

In this approach, both **text and images** converted into a **shared vector space** using models like **CLIP**. Queries, whether text or image yare embedded into the same space, allowing semantic similarity searches to work across modalities. For answer generation, a **multimodal LLM** is used to combine retrieved content into coherent responses.

- **How it works:** Replace the text embedding model with a multimodal embedding model. Retrieval pipeline remains mostly unchanged. Generation uses a multimodal LLM.
- **Example:** Searching "worker without helmet" retrieves images of workers lacking PPE alongside relevant text notes.
- **Pros:** Simple pipeline; seamless integration into existing text-based RAG systems.
- **Cons:** Requires embeddings that capture fine-grained visual and textual details; may struggle with complex tables or annotations in images.

---

2. Ground all modalities into one primary modality:

Here, a **primary modality** is chosen based on the main focus of the application (commonly text). Other modalities, such as images or tables, are **converted into text or metadata** during preprocessing. Retrieval operates mainly over the primary modality, while generation can leverage both LLMs and multimodal LLMs as needed.

- **How it works:** Images are described with text and metadata, stored alongside text chunks. Retrieval matches queries to these descriptions. The actual image can still be referenced if required for verification.
- **Example:** Diagrams are converted into descriptive text: "Scaffolding shows worker without a hard hat." Queries like "Find unsafe workers" match these descriptions.
- **Pros:** Works with existing text pipelines; rich metadata from images improves answer accuracy.
- **Cons:** Preprocessing overhead; some visual nuances may be lost in text conversion.

---

3. Separate stores for different modalities:

Each modality (text, images, tables) is stored in a **separate vector database**. Queries retrieve top-N results from each store, which are then combined and **re-ranked by a multimodal re-ranker** to generate the final response.

- **How it works:** Separate indices maintain modality-specific embeddings. After retrieval, a re-ranker evaluates relevance across modalities and merges results.
- **Example:** Text DB returns inspection notes; image DB returns photos of cranes. A re-ranker integrates both to report "Inspection passed, but photo shows worn cable."
- **Pros:** Avoids forcing one model to handle multiple modalities; each store can use optimized models.
- **Cons:** Requires additional re-ranking step; retrieval complexity increases.

4. Categorization of Multimodal RAG Techniques into the Three Core Strategies:

Strategy 1: Embed All Modalities into the Same Vector Space

1. Vision-Language Embedding (CLIP, BLIP, SigLIP)

Description:
These models are trained on huge datasets of images paired with captions (e.g., "a dog chasing a ball"). The model learns to place the text "dog" close to an actual dog picture in vector space. This lets you search across modalities: a text query can pull out an image, and an image can retrieve related text.

Flow with example:

- **Query:** "worker without helmet"
- **Process:**
    1. CLIP embeds the text query into vector space.
    2. Stored image embeddings of site photos are compared using cosine similarity
- **Retrieved Image:** Photo of a worker on scaffolding not wearing a hard hat.
- **Response (via LLM):** "Found worker without hard hat on the east scaffolding section."

Why use it? Perfect for general-purpose image–text retrieval like **shopping catalogs, memes, stock photos**.

Pros: Cross-modal, fast, widely used.
Cons: Coarse – only recognizes broad patterns, might miss small details, not for graphs/charts

---

2. Multi-Vector Representation (ColBERT, ColPali)

Description:
Instead of representing an image with one single vector, break it into multiple smaller vectors. Each patch (like "window," "fire exit," "stairs") is stored separately. At search time, the query can match just the region that's relevant.

Flow with example:

- **Image:** Blueprint of a floor plan with fire exits, walls, and windows.

1. **Process:**
    1. Blueprint is divided into multiple patches (e.g., fire exit on east wall, window section, stairs).
    2. Each patch is embedded separately into the vector space.

- **Query:** "fire exit on east wall"
- **Retrieved Patch:** Embedding corresponding to the fire exit region.
- **Response (via LLM):** "Fire exit located on east wall near stairwell #3, clearly marked on blueprint."

Why use it? Great for fine-grained search where **specific parts of an image** matter (e.g., product logos, specific organs in medical scans).

Pros: Very precise, supports "region-aware" search.
Cons: Needs more storage(many vectors per img) and compute, complex architecture

---

3. Hybrid OCR + Vision Embedding

Description:
Some images contain both **visual elements** and **text inside the image** (like signs, posters, diagrams). This method captures both: OCR extracts text, while a vision model embeds the visual appearance. Both embeddings are stored so retrieval works even if a user asks about text inside the image.

Flow with example:

- **Image:** A safety sign on-site saying **"Wear safety goggles"** with a worker wearing goggles.
- **OCR extracts:** "Wear safety goggles" → stored as text embedding
- **CLIP embeds image:** visual features of worker + sign → stored as visual embedding
- **Query:** "safety posters about goggles" → retrieval finds the image via text or visual similarity
- **Response: LLM generates:** "Safety poster shows worker wearing goggles. Text: 'Wear safety goggles'"

Why use it? Best for **mixed-content images** (infographics, street photos, memes).

Pros: Covers both worlds — text + visual.
Cons: Double storage & extra compute needed.

Strategy 2: Ground All Modalities into Text/Structured Data

4. Image Captioning + Text RAG

Description:
Use a captioning model to turn images into short sentences like "A brown dog is running in a park." These captions are stored as text chunks alongside other documents. When you search, retrieval happens purely on text.

Flow:

- **Image:** Construction worker on scaffolding wearing safety vest but no helmet
- **Caption generated using BLIP:** "Construction worker on scaffolding wearing a safety vest"
- Store caption as text chunk in DB
- **Query:** "Find workers not wearing hard hats"
- Text embedding search retrieves relevant caption
- LLM generates final response using caption + other text chunks

Why use it? Simple, works with **existing text-only infrastructure** without any multimodal indexing.

Pros: Cheap, easy to implement.
Cons: Captions may be too generic or miss details (e.g., "animal" instead of "Siberian Husky").

---

5. Image-to-Text with VLMs (GPT-4V, Gemini, Claude)

Description:
A multimodal LLM can look at an image and produce a much richer description than a captioning model. For example, not just "a cat on a sofa," but "a gray tabby cat curled up on a red sofa with sunlight coming through a window." This detail gives better recall during retrieval.

Flow:

- **Image:** Construction site photo showing a worker on scaffolding without a hard hat, with a crane in the background
    1. **GPT-4V prompt**: "Describe all safety issues in this image"
    2. **VLM output:** "Worker on scaffolding is not wearing a hard hat. A crane is positioned nearby, and the area around the scaffolding is cluttered with materials."
    3. Store description as text chunk in DB
- **Query:** "Workers missing safety gear near cranes"
    1. Text search retrieves the detailed description
    2. LLM generates final response combining retrieved info

Why use it? Best when you need **very detailed understanding** of images (complex scenes, diagrams).

Pros: Captures rich context.
Cons: More costly, slower per image.

8. OCR + Document Layout Understanding and Structured Data Extraction (LayoutLM, Donut, Pix2Struct)

Description:
This technique goes beyond simple OCR. It **reads the text inside an image**, preserves **layout information**, and understands **document structure and hierarchy** (headers, tables, paragraphs, fields). Useful for scanned documents, inspection reports, blueprints, forms, or any structured document where precise context matters.

Flow:

• **Image:** Scanned equipment inspection report or floor plan PDF with labels, dimensions, and tables

1. **OCR extracts all text:**
    1. "Wall A: Load Bearing, 3.5m",
    2. "Wall B: Non-load bearing, 2.8m",
    3. "InspectorName: John Smith",
    4. "Date: 2024-10-07",
    5. "EquipmentID: Crane-12",
    6. "Status: Passed"
2. **Layout & hierarchy analysis preserves positions, titles, table cells, and field relationships:**
    1. Wall labels vs. dimensions
    2. Table rows and columns
    3. Key-value pairs in forms or inspection reports
3. **Advanced document model (LayoutLM, Donut, Pix2Struct) interprets semantic structure, creating structured JSON:**

```
{
 "Walls": {
  "Wall A": {"Type": "Load Bearing", "Height": "3.5m"},
  "Wall B": {"Type": "Non-load bearing", "Height": "2.8m"}
 },
 "Inspection": {
  "InspectorName": "John Smith",
  "Date": "2024-10-07",
  "EquipmentID": "Crane-12",
```

"Status": "Passed"
}

4. **Store structured text with layout metadata in the database.**

- **Query:** "Who inspected Crane-12 on October 7, 2024?"
  1. Search retrieves InspectorName field from structured JSON
  2. LLM generates response: "John Smith inspected Crane-12 on October 7, 2024."

Why use it? Designed for **invoices, forms, academic papers**.

Pros: Maintains exact structure.
Cons: Heavy models, more complex to deploy.

Strategy 3: Separate Stores for Different Modalities

9. Separate Vector Stores (Dual-Index)

Description:
In this approach, **each modality has its own vector database**—one for text, one for images, and optionally others like tables or drawings. Queries retrieve top-N results from each store, which are then **merged and ranked** to generate the final response. This is useful when **text and image content have very different retrieval characteristics**, like project notes versus site photos

Flow:

- **Query**: "Crane inspection on Site A"
  1. **Text DB:** Retrieves inspection notes: "Crane-12 inspected on 2024-10-07, minor cable wear noted"
  2. **Image DB:** Retrieves site photo of Crane-12 with visible cable wear
  3. Multimodal re-ranker merges results based on relevance
  4. **LLM generates final response:** "Inspection on Site A for Crane-12 shows minor cable wear (photo attached). Notes confirm inspection completed on 2024-10-07."

Why use it? Best when modalities have **different query patterns** (e.g., sometimes text-heavy, sometimes image-heavy).

Pros: Flexible, modular.
Cons: Requires fusion logic to combine results well.

---

10. Late Fusion (Retrieve Then Describe)

Description:
In this approach, **images are not processed or embedded upfront**. Instead, the retrieval

pipeline first identifies the most relevant candidates using **text or coarse embeddings**, then a **vision-capable LLM (VLM)** analyzes only these top results for detailed understanding. This is ideal when **images are expensive to process** or appear infrequently in queries.

Flow:

- **Query:** "Worker on scaffolding without safety harness"
  1. Retrieve top 20 site photos based on text notes or general embeddings
  2. GPT-4V / Multimodal LLM analyzes top 5 images
  3. Identifies image showing worker missing harness
  4. **LLM generates final response:** "Photo shows a worker on scaffolding without a safety harness. Immediate safety action required."

Why use it? Saves cost when images are **rare** or **expensive to process**.

Pros: Efficient, scalable.
Cons: Slower per query, risk of missing info.

5.Hybrid Technique: Multimodal Floor Plan Understanding & Retrieval:

1.Problem Statement:

Traditional single-technique methods for analyzing floor plans and technical drawings fall short when applied to real-world AEC (Architecture, Engineering, Construction) use cases.

- **OCR-only approaches** can extract text but fail to preserve spatial relationships.
- **Vision-only models** recognize components but cannot link them to dimensions or annotations.
- **Embedding-only methods** allow retrieval but lack structural reasoning and adjacency checks.
- **Graph-only methods** enable connectivity analysis but require accurate component metadata from other sources.

As a result, existing systems cannot reliably:

1. Identify all components (rooms, doors, windows, furniture) with dimensions,
2. Preserve layout and adjacency for spatial queries,
3. Answer compliance and accessibility-related questions,
4. Scale to multi-floor or multi-building search.

To overcome these limitations, we propose a **Hybrid Multimodal Approach** that combines the strengths of segmentation, Document AI, vision transformers, graph reasoning, and hierarchical embeddings to deliver both **basic extraction tasks** and **complex spatial reasoning** within the same pipeline.

2.Description:

The Hybrid Approach integrates **Document AI** for OCR and layout, **Fine-tuned Vision Transformers** for domain-specific classification, **Semantic Segmentation** for boundary detection, **Symbol Recognition** for component identification, **Graph Neural Networks** for spatial reasoning, **Hierarchical Embeddings** for semantic search, and **RAG with LLM** for intelligent query handling.

3.Technqiues integrated into tiers:

To better structure the hybrid approach, the techniques are organized into **three tiers** based on their role in the pipeline:

- The **Extraction Layer** handles raw floor plan processing, separating components and linking annotations.
- The **Reasoning Layer** builds structured relationships and embeddings for advanced queries.
- The **Interaction Layer** connects everything to the user, enabling natural language queries and intuitive visualization.

This tiered design makes the system easier to understand, while also reflecting the actual order of execution—from raw data to user-facing insights.

Example (single scenario used throughout):

Input image: floorplan_floor2.png - a scanned 2D office floor plan (Floor 2).
Key visible elements: Lobby, Main Corridor, Cafeteria, Conference Room, Manager Office, Stairs, Elevator, Emergency Exit, many doors and walls with annotated dimensions.
Title block / scale: Scale: 1:100 (extracted by Document AI).

User query: "Find an accessible path from **Lobby** to **Cafeteria** avoiding stairs; list rooms adjacent to **Main Corridor** whose door to the corridor is **wider than 1.2 m**; and identify load-bearing walls on this floor longer than **5.0 m**."

Tier 1- Extraction Layer (Converts raw blueprint into structured components, annotations & symbols):

1. Semantic Segmentation for Boundary Detection (SAM / Mask R-CNN):

Goal: Separate the image into pixel masks for rooms, doors, windows, walls, stairs, elevator, and furniture.

How it works:

- Preprocess image (binarize/deskew / line enhancement) for crisp boundaries.
- SAM/Mask R-CNN runs instance segmentation → returns masks and bounding boxes.
- Post-process: polygonize masks, compute mask centroids and per-mask pixel extents.

Representative output:


```
{
  "masks": [
    {"id":"room_lobby","type":"room","polygon":[...],"bbox":[40,120,260,300],"area_pixels":42000},
    {"id":"room_corridor","type":"corridor","polygon":[260,120,620,300],"bbox":[260,120,620,300]},
    {"id":"room_cafeteria","type":"room","polygon":[620,120,860,420],"bbox":[620,120,860,420]},
    {"id":"door_001","type":"door","polygon":[250,200,260,220],"bbox":[245,198,263,222],"width_pixels":150},
    {"id":"door_002","type":"door","polygon":[600,220,610,240],"bbox":[598,218,612,242],"width_pixels":130},
    {"id":"stairs_01","type":"stairs","polygon":[900,80,960,180],"bbox":[900,80,960,180]}
  ]
}
```


Notes about pixels → meters: segmentation reports width_pixels. We convert to meters after Document AI provides the scale (next step).

Edge-case risk: overlapping text/lines can create split or merged masks (so post-processing heuristics are used).

---

2. Document AI (LayoutLMv3 / Pix2Struct):

Goal: Extract text/annotations and map them to the exact spatial locations (crucial for dimensions & scale).

How it works:

- OCR + layout parse entire plan.
- Identify title-block metadata (e.g., Scale: 1:100) and dimension annotations near elements.
- Map textual dimension strings to the nearest segmented component (point-in-polygon or proximity with vector offsets).

Representative output:


```
{
  "metadata": {"scale":"1:100","units":"m"},
  "annotations": [
    {"text":"Lobby","coords":[120,160]},
    {"text":"Main Corridor","coords":[340,160]},
    {"text":"Cafeteria (Canteen)","coords":[740,220]},
    {"text":"Door D1: 1.50m","coords":[250,210],"linked_to":"door_001"},
    {"text":"Door D2: 1.30m","coords":[605,230],"linked_to":"door_002"},
    {"text":"Wall W1: 6.20m","coords":[480,90],"linked_to":"wall_01"}
  ]
}
```


Scale Conversion Example:

- **Input:** Scale 1:100 from title block
    1. door_001.width_pixels = 150 pixels
- **Calculation:**
    2. 1 pixel = 100 cm (from scale)
    3. 150 pixels = 150 × (100 cm / 100) = 150 cm = 1.50 m
- **Output:** door_001.width_m = 1.50

Important action: use the title-block scale to convert segmentation width_pixels to real-world meters (e.g., door_001.width_pixels -> 150 px -> 1.50 m).

Confidence score: How certain the OCR is about the extracted text AND its spatial relationship to visual elements.

Edge-case risk: Dimension text too close to lines may be mis-located — system flags any low-confidence mappings for human review.

---

3. Fine-tuned Vision Transformer (ViT/Swin):

Goal: Produce domain-aware labels and identify structural element properties not explicitly annotated (e.g., load-bearing vs partition walls).

How it works:

- For each segmented region (room, wall, door), crop region and feed to fine-tuned ViT.
- ViT outputs labels and classification confidences and may estimate wall thickness patterns, hatchings → infer material.

Representative output:


```
{
  "rooms": [
    {"id":"room_lobby","label":"Lobby","confidence":0.98},
    {"id":"room_cafeteria","label":"Cafeteria","confidence":0.96}
  ],
  "walls": [
    {"id":"wall_01","classification":"load-bearing","confidence":0.92,"length_text_from_DocAI":"6.20m"},
    {"id":"wall_02","classification":"partition","confidence":0.88}
  ]
}
```


Important action: ViT flags wall_01 as load-bearing and Document AI says it's 6.20 m — this satisfies "> 5.0m" requirement.

Confidence score: How certain the model is about room type, wall classification, or material identification.

Edge-case risk: If drawing conventions differ, ViT might misclassify (requires retraining or active learning).

---

4. Symbol Recognition (Custom YOLO / classifier):

Goal: Detect small but crucial symbols: stairs, elevator, wheelchair/ramp, emergency exits, fire equipment.

How it works:

- Run symbol detector across image.
- For each detection, find which room polygon contains it (spatial grounding via point-in-polygon).

Representative output:


```
{
  "symbols":[
    {"id":"sym_stairs_01","type":"stairs","coords":[930,120],"in_room":"stair_core"},
    {"id":"sym_elevator_01","type":"elevator","coords":[880,210],"in_room":"service_core"},
    {"id":"sym_emexit_01","type":"emergency_exit","coords":[860,420],"in_room":"corridor"}
  ]
}
```


Effect on query: We now know where the stairs are and can program the pathfinder to **avoid** any routes that require stairs.

Edge-case risk:

- Tiny or unconventional symbol drawings may be missed → fallback: OCR for legend mapping or human verification.

- What happens if symbol is ON a boundary (e.g., door between two rooms)?
    1. **Edge Case Handling:**
        - Symbol inside room polygon → assign to that room
        - Symbol on boundary (door/wall between rooms) → assign to both rooms with relationship:

```
{
  "symbol_id": "door_001",
  "type": "door",
  "connects": ["room_lobby", "room_corridor"],
  "location": "boundary"
}
```

Tier 2- Reasoning Layer (Use extraction outputs to build reasoning artifacts and run the actual query):

5. Graph Construction & Graph Neural Network (GNN):

Goal: Represent rooms and connections as a graph so we can compute paths and apply filters like door_width > 1.2m and avoid stairs.

How it works:

- **Nodes:** created for each room/space (lobby, corridor, cafeteria, conference, manager_office, stair_core).
- **Edges:** create an edge for each door connection; annotate edges with attributes door_id, width_m, door_type.
- Build graph in PyTorch Geometric / store in Neo4j if persistence/queries needed.
- Use graph algorithms (or a trained GNN) to compute constrained shortest paths.
- Door-to-Room Assignment Logic:
    1. For each door from segmentation (door_001, door_002...):
    2. Find door centroid coordinates: door_001 at (125, 145)
    3. Check which room boundaries the door touches/overlaps:
        - Overlaps room_lobby boundary? YES (edge pixels match)
        - Overlaps room_corridor boundary? YES (edge pixels match)
    5. Create graph edge:
    6. edge: (room_lobby ↔ room_corridor)
    7. attributes: {door_id: "door_001", width_m: 1.50}
    8. If door only overlaps one room → external door (to outside)

Representative output:


```
{
  "nodes":["room_lobby","room_corridor","room_cafeteria","room_conf","room_mgr","stair_core"],
  "edges":[
    {"from":"room_lobby","to":"room_corridor","door":"door_001","width_m":1.50},
    {"from":"room_corridor","to":"room_cafeteria","door":"door_002","width_m":1.30},
    {"from":"room_corridor","to":"room_conf","door":"door_003","width_m":0.90},
    {"from":"room_corridor","to":"room_mgr","door":"door_004","width_m":0.85},
    {"from":"room_corridor","to":"stair_core","door":"door_005","width_m":0.9,"stairs_access":true}
  ]
}
```


Run the example query on the graph:

- Pathfinding constraint: avoid nodes/edges that require going through stair_core or edges marked stairs_access:true.
- Accessibility constraint (optional): ensure doors widths >= 0.85 m (wheelchair) — this example focuses on avoiding stairs, but we also have widths.

Resulting path:
room_lobby → (door_001, 1.50 m) → room_corridor → (door_002, 1.30 m) → room_cafeteria. All edges avoid stairs and exceed 0.85 m.

Also run adjacency filter: find nodes adjacent to room_corridor whose connecting door width > 1.2 m.
→ room_lobby (1.50 m), room_cafeteria (1.30 m). (Conference and Manager Office have doors 0.90 m and 0.85 m and are excluded.)

Edge-case risk: if a door was missed in segmentation, the graph would be incomplete — system logs missing/low-confidence edges and can fallback to manual check.

6.Hierarchical Multi-Scale Embeddings (FAISS / SentenceTransformers / CLIP)

Goal: Support semantic search (synonym handling, fuzzy queries) and multi-granularity retrieval.

How it works:

- **Purpose 1:**
  1. Synonym/Fuzzy Matching - User says "cafeteria", plan says "canteen"
  2. Semantic embeddings match despite different words
- **Purpose 2:**
  - **Multi-Granularity Search –**
    1. **Building-level:** "Find similar office buildings"
    2. **Floor-level:** "Show me layouts like Floor 2"
    3. **Room-level:** "Find conference rooms like this"
    4. **Component-level:** "Show similar window configurations"
- Each level has its own embeddings, enabling queries at any zoom level.

Representative output:

- **Text query** → embed ("cafeteria" or "canteen").
- **Room-level embeddings searched** → finds room_cafeteria with high cosine similarity.
- **Embedding retrieval helps confirm target node for graph queries.**

Edge-case risk: When Document AI misses a small label, embeddings + visual similarity can still find the intended room.

Tier 3 - Interaction Layer (Orchestrate and present the result.):

7. RAG with LLM Orchestration (LangChain / LlamaIndex + GPT/GGML):

Goal: split the complex natural query into sub-tasks, route them to the right systems, and synthesize the final answer.

How it works:

LLM Decision Logic (Routing):

- **Sub-task A:** "Accessible path Lobby → Cafeteria, avoid stairs"
    1. **Keywords:** "path", "avoid"
    2. **Requires:** Pathfinding + constraints
    3. **Route to**: Graph Neural Network (shortest path with filters)
- **Sub-task B:** "Rooms adjacent to Main Corridor where door_width > 1.2m"
    1. **Keywords:** "adjacent", numeric filter "> 1.2m"
    2. **Requires:** Graph traversal + attribute filtering
    3. **Route to:** GNN (adjacency) + SQL-style filter on edges
- **Sub-task C:** "Load-bearing walls > 5.0m"
    1. **Keywords:** structural property + dimension filter
    2. **Requires:** Classification data + measurements
    3. **Route to**: Document AI (dimensions) + ViT (classification) + SQL filter
- **LLM Prompt Example:**

  "Analyze query and return routing plan as JSON:
  {
    'sub_tasks': [
      {'task': '...', 'requires': ['graph', 'spatial_filter'], 'route_to': 'GNN'},
      ...
    ]
  }"

Representative output:


```
{
  "query":"accessible_path_and_checks",
  "results": {
    "path": {
      "sequence":["room_lobby","room_corridor","room_cafeteria"],
      "doors":[{"id":"door_001","width_m":1.50},{"id":"door_002","width_m":1.30}],
      "avoids_stairs": true,
      "total_distance_m": 18.0
    },
    "adjacent_rooms_with_doors_gt_1.2m":[
      {"room":"room_lobby","door_id":"door_001","width_m":1.50},
      {"room":"room_cafeteria","door_id":"door_002","width_m":1.30}
    ],
    "load_bearing_walls_gt_5m":[
      {"wall_id":"wall_01","length_m":6.20,"location":"north exterior"}
    ]
  },
  "nl_summary":"An accessible path from the Lobby to the Cafeteria is: Lobby → Main Corridor → Cafeteria. Doors
along the route are 1.50 m and 1.30 m (both avoid stairs). Rooms adjacent to the Main Corridor with doors wider
than 1.2 m are the Lobby (1.50 m) and Cafeteria (1.30 m). One load-bearing wall longer than 5.0 m was found:
wall_01 (6.20 m, north exterior)."
}
```


Edge-case risk: If any subcomponent has low confidence, the response should include a confidence flag and suggest manual review (e.g., "door_002 width confidence 0.65 — verify").

---

8. Interactive Visualization (OpenCV / Plotly / Web UI):

Goal: Communicate results visually and enable validation.

What gets rendered:

- **Path overlay:** highlight corridor polygon and successive door bounding boxes (green line/arrows).
- **Adjacent-room highlights:** Lobby and Cafeteria shaded with a border and annotation showing door widths.
- **Load-bearing wall highlight:** wall_01 colored red with label 6.20 m (load-bearing).
- **Sidebar:** structured JSON and confidence scores; a toggle to show/hide each overlay layer.
- **Export:** annotated PNG / interactive HTML with clickable components (shows underlying JSON).

Final consolidated answer (what the system returns to the user):

- **Natural language:**

Accessible path found: Lobby → Main Corridor → Cafeteria.
Doors on route: Door D1 = 1.50 m, Door D2 = 1.30 m (both pass width checks and avoid stairs).
Rooms adjacent to the Main Corridor with door width > 1.2 m: Lobby (1.50 m) and Cafeteria (1.30 m).
Load-bearing walls on Floor 2 longer than 5.0 m: Wall W1 (6.20 m) on the north exterior.

Confidence: high for segmentation and Document AI mappings; one symbol detection had medium confidence (marked for review).

- **Machine-readable:** the JSON shown above (structured results, coordinates, confidence levels).
- **Visual:** annotated floor plan with overlays for path, doors, and wall.

7.Use Cases with Example User Queries (Simple + Advanced):

Simple User Queries (Baseline Scenarios):

Query Type 1: Room & Area Checks

- *"What's the area of the kitchen?"*
- *"List all rooms with their dimensions."*
  Handled by: Document AI + Segmentation (text + masks).

Query Type 2: Component Counts

- *"How many windows are in this floor plan?"*
- *"Count the doors in the corridor."*
  Handled by: Semantic Segmentation + Symbol Recognition.

Query Type 3: Object Location

- *"Where is the elevator located?"*
- *"Show me all fire extinguishers."*
  Handled by: Symbol Recognition + Spatial Grounding.

Query Type 4: Simple Spatial Adjacency

- *"Which room is next to the living room?"*
- *"What's beside the staircase?"*
  Handled by: Graph Construction (adjacency lookup).

Query Type 5: Metadata Extraction

- *"What is the scale of this drawing?"*

- *"Who is the architect listed in the title block?"*
  Handled by: Document AI (title block parsing).

Use Case 1: Understanding 2D Floor Plans:

User Query Examples:

- *"What's the total area of the living room and kitchen?"*
- *"How long is the wall between the two bedrooms?"*
- *"List all rooms larger than 20 sqm with at least 2 windows."*

Hybrid Solution:

1. **Document AI (LayoutLM / Pix2Struct)**
   → Extracts text + dimensions.
   → Output: {"Living Room": "24 sqm", "wall_length": "5.2m"}
2. **Semantic Segmentation (SAM / Mask R-CNN)**
   → Identifies room boundaries + component masks.
3. **Hierarchical Embeddings**
   → Enables similarity queries ("show layouts like this").

Coverage: Area, dimensions, annotations linked to correct rooms.

Use Case 2: Understanding 3D Floor Plans / Renderings:

User Query Examples:

- *"Show me the 3D path from the lobby to the conference room."*
- *"Which rooms on Floor 2 have south-facing windows?"*
- *"Is there a step-free path to the elevator in this 3D layout?"*

Hybrid Solution:

- **Preprocessing:** Convert 3D render into multiple 2D views.
- **Pipeline:**
    1. Document AI extracts annotations (text/dimensions).
    2. Segmentation detects rooms & components.
    3. Fine-tuned ViT recognizes 3D spatial relationships.
    4. Aggregate embeddings across views into single floor representation.

Coverage: Complex 3D → 2D → structured reasoning.

Use Case 3: Complex Technical Diagrams (MEP – Mechanical, Electrical, Plumbing)

User Query Examples:

- *"Trace the water flow from the main supply to Zone 3."*
- *"List all valves connected to Chiller-01."*
- *"Find ducts longer than 10m connected to Air Handling Unit."*

Hybrid Solution:

1. Document AI → Extracts labels like *"6-inch supply pipe"*.
2. Symbol Recognition → Detects valves, pumps, ducts, panels.
3. **Graph Construction** → Nodes (equipment), edges (connections).
4. Region-based embeddings (ColPali) → Equipment groupings.

Coverage: Tracing connectivity, filtering by capacity/length.

Use Case 4: Symbol Recognition & Location Tracking:

User Query Examples:

- *"Count all smoke detectors on Floor 1."*
- *"Which rooms have fire extinguishers within 5m of the door?"*
- *"Locate all electrical outlets in the kitchen."*

Hybrid Solution:

1. Symbol Detection (YOLO).
2. Spatial Grounding → Assign symbols to rooms/walls.
3. Structured Storage (SQL + GeoJSON).

Coverage: Safety checks, equipment counts, geospatial queries.

Use Case 5: Compliance & Code Checking

User Query Examples:

- *"Does every bedroom have at least 2 outlets?"*
- *"Check if exit distance from each room is < 15m."*
- *"Are all bathroom doors at least 80 cm wide (wheelchair standard)?"*

Hybrid Solution:

1. Collect data (outlet counts, door widths, exits).
2. Graph Traversal → Path/distance checks.
3. Structured Queries → Count/dimension validation.
4. LLM Reasoning → Interpret building codes.
5. Report Generation → JSON + human-readable results.

Coverage: Automated safety compliance verification.

Use Case 6: Semantic Search ("Find layouts like this")

User Query Examples:

- *"Show me layouts similar to this office floor plan."*
- *"Find houses with open kitchens next to living rooms."*
- *"Retrieve buildings with 3 bedrooms on the same floor."*

Hybrid Solution:

1. Multi-scale embeddings (building, floor, room, component).
2. Vector similarity search (cosine similarity).
3. Fine-tuned ViT → Understands style (modern vs traditional).

Coverage: Design inspiration, layout retrieval.

Use Case 7: Spatial Relationship Queries

User Query Examples:

- *"List all rooms directly connected to the main corridor."*
- *"Find the shortest accessible path from kitchen to bedroom."*
- *"Which windows face south?"*

Hybrid Solution:

1. **Graph Construction** → Nodes = rooms, Edges = doors/walls.
2. **Query Processing:**
    - "Adjacent to corridor" → Graph traversal.
    - "Path from A to B" → Shortest path search.
    - "South-facing windows" → Orientation analysis.

3. **Filters:** Apply width/type constraints.

Coverage: Advanced spatial queries beyond OCR or segmentation alone.

---

8. Optimal Multimodal Embedding Methods:

1. Hybrid Vision Embedding Method:

Brief Description:
The Hybrid Vision Embedding Method is a multimodal embedding strategy designed to jointly capture information from both **OCR-extracted text** and **visual elements** (images, drawings, floor plans, diagrams) in construction documents. Instead of relying on a single modality, this method generates **separate embeddings for text and images**, then combines them into a unified hybrid representation that improves cross-modal retrieval accuracy.

PHASE 1: INDEX GENERATION (ONE-TIME SETUP)

1. **Initialization:**
   The indexing process begins by running build_index.py. The system loads the construction PDF from the data directory and prepares all components for text and image extraction.
2. **Text Extraction:**
   The PDF is processed using the PyMuPDF library. Each page is scanned for text blocks. Section headers are detected through layout patterns and uppercase formatting, and related content is grouped under these headings. The system then creates semantically coherent text chunks, resulting in approximately 180 text segments representing the document's structure.
3. **Image Extraction:**
   The same PDF is scanned for embedded images, including diagrams, drawings, and photographs. Around 40 images are extracted, saved as PNG files, and stored under outputs/hybrid_index/extracted_images with structured naming based on page and index.
4. **OCR and Caption Processing:**
   For every extracted image, the system retrieves nearby page text to capture captions or labels. Image content is processed using OCR to extract any text present inside the image. Both sources are merged to form a descriptive text representation of the image.
5. **Model Loading:**
   Two pretrained models are loaded: Sentence-BERT for text embeddings and CLIP for image embeddings. These models convert textual and visual information into numerical vector representations.

6. **Text Embedding Generation:**
   Each text chunk is passed through Sentence-BERT to produce a 384-dimensional semantic embedding. Because text chunks do not contain visual content, a 512-dimensional zero vector is appended, forming a unified 896-dimensional hybrid vector for consistency across modalities.

7. **Image Embedding Generation:**
   For each extracted image, the merged OCR and caption text is embedded using Sentence-BERT (384 dimensions), while the image itself is encoded using CLIP (512 dimensions). Concatenating these vectors produces an 896-dimensional hybrid embedding representing both the textual and visual characteristics of the image.

8. **Vector Normalization:**
   All hybrid vectors are normalized to unit length to ensure consistent scaling and improve similarity search accuracy. This step ensures equal contribution between text-derived and image-derived embedding components.

9. **FAISS Index Construction:**
   A FAISS index is created using the set of normalized 896-dimensional vectors. The index is optimized for fast similarity search and is paired with a metadata store containing page numbers, chunk text, image paths, and content type markers.

10. **Persistence:**
    The final index is saved as faiss_index.bin, while metadata is stored in metadata.json. Extracted images remain in the corresponding output directory. This completes the one-time indexing process.

PHASE 2: QUERY PROCESSING (RUNTIME)

1. **System Initialization:**
   Running main.py loads the FAISS index, metadata, test queries, and the pre-trained embedding models required for query encoding.

2. **Query Embedding:**
   Each input question is encoded using Sentence-BERT to produce a 384-dimensional embedding. A 512-dimensional zero vector is appended to maintain the 896-dimensional hybrid embedding format. The resulting query vector is normalized before retrieval.

3. **Similarity Search:**
   FAISS performs a cosine similarity search between the query vector and all stored embeddings. The system ranks all matches and returns the top results along with similarity scores.

4. **Content Retrieval:**
   Metadata for the retrieved embeddings is loaded. Low-confidence results are filtered

out. The remaining text segments or image descriptions are collected as candidate answer material.

5. **Answer Extraction:**
   Relevant sentences are selected from the matched text and descriptions. The system removes redundancies, filters by semantic relevance, and constructs a coherent final response.

6. **Output Generation:**
   Each query and its corresponding answer are stored in a structured dictionary. Once all queries are processed, results are written to outputs/query_results.json.

DATA TRANSFORMATION SUMMARY:

PDF content is transformed through text extraction, image extraction, OCR processing, semantic embedding generation using Sentence-BERT and CLIP, concatenation into 896-dimensional hybrid vectors, normalization, FAISS indexing, and final retrieval through similarity search.

---

2. OCR + Document Layout Understanding and Structured Data Extraction (LayoutLM / Donut / Pix2Struct):

Brief Description:

This method extends traditional OCR by not only reading text from scanned documents or images but also understanding the document's visual layout, structure, and semantic relationships. Using advanced models such as LayoutLM, Donut, or Pix2Struct, the system captures both textual content and its spatial positioning—such as headers, tables, key-value fields, labels, and section hierarchies. The output is a structured, machine-readable representation (typically JSON) that preserves the document's organization. This approach is particularly effective for forms, inspection reports, engineering drawings, floor plans, invoices, and any document where context depends on layout accuracy. It enables precise field-level retrieval, reduces ambiguity, and supports highly accurate downstream querying and analysis.

Technical Process Flow:

PHASE 1: DOCUMENT INGESTION AND STRUCTURE ANALYSIS

Step 1: Input Preparation:
A scanned document or image-based PDF (inspection report, floor plan, form, blueprint, or equipment checklist) is loaded into the pipeline. The system prepares it for OCR and layout interpretation.

Step 2: Text Extraction (OCR Layer):
The OCR engine reads all textual content present in the document.
Example extracted tokens:

- "Wall A: Load Bearing, 3.5m"
- "Wall B: Non-load bearing, 2.8m"
- "InspectorName: John Smith"
- "EquipmentID: Crane-12"
- "Status: Passed"

The output at this stage contains raw text without structure.

Step 3: Layout and Hierarchy Detection:
A document layout understanding module analyzes the spatial structure of the page:

- Identifies bounding boxes, text blocks, key-value pairs
- Distinguishes between headers, paragraphs, tables, form fields
- Detects semantic zones such as section titles, labels, dimensions, legends, table rows

This preserves **positional relationships**, such as:

- Wall label → corresponding measurements
- Field name → field value
- Table cell ordering
- Form section hierarchy

Step 4: Structured Interpretation Using a Document Model:
A model such as **LayoutLM, Donut, or Pix2Struct** processes both the text and layout information together.
It converts the entire document into a structured, machine-readable format (typically JSON).

This removes ambiguity and ensures the document is stored with **clear relationships and hierarchy**.

Step 5: Persisting Structured Output:
The structured representation is stored in the system (database or index) along with layout metadata.
This enables precise retrieval of individual fields, tables, or sections.

PHASE 2: QUERYING AND RETRIEVAL

Step 6: Query Processing:
When a user asks a question such as:
"What is the area of xyz room?"
The query engine maps keywords to structured fields (area, dimension).

Step 7: Structured Search:
The system looks up the corresponding values in the structured JSON data.
Since relationships are explicit, no semantic guessing is required.

Step 8: Response Generation:
The LLM or rule-based formatter synthesizes the answer:
"The area of this room is 250sqft."

9.Comparison of Two Optimal Methods for Multimodal RAG

Hybrid Vision Embedding vs. OCR + Layout Understanding

---

1. Hybrid Vision Embedding (Sentence-BERT + CLIP)

Purpose

To build a unified embedding space where text and images can be compared, retrieved, and ranked by semantic similarity.

Why Implement It?

- Supports both textual and visual queries.
- Can retrieve information from diagrams, construction drawings, or image-based content without needing structured fields.
- Ideal for broad semantic search across mixed modalities.
- Enables scalable multimodal RAG systems.

Strengths (Pros):

- Works for any type of image: diagrams, photos, sketches, floor plans, scanned pages.
- Captures high-level meaning, not just text.
- Allows joint search of text + image content.
- Fast retrieval using FAISS.
- Minimal preprocessing—does not require layout detection.
- Robust even when OCR fails or images contain very little text.

Limitations (Cons):

- Does not understand structural relationships (table cells, form fields, hierarchical document layout).
- Cannot extract exact key-value fields from documents.
- Embeddings may mix unrelated visual and textual features if not tuned.
- Not suitable when precise field-level extraction is needed.

Best Use Cases:

- Construction drawings, diagrams, sketches
- Multi-page reports with both text and visuals
- General question answering across mixed content
- Searching for concepts, not exact fields

---

2. OCR + Layout Understanding (LayoutLM, Donut, Pix2Struct)

Purpose:

To read text from images and understand the document's structural layout—sections, tables, labels, and key-value fields—to generate structured data.

Why Implement It?

- Extracts field-level details from technical documents, inspection reports, blueprints, or labeled diagrams.
- Converts complex pages into machine-readable JSON.
- Enables precise querying of specific fields.

Strengths (Pros):

- Preserves document layout, hierarchy, and relationships.
- Highly accurate for forms, tables, labels, and structured content.
- Produces structured output suitable for databases.
- Excellent for field extraction, compliance documents, invoices, inspection checklists, or dimensioned floor plans.

Limitations (Cons):

- Heavier and more complex models.
- Slower inference compared to embedding methods.

- Requires more preprocessing and GPU resources.
- Not suitable for photos or images that lack a clear document-like layout.

Best Use Cases:

- Inspection reports
- Floor plans with text labels
- Forms, invoices, structured PDFs
- Pages where meaning depends on position (top-left field, table row, section title)