Solstice: LLM-Orchestrated System for Medical Document Fact-Checking

An AI-Native Approach to Evidence Extraction and Verification

1 Introduction

Solstice is a multi-step system that automatically fact-checks medical claims against scientific literature and clinical documents. It combines layout detection, multimodal language models, and an orchestrated chain of LLM calls to extract and verify evidence from PDFs that contain text, tables, and figures.

The system is designed to handle real-world medical documentation at scale, processing complex clinical documents containing text, tables, and figures.

2 System Architecture

2.1 Document Ingestion Pipeline

The ingestion pipeline transforms unstructured PDFs into queryable structured documents through multiple stages:

1. **Layout Detection**: Uses Detectron2 with PubLayNet-trained models (Mask R-CNN + ResNet-50-FPN). The pipeline relies on pre-trained weights rather than custom training because PubLayNet provides extensive annotated data.
2. **Box Consolidation**: Resolves overlapping detections through IoU-based merging (configurable threshold, default 0.1) and hierarchical nesting. Medical PDFs often contain overlapping layout elements, so the implementation includes domain-specific rules.
3. **Text Extraction**: Uses PyMuPDF for vector text extraction within bounding boxes, and falls back to Tesseract OCR for scanned pages. SymSpell correction fixes common OCR artefacts (e.g., "rn" → "m").
4. **Figure/Table Extraction**: Saves visual elements as PNG at 400 DPI. Images are stored separately for later multimodal analysis rather than embedded as base64.
5. **Reading Order**: Computes reading order with column detection and vertical positioning; this is necessary for the multi-column layouts common in medical journals.

2.2 LLM-Based Fact-Checking System

The fact-checking pipeline processes text evidence through three sequential stages (Extract → Completeness → Verify) followed by parallel image analysis:

- **Evidence Extraction Step**: Searches document text for claim-relevant quotes using gpt-4.1 with temperature=0. Preserves exact quotes while correcting OCR errors and returns structured evidence with relevance explanations.
- **Completeness Checker**: Takes the raw extracted evidence and searches for any additional quotes that weren't initially found. Merges evidence from multiple sources to ensure comprehensive coverage.
- **Evidence Verification Step (V2)**: Validates that all extracted quotes (from both extraction and completeness steps) exist in the source document. Uses semantic matching to handle OCR variations and filters out tangentially related content, achieving high verification rates.
- **Image Evidence Analyzer**: Analyzes figures and tables using vision models to identify supporting visual evidence. Processes images in parallel with semaphore control (max 5 concurrent) and provides detailed explanations.

After all LLM processing completes, an Evidence Presenter step consolidates all verified text and image evidence into structured JSON reports with coverage assessment.

3 Design Decisions

3.1 Step Communication Pattern

Chose filesystem-based communication over message passing:

- Each step reads inputs from disk and writes outputs to disk
- No shared memory or direct step-to-step communication
- Provides debuggability, resumability, and testability in isolation
- Introduces disk I/O overhead, which is acceptable at the current throughput requirements

3.2 Prompt Strategy

Prompt engineering decisions:

- **Structured Output**: Always request JSON with explicit schemas
- **Few-Shot Examples**: Avoided in favor of clear instructions (reduced token usage)
- **Error Context**: Include previous failures in retry prompts
- **Temperature=0**: Consistency critical for medical facts

3.3 Caching Strategy

Multi-level caching approach:

- **Document Cache**: Extracted content never reprocessed
- **Call Cache**: Each step output stored by claim ID
- **No LLM Response Cache**: The pipeline always fetches fresh responses
- Caching is applied at semantic boundaries rather than individual API calls

4 Technical Implementation

4.1 Orchestration Layer

The system uses asynchronous Python with strategic architectural decisions:

- **Hierarchical Orchestration**: Two-level design with StudyOrchestrator → ClaimOrchestrator → processing steps. Enables both study-level and claim-level parallelism control.
- **Step Isolation**: Each processing step runs independently with explicit input/output contracts via Pydantic models. Enables testing and development in isolation.

4.2 Where to Find the Data Artifacts

Solstice writes intermediate and final results to disk in human-readable form. This allows users to inspect the system, debug individual stages, or create visualisations without rerunning the full pipeline.

1. Formatted Claims & Evidence After you run a fact-checking study (`python -m src.cli run-study`) the full, merged JSON for each claim is written to:

```
data/studies/<study_name>/
|-- study_report.json            # roll-up across all claims
`-- claim_<id>/
    `-- evidence_report.json     # structured claims + supporting / refuting evidence
```

The `evidence_report.json` file can be consumed directly by any UX layer.

If you want the raw, per-step outputs (e.g. to analyse the verifier's chain-of-thought) look under the same `claim_<id>/agent_outputs/` folder where each processing step writes an `output.json` plus helper metadata.

2. Marketing-Material Cache PDFs passed through the dedicated marketing pipeline end up in:

```
data/marketing_cache/<pdf_name>/
|-- extracted/content.json   # text + layout (marketing-tu
`-- figures/                 # high-resolution figure crop
```

The structure mirrors `scientific_cache` so the same downstream tools can ingest either kind of document.

3. Quick Directory Cheatsheet For newcomers, these are useful places to explore:

- `src/cli/` – entry-point scripts that orchestrate the pipelines.
- `src/injestion/shared/processing/` – layout detection, reading-order logic, text extractors.
- `src/fact_check/agents/` – implementation of the four specialised LLM processing steps plus formatting.
- `data/scientific_cache/` – processed scientific PDFs (inspect `content.json`).
- `data/marketing_cache/` – processed marketing PDFs.
- `data/studies/` – end-to-end fact-checking results ready for consumption.

4.3 Model Integration

Models were selected based on empirical performance:

- **gpt-4.1 (Primary)**: Evidence extraction and verification with temperature=0 to prioritise consistency over creativity.
- **Vision Models**: o4-mini for image analysis. Images are processed individually with focused prompts rather than in batches.
- **Gateway Pattern**: All LLM calls go through an HTTP gateway service, which handles rate limiting, cost tracking, and provider abstraction.
- **Prompt Engineering**: Uses structured output formats with explicit JSON schemas. Token limits were removed after issues with truncation.

4.4 Robust Error Handling

The implementation includes several measures to improve reliability:

- **Defensive JSON Parsing**: Custom parser handles markdown code blocks, trailing commas, and incomplete responses. Learned from: 15% of GPT-4 responses wrapped JSON in markdown.

- **Smart Retry Strategy**: Exponential backoff with error context injection. Including parsing errors in retry prompts increased the success rate to 95%+.
- **Pydantic Validation**: Type-safe data flow between processing steps; the pipeline fails fast with clear errors rather than propagating invalid data.
- **Context Window Management**: The max_tokens parameter was removed after 30% of evidence was found to be truncated; this increases token cost but keeps responses complete.

5 Implementation Notes

5.1 Multimodal Evidence Integration

Visual evidence is handled in a separate image extraction and analysis pipeline:

- Images are stored as files, not base64, which makes browser-based inspection easier
- Images are analysed individually with focused prompts rather than in batches
- Parallel processing is limited with a semaphore (max 5 concurrent calls) to avoid API rate limits

5.2 Three-Stage Evidence Pipeline

Extraction, verification, and completeness are run as distinct LLM-driven steps:

- **Stage 1**: Extract all potentially relevant quotes (high recall)
- **Stage 2**: Find additional quotes not caught in initial extraction (expand coverage)
- **Stage 3**: Verify all quotes exist and support claim (high precision)

This separation allows the stages to be optimised and tested independently.

5.3 OCR-Resilient Text Matching

The pipeline includes a fuzzy matching algorithm to address common OCR issues:

- Character-level substitution rules ("0"/"O", "1"/"l", "rn"/"m")
- Whitespace normalization for column-spanning text
- Semantic similarity fallback using sentence embeddings

5.4 Filesystem-Centric Architecture

The pipeline stores intermediate data on the filesystem rather than in a database:

- Human-readable JSON can be inspected without extra tools
- Hierarchical directories (document/claim/step) reflect the processing flow
- No additional infrastructure is required
- Files can be version-controlled in Git

6 Conclusion

Solstice combines layout understanding, multimodal analysis, and orchestrated LLM calls to perform medical fact-checking. Tests on multiple documents and claims indicate that the architecture can process real-world medical documentation at moderate scale.