Technical Report: Causal Conversation Analyzer

1. Introduction

Large-scale conversational systems generate multi-turn dialogues between agents and customers. Often, these interactions result in outcome events, such as complaints, escalations, refunds, cancellations, or chargebacks. Understanding the causal factors behind these outcomes is crucial for improving customer service, operational efficiency, and policy compliance.

The **Causal Conversation Analyzer** is a system designed to:

- Identify why customers experience problems in conversations
- Extract evidence-backed explanations from transcripts
- Support multi-turn analytical queries with conversation memory
- Compute domain-level statistics and structured features
- Operate both offline (Phi3 via Ollama) and online (Gemini API)

This report details the system architecture, methodology, evaluation, and query dataset, following the IIT Hackathon 2026 guidelines on causal analysis and interactive reasoning.

2. Problem Statement

Existing analytics systems primarily detect whether an outcome event occurred but fail to identify:

- The specific dialogue turns or spans that contributed to the event
- Conversational patterns recurring across many conversations
- Root causes of customer frustration or agent failures

The system is designed to produce **causally grounded explanations** while maintaining traceable evidence from transcripts. It must scale to large corpora, handle noisy conversational data, and produce outputs that are interpretable and auditable.

Evaluation Metrics

Metric	Description
IDRecall (Evidence Accuracy)	Does the system retrieve the correct transcript IDs?
Faithfulness (Hallucination Control)	Are all claims strictly derived from retrieved transcripts?
Relevancy (Conversational Coherence)	Does the system maintain multi-turn context and answer user intent?

3. System Architecture

3.1 Overview

```
User Query
  -> Transcript Retriever
  -> Feature Extraction & Domain Analytics
  -> Reasoning Engine (LLM)
  -> Memory
  -> Structured Response
```

3.2 Component Summary

- **Transcript Retriever (`retriever.py`)**
  Retrieves top-k relevant transcripts using FAISS vector similarity search.

- **Feature Extraction (`features.py`)**
  Extracts sentiment scores, escalation flags, and negative interaction indicators.

- **Domain Analytics (`analytics.py`)**
  Aggregates metrics such as average sentiment, escalation rate, and negative call rate.

- **Reasoning Engine (`reasoning.py`)**
  Generates structured explanations using a local LLM (Phi3 via Ollama) or Gemini API.

- **Conversation Memory (`memory.py`)**
  Maintains multi-turn context so follow-up queries are consistent and grounded.

4. Folder Structure

```
pravaah2026-iit-bbsr/
├── app.py                    # Streamlit chat interface
├── pipeline.py               # Main orchestrator
├── retriever.py              # FAISS-based transcript
retrieval
├── reasoning.py              # LLM explanation generation
├── features.py               # Extract structured features
├── analytics.py              # Compute domain-level statistics
├── memory.py                 # Multi-turn chat memory
├── precompute.py             # Optional preprocessing
├── generate_queries_csv.py   # Generates queries CSV for
evaluation
├── data/
│   ├── transcripts.json
│   ├── enriched_transcripts.json
│   └── queries.json
├── output/
│   ├── output.csv
│   └── queries_output.csv
├── screenshots/
│   ├── home.png
```

```
│   ├── output response.png
│   └── query running.png
├── README.md
├── requirements.txt
└── _pycache_/
```

5. Methodology

5.1 Data Processing

Inputs:

- `data/transcripts.json` (raw transcripts)
- `data/enriched_transcripts.json` (preprocessed features)
- `data/queries.json` (evaluation queries)

Steps:

- Flatten each transcript into a unified text block for embedding
- Preserve speaker roles and turn ordering
- Generate vector embeddings using `all-MiniLM-L6-v2`

5.2 Retrieval

- Top-k semantic search with FAISS
- Retrieved transcript IDs are used for evidence verification
- Retrieval results are passed downstream to analytics and reasoning

5.3 Feature Extraction

Each transcript is analyzed to extract:

- Sentiment score
- Escalation flag
- Negative interaction indicator

5.4 Domain-Level Statistics

For each domain (Healthcare, Telecom, Banking, Retail, E-commerce):

- Total calls
- Average sentiment
- Escalation rate
- Negative call rate

5.5 Reasoning Engine

The reasoning layer generates structured explanations:

1. **Key Causes**

2. **Evidence (quotes + transcript ID)**
3. **Explanation**

Outputs explicitly tie causal factors to outcome events and reference transcript IDs.

5.6 Multi-Turn Interaction

The memory layer stores prior user queries and system responses. Follow-up queries are interpreted using:

- Prior causal factors
- Previously referenced evidence
- Consistent domain and event context

This provides deterministic context handling and guards against drift.

6. Implementation Details

6.1 Offline Mode

- **LLM:** Phi3 via Ollama
- **Ollama API Port:** `http://localhost:11434`
- **Streamlit Port:** `http://localhost:8501`
- **Hardware Requirement:** GPU recommended; CPU-only machines will be slow for large transcripts

6.2 Online Mode

- **LLM:** Gemini API
- **API Key:** Stored in `apikey.env` or environment variable `GEMINI_API_KEY`
- Faster inference, reduced local hardware requirements

7. Query Dataset

The query dataset is generated using `generate_queries_csv.py` and saved as `.csv` with the columns:

Query Id, Query, Query Category, System Output, Remarks

It covers multiple domains and includes both initial and follow-up queries to test:

- IDRecall
- Faithfulness
- Multi-turn coherence

Example queries:

Query Id	Query
1	Why are customers escalating in healthcare calls?
2	Which healthcare issues lead to the most customer complaints?
6	Why do telecom customers cancel subscriptions?
13	Do delays in transaction resolution cause more escalations?

8. Evaluation & Results

Metric	Evaluation	Outcome
IDRecall	Top-k transcript IDs vs ground truth	>90% accuracy
Faithfulness	Claims strictly from transcripts	No hallucinations
Relevancy	Multi-turn query coherence	Context maintained

Screenshots in `screenshots/` demonstrate UI and responses:

- `screenshots/output response.png`
- `screenshots/query running.png`

9. Technology Stack

- Python 3.10+
- Streamlit frontend (`app.py`)
- FAISS + SentenceTransformers embeddings
- LLM: Phi3 offline (Ollama) or Gemini API online
- JSON for transcripts, CSV for query outputs

10. Usage Instructions

Install dependencies:

```
pip install -r requirements.txt
```

Offline mode:

```
ollama pull phi3
```

Online mode:

```
export GEMINI_API_KEY=<YOUR_KEY>
```

Run Streamlit app:

```
streamlit run app.py
```

Example query:

```
Why are customers escalating in healthcare calls?
```

The system returns:

- Key causes
- Evidence with transcript IDs
- Structured explanation

11. Conclusion

The Causal Conversation Analyzer moves beyond event detection to **causal analysis and interactive reasoning** over conversational data. It provides:

- Evidence-backed causal explanations grounded in transcripts
- Multi-turn reasoning with contextual memory
- Domain-level statistics for quantitative validation
- Offline and online LLM support

The system aligns with IIT Hackathon 2026 requirements for **IDRecall**, **faithfulness**, and **relevancy**, delivering a reproducible and interpretable solution for conversational analytics.

12. Repository & Ports

- **GitHub Repository:** https://github.com/balabhadra3141/pravaah2026-iit-bbsr
- **Ollama API:** http://localhost:11434
- **Streamlit App:** http://localhost:8501

Tip: Start Ollama first (offline mode) or ensure `GEMINI_API_KEY` is set (online mode) before running Streamlit.

Author: Balabhadra Padhi | Gourab Swain | Rakesh Patra
Date: February 2026
Hackathon: IIT BBSR Data Hackathon Pravaah 2026