Intelligent Underwriting Workbench

An Agentic AI Co-Pilot for Commercial Risk Assessment

LangChain | Ollama | Streamlit | Pydantic

Presented by: Gurupreet Dhande, Khushi Dekate, Arnav Kalambe

The Business Challenge

Problem Statement

Unstructured Data Overload

Commercial underwriting involves analyzing messy, unstructured documents:

- PDF Application Forms
- Loss Run Reports (Claims History)
- External Credit Reports
- Broker Emails & Notes

Operational Impact

Manual processing leads to:

- **Slow Turnaround:** Days to quote.
- **Inconsistency:** Human error/bias.
- **High Cost:** Experts doing data entry.

The Solution: Agentic AI Workbench

A **"Human-in-the-Loop" Co-Pilot** that automates data extraction, analysis, and initial decisioning.

Ingest

Reads messy PDFs and raw text instantly.

Analyze

Parallel agents assess Claims, Credit, and Legal risks.

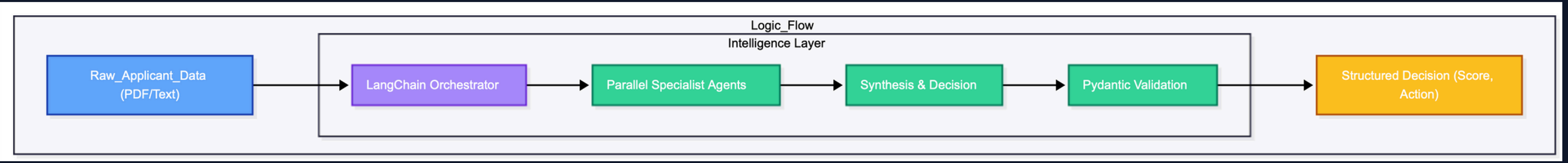
Decide

Generates a consistent Risk Score (0-10).

Act

Drafts formal decision letters automatically.

Detailed Components

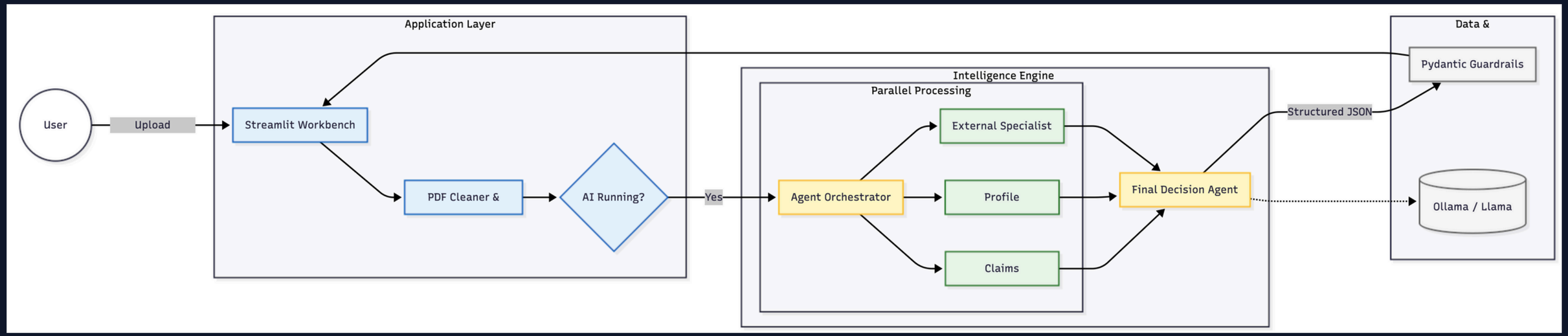


Core Layers

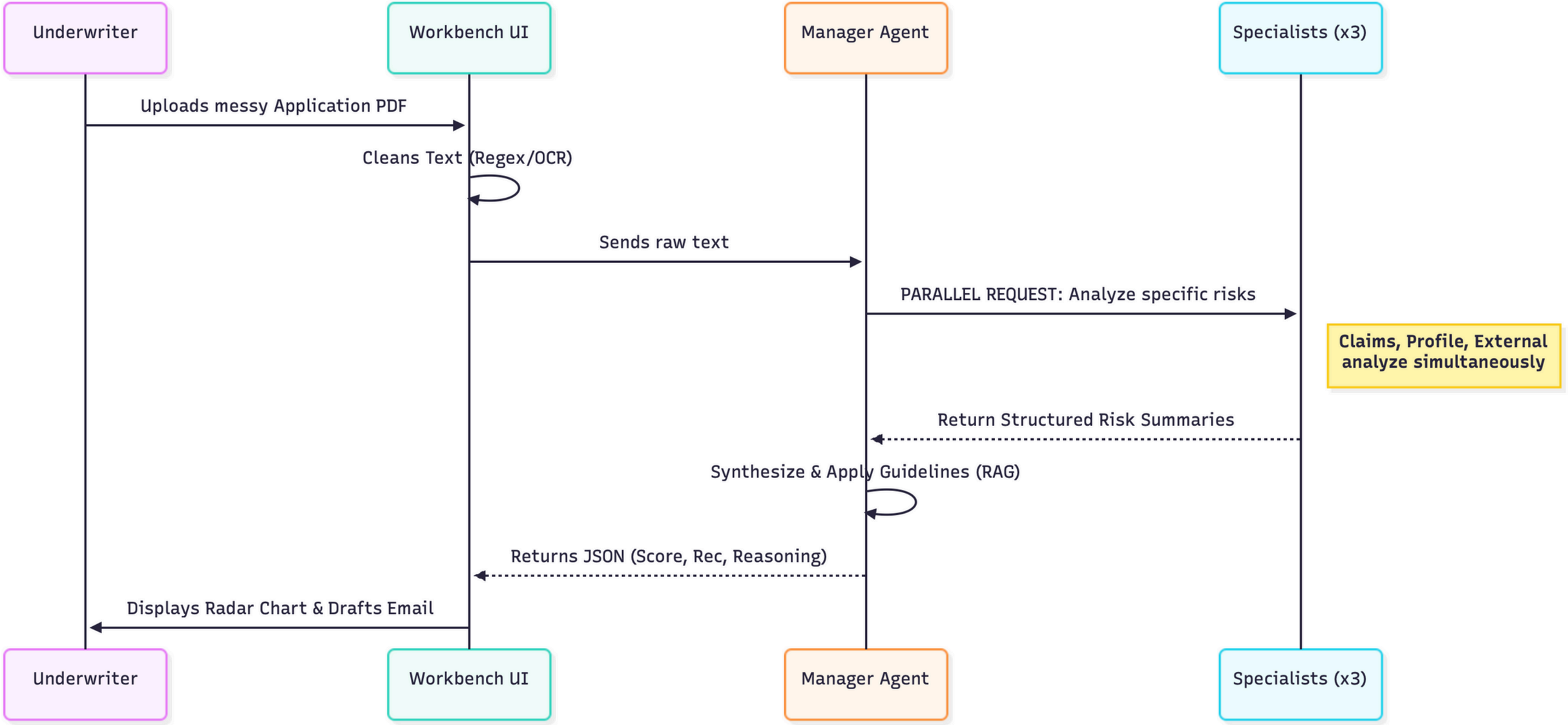
- **Ingress:** Nginx/Traefik Load Balancer.
- **Frontend:** Streamlit (Dashboard, Visuals).
- **Backend:** FastAPI (Async endpoints, Scaling).
- **Intelligence:** LangChain + Ollama + XGBoost.
- **Storage:** PostgreSQL (Audit), S3 (PDFs).

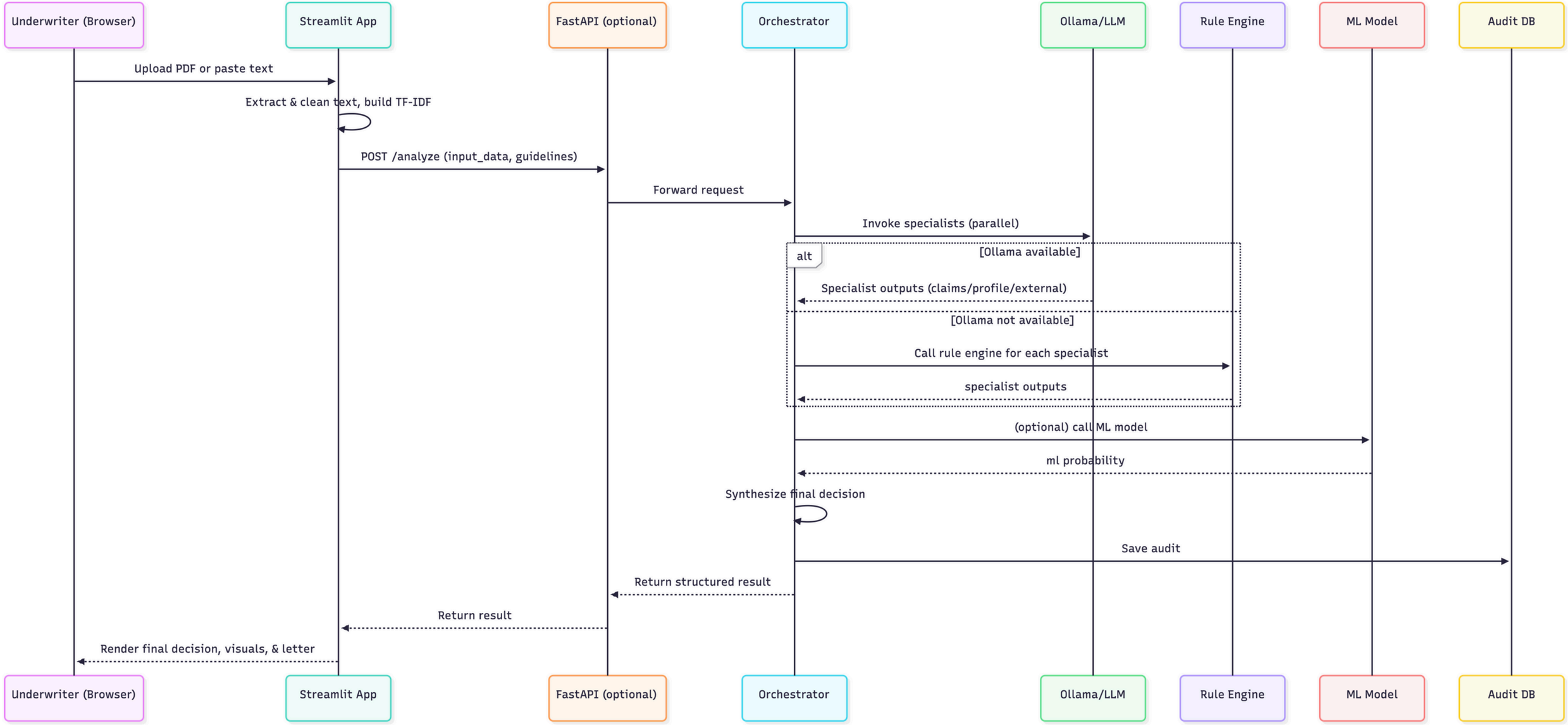
System Architecture

High Level Design



The design of the Intelligent Underwriting Workbench, a robust, hybrid system. It focuses on demonstrating the Agentic Architecture, showcasing solutions to technical challenges like PDF parsing and JSON reliability, and concluding with the business impact of rapid, consistent risk assessment.


Underwriter
Workbench UI
Manager Agent
Specialists (x3)
Uploads messy Application PDF
Cleans Text (Regex/OCR)
Sends raw text
PARALLEL REQUEST: Analyze specific risks
Claims, Profile, External analyze simultaneously
Return Structured Risk Summaries
Synthesize & Apply Guidelines (RAG)
Returns JSON (Score, Rec, Reasoning)
Displays Radar Chart & Drafts Email
Underwriter
Workbench UI
Manager Agent
Specialists (x3)


Underwriter (Browser)
Streamlit App
FastAPI (optional)
Orchestrator
Ollama/LLM
Rule Engine
ML Model
Audit DB
Upload PDF or paste text
Extract & clean text, build TF-IDF
POST /analyze (input_data, guidelines)
Forward request
Invoke specialists (parallel)
alt
[Ollama available]
Specialist outputs (claims/profile/external)
[Ollama not available]
Call rule engine for each specialist
specialist outputs
(optional) call ML model
ml probability
Synthesize final decision
Save audit
Return structured result
Return result
Render final decision, visuals, & letter
Underwriter (Browser)
Streamlit App
FastAPI (optional)
Orchestrator
Ollama/LLM
Rule Engine
ML Model
Audit DB

The "Multi-Agent" Methodology

Parallel Processing

Why multiple agents? Because a single LLM gets overwhelmed. We use **Separation of Concerns**.

1. Claims Specialist

Role: Looks ONLY for loss history.

Task: Extract frequency, severity, and recency of claims.

```
{"score": 6, "summary": "Large fire claim in 2021..."}
```

2. Profile Specialist

Role: Looks ONLY for business health.

Task: Analyze credit score, years in business, and industry risk.

```
{"score": 2, "summary": "Strong credit, 15yr tenure..."}
```

3. External Specialist

Role: Looks ONLY for legal/compliance.

Task: Find OSHA violations, lawsuits, or negative news.

```
{"score": 4, "summary": "Minor safety violations..."}
```

Feature 1: Robust PDF Ingestion

Technical Deep Dive

The Challenge

PDF extraction often results in "broken text" where lines split mid-sentence or tables lose structure.

Example Raw Output:

```
Appli-
cant Name: Ac-
me Const...
```

The Solution (Regex Pipeline)

We built a custom cleaning pipeline in Python:

1. **Merge Hyphens:** Reconnects split words.
2. **Fix Newlines:** Aggressive regex `([^\n])\n([^\n])` to merge broken lines while keeping paragraphs.
3. **Standardize Bullets:** Converts confusing PDF symbols to standard text bullets.

Feature 2: Structured Guardrails

Pydantic Integration

Without Guardrails

LLMs are unpredictable. They might return:

- A long paragraph.
- Invalid JSON.
- A score of "High" instead of a number.

Result: The application crashes.

With Pydantic & LangChain

We define a strict **Data Contract**:

```
class RiskOutput(BaseModel):
  score: int = Field(ge=0, le=10)
  decision: Literal["Approve", "Decline"]
```

Result: 100% Reliable, machine-readable output.

Feature 3: Advanced Capabilities

RAG & Analytics

Context-Aware RAG

The agent doesn't just guess; it reads **Underwriting Guidelines** (injected via sidebar) to make policy-compliant decisions.

Risk Visualization

Used **Plotly** to generate interactive Radar Charts, visualizing the balance between Credit, Claims, and Legal risks.

Explainability (XAI)

Includes a "Deep Dive" dashboard using **SHAP** concepts to show exactly *why* a score was given (e.g., "Credit score reduced risk by 10%").

Deployment Strategy

Production Scaling

- **Orchestration:** Kubernetes (K8s).
- **Async Tasks:** Redis + Celery for batch jobs.
- **Storage:** PostgreSQL (Audit) + S3 (Docs).

Containerization

- **Frontend Container:** Streamlit
- **Backend Container:** FastAPI + Orchestrator
- **Model Container:** Ollama (GPU enabled)

Demo Flow

Why This Tech Stack?

Technology Choices

Why Ollama / Llama 3?

Privacy & Security. Insurance data is sensitive. Cloud APIs (like GPT-4) send data off-premise. Ollama runs 100% locally, ensuring zero data leakage.

Why LangChain?

Orchestration. Managing 4 distinct agents and passing data between them requires a robust framework. LangChain handles the "plumbing" effortlessly.

Why Streamlit?

Speed to Value. Allowed us to build a full-stack, interactive dashboard in Python without needing a separate frontend team.

Why Pydantic?

Reliability. In enterprise software, AI outputs must be predictable types, not random text.

Impact & Results

Business Value

90%

Faster Triage

Reduced initial review time from 30 mins to 30 seconds.

100%

Consistency

Eliminates human variability in risk assessment.

0$

API Costs

Runs completely locally using Open Source models.

Future Roadmap

Next Steps

Phase 1 (Current)

- ✓ Multi-Agent Analysis
- ✓ PDF Ingestion
- ✓ Rule-Based Fallback

Phase 2 (Next)

- **Full RAG:** Connect to vector DB of 10,000+ past policies.
- **Human Feedback Loop:** Allow underwriters to correct AI to improve model.

Phase 3 (Scale)

- **API Integration:** Connect directly to Policy Admin Systems.
- **Multi-Modal:** Analyze photos of property damage.

Thank You