Ramayana Fact Checker

Algorithm Implementation Report

Codebase

ollama_fact_checker.py, model_report.py, generate_report_v2.py

EXECUTIVE SUMMARY

The Ramayana Fact Checker implements a Retrieval-Augmented Generation (RAG) pipeline using Ollama's local Llama 3.8B model for offline verification of claims about Valmiki's Ramayana. The system achieves reliable fact-checking through keyword-based retrieval, structured prompt engineering, and comprehensive evaluation tools.

ALGORITHM PIPELINE

Core Processing Flow

```
Input Claim -> NLTK Keywords -> LLM Query Generation -> Chunk Retrieval ->
Context Assembly -> Ollama LLM -> Response Parsing -> JSON Output
```

1. Keyword Extraction (NLTK)

The core pipeline is implemented in three main code components:

1. Keyword Extraction (NLTK)
2. Query Generation & Chunk Retrieval
3. Prompt Engineering & LLM Processing

These three steps cover the main logic of the fact-checking pipeline. Other supporting code (e.g., response parsing, evaluation, error handling) is not shown here for brevity.

```
def get_keywords_from_claim(self, claim: str) -> List[str]:
    words = re.findall(r'\b\w+\b', claim.lower())
    keywords = [w for w in words if w not in self.stopwords and len(w) > 2]
    freq = nltk.FreqDist(keywords)
    return [kw for kw, _ in freq.most_common(10)]
```

2. Query Generation & Chunk Retrieval

```
def retrieve_relevant_chunks(self, claim: str, chunks: List[Dict]) -> List[Dict]:
    keywords = self.get_keywords_from_claim(claim)
    queries = self.generate_search_queries(claim)
    combined_keywords = set(keywords)
    for q in queries:
        combined_keywords.update(self.get_keywords_from_claim(q))
    scored = []
    for chunk_data in chunks:
        score = sum(1 for kw in combined_keywords if kw in chunk_data["text"].lower())
        if score > 0:
            scored.append({"score": score, **chunk_data})
    return sorted(scored, key=lambda x: x["score"], reverse=True)[:5]
```

3. Prompt Engineering & LLM Processing

```
def create_fact_checking_prompt(self, claim: str) -> str:
    chunks = self.retrieve_relevant_chunks(claim, self.all_chunks)
    context = "\n\n---\n\n".join([c["text"] for c in chunks])
    return f'''Fact-check: "{claim}"
CONTEXT:
{context}
RULES:
1. EXACT MATCHING: Details must match context.
2. COMPONENT ANALYSIS: Break into WHO/WHAT/WHEN etc.
3. NO ASSUMPTIONS: Verify explicit facts only.
FORMAT:
VERDICT: [TRUE/FALSE/INSUFFICIENT_DATA]
CONFIDENCE: [0.0-1.0]
EVIDENCE: [Supporting text]
EXPLANATION: [Reasoning]'''
```

TECHNICAL CONFIGURATION

Key Parameters

Parameter	Value
CHUNK_SIZE	1500 characters
CHUNK_OVERLAP	200 characters
MAX_RETRIEVED_CHUNKS	5 chunks
TOP_N_KEYWORDS	10 keywords
Ollama Model	ramayana-fact-checker:latest
LLM Temperature	0.3 (for consistency)
LLM Top-p	0.9 (for quality)
API Timeout	180 seconds
Source Texts	6 Ramayana books (All Kandas)

Performance Metrics

Metric	Value
Avg. Processing Time	~20-35 seconds per claim
Model Memory Usage	8-16GB (Llama 3.8B)
Text Corpus Size	6 books, approx. 1500 chunks
Keyword Extraction	<100ms (NLTK)
Chunk Retrieval	~50ms (keyword matching)

EVALUATION SYSTEM

model_report.py Framework

```
# Evaluation Process Snippet
for item in evaluation_data:
    claim = item.get("statement")
    expected = item.get("label")
    result = fact_checker.fact_check(claim)
    log = {"claim": claim, "expected": expected, **result}
    # ... (logging and accuracy calculation) ...
accuracy = (correct / total) * 100
```

Debug Tools

- debug_specific_claims(): Analyze problematic cases with full retrieval visibility.
- search_pushpaka_origin(): Targeted keyword searches for specific elements.
- Chunk Analysis: Detailed scoring and source tracking for retrieved text segments.

CRITICAL VERIFICATION RULES

- Exact Matching: All claim details must precisely match the provided context; no assumptions allowed.
- Component Analysis: Complex claims are broken down into core components (WHO, WHAT, WHEN, WHERE, HOW, WHY).
- Timing Verification: Specific temporal assertions must be explicitly verifiable from the context.
- Character Precision: Similar characters and their roles must be distinguished accurately.

ERROR HANDLING & RELIABILITY

- Connection Testing: Automatic verification of Ollama server connectivity.
- Model Validation: Checks for the availability of the specified model.
- Graceful Degradation: Provides fallback facts if source text files are unavailable.
- NLTK Dependency Management: Automatically downloads missing NLTK resources (e.g., stopwords).
- Comprehensive Logging: INFO level logging provides detailed insights into processing steps.

FUTURE ENHANCEMENTS

- Vector Embeddings: Implement semantic search using sentence transformers for improved retrieval relevance.
- Model Fine-tuning: Further train Llama models specifically on the Ramayana corpus for enhanced domain expertise.
- Multi-source Validation: Cross-reference information from multiple Ramayana translations and commentaries.
- Performance Optimization: Introduce caching mechanisms and batch processing for faster response times.
- Web Interface: Develop a user-friendly web interface for interactive fact-checking.