Initial Problem Analysis

When I first approached the medical claims fact-checking problem, I realized that traditional text-only verification systems were insufficient. Medical literature heavily relies on visual content - charts showing clinical trial results, diagrams explaining mechanisms of action, and microscopy images demonstrating effects. This led to my first key insight:

"We need a system that can understand both textual and visual evidence to properly verify medical claims."

Architectural Evolution

I started by breaking down the problem into three core challenges:

1. **Content Understanding Challenge**

```
def extract_text_from_pdf(pdf_path: str, save_markdown: bool =
        False) -> str:
    try:
        # First attempt: Use docling for better PDF processing
        converter = DocumentConverter()
        result = converter.convert(pdf_path)
        markdown_text = result.document.export_to_markdown()
    except Exception as e:
        # Fallback to simpler extraction if needed
        reader = PdfReader(pdf_path)
        text = ""
        for page in reader.pages:
            text += page.extract_text() + "\n"
```

I realized early on that medical documents are complex - they contain formatted text, tables, references, and figures. A simple text extractor wouldn't suffice. This led me to implement a multi-layered extraction approach with fallbacks.

1. **Visual Content Challenge** The next major challenge was handling visual content. I thought: "How would a medical expert analyze a research paper? They don't just look at figures in isolation - they consider captions, surrounding text, and overall context."

This led to implementing a sophisticated image processing pipeline:

```
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 with caption matching
    figures = detect_and_extract_figures(page, pdf_name,
        page_num, output_dir)
```

1. **Integration Challenge** The biggest breakthrough came when thinking about how to integrate text and visual evidence. I realized we needed a common representation space:

```
def get_multimodal_embeddings(image_data: bytes, description: str
    = None):
    # Generate embeddings that capture both visual and textual
        aspects
    result = genai.embed_content(
        model="models/embedding-001",
        content=description,
        task_type="retrieval_document",
        title="Research paper image"
    )
```

Solving the Search Problem

A critical realization came when testing early versions: simple vector similarity search wasn't enough. Medical claims often contain specific technical terms that need exact matching alongside semantic understanding. This led to the hybrid search architecture:

```
def search_evidence(claim: str, top_k: int = 5):
    embeddings = get_embedding(claim)
    query_params = {
        "vector": embeddings["dense"],
        "sparse_vector": embeddings["sparse"]
    }
```

Evidence Assessment Evolution

The evidence assessment system went through several iterations. Initially, I tried simple scoring, but realized medical claims often need nuanced evaluation. This led to implementing a multi-faceted assessment:

```
def generate_explanation(claim: str, evidence_list: List[Dict[str,
    Any]]):
    prompt = """
    Analyze how the evidence supports or refutes this claim
        considering:
    1. Direct support or contradiction
```

```
    2. Partial support with caveats
    3. Statistical significance
    4. Methodology context
    5. Visual evidence interpretation
    """
```

Handling Scale and Reliability

As the system grew, new challenges emerged around scale and reliability. I implemented:

1. **Smart Rate Limiting**:

```
@rate_limited_api_call
def generate_explanation(claim: str, evidence_list: List[Dict[str,
        Any]]):
    # Ensures we don't exceed API quotas
```

1. **Resumable Processing**:

```
def mark_as_processed(filename, pdf_path):
    with open(filename, 'a') as f:
        f.write(f"{pdf_path}\n")
```

Key Learnings

Throughout development, several insights proved crucial:

1. **Context is King**: Medical documents need to be understood in context. This led to implementing overlap in text chunking:

```
def chunk_text_by_paragraphs(text: str, overlap: int = 1):
    # Create overlapping chunks to maintain context
    if overlap > 0:
        for i in range(0, len(final_paragraphs), max(1,
        overlap)):
            chunk_size = min(overlap * 2, len(final_paragraphs) -
        i)
```

1. **Source Diversity**: Early testing showed the system sometimes over-relied on a single source. This led to implementing source diversity checks:

```
if ensure_source_diversity:
    source_counts = {}
    for e in evidence:
```

```
        doc_name = e.get("document_name", "Unknown")
        source_counts[doc_name] = source_counts.get(doc_name, 0)
        + 1
```

1. **Explainability**: Medical fact-checking needs to be transparent. Each decision in the pipeline needed to be traceable and explainable, leading to detailed logging and structured outputs.

The final system represents an evolution in thinking from “how do we verify claims?” to “how would a medical expert verify claims?” - considering all available evidence, both textual and visual, while maintaining rigorous standards for verification and explanation.

This approach has proven effective in handling complex medical claims, providing nuanced assessments backed by diverse evidence types, while maintaining the scalability and reliability needed for practical applications.