ClaimAssist

Are you submitting an insurance claim? Let ClaimAssist help you validate your claim based on your unique insurance policy

Demo

1. Problem Description

The cards are stacked against you when trying to submit a successful insurance claim because insurance policies can be tedious and cryptic to read.

2. Why this problem is important to users

Insurance policyholders are usually average people. They face significant challenges when filing claims because they lack expertise to interpret complex policy language and determine if their specific incident is covered. This leads to uncertainty, delayed submissions, and often claim denials due to missing information or misunderstanding of coverage terms.

In addition, insurance companies purposefully make you jump through as many hoops as possible to submit a claim. Their business is built around making it as hard as possible to claim the coverage you've been paying into even if your claim is more than valid.

3. Proposed Solution

ClaimAssist provides an Claim Consultant Agent allowing users upload their insurance policy and describe their incident to receive instant, detailed analysis of claim validity. The interface feels like having a knowledgeable insurance consultant available 24/7 - users simply describe what happened, and the system provides clear guidance on whether their claim is likely to be approved, what policy sections support this decision, and actionable next steps including draft emails to insurance companies.

4. Technology Stack

- **LLM: OpenAI GPT-4o-mini**

    - for claim analysis because it provides excellent reasoning capabilities at low cost for insurance document interpretation.
- **Embedding Model:** OpenAI text-embedding-3-small
    - for policy document vectorization due to its strong performance on domain-specific text and wide compatibility.
- **Orchestration:**
    - LangGraph for agent workflow management because it provides clear state management and tool integration for multi-step reasoning processes.
    - LangChain for agent tooling such as pdf loading & chunking, tavily search and different RAG retrieval strategies
- **Vector Database:** Qdrant
    - for semantic search of policy documents due to its excellent performance and easy deployment.
- **Monitoring:** LangSmith
    - for detailed agent tracing and debugging
    - Console logging and visual strategy indicators for debugging and user transparency because it's lightweight and provides immediate feedback.
- **Evaluation:**
    - RAGAS framework
        - for retrieval assessment because it provides standardized metrics for RAG pipeline performance measurement.
    - LangSmith
        - for evaluating agent latency and performance
- **User Interface:** Next.js with TypeScript and Tailwind CSS
    - for responsive, type-safe frontend development with excellent developer experience.
- **Serving & Inference:** FastAPI
    - for backend API due to its automatic OpenAPI documentation, excellent performance, and seamless integration with Python AI libraries

5. Agent Use and Agentic Reasoning

The Claim Consultant agent uses agentic reasoning to dynamically decide between RAG policy search and web research tools based on claim complexity. It determines when additional context is needed, formulates targeted queries for policy documents, validates findings against external sources when necessary, and synthesizes all information into structured evaluations with reasoning chains.

6. Data Sources and External APIs

- **Insurance Policy PDFs:** Primary data source for coverage determination and policy interpretation

- **Tavily Search API:** External web search for clarifying complex insurance terminology, weather events, or technical scenarios not clearly defined in policies
- **User Claim Descriptions:** Structured input containing incident details, dates, locations, and circumstances
- **OpenAI API:** For embeddings generation and LLM inference in claim evaluation
- **Cohere API:** Optional premium reranking for improved retrieval accuracy in advanced strategies

7. Default Chunking Strategy

We use Recursive Character Text Chunking with 1000-character chunks and 200-character overlap for policy documents because this preserves context around important clauses while maintaining manageable chunk sizes for embedding and retrieval - insurance policies contain interconnected provisions that require contextual understanding.

8. Additional Data Requirements

None

9. End-to-End Prototype

Demo: https://www.loom.com/share/5b0a18237f14415bac3949a47c88f8ca

10. RAGAS Metrics

Metric	Value
Context Recall	0.5802
Faithfulness	0.3125
Factual Correctness (F1)	0.5155
Answer Relevancy	0.5676
Context Entity Recall	0.0265
Noise Sensitivity (Relevant)	0.1366

11. Performance and Effectiveness Conclusions

Moderate Context Recall (0.5802): The basic k=5 retrieval captures approximately 58% of relevant policy
information, indicating significant room for improvement in finding all pertinent policy clauses for claim evaluation.

Poor Faithfulness (0.3125): Only 31% of generated responses are grounded in the retrieved context, meaning the agent frequently hallucinates or adds information not found in the policy documents - a critical flaw for insurance applications requiring strict adherence to policy language.

Decent Factual Correctness (0.5155): The F1 score suggests moderate accuracy in factual claims, but with significant room for improvement given the high-stakes nature of insurance decisions.

Moderate Answer Relevancy (0.5676): Responses are reasonably relevant to user queries, though 43% of content may be tangential or off-topic.

Very Poor Entity Recall (0.0265): The system fails to capture important policy-specific entities like coverage limits, deductibles, and specific terms - critical information for accurate claim evaluation.

Low Noise Sensitivity (0.1366): The system shows minor susceptibility to irrelevant information, which means the agent is less likely to generate response with inaccurate information.

Overall Assessment: Basic retrieval provides a functional but unreliable foundation that requires significant enhancement for production insurance use cases.

12. Planned Retrieval Techniques for Experimentation

- Parent Document Retrieval: This technique will provide larger context windows by retrieving full document sections rather than small chunks, ensuring policy clauses aren't fragmented and maintaining the logical flow of insurance policy language that spans multiple sentences.

- Contextual Compression with FlashRank: This approach will first cast a wide net with k=20 retrieval, then use reranking to select the most relevant documents, improving precision while maintaining recall for complex insurance scenarios that require multiple policy sections.

Note: These are the two advanced strategies I will be testing as indicated in the prototype notebook, moving beyond basic k=5 retrieval.

14. Performance Comparison Table (Actual Results)

Metric	Basic Retrieval	Parent Doc Retrieval	% Δ (Parent vs. Basic)	Contextual Compression	% Δ (Compression vs. Basic)
Context Recall	0.5802	0.7788	+34.24%	0.7879	+35.77%
Faithfulness	0.3125	0.3511	+12.36%	0.3572	+14.24%
Factual Correctness (F1)	0.5155	0.4845	−6.02%	0.4591	−10.97%
Answer Relevancy	0.5676	0.4049	−28.65%	0.4057	−28.44%
Context Entity Recall	0.0265	0.0673	+153.96%	0.0866	+226.04%
Noise Sensitivity (Relevant)	0.1366	0.1330	−2.64%	0.0925	−32.29%

Based on the evaluation comparison image, here are the actual test results across all strategies:

Context Recall Performance:

- Parent Document: 0.7788 (+34%)
- 🏆Contextual Compression: 0.7879 (+36%) 🏆

Faithfulness Trends:

- Parent Document: 0.3511 (+12%)
- 🏆 Contextual Compression: 0.3572 (+14%) 🏆

Factual Correctness:

- Note:
    - This metric compares the reference answer to the actual agent response. However, due to the nature of the RAGAS query-answer generator the actual agent response is much more detailed then the reference answer. The reference answer provides a simplified analysis of the claim, the actual response from the agent is much more detailed including policy citations and an email if the claim is valid. To better compute this metric the RAGAS query generator must be further customised there this metric is not reliable for our analysis.
- Parent Document (-6%)
- Contextual Compression (-10%)

Answer Relevancy Trade-offs:

- **Note:** This metric compares the user input to the actual agent response. Because the user input is formulated as a claim description and the response is an analysis of the claim, it is possible that the computation of this metric the two entities are not correctly compared. This metric must be further fine tuned and is not reliable for this analysis.
    - Parent Document: 0.4049 (-29%)

    - Contextual Compression: 0.4057 (-29%)

Entity Recognition Improvements:

- Parent Document: 0.0673 (+154%)
- Contextual Compression: 0.0866 (+227%)

Noise Reduction:

- Parent Document: 0.1330 (-3%)
- 🏆Contextual Compression: 0.0925 (-32%)🏆

🏆**Best Option:** Contextual Compression 🏆

15. Future Plans

- Multi-Agent Approach
    - The claim analysis could benefit from a more structured reasoning approach
    - For example
        - Break the claim into subclaims
        - For each sub claim trigger another agent to perform focused research:
            - What policies relate to the subclaim?
                - Is the subclaim valid according the policy
            - In what ways could the claim be valid?
                - Use RAG to find exclusions and compare it to the claim
        - ... (Use deep research to come up with a systematic approach to validating a claim)
- Customized Synthetic Data Generation and RAGAS metrics
    - The input and output to this agent differs from the question-answer standard
    - Must apply further customization to the RAGAS strategy to generate data that better matches the use case and metrics that better evaluate the use case