Technical Report: Agentic Workflow for Oil Well Test Anomaly Interpretation

# 1. Overview

This report describes the design and implementation of a Proof of Concept (POC) agentic workflow system for interpreting oil well test anomalies using a combination of:

- Data-driven feature engineering

- Vector similarity search on historical knowledge base (Qdrant)

- Large Language Model (LLM) inference with prompt engineering

- Directed state graph orchestration via LangGraph framework

- Interactive visualization via Streamlit

The POC enables a reservoir engineer to select a well test, analyze changes compared to previous tests, retrieve similar historical cases, and generate a concise expert interpretation with actionable recommendations.

# 2. Objectives

- Automate anomaly detection interpretation leveraging domain knowledge and past cases

- Use an agentic architecture to modularize the workflow into interpretable and maintainable steps

- Enable easy integration of vector databases and LLMs for retrieval-augmented generation

- Provide an interactive UI to visualize well test trends, workflow state, and AI-generated insights

# 3. Architecture

## 3.1 Agentic Workflow (LangGraph)

The core orchestration uses a StateGraph pattern with three sequential agents:

|  |  |
| --- | --- |
| Agent | Responsibility |
| describe\_welltest | Generate summary prompt based on recent well test changes |
| kb\_search | Query Qdrant vector DB for semantically similar historical interpretations |
| generate\_interpretation | Generate final expert interpretation by sending consolidated prompt + KB matches to LLM |

The workflow is defined as a directed graph, enabling clear data flow and potential extensibility.

## 3.2 Key Components

|  |  |  |
| --- | --- | --- |
| Component | Technology / Library | Purpose |
| Data Processing | Pandas, NumPy | Well test data manipulation and feature engineering |
| Vector Database | Qdrant | Store and retrieve historical anomaly explanations |
| Embeddings | Custom embedding model | Convert text prompts to 1536-dimensional vectors |
| LLM | LangChain/OpenAI | Generate textual interpretations |
| Orchestration | LangGraph (custom StateGraph class) | Manage agent workflow and state propagation |
| Visualization | Streamlit, Plotly | Interactive UI and time series visualization |

# 4. Detailed Module Description

## 4.1 Data Preparation and Feature Engineering

- Input CSV data contains daily well test metrics including oil rate (WT Oil), water cut (WT WCT), and zonal bottom-hole pressures (Z1 BHP, Z2 BHP, Z3 BHP).

- Features are augmented with calculated deltas, rates, and z-scores to highlight deviations from historical trends.

- The last well test row and its predecessor form the basis for the anomaly description prompt.

## 4.2 Agent 1: describe\_welltest

- Inputs: DataFrame subset including selected test row and previous row.

- Outputs: Natural language prompt describing observed changes and domain-specific anomalies.

- Displays the generated prompt in an expandable section within Streamlit for user inspection.

- Signals threshold for the description generation function is decided by looking at the log distribution of the signals. This can be refined further by looking at the complex patterns in the signal to define more robust thresholds.

A graph of a distribution of log change

AI-generated content may be incorrect.

## 4.3 Agent 2: kb\_search

- Inputs: Prompt from Agent 1.

- Uses embedding model to encode the prompt.

- Queries Qdrant vector database for top 44 similar vectors using cosine similarity.

- For each candidate, evaluates entailment score with the prompt to rank true semantic matches.

- Returns top 3 historical interpretations to the state and displays them interactively.

## 4.4 Agent 3: generate\_interpretation

- Inputs: Prompt and top KB matches.

- Constructs a composite prompt combining anomaly description and historical context.

- Sends prompt to LLM (OpenAI via LangChain) requesting a structured interpretation with:

- Zone status (open/shut-in)

- Well phase (e.g., transient, zonal test, stimulation)

- Recommendations (e.g., shut-in, acid stimulation, continue monitoring)

- LLM response is rendered in Streamlit and stored in the workflow state.

## 4.5 Data Storage: store\_to\_qdrant

- Function to upsert new anomaly explanations and their embeddings into Qdrant.

- Embeddings are generated for textual explanation summaries.

- Payload stores original data, prompt, and explanation for traceability.

## 4.6 Visualization

- Time-series plots of key well test parameters using Plotly.

- Custom network graph visualization of the workflow state graph using NetworkX and Matplotlib.

- Streamlit expander panels organize outputs and reduce visual clutter.

# 5. Usage Workflow

1. User selects well test CSV from dropdown.

2. Well test data table and trends are displayed.

3. User selects specific well test entry for analysis.

4. Agentic workflow executes:

- Summarize anomalies compared to previous test.

- Retrieve and rank historical similar cases from vector DB.

- Generate expert interpretation from LLM.

5. Outputs presented interactively for user review.

6. Optional: Store new interpretations back to vector DB.

# 6. Production Considerations & Recommendations

|  |  |
| --- | --- |
| Aspect | Recommendations |
| Scalability | Use asynchronous execution for LLM calls and Qdrant queries; batch process multiple tests. |
| Robustness | Add comprehensive error handling for missing data, API failures, empty results. |
| Security | Secure API keys, use environment variables, and ensure GDPR compliance for data. |
| Performance | Cache embedding computations, Qdrant collections metadata; optimize prompt size. |
| Extensibility | Modularize agents to allow parallel or conditional execution paths. |
| Logging & Monitoring | Implement detailed logging for debugging and audit trail of interpretations. |
| UI/UX | Provide richer interactivity, filterable historical cases, downloadable reports. |
| Testing | Develop unit/integration tests for agents and end-to-end workflows with sample data. |
| Deployment | Containerize with Docker, orchestrate with Kubernetes, monitor resource usage. |
| Versioning | Track versions of embedding models, LLM prompts, and vector DB schema for reproducibility. |

# 7. Next Steps for Vendor

- Refine embedding model and Qdrant schema to support richer metadata and filtering.

- Integrate production-grade LLM APIs with rate limiting and retries.

- Build modular, parameterized agent implementations with plug-and-play capabilities.

- Enhance UI for collaborative review and feedback loops with domain experts.

- Automate data ingestion pipelines to update vector DB continuously with new test results.

- Develop documentation and API contracts for seamless integration with client infrastructure.

# 8. Appendix

- Code snippets for main agents and workflow construction (as provided).

- Sample data format and column definitions.

- Description of LangGraph and its role in orchestrating multi-agent workflows.

If you want, I can also prepare a vendor-ready API specification or detailed deployment guide next. Would you like me to?

Initial Approach:

Approach 1:

During the initial phase of anomaly detection exploration, an XGBoost classifier was trained using engineered features such as zonal pressure differences, gas-oil ratio (GOR), and normalized production. The model performed reasonably well in classifying labeled anomalies using traditional supervised learning metrics. However, it exhibited limitations in reliably detecting subtle or emerging changes in the well test data, particularly for transient events or gradual zonal behaviors that did not have strong separability in the feature space. While the classifier achieved good accuracy for distinct anomaly classes, it failed to generalize well across the full spectrum of operational scenarios, prompting a shift toward agentic analysis and LLM-based interpretability for more nuanced and context-aware detection.

Approach 2:

In the second approach, a domain-informed rule-based anomaly detection strategy was developed to overcome the limitations of purely statistical or machine learning models. This method combined zonal pressure diagnostics with decline curve analysis to flag anomalous behaviors based on deviations from expected production trends. A hyperbolic decline model was first fitted to the well’s oil rate over time, capturing the baseline performance under commingled flow. Then, for each well test entry, zonal BHPs were compared against the average to classify each zone as “open,” “shut,” or “commingled.” Anomalies were flagged when actual oil rates deviated significantly from the fitted decline, particularly when correlated with high water cut or previous optimization actions. This rule-based system proved more interpretable and robust in capturing domain-specific anomalies, such as transient flow events or the impact of zonal shut-ins, which were not always well detected by the initial XGBoost classifier.

Approach 3:

The third approach utilized an unsupervised machine learning technique—**Isolation Forest**—to detect anomalies in well test data without relying on predefined class labels. This method involved selecting key engineered features such as oil and liquid rates, tubing head pressure, zonal pressure differences, and derived metrics like GOR and normalized production. These features were standardized using StandardScaler to ensure uniform scaling across variables before fitting the model. The Isolation Forest was then trained to isolate anomalies by recursively partitioning the data and scoring each observation’s degree of “isolation.” Observations with significantly different patterns received lower anomaly scores and were labeled as outliers (−1). This approach proved effective in capturing subtle multivariate deviations and rare patterns in production behavior, complementing the insights from the rule-based and XGBoost models with a purely data-driven perspective.

Below plot show a comparison of the 3 approaches. Also added as an artifact in the github repo.

