Core Architecture and Pipeline Design

The system implements a sophisticated three-stage pipeline for processing and verifying medical claims against both textual and visual evidence:

1. **Document Processing Stage** The first stage handles the extraction and processing of content from medical PDFs. The implementation in `process_all_pdfs.py` shows a comprehensive approach that:

- Uses multiple extraction methods to ensure robust content capture:

```
# Extract text using primary method with fallback
text = extract_text_from_pdf(str(pdf_file),
        save_markdown=False)

# Chunk text with context preservation
paragraphs = chunk_text_by_paragraphs(text, max_length=max_length,
                                       min_length=min_length,
        overlap=overlap)
```

- Implements sophisticated figure extraction:

```
def extract_images_from_pdf(pdf_path: str, output_dir: str =
        None) -> List[Dict]:
    # Method 1: Direct image extraction
    image_list_per_page = page.get_images(full=True)

    # Method 2: Full page capture for annotations
    pix = page.get_pixmap(matrix=fitz.Matrix(2, 2))

    # Method 3: Smart figure detection
    figures = detect_and_extract_figures(page, pdf_name,
        page_num, output_dir)
```

1. **Multimodal Embedding Generation** The system generates hybrid embeddings for both text and images:

```
def get_multimodal_embeddings(image_data: bytes, description: str
        = None) -> Dict[str, List[float]]:
    # Generate embeddings for image content
    result = genai.embed_content(
        model="models/embedding-001",
        content=description,
        task_type="retrieval_document",
        title="Research paper image"
    )
    return {
        "dense": result["embedding"]
    }
```

For images, the system: - Extracts figures and diagrams using segmentation - Generates detailed descriptions using Gemini Vision - Creates embeddings that capture both visual and textual aspects - Stores metadata including image type, captions, and locations

1. **Hybrid Search and Evidence Assessment** The verification process combines dense and sparse embeddings:

```
def search_evidence(claim: str, top_k: int = 5,
        max_duplicates_per_source: int = 2) -> List[Dict[str,
        Any]]:
    embeddings = get_embedding(claim)
    query_params = {
        "vector": embeddings["dense"],
        "top_k": top_k * 3,
        "include_metadata": True
    }

    if embeddings["sparse"] is not None:
        query_params["sparse_vector"] = embeddings["sparse"]
```

Multimodal Processing Deep Dive

The multimodal aspect is particularly sophisticated, handling various types of visual content:

1. **Image Processing Pipeline**

```
def classify_research_image(image_data: bytes, caption: str =
        None) -> str:
    # Create classification prompt
    prompt = """
    Classify this research paper image into EXACTLY ONE of these
        categories:
    - Table: structured arrangement of data
    - Chart/Graph: visual representation of data
    - Diagram: illustration explaining a concept
    - Flowchart: diagram representing a process
    - Microscopy/Medical Image: biological/medical specimen
        images
    - Chemical Structure: molecular visualization
    - Equation/Formula: mathematical expressions
    """
```

1. **Visual Content Analysis** The system generates comprehensive descriptions of visual content:

```
def get_image_description(image_data: bytes, caption: str = None)
        -> str:
    prompt = """
    Please analyze this image thoroughly and provide a detailed
        description.
    Focus on:
    1. The key information being conveyed
    2. ALL text content visible in the image
    3. Any numeric data or measurements
    4. Any legends, labels, or annotations
    5. The relationships between elements
    """
```

1. **Evidence Integration** The system seamlessly integrates visual and textual evidence:

```
def generate_explanation(claim: str, evidence_list: List[Dict[str,
        Any]]) -> Dict[str, Any]:
    # Handle different content types
    if content_type == "image" or content_type ==
        "image_description":
        text = match.metadata.get("description", "No description
        available")
        caption = match.metadata.get("caption")
        if caption:
            text = f"[Image Caption: {caption}]\n\n{text}"
```

Technical Innovations

1. **Hybrid Search Architecture** The system combines dense embeddings (semantic understanding) with sparse embeddings (keyword matching):

```
def get_embeddings(text: str, context_paragraphs: List[str] =
        None) -> Dict[str, List[float]]:
    # Dense embeddings for semantic understanding
    result = genai.embed_content(
        model="models/embedding-001",
        content=text,
        task_type="retrieval_document"
    )
    # Sparse embeddings for keyword matching
    bm25 = BM25Encoder.default()
    sparse_vector = bm25.encode_documents(text)
```

1. **Context-Aware Processing** The system maintains context during processing:

```
def chunk_text_by_paragraphs(text: str, max_length: int = 2500,
        min_length: int = 1000, overlap: int = 1) -> List[str]:
    # Create overlapping chunks
    if overlap > 0 and len(final_paragraphs) > 1:
        overlapped_paragraphs = []
        for i in range(0, len(final_paragraphs), max(1,
        overlap)):
            chunk_size = min(overlap * 2, len(final_paragraphs) -
        i)
            chunk_text =
        "\n\n".join(final_paragraphs[i:i+chunk_size])
            overlapped_paragraphs.append(chunk_text)
```

This comprehensive architecture enables the system to: - Process complex medical documents with both text and images - Generate accurate embeddings for multimodal content - Provide evidence-based verification using both visual and textual sources - Maintain context and relationships between different content types - Scale efficiently while handling rate limits and API quotas

The multimodal capabilities particularly shine in handling medical literature where diagrams, charts, and images are crucial for understanding complex concepts and supporting claims.