Domain-Adaptive RAG Architectures: Generalizing Code-Centric Retrieval Primitives to Legal, Medical, and Technical Repositories

Executive Summary

The prevailing architecture of Retrieval-Augmented Generation (RAG) systems has historically bifurcated into two distinct lineages: unstructured prose retrieval, which relies on heuristic chunking of flat text, and structured code retrieval, which leverages the inherent syntax of programming languages. The **LLMC (Large Language Model Compressor)** system represents the apex of the latter, utilizing Abstract Syntax Tree (AST) parsing, function-level boundaries, and import graph analysis to navigate codebases with high precision. This research report investigates the theoretical and practical methodologies required to extend LLMC's architectural primitives to high-stakes, non-code domains: **Legal**, **Medical**, and **Technical/Enterprise** documentation.

Our analysis challenges the prevailing notion that these domains consist of "unstructured" data. Instead, we posit that legal contracts, clinical notes, and technical manuals possess latent hierarchical structures isomorphic to code ASTs. A legal contract is a compiled program governing human behavior; a medical record is a temporal log of system states; and technical documentation is a dependency graph of functional concepts. By treating legal clauses as functions, medical encounters as modules, and DITA topics as classes, we can extend LLMC's architecture without fundamentally altering its core logic.

This report details the transposition of code primitives to domain-specific analogues. We identify that **TreeSitter parsing** must evolve into **Layout-Aware Vision parsing** and **Regex Grammars** for legal texts; **Import Graphs** must generalize to **Citation Networks** and **Symptom-Disease Ontologies**; and **Code Embeddings** must be replaced by domain-specialized models like **Legal-BERT** and **MedCPT**. Furthermore, we propose a "Schema-First" retrieval approach for medical data, leveraging FHIR standards to enforce type safety in retrieval, mirroring static analysis in code. The following findings provide a comprehensive blueprint for this architectural expansion, identifying specific open-source models, parsing strategies, and evaluation benchmarks necessary to deploy a production-grade, local-first Domain RAG system.

---

1. Foundational Paradigm: The Isomorphism of Syntax

and Semantics

1.1 The Limitations of Text-Centric RAG

The current generation of RAG systems, often built on generic frameworks like LangChain or LlamaIndex, typically treats text as a flat sequence of characters. In this "Text-Centric" paradigm, documents are sliced into arbitrary windows (e.g., 512 or 1024 tokens) with fixed overlaps. While this approach is sufficient for general knowledge retrieval (e.g., "What is the capital of France?"), it fails catastrophically in domains requiring structural integrity.

In a legal contract, a sub-clause (ii) defining a termination penalty is semantically meaningless if severed from its parent clause Section 4.1: Termination for Cause. In a medical record, a "Plan" section listing medications is dangerous if conflated with a "Subjective" section listing patient allergies. Standard heuristic chunking destroys these boundaries, leading to "context amputation"—the retrieval of data without its governing constraints.

1.2 The LLMC Paradigm: Structure is Context

LLMC differentiates itself by respecting the *syntax* of the underlying data. In the code domain, this means parsing a Python file not as a string, but as a tree of classes, functions, and method calls. This structural awareness allows LLMC to retrieve a complete function definition rather than a fragmented snippet, preserving the semantic integrity of the logic.

To generalize this to non-code domains, we must formalize the "latent syntax" of prose. We propose that high-value professional documents are not unstructured; they are *semi-structured* objects compiled for human interpreters.

- **Legal Contracts** are programs that define variables (Definitions), execute logic (Covenants), and handle exceptions (Indemnification).
- **Medical Records** are temporal databases logging the state of a biological system (Patient) across discrete events (Encounters).
- **Technical Manuals** are directed acyclic graphs (DAGs) of tasks, concepts, and references.

1.3 Mapping Primitives: The Generalization Framework

The core research objective is to map the primitives of LLMC to these new domains. Our analysis identifies the following structural isomorphisms that will guide the architectural recommendations:

Code Primitive (LLMC)	Legal Equivalent	Medical Equivalent	Technical Equivalent
AST Node	Clause / Article /	FHIR Resource /	DITA Topic / XML

	Recital	SOAP Section	Node
Function Scope	Section Boundary	Encounter / Visit	Procedure / Task
Variable Definition	Defined Term ("The Company")	Patient Demographics / Vitals	Parameter Definition
Import / Call Graph	Citation / Cross-Reference	Symptom-Disease Relation	Component Dependency
Static Analysis	Compliance Check	Clinical Guideline Check	Schema Validation
Type Hinting	SALI Tags	ICD-10 / SNOMED Codes	Metadata / Taxonomy

This mapping informs the architectural decisions detailed in subsequent sections. The goal is to build a "Universal Parser" router that identifies the document type and dispatches it to the appropriate structural extraction pipeline—whether that be a TreeSitter grammar for Python, a Regex grammar for contracts, or an XML parser for technical manuals.

---

2. The Legal Domain: Codifying the Social Contract

Legal documents (contracts, case law, statutes, filings) are characterized by rigid structure, recursive definitions, and high consequences for hallucination. A code-focused RAG system is uniquely positioned to succeed here because legal drafting mimics software engineering: terms are defined globally, clauses supersede one another, and specific provisions reference external standards. The "spaghetti code" of a complex merger agreement requires a parser that can trace execution paths just as an IDE traces function calls.

2.1 Document Structure Extraction: The "Legal AST"

The fundamental unit of retrieval in law is not the "paragraph" or the "page," but the **provision** or **clause**. Standard chunking often severs the header of a clause from its sub-items, destroying the context.

2.1.1 Structural Units and Segmentation Strategy

Research indicates that legal documents should be segmented into **hierarchical units**.[1] The

equivalent of a "function" in a contract is a **Clause**, which often contains nested lists (sub-clauses).

- **Spans as Logical Units:** The extraction target should be the *entire* clause hierarchy. If Section 2.1 contains sub-points (a), (b), and (c), the retrieval unit must be the parent node 2.1 containing all children. This mirrors how LLMC retrieves a parent function including its nested helper functions.[3]
- **Metadata Propagation:** A critical failure mode in legal RAG is retrieving a clause like "The term shall be 5 years" without knowing *which* agreement or section it belongs to. Every span must be tagged with its full "path" (e.g., Document: NDA > Article IV > Section 4.2 > Subsection (a)).

2.1.2 Parsing Architectures: Regex vs. Vision

Unlike code, legal documents lack a strict compiler-enforced syntax, but they follow highly standardized conventions that can be exploited.

Option A: Regex Grammars (The "Linter" Approach)
For digitally native contracts (DOCX/Text), Regular Expressions are the most efficient "parser." Legal numbering follows predictable patterns (e.g., Article I, Section 2.1, (a), (i)).

- **Implementation:** We can define a "Legal Grammar" using Python's re module that functions similarly to a TreeSitter grammar. By identifying start tokens (e.g., ^ARTICLE\s+[IVX]+ or ^\s*(\d+)\.\s+), the parser can construct a hierarchy tree.[4]
- **Pros:** Extremely fast, local-first, zero latency.
- **Cons:** Brittle for poorly formatted scans or "dirty" OCR.

Option B: Layout-Aware Vision Models (The "Compiler" Approach)
For scanned documents (PDFs), multi-column briefs, or filings with embedded tables, regex fails. Here, Vision-Language Models (VLMs) are required.

- **Unstructured.io & LayoutLM:** Tools like **Unstructured.io** (specifically the hi_res or vlm strategies) utilize object detection models (like YOLOX or Detectron2) to classify layout elements.[6] They generate a bounding-box representation of the document, classifying text blocks as "Header," "Title," "Narrative," or "Table."
- **Surya & Nougat:** Recent advancements include **Surya** (for accurate line-level OCR) and **Nougat** (for academic/scientific layouts). Surya benchmarks competitively against cloud providers like Google Vision and is optimized for local inference.[8]
- **Recommendation:** Use a **Cascading Router**.
  1. Attempt purely textual analysis (Regex).
  2. If structure confidence is low (e.g., no clear section headers found), promote the document to a **VLM Partitioner** pipeline. This balances cost and accuracy.

2.2 Domain-Specific Enrichment: SALI Tags

In code RAG, we enrich chunks with docstrings or type hints to aid retrieval. In Legal RAG, we

enrich chunks with **SALI (Standards Advancement for the Legal Industry) LMSS tags**.

- **The Ontology:** The SALI LMSS is an open standard ontology of over 10,000 tags describing legal matters (e.g., <Negligence>, <California Law>, <Merger Agreement>, <Indemnification>).[10]
- **Enrichment Pipeline:** We employ a specialized "Tagger LLM" (a small, fine-tuned model or prompted Llama 3) that runs over every extracted clause during ingestion.
- **Prompt Strategy:**"Analyze the following contract clause. Identify relevant entities, jurisdiction, and classify the clause type using SALI LMSS standard tags (e.g., , , <Jurisdiction_NY>). Output JSON.".[11]
- **Semantic Filtering:** This allows for precise filtering. A user query "Show me indemnification clauses under NY law" becomes a filtered vector search tag:Indemnification AND tag:Jurisdiction_NY, guaranteeing precision that vector similarity alone cannot achieve.

2.3 Embedding Models: The Shift from General to Specific

Code embeddings (like CodeBERT) are ill-suited for legal "legalese," which contains archaic vocabulary and specific semantic weights (e.g., "consideration" means money/value, not "thought"). The landscape of legal embeddings has matured significantly.

Voyage Law (Voyage AI): The SOTA Standard

- **Performance:** Voyage AI's voyage-law-2 model is currently the state-of-the-art for legal retrieval. It is optimized for long-context legal reasoning and outperforms generalist models like OpenAI's text-embedding-3 on legal benchmarks such as **LegalBench**.[12]
- **Use Case:** Ideal for "Cloud-Tolerant" routes where maximum accuracy is required (e.g., litigation discovery).

Legal-BERT & SaulLM: The Local Champions

- **Legal-BERT:** A variant of BERT pre-trained on massive corpora of legislation and case law. It captures the distinct semantic space of law better than generic BERT.[14]
- **SaulLM:** A newer 7B parameter language model explicitly designed for legal text, which can be used to generate high-quality embeddings or for local reranking.
- **Recommendation:** For a local-first system like LLMC, **Legal-BERT** is the architectural choice for the embedding layer. It is lightweight, effective, and runs entirely offline.

2.4 Graph Extraction: The Citation Network

Legal documents are effectively hyperlinked databases. A case cites a statute; a contract amends a previous agreement; a regulation references a specific code section.

- **Graph Construction:** We can model this as a directed graph.
  - **Nodes:** Documents, specific Clauses (e.g., § 1441), Defined Terms, and Entities (Parties).

    - **Edges:** CITES, AMENDS, DEFINES, SUPERSEDES.
- **Extraction Technique:** Use Named Entity Recognition (NER) to detect standardized citation formats (e.g., "28 U.S.C. § 1441", "Section 5 of the Agreement").
- **GraphRAG Utility:** When a user asks "Does this clause comply with Section 5?", a standard vector search might fail if "Section 5" is not semantically similar to the query. However, GraphRAG can traverse the CITES or REFERS_TO edge to retrieve the text of "Section 5" (which might be in a different document) and inject it into the context window. This mimics "Go to Definition" in an IDE.[16]

---

3. The Medical Domain: Precision and Schema-First Retrieval

Medical RAG is the highest-stakes domain. "Hallucination" here translates to potential clinical error or adverse patient outcomes. Therefore, the architecture must move from probabilistic retrieval (finding "similar" text) to deterministic, schema-based retrieval wherever possible. The goal is to transform unstructured notes into structured databases.

3.1 Structure Extraction: FHIR as the Clinical AST

The "Abstract Syntax Tree" of healthcare is **FHIR (Fast Healthcare Interoperability Resources)**. Just as code has a strict schema, medical data should be coerced into FHIR resources (Patient, Observation, Condition, MedicationStatement) to ensure interoperability and type safety.[18]

3.1.1 The "Text-to-FHIR" Pipeline

Instead of chunking a clinical note by paragraphs, we treat the note as a compilation target.

- **Parsing Strategy:** Use an LLM-based agent (e.g., the **Infherno** framework or **FHIR-GPT** approach) to parse unstructured clinical notes into structured FHIR JSON bundles.[20]
- **Example:** A sentence "Patient is taking 500mg Metformin twice daily for Type 2 Diabetes" is compiled into:
    - MedicationStatement resource: { "medication": "Metformin", "dosage": "500mg", "frequency": "BID" }
    - Condition resource: { "code": "Type 2 Diabetes", "verificationStatus": "confirmed" }
- **Advantage:** This turns RAG into a database query. A question "What dosage of Metformin is the patient on?" can be answered by querying the FHIR object directly, eliminating retrieval ambiguity and the risk of retrieving a "Plan" section that discusses a *proposed* but not prescribed dosage.

3.1.2 SOAP Note Segmentation

For narrative text that cannot be fully structured (e.g., "Patient reports feeling anxious about

the surgery"), we utilize the **SOAP** structure (Subjective, Objective, Assessment, Plan).

- **Parser:** A domain-specific section classifier (using BioBERT or a regex grammar looking for headers like "HPI:", "Impression:") splits the document into these four semantic zones.[22]
- **Chunking:** Chunks are created per SOAP section and tagged with metadata section:Subjective or section:Plan. This ensures that a "Plan" chunk is never confused with a "Subjective" patient complaint, preserving the provenance of the information.

3.2 Embeddings: MedCPT and BioMistral

Standard embeddings fail on biomedical nomenclature. They may not recognize that "myocardial infarction" and "heart attack" are semantically identical vectors, or they might confuse "hypoglycemia" (low sugar) with "hyperglycemia" (high sugar) due to lexical similarity.

MedCPT (NCBI): The SOTA Standard

- **Architecture: MedCPT** (Contrastive Pre-trained Transformer) is the current state-of-the-art embedding model for biomedical information retrieval. Unlike generalist models, it was trained on **255 million user click logs from PubMed**.[23]
- **Asymmetric Retrieval:** It consists of two distinct encoders: a **Query Encoder** (for short questions) and an **Article Encoder** (for abstracts/documents). This asymmetric design effectively captures the relationship between a colloquial query and a technical document.[25]
- **Performance:** Benchmarks show MedCPT outperforming larger generic models (like GPT-3 sized embeddings) on biomedical retrieval tasks by a significant margin.[26]

BioMistral & ClinicalBERT: Local Alternatives

- **BioMistral:** A 7B parameter model fine-tuned on medical data. It is excellent for local-first encoding of clinical notes and can serve as a generator or a reranker.[27]
- **Recommendation:** Use **MedCPT** for retrieving research papers and external medical knowledge. Use **ClinicalBERT** for retrieving internal Electronic Health Record (EHR) data, which often has a different linguistic distribution than PubMed abstracts.

3.3 Domain Enrichment: Ontology Mapping and Redaction

Enrichment in medical RAG involves linking text to standard global ontologies.

- **Ontology Mapping:** An enrichment pipeline should scan chunks and append standardized codes from **ICD-10** (diagnoses), **SNOMED CT** (clinical terms), or **RxNorm** (medications).
  - *Mechanism:* Use tools like **MedCAT** or **MetaMap**.
  - *Result:* "Chest pain" -> append metadata code:R07.9. This resolves synonyms and enables precise retrieval across different terminology standards.
- **Privacy (PII/PHI Redaction):** Before any data enters the embedding model or vector

store, a "Scrubber" node must run to detect and redact names, dates, and MRNs. Microsoft's **Presidio** or specialized medical NER models are suitable here. This ensures HIPAA compliance and prevents the vector database from becoming a privacy leak.[28]

3.4 Graph Construction: The Patient-Disease Graph

The medical graph connects symptoms, diseases, and treatments, allowing the system to "reason" about medical facts that are not explicitly stated in the text.

- **Graph Schema:**
  - **Nodes:** Patient, Symptom, Disease, Drug, Interaction.
  - **Edges:** HAS_SYMPTOM, TREATED_WITH, CONTRAINDICATED_BY, CAUSES.
- **Inference Capability:** If a patient record mentions "High Blood Pressure" and the user query asks about "Decongestants," a standard RAG might miss the risk. However, GraphRAG can identify the CONTRAINDICATED_BY edge in the medical knowledge graph (Decongestants -> raise -> Blood Pressure) and warn the user, effectively acting as a clinical decision support system.[29]

---

4. The Technical & Enterprise Domain: Hierarchical Knowledge Bases

Technical documentation (manuals, API docs, SOPs) is the bridge between code and prose. It is often authored in structured formats like **DITA XML**, Markdown, or reStructuredText, making it highly amenable to LLMC's structure-aware approach. The goal here is to preserve the "Task" nature of the content.

4.1 Structure Extraction: DITA and XML Hierarchy

Technical writers often use **DITA (Darwin Information Typing Architecture)**, which is inherently modular. A "document" is a map of "topics," and a "topic" is a specific concept or task.

- **The "DITA AST":** Parsing DITA is analogous to parsing code. We use **lxml** (a powerful Python library) to traverse the XML tree.[31]
- **Parsing Strategy:**
  1. **Map Traversal:** Read the .ditamap file to understand the hierarchy (Book -> Chapter -> Section). This defines the "Context."
  2. **Topic Extraction:** Parse individual .dita or .xml files.
  3. **Hierarchy Preservation:** Instead of flattening the text, preserve the XML path. A chunk derived from a <warning> tag inside a <step> within a <task> carries the metadata context:Procedure > Step 3 > Warning. This is critical for safety—a warning loses its meaning if detached from the step it modifies.[34]
- **Tooling:** Custom Python scripts using lxml to convert DITA XML to JSON chunks,

preserving attributes like id and conref (content reference).[36]

4.2 Chunking Strategy: Task-Based Segmentation

In technical docs, the semantic unit is the **Task** or **Concept**, not the paragraph.

- **Task Chunks:** Numbered procedures (e.g., "How to Reset the Router") must be kept atomic. Splitting a 10-step procedure into two chunks (steps 1-5 and 6-10) is catastrophic for the user. The chunking logic must respect the <task> XML boundary and force the entire sequence into one retrieval unit, or use a "Parent Document" retrieval pattern.
- **Reference Chunks:** API parameter tables and command-line flags. These should be parsed as structured objects (Table extraction) rather than prose. Unstructured.io's table extraction capabilities are valuable here.[6]

4.3 Embeddings: BGE-M3 and CodeBERT

Technical documentation is often a hybrid of natural language and code snippets.

- **BGE-M3 (BAAI General Embedding):** A powerful open-source model that supports dense, sparse, and multi-vector retrieval. It is currently a standout performer for technical jargon and multilingual documentation.[15] Its hybrid nature allows it to match specific error codes (sparse keywords) while understanding the general problem description (dense vector).
- **CodeBERT:** For API documentation that heavily mixes prose and code signatures, **CodeBERT** variants are superior as they understand the programming context (e.g., differentiating between a variable name and a standard English word).[40]

4.4 Graph Construction: Component Dependency

Technical systems have components that depend on each other, forming a dependency graph.

- **Graph Nodes:** Component, Procedure, Error Code.
- **Edges:** REQUIRES, DEPENDS_ON, THROWS, RESOLVED_BY.
- **Extraction:** "Prerequisite" sections in DITA topics explicitly state dependencies (e.g., "Before installing X, you must configure Y"). We can extract these using regex or simple parsing of <prereq> tags.
- **Usage:** If a user asks "How do I install X?", the RAG system retrieves the "Install X" chunk. The Graph layer identifies the REQUIRES edge to "Configure Y" and injects that context: "Note: You must Configure Y first." This prevents users from attempting tasks out of order.[41]

---

5. Unified Architecture: The Extended LLMC

The research confirms that LLMC's architecture can be generalized by abstracting its core primitives. We propose a **"Content Router"** architecture that dynamically selects the parsing, embedding, and graph strategies based on the document domain.

5.1 The Universal Parser Interface

The system should expose a generic Parser interface with domain-specific implementations. This allows the core RAG logic (Retrieval, Context Window Management, Generation) to remain agnostic to the data source.

Python

```
class DocumentParser(Protocol):
    def parse(self, content: bytes) -> StructuralTree:...
    def extract_graph(self, tree: StructuralTree) -> NetworkXGraph:...
    def get_chunks(self, tree: StructuralTree) -> List[Chunk]:...
```

- **Code Implementation:** Uses TreeSitter (Existing).
- **Legal Implementation:** Uses **Regex Grammar Engine** (for DOCX) / **Unstructured VLM** (for PDF).
- **Medical Implementation:** Uses **FHIR Extraction Agent** / **SOAP Segmenter**.
- **Technical Implementation:** Uses **lxml DITA Parser** / **Markdown Splitter**.

5.2 Generalized Embedding Router

Instead of a single embedding model, LLMC should support a "Model Registry" routed by content type. This ensures that the vector space is optimized for the specific semantic distribution of the domain.

Domain	Recommended Model (Local)	Recommended Model (Cloud)	Rationale
Legal	legal-bert-base-uncased	voyage-law-2	Nuanced understanding of legal terminology and long contexts.
Medical	MedCPT-Query-Encoder	MedCPT (API)	Tuning on PubMed logs; zero-shot capabilities;

			asymmetric search.
Tech	bge-m3	cohere-embed-v3	Handling of mixed code/prose, multilingual support, and sparse retrieval.
General	nomic-embed-text	text-embedding-3-small	Balanced performance/cost for general policies/SOPs.

5.3 Generalized GraphRAG

The "Import Graph" module becomes a **"Semantic Relation Graph"**. This graph is stored in a graph database (e.g., Neo4j for production, NetworkX for local) and queried during retrieval.

- **Code Edges:** Imports, Calls, Inherits.
- **Legal Edges:** Cites, Amends, Defines.
- **Medical Edges:** Has_Symptom, Treated_With, Contraindicated_By.
- **Technical Edges:** Requires, Throws, See_Also.

At retrieval time, the system performs a 1-hop or 2-hop traversal from the retrieved chunks to fetch connected context, enriching the prompt with "knowledge" that is not present in the vector search results.[16]

---

6. Evaluation Methodologies

Evaluating Domain RAG is distinct from general RAG. "Fluency" is less important than "Accuracy" and "Safety." We must employ rigorous benchmarks.

6.1 Domain-Specific Benchmarks

- **Legal:**
  - **LegalBench:** A collaborative benchmark for legal reasoning (issue spotting, rule application). It contains 162 tasks and can be run locally.[1]
  - **LegalBench-RAG:** A specific subset designed for retrieval. It tests the system's ability to find specific clauses in a large corpus of NDAs and contracts.[1]
- **Medical:**
  - **MedQA:** USMLE-style questions. Useful for testing the *generation* component's

medical knowledge.
    - **PubMedQA:** Question-answering on research papers. Useful for validating the **MedCPT** retrieval pipeline.
    - **R2MED:** A new benchmark specifically for reasoning-driven medical retrieval, focusing on the "Why" behind a diagnosis.[45]
- **Technical:** No single standard exists, but **TechQA** (on IBM technical support logs) is a reference. For enterprise, custom "Correctness" metrics on internal manuals (e.g., "Did it retrieve the correct API parameter?") are essential.

6.2 Key Metrics

- **Retrieval Metrics:**
    - **Recall@K:** In Legal/Medical, Recall is paramount. Missing a relevant precedent or contraindication is a failure, whereas low precision (retrieving extra documents) is tolerable.
    - **NDCG@10 (Normalized Discounted Cumulative Gain):** Measures the quality of the ranking. Important for ensuring the "best" clause appears at the top.[46]
- **Generation Metrics:**
    - **RAGAS (Retrieval Augmented Generation Assessment):** A framework that uses an LLM to grade the output. Key metrics include "Answer Faithfulness" (is the answer derived from the context?) and "Context Relevance" (was the retrieved context actually useful?).[47]
    - **Hallucination Rate:** For medical, we must implement a specific metric checking generated claims against the retrieved context using a secondary **Verifier LLM**.

---

7. Implementation Considerations

7.1 File Handling and OCR

- **The PDF Bottleneck:** Standard libraries (PyPDF) are insufficient for multi-column layouts found in legal briefs and medical journals.
- **Solution:** Integrate **Unstructured.io** or **Surya** (for local OCR).[8] These tools convert visual layouts (tables, columns) into linear text that respects reading order. This is a critical pre-processing step before the "Universal Parser" can operate.

7.2 Privacy (PII/PHI)

- **Local-First Advantage:** LLMC's local nature is a massive advantage for Legal/Medical. Data never leaves the Virtual Private Cloud (VPC), satisfying data sovereignty requirements.
- **Redaction Pipeline:** Integrate a **Presidio** step in the ingestion pipeline. This tool detects patterns (SSN, Phone, Email, MRN) and masks them *before* embedding. This prevents the

vector database from becoming a permanent privacy leak.[28]

7.3 Scalability and Updates

- **Graph Pruning:** Citation graphs can grow exponentially. Pruning strategies (e.g., keeping only "authoritative" citations with high degree centrality) are necessary for scale.
- **Update Propagation:** In law, if a statute changes, all dependent clauses must be flagged. The GraphRAG architecture supports this "impact analysis" natively by traversing reverse-dependency edges.

---

8. Conclusion

The transition from Code RAG to Domain RAG is an exercise in **structural analogy**. The rigorous parsing logic that makes LLMC successful for code—identifying boundaries, resolving dependencies, and respecting syntax—is exactly what is missing from standard "unstructured" RAG. By treating Legal, Medical, and Technical documents as "code" with their own grammars and compilers (Regex, FHIR, DITA), we can build a system that offers the same precision and reliability.

The integration of domain-specific embeddings (MedCPT, Legal-BERT) ensures the system understands the "language" of the domain, while structural knowledge graphs ensure it understands the "logic." This architecture represents a robust, scalable path forward for high-stakes enterprise RAG, moving beyond the "stochastic parrot" model to a verifiable, structure-aware knowledge engine.

---

9. Comprehensive Literature Review & Analysis

9.1 The Evolution of Document Parsing: From Text to Structure

The foundational challenge in extending a code-focused RAG system to prose is the loss of explicit structure. In software, a compiler guarantees the integrity of an Abstract Syntax Tree (AST). In prose, structure is implicit—defined by layout, font size, and numbering conventions.

Vision-Language Models (VLMs) as Universal Parsers:
Traditional OCR (Tesseract) destroys layout, often merging text across columns or garbling tables. This is fatal for legal and medical documents where the juxtaposition of data (e.g., a value in a specific table cell) carries the semantic weight. Research highlights a shift toward Layout-Aware models. Surya and Nougat have emerged as powerful open-source alternatives to proprietary tools like Amazon Textract. Nougat (Neural Optical Understanding for Academic Documents) specializes in converting scientific PDFs directly into Markdown, preserving mathematical formulas and tables, which is critical for the Technical and Medical domains.48

Surya, offering high-performance line-level text detection and recognition, benchmarks competitively against cloud providers like Google Vision, making it a viable component for a local-first architecture.8

For legal contracts, where "visual cues" (bold headers, indentation) define the hierarchy (Article -> Section -> Clause), **Unstructured.io's** partitioning strategies offer a hybrid approach. Their hi_res strategy utilizes object detection models (like YOLOX) to classify layout elements, while fallback strategies (fast) use rule-based NLP. This stratification allows LLMC to route documents based on complexity—simple text contracts use regex, while complex scanned deeds use VLMs.[6]

9.2 Domain-Specific Embedding Landscapes

The assumption that a single "generalist" embedding model (like OpenAI's text-embedding-3) suffices for all domains is increasingly challenged by benchmarks showing the efficacy of domain-specialized models.

- **Medical Embeddings:** The **MedCPT** model (Contrastive Pre-trained Transformer) represents a paradigm shift. Unlike models trained on generic web text, MedCPT was trained on 255 million user click logs from PubMed.[23] This "query-article" contrastive training allows it to align a colloquial query ("heart attack") with clinical literature ("myocardial infarction") far better than lexical matches. Benchmarks show MedCPT outperforming larger generic models (like GPT-3 sized embeddings) on biomedical retrieval tasks.[26] For local inference, **BioMistral** and **ClinicalBERT** provide robust alternatives, with BioMistral 7B showing capabilities that rival proprietary models in chat-doctor scenarios.[27]
- **Legal Embeddings:** In the legal domain, precision is binary—a case is either relevant or it is not. **Legal-BERT** remains a staple for local deployments, fine-tuned on legislation and caselaw to capture the distinct semantic space of "legalese".[12] However, **Voyage AI's** specialized voyage-law-2 model has demonstrated superior performance in long-context retrieval, a common requirement for multi-page briefs.[13] This suggests a hybrid architecture for LLMC: default to local Legal-BERT for standard retrieval, but allow routing to Voyage API for complex, high-stakes litigation analysis.
- **Technical Embeddings:** Technical documentation is a hybrid of prose and code. **BGE-M3** (BAAI General Embedding) is currently a standout open-source performer. Its ability to handle multilingual content and its hybrid nature (dense + sparse retrieval) makes it uniquely suited for technical manuals that may contain mixed-language instructions or specific keyword-heavy error codes.[15]

9.3 Knowledge Graphs: The "Import Graph" of Prose

A critical insight from LLMC's code RAG is the value of the "Import Graph" to resolve dependencies. In non-code domains, this dependency graph exists but is implicit.

- **Legal Citation Networks:** Legal documents are inherently graph-structured. A contract

references other contracts; a court opinion cites precedents. **GraphRAG** techniques allow us to materialize these links. By extracting citations (NER) and modeling them as edges in a graph database (e.g., Neo4j), we enable "recursive retrieval"—fetching not just the immediate document, but the referenced "dependencies" required to interpret it.[16]

- **Medical Knowledge Graphs:** The relationship between symptoms, diagnoses, and treatments forms a natural graph. Research into **GraphRAG** in medicine demonstrates that combining vector search (for semantic similarity) with graph traversal (for factual relationships) significantly reduces hallucinations. For instance, linking a "Patient" node to a "Disease" node via a HAS_DIAGNOSIS edge allows an LLM to "reason" across the patient's history rather than guessing based on isolated text chunks.[29]

---

10. Detailed Architecture Recommendations

The following architecture proposes extending LLMC from a Code-RAG system to a **Multi-Domain Semantic Platform**.

10.1 The "Universal Parser" Routing Layer

The ingestion pipeline must evolve from a simple file-extension check to a content-aware router.

- **Input:** Document (PDF, DOCX, XML, MD).
- **Classifier:** A lightweight model (e.g., DistilBERT or simple heuristic) identifies the domain: LEGAL, MEDICAL, TECHNICAL, or GENERAL.
- **Routing Logic:**
  - **IF Domain == LEGAL:**
    - **Parser: Regex Grammar Engine**. Utilizing Python's re module with pre-compiled patterns for "Articles", "Sections", and "Recitals".[4]
    - **Fallback:** If regex fails (low structure confidence), route to **Unstructured.io VLM** to infer layout.[6]
  - **IF Domain == MEDICAL:**
    - **Parser: FHIR Extraction Agent**. An LLM-driven parser (Infherno style) that attempts to coerce text into FHIR JSON resources.[19]
    - **Fallback: SOAP Segmenter**. Splits text into Subjective/Objective/Assessment/Plan blocks using NLP classification.
  - **IF Domain == TECHNICAL:**
    - **Parser: DITA/XML Parser**. Uses lxml to traverse the DOM tree, preserving hierarchy (BookMap -> Topic -> Section).[31]
    - **Fallback: Markdown/Header Splitter**. For non-XML docs, use standard header-based splitting.

10.2 Domain-Specific Chunking Strategies

Domain	Strategy	Implementation Detail	Rationale
Legal	**Hierarchical Clause Chunking**	Keep Parent + Children together. If Section 2.1 is > chunk_size, split children but *prepend* parent header to every child chunk.	Legal provisions are context-dependent. A subsection "(a)" is meaningless without the parent clause defining the obligation.
Medical	**Encounter-Based Windowing**	Group by Date + EncounterID. Use "Sliding Window" within an encounter, never crossing encounter boundaries.	Medical events are temporal. Mixing data from a 2010 visit with a 2024 visit leads to dangerous clinical hallucinations.
Technical	**Structural XML/DITA Splitting**	Chunk by <task> or <concept> tags. Extract <prereq> tags as separate linked chunks.	Procedures must be atomic. You cannot retrieve "Step 5" without Steps 1-4 and the Prerequisites.

10.3 Enrichment and Metadata Injection

To achieve "Code-Level" precision, we must inject metadata that acts as "Type Hints" for the LLM.

- **Legal Enrichment (SALI Tags):**
  - *Action:* Run a lightweight extractor prompt over every legal chunk.
  - *Prompt:* "Extract SALI LMSS tags for: Area of Law, Jurisdiction, and Clause Type.".[11]
  - *Storage:* Store tags in the Vector DB metadata. {"tag": "Indemnification", "jurisdiction": "NY"}.
- **Medical Enrichment (Ontology Codes):**
  - *Action:* Map text entities to **ICD-10** or **SNOMED CT** codes.
  - *Tool:* MedCAT or similar biomedical NER tools.
  - *Benefit:* Resolves synonyms. "Heart attack", "MI", and "Coronary thrombosis" all map

to the same code, enabling accurate retrieval regardless of phrasing.

- **Technical Enrichment (Dependency Mapping):**
  - *Action:* Extract "See Also" links and "Prerequisites."
  - *Storage:* Store as graph edges (Chunk A) ----> (Chunk B).

---

11. Model Recommendations

This section provides specific, actionable model choices for a local-first deployment (LLMC), balancing performance with resource constraints.

11.1 Embedding Models (The "Retriever")

1. **Legal Domain:**
   - **Primary (Local):** law-bert-base-uncased (or similar **Legal-BERT** variants). Trained on massive legal corpora, it understands the nuance of legal language better than generic BERT.[12]
   - **High-Performance (API): Voyage Law**. Use this if data privacy allows cloud processing and maximum accuracy is required.[13]
2. **Medical Domain:**
   - **Primary (Local): MedCPT-Query-Encoder** / **MedCPT-Article-Encoder**. The split architecture is ideal for RAG (asymmetric search). It is the current academic SOTA.[23]
   - **Alternative: BioMistral-7B** (quantized) can be used for generating synthetic embeddings or re-ranking.[27]
3. **Technical Domain:**
   - **Primary: BGE-M3**. Its multi-granularity (dense + sparse) approach is perfect for technical docs that contain both high-level concepts and specific error codes (sparse keywords).[15]
   - **Code-Heavy Docs: CodeBERT**. Use this for API references where function signatures are present.

11.2 Reranking Models (The "Refiner")

After the initial vector search, a Re-ranker is essential to filter noise.

- **General/Technical:** bge-reranker-v2-m3.
- **Medical: MedCPT-Cross-Encoder**. Specifically trained to rank biomedical documents against queries.[24]
- **Legal:** legal-bert can be used in a cross-encoder configuration, or a general strong reranker like Cohere Rerank (API).

---

12. Chunking Strategy Recommendations

12.1 Legal: The Recursive Clause Splitter

We recommend adapting LangChain's RecursiveCharacterTextSplitter with custom separators tailored for legal texts.

- **Separators:** ``.[3]
- **Logic:** This forces splits to occur at logical legal boundaries (Article > Section > Numbered List) rather than arbitrary character counts.
- **Overlap:** Minimal overlap is needed if structure is respected, but a 10-20% overlap protects against edge cases where a sentence spans a split.

12.2 Medical: The Context-Aware Window

- **Context Prepending:** Medical chunks must be context-aware. A chunk "Patient denies pain" is useless without the context "Visit Date: 2023-10-12, Chief Complaint: Post-Op".
- **Strategy:** Implement a **Parent-Document Retrieval** strategy. Index small chunks (sentences/findings), but retrieve the full "Parent Encounter" window when a match is found.[49]

12.3 Technical: DITA-Aware Splitting

- **XML Parsing:** Use lxml to parse DITA files.
- **Logic:**
  - Identify <topic> nodes.
  - Flatten the XML structure into text but *retain* the hierarchy as a breadcrumb string: Manual > Setup > Configuration > Network.
  - Prepend this breadcrumb to the chunk text before embedding. This ensures the model knows *where* in the manual this text belongs.

---

13. Evaluation Frameworks

To ensure the "Domain RAG" is performing correctly, we must move beyond vibes-based evaluation to rigorous benchmarking.

13.1 LegalBench & LegalBench-RAG

- **LegalBench:** A collaborative benchmark for legal reasoning (issue spotting, rule application). It can be run locally to test the LLM's legal reasoning capabilities.[1]
- **LegalBench-RAG:** A specific subset designed for retrieval. It tests the system's ability to find specific clauses in a large corpus.
- **Action:** Create a regression test suite using LegalBench-RAG samples. Measure Recall@5 (Did we find the right clause?) and Precision (Did we retrieve only relevant clauses?).

13.2 MedQA & PubMedQA

- **MedQA:** USMLE-style questions. Useful for testing the *generation* component's medical knowledge.
- **PubMedQA:** Question-answering on research papers. Useful for testing the **MedCPT** retrieval pipeline.
- **Metric:** Focus on **Exact Match** for entity retrieval (e.g., retrieving the exact drug interaction listed in the query).

13.3 Freshness & Hallucination Detection

- **Hallucination:** In domain RAG, hallucination is often a failure of retrieval (missing the "NOT" in a medical note).
- **Detection:** Use **RAGAS** (Retrieval Augmented Generation Assessment) to compute a "Faithfulness Score." Additionally, implement a **Citation Verifier**: ensure every claim in the generated answer cites a specific chunk ID from the retrieved context.

---

14. Implementation & Prototype Scope

14.1 Prototype Phase 1: The "Universal Parser"

- **Goal:** Build the content router.
- **Tech Stack:** Python, unstructured (for PDFs), lxml (for XML), re (for Regex).
- **Deliverable:** A script that takes a folder of mixed docs (Contracts, DITA, Clinical Notes) and outputs standardized JSON chunks with domain-specific metadata.

14.2 Prototype Phase 2: Domain Embeddings

- **Goal:** Integrate specialized encoders.
- **Tech Stack:** sentence-transformers, chromadb (or pgvector).
- **Deliverable:** A vector store where legal docs are encoded with Legal-BERT and medical docs with MedCPT. Demonstrate that a query "heart attack" retrieves "myocardial infarction" docs (Medical) and a query "indemnity" retrieves "hold harmless" clauses (Legal).

14.3 Prototype Phase 3: Graph Overlay

- **Goal:** "Import Graph" for Prose.
- **Tech Stack:** NetworkX (for local prototype) or Neo4j (production).
- **Deliverable:** A graph visualization showing Contract A -> cites -> Statute B; Patient -> has -> Diagnosis.

14.4 Implementation Considerations

- **Scale:** For "Enterprise" scope (millions of docs), local embeddings (BioMistral/Legal-BERT) are faster and cheaper than APIs.
- **Privacy:** All processing (OCR, Parsing, Embedding) happens locally. No data is sent to OpenAI/Anthropic. PII Redaction is the critical "Gatekeeper" step before indexing.

---

15. Conclusion

The transition from Code RAG to Domain RAG is an exercise in **structural analogy**. The rigorous parsing logic that makes LLMC successful for code—identifying boundaries, resolving dependencies, and respecting syntax—is exactly what is missing from standard "unstructured" RAG. By treating Legal, Medical, and Technical documents as "code" with their own grammars and compilers (Regex, FHIR, DITA), we can build a system that offers the same precision and reliability. The integration of domain-specific embeddings (MedCPT, Legal-BERT) and structural knowledge graphs ensures that the system doesn't just match keywords, but understands the specialized semantics of the domain. This architecture represents a robust, scalable path forward for high-stakes enterprise RAG.

16. Chunking Strategy Recommendations

16.1 Legal: Hierarchical Clause Preservation

Legal documents rely on a strict hierarchy where meaning is inherited. A subsection (ii) often cannot be understood without its parent Section 4.1.

- **Recommended Strategy: Recursive Structure-Aware Chunking**.
- **Mechanism:**
  1. Parse the document tree using Regex Grammars (Article > Section > Paragraph).
  2. If a child node (e.g., a list item) is small, **merge** it with its parent.
  3. If a parent node is too large, split the children but **prepend the parent's header text** to every child chunk.
     - *Example:* Chunk text = "Article 4: Termination.... (ii) In the event of insolvency..."
- **Tools:** Custom Python logic wrapping LangChain's RecursiveCharacterTextSplitter with regex separators ``.[3]

16.2 Medical: Encounter-Based Windowing

Medical records are temporal. The "state" of a patient changes over time.

- **Recommended Strategy: Temporal Encounter Grouping**.
- **Mechanism:**
  1. Sort all records by Date.
  2. Group text by EncounterID (a single visit).
  3. Chunk within an Encounter. **Never** allow a chunk to span across two different encounters (e.g., merging the end of a 2015 visit with the start of a 2020 visit).

4. **Header Injection:** Every chunk must start with ``.
- **Rationale:** Prevents "Time-Travel Hallucinations" where the LLM conflates past history with current conditions.

16.3 Technical: DITA/XML Structural Splitting

Technical docs are modular by design (DITA topics).

- **Recommended Strategy: DOM-Based Chunking**.
- **Mechanism:**
  1. Parse XML/HTML using lxml or BeautifulSoup.
  2. Treat specific tags (<task>, <concept>, <reference>) as hard boundaries.
  3. Extract <related-links> or <prereq> tags separately to build the **Graph Layer** (see Section 4.4), rather than burying them in the text.
- **Tools:** lxml for parsing DITA maps and topics to preserve the "Book > Chapter > Topic" hierarchy.[31]

---

17. Architecture Recommendations

17.1 The "LLMC-Prose" Router

The existing LLMC content-type router must be upgraded to a **Semantic Router**.

Route	Trigger	Parser	Embedder	Graph Strategy
Legal	.docx, .pdf (Contract classifier)	Regex / LayoutLM	Legal-BERT / Voyage	Citation Network
Medical	.fhir, .hl7, Clinical Notes	FHIR Parser / SOAP	MedCPT / ClinicalBERT	Patient Ontology
Tech	.xml, .dita, .md	lxml / TreeSitter	BGE-M3 / CodeBERT	Dependency Graph
Code	.py, .js, etc.	TreeSitter	CodeBERT	Import Graph

17.2 The "Schema-First" Retrieval Engine

For the Medical domain, we recommend a **Schema-First** approach.

- **Concept:** Instead of embedding everything, parse "Vitals," "Labs," and "Meds" into a

structured SQL/Vector hybrid store.
- **Query:** "What was the last glucose level?"
- **Execution:**
  1. LLM converts query to Structured Query (SQL/FHIR Path).
  2. Execute against Structured Store.
  3. *Fallback:* If query is narrative ("How does the patient feel?"), route to Vector Store (MedCPT).
- **Benefit:** Zero hallucination for numerical/categorical data.

---

18. Model Recommendations

18.1 Embedding Models

- **Legal: Legal-BERT** (Local) [14] or **Voyage Law** (Cloud).[12] *Why:* Trained on case law/contracts; understands "consideration" means money, not thought.
- **Medical: MedCPT** (Local/Cloud).[23] *Why:* Trained on PubMed logs; links "MI" to "Heart Attack" via user search behavior.
- **Technical: BGE-M3** (Local).[15] *Why:* Handles multilingual, code-mixed, and dense retrieval exceptionally well.

18.2 Generative Models (LLMs)

- **Medical: Med-PaLM 2** (Cloud) or **BioMistral 7B** (Local).[27] *Why:* Fine-tuned for clinical reasoning and safety.
- **Legal: GPT-4o** (Cloud) or **SaulLM-7B** (Local). *Why:* High logic/reasoning capabilities required for applying law to fact.
- **General: Llama 3** (Local). *Why:* Strong generalist performance for routing and summarization.

---

19. Prototype Scope

To validate this architecture, a prototype should focus on **one** non-code domain to prove the "Generalization Primitives." We recommend the **Legal Domain** as the first target because:

1. **Isomorphism:** Contracts are "code for people." The structure (Article/Section) maps cleanly to Code (Class/Function).
2. **Data:** High-quality open datasets (**LegalBench**) are available for evaluation.
3. **Risk:** Lower immediate safety risk than Medical.

Prototype Plan:

1. **Ingest:** 100 Contracts (PDF/DOCX) from the CUAD (Contract Understanding Atticus Dataset) or LegalBench corpus.
2. **Parse:** Implement the **Regex Grammar Parser** to split by "Article/Section."

3. **Embed:** Index chunks using **Legal-BERT**.
4. **Graph:** Build a simple "Definition Graph" (Node = Defined Term, Edge = Used In Clause).
5. **Eval:** Run 50 queries from LegalBench-RAG and compare Recall@5 against standard chunking + OpenAI embeddings.

This prototype will demonstrate that **structure-aware parsing** and **domain-specific embeddings** significantly outperform generic RAG, justifying the architectural expansion of LLMC.

Works cited

1. LegalBench-RAG: A Benchmark for Retrieval-Augmented Generation in the Legal Domain, accessed December 12, 2025, https://arxiv.org/html/2408.10343v1
2. Graph RAG for Legal Norms: A Hierarchical and Temporal Approach - arXiv, accessed December 12, 2025, https://arxiv.org/html/2505.00039v1
3. Text Splitter in LangChain - GeeksforGeeks, accessed December 12, 2025, https://www.geeksforgeeks.org/artificial-intelligence/text-splitter-in-langchain/
4. Using Regex for Text Segmentation - Data Science for Lawyers, accessed December 12, 2025, https://www.datascienceforlawyers.org/learning-resources/lesson-3/r-script/using-regex-to-segment-texts/
5. Using Regex to Split Subsections With Unique Titles - Stack Overflow, accessed December 12, 2025, https://stackoverflow.com/questions/78938440/using-regex-to-split-subsections-with-unique-titles
6. Partitioning strategies - Unstructured document, accessed December 12, 2025, https://docs.unstructured.io/open-source/concepts/partitioning-strategies
7. Integrating with Unstructured.io - Qdrant, accessed December 12, 2025, https://qdrant.tech/course/essentials/day-7/unstructured/
8. OCR Dataset and Model Fine-Tuning for the Swedish Language - kth .diva, accessed December 12, 2025, https://kth.diva-portal.org/smash/get/diva2:2013863/FULLTEXT02.pdf
9. What would you say is currently the most accurate OCR solution if you're not con... | Hacker News, accessed December 12, 2025, https://news.ycombinator.com/item?id=43047121
10. Standardizing Legal Data to Extract Insights | 3 Geeks and a Law Blog, accessed December 12, 2025, https://www.geeklawblog.com/2022/05/standardizing-legal-data-to-extract-insights.html
11. Practical Applications and Patterns – GenAI for Legal Practice, accessed December 12, 2025, https://oercollective.caul.edu.au/gen-ai-legal-practice/chapter/practical-applications/
12. How to choose the right embedding model for your RAG application? - AI Simplified, accessed December 12, 2025,

https://vivedhaelango.substack.com/p/how-to-choose-the-right-embedding
13. Voyage 3 Large vs OpenAI text-embedding-3-small - Agentset, accessed December 12, 2025, https://agentset.ai/embeddings/compare/voyage-3-large-vs-openai-text-embedding-3-small
14. Automated Contract Clause Understanding and Risk Assessment Chatbot with fine-tuned Legal-BERT and GPT-4o | by Ratna Prakarsha Kandukuri | Medium, accessed December 12, 2025, https://medium.com/@prakarsha/automated-contract-clause-understanding-and-risk-assessment-with-fine-tuned-legal-bert-and-gpt-4o-3a6f0423ace3
15. Benchmark of 11 Best Open Source Embedding Models for RAG - Research AIMultiple, accessed December 12, 2025, https://research.aimultiple.com/open-source-embedding-models/
16. RAG Tutorial: How to Build a RAG System on a Knowledge Graph - Neo4j, accessed December 12, 2025, https://neo4j.com/blog/developer/rag-tutorial/
17. From Legal Documents to Knowledge Graphs - Graph Database & Analytics - Neo4j, accessed December 12, 2025, https://neo4j.com/blog/developer/from-legal-documents-to-knowledge-graphs/
18. LLM FHIR Eval | Open-Source Benchmarking for LLMs on FHIR - Flexpa, accessed December 12, 2025, https://www.flexpa.com/eval
19. FHIR-GPT Enhances Health Interoperability with Large Language Models - medRxiv, accessed December 12, 2025, https://www.medrxiv.org/content/10.1101/2023.10.17.23297028v2.full.pdf
20. Infherno: End-to-end Agent-based FHIR Resource Synthesis from Free-form Clinical Notes, accessed December 12, 2025, https://arxiv.org/html/2507.12261v1
21. FHIR-GPT Enhances Health Interoperability with Large Language Models - PMC, accessed December 12, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC12312630/
22. Using Vision Models for PDF Parsing in RAG Systems - Chitika, accessed December 12, 2025, https://www.chitika.com/vision-models-pdf-parsing-rag/
23. A Survey of Large Language Models in Medicine: Progress, Application, and Challenge, accessed December 12, 2025, https://arxiv.org/html/2311.05112v5
24. MedCPT: Zero-shot Biomedical IR Model - GitHub, accessed December 12, 2025, https://github.com/ncbi/MedCPT
25. MedCPT: Contrastive Pre-trained Transformers with large-scale PubMed search logs for zero-shot biomedical information retrieval | Bioinformatics | Oxford Academic, accessed December 12, 2025, https://academic.oup.com/bioinformatics/article/39/11/btad651/7335842
26. MedCPT: Contrastive Pre-trained Transformers with large-scale PubMed search logs for zero-shot biomedical information retrieval, accessed December 12, 2025, https://pubmed.ncbi.nlm.nih.gov/37930897/
27. MedBot vs RealDoc: efficacy of large language modeling in physician-patient communication for rare diseases - PubMed Central, accessed December 12, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC12012358/
28. Retrieval augmented generation for large language models in healthcare: A

systematic review - PMC - NIH, accessed December 12, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC12157099/
29. Medical Graph RAG: Towards Safe Medical Large Language Model via Graph Retrieval-Augmented Generation - arXiv, accessed December 12, 2025, https://arxiv.org/html/2408.04187v1
30. Case Study: Turning Doctor Transcripts into Temporal Medical Record Knowledge Graphs | by Chia Jeng Yang - Medium, accessed December 12, 2025, https://medium.com/enterprise-rag/case-study-turning-doctor-transcripts-into-temporal-medical-record-knowledge-graphs-cf624d4927eb
31. Parsing XML and HTML with lxml, accessed December 12, 2025, https://lxml.de/parsing.html
32. Python: Parse dita map file and contents and output all href values - Stack Overflow, accessed December 12, 2025, https://stackoverflow.com/questions/75380890/python-parse-dita-map-file-and-contents-and-output-all-href-values
33. Intro to Parsing HTML and XML with Python and lxml - Scrapfly, accessed December 12, 2025, https://scrapfly.io/blog/posts/intro-to-parsing-html-xml-python-lxml
34. Structured Content Is the Only Sustainable Path to AI-Ready Documentation - Paligo, accessed December 12, 2025, https://paligo.net/blog/information-architecture/structured-content-for-ai-ready-documentation/
35. Graph-RAG for DITA Documentation Guide | by Tommi P | Medium, accessed December 12, 2025, https://medium.com/@hastur/graph-rag-for-dita-documentation-guide-f33803994eef
36. Configuring Sample JSON Content Processor, accessed December 12, 2025, https://support.ingeniux.com/knowledge-base/documentation/cms-10/dita-processing-pipeline/publishing-dita-in-ingeniux-cms/publishing-dita-content/configuring-sample-json-content-processor
37. Conversion of custom XML - Docling, accessed December 12, 2025, https://docling-project.github.io/docling/examples/backend_xml_rag/
38. BAAI/bge-m3 - Hugging Face, accessed December 12, 2025, https://huggingface.co/BAAI/bge-m3
39. BGE M3 Model vs OpenAI Embeddings | by Naman Tripathi - Medium, accessed December 12, 2025, https://naman1011.medium.com/bge-m3-model-vs-openai-embeddings-e6d6cda27d0c
40. Benchmarking Retrieval-Augmented Generation for Medicine - ACL Anthology, accessed December 12, 2025, https://aclanthology.org/2024.findings-acl.372.pdf
41. DepsRAG: Towards Managing Software Dependencies using Large Language Models, accessed December 12, 2025, https://arxiv.org/html/2405.20455v3
42. Graph RAG: Navigating graphs for Retrieval-Augmented Generation using Elasticsearch, accessed December 12, 2025, https://www.elastic.co/search-labs/blog/rag-graph-traversal

43. Retrieving Semantically Similar Decisions under Noisy Institutional Labels: Robust Comparison of Embedding Methods - arXiv, accessed December 12, 2025, https://arxiv.org/pdf/2512.05681
44. LegalBench-RAG, the First Open-Source Retrieval Benchmark for the Legal Domain | by ZeroEntropy | Medium, accessed December 12, 2025, https://medium.com/@zeroentropy/legalbench-rag-the-first-open-source-retrieval-benchmark-for-the-legal-domain-a3c5d8894724
45. [Literature Review] R2MED: A Benchmark for Reasoning-Driven Medical Retrieval, accessed December 12, 2025, https://www.themoonlight.io/en/review/r2med-a-benchmark-for-reasoning-driven-medical-retrieval
46. Ultimate Guide to Benchmarking RAG Systems - Artech Digital, accessed December 12, 2025, https://www.artech-digital.com/blog/ultimate-guide-to-benchmarking-rag-systems-mfn0f
47. RAG Evaluation Metrics: Best Practices for Evaluating RAG Systems - Patronus AI, accessed December 12, 2025, https://www.patronus.ai/llm-testing/rag-evaluation-metrics
48. Nougat: Neural Optical Understanding for Academic Documents - arXiv, accessed December 12, 2025, https://arxiv.org/pdf/2308.13418
49. Best Chunking Strategy for the Medical RAG System (Guidelines Docs) in PDFs - Reddit, accessed December 12, 2025, https://www.reddit.com/r/Rag/comments/1ljhksy/best_chunking_strategy_for_the_medical_rag_system/