

Team 2025107

Contents

1	Introduction	3
1.1	Project Goal	3
2	System Overview & Features	4
2.1	Data Preprocessing	4
2.2	Feature Engineering	4
3	Anomaly Detection	6
3.1	Multi-DataFrame Batch Clustering	6
3.2	Unsupervised Detection of Anomalous Employee Segments	6
3.3	Explainability via Decision Trees	7
4	Modeling	8
4.1	Key Steps	8
5	Explainability	9
5.1	Method	9
5.2	Output	9
6	Aggregation	10
6.1	Focus	10
6.2	Techniques	10
7	High-Level Design	11
7.1	Conceptual Workflow	11
8	References	12
9	Chatbot Functionality	13
9.1	Pipeline Architecture	13
9.1.1	Data Extraction and Classification (SHAP-Based)	13
9.1.2	Summary Generation and Embedding	13
9.1.3	Intelligent Question Selection	14
9.2	Conversational Chatbot Layer	14
9.2.1	LLM-Driven Dialogue System	14
9.2.2	Functional Components	14
10	Data Storage and Reporting	15
10.1	PostgreSQL Database Integration	15
10.2	Report Generation and HR Integration	15
11	Scalability and Reliability	16
12	Conclusion	17
13	API Workflow	18
14	Voice Transcription and Text-to-Speech Integration	19
15	Question Bank	20
16	Annexure	21

Project Documentation

Employee Engagement Analysis System

1. Introduction

Project Goal: The primary objective of this system is to proactively identify employees who might be disengaged or facing challenges by analyzing diverse datasets related to their work life. It aims to provide a holistic view by integrating information from various sources including performance reviews, daily activity, sentiment surveys (Vibemeter), leave patterns, onboarding experiences, and rewards data.

By applying data processing, anomaly detection, machine learning modeling, and explainability techniques, the system generates a prioritized list of employees requiring attention, along with insights into the key factors contributing to their status. The ultimate goal is to enable timely and targeted interventions to support employees.

2. System Overview & Features

The system functions as a modular pipeline, processing data through distinct stages:

Data Preprocessing: The raw data from various sources is properly structured, cleaned, and enriched before any analysis is performed in the following steps:

- **Loading:** Importing relevant datasets (leave tracker, performance tracker, etc.) as Pandas DataFrames for further processing.
- **Standardization:** Ensuring uniform formats, especially for dates which are converted to pandas datetime objects. The categorical variables are mapped with appropriate numeric scores in all datasets.

Feature Engineering: Creating new, potentially more informative features from existing data. Examples include:

- Calculating time durations (e.g., time since last login/activity, tenure).
- Aggregating metrics over time (e.g., average Vibemeter score, total leave days).
- Creating flags or categorical indicators based on thresholds or conditions (e.g., low-performance flag).

The EmployeeDataTransformer class encapsulates the logic for transforming each specific dataset to create new features and clean the data for the subsequent steps.

Function	Input Columns	Output Columns
transform.vibe.dataset	Response_Date, Vibe_Score	Average_Vibe_Score, Latest_Vibe_Score
transform.performance.data	Review_Period, Performance_Rating, Manager_Feedback, Promotion_Consideration	Average_Performance_Rating, Latest_Performance_Rating, Average_Manager_Feedback_Score, Promotion_Consideration_Ratio, Latest_Promotion_Consideration
transform.onboarding.dataset	Joining_Date, Onboarding_Feedback, Mentor_Assigned, Initial_Training_Completed	Days_since_joining, Onboarding_Feedback, Mentor_Assigned, Initial_Training_Completed
transform.rewards.data	Award_Type, Award_Date, Reward_Points	Total_Rewards_Received, Total_Reward_Points, Average_Reward_Points, Days_since_last_reward
transform.leave.data	Leave_Type, Leave_Days, Leave_Start_Date, Leave_End_Date	Sick_Leaves, Annual_Leaves, Unpaid_Leaves, Casual_Leaves, Days_since_last_leave
transform.activity.data	Date, Teams_Messages_Sent, Emails_Sent, Meetings_Attended, Work_Hours	Days_since_last_activity, Average_Team_Messages_Sent, Average_Emails_Sent, Average_Work_Hours, Latest_Work_Hours

Table 1: Transformation Functions and Their Input/Output Columns

3. Anomaly Detection

Overview: This module is designed to surface anomalous patterns in employee behavior by identifying small, distinct subgroups within multiple datasets—representing potential disengagement, performance concerns, or other atypical traits. Unlike traditional supervised approaches, this unsupervised method operates without the need for predefined labels, making it well-suited for early warning systems and exploratory diagnostics.

Multi-DataFrame Batch Clustering: The module accepts a dictionary of named pandas DataFrames, enabling batch analysis across multiple organizational units or time periods. It automatically selects numerical columns and applies feature standardization using StandardScaler. This ensures uniformity across all inputs before initiating K-Means clustering (with n_clusters = 2).

Unsupervised Detection of Anomalous Employee Segments: Each DataFrame undergoes K-Means clustering, with a key focus on identifying the smallest cluster—presumed to represent anomalous or outlier-like behavior. A binary `Should_Reach_Out` flag is added to all rows, marking employees within this minimal group. A positive value of this flag signifies potential concern in that dataset.

Explainability via Decision Trees: To explain the characteristics defining each cluster, the module trains a `DecisionTreeClassifier` on the standardized features and cluster labels. The resulting tree provides global feature importance scores and row-level explanations. These are appended to the original DataFrames along with cluster assignments.

4. Modeling

Goal: This module applies supervised machine learning to estimate engagement scores, classify employees into engagement levels, or predict behavioral outcomes. It leverages XGBoost classifiers on the transformed datasets, using the `Should_Reach_Out` flag as the target.

Key Steps:

- **Class Imbalance Handling:** ADASYN (Adaptive Synthetic Sampling) is used to balance classes when the minority class is significantly underrepresented (e.g., when one class is 1.5x the other). This helps create synthetic samples to support robust model learning.
- **Data Splitting:** Each dataset is split into 80% training and 20% testing subsets to validate model performance.
- **Model Training and Hyperparameter Tuning:** An `XGBoostClassifier` is trained per dataset. Hyperparameter tuning is done via `RandomizedSearchCV`, enabling efficient optimization without the computational cost of grid search.
- **Model Evaluation:** The performance of each tuned model is evaluated using appropriate classification metrics such as accuracy and f1 scores.
- The best-performing model for each dataset is saved for downstream inference. These models are then used to generate predictions or engagement scores for each employee, which can be utilized for decision-making and intervention planning.

Dataset	No. of samples	Accuracy	Precision	Recall	F1	ADASYN_Applied
onboarding	320	1	1	1	1	No
rewards	318	0.984375	1	0.966667	0.983051	No
vibemeter	315	0.984127	1	0.964286	0.981818	No
leave	311	0.984127	0.944444	1	0.971429	Yes
activity	312	0.968254	0.965517	0.965517	0.965517	No
performance	322	0.953846	0.909091	1	0.952381	No

Table 2: Model Evaluation Metrics across Different Datasets

Explainability

Understanding why a model makes a certain prediction is crucial for trust and actionability. This module uses the SHAP (SHapley Additive exPlanations) library, which is a unified approach to explain the output of any machine learning model. SHAP assigns each feature an importance value for a particular prediction based on cooperative game theory (Shapley values).

Method

SHAP explainers, specifically `shap.TreeExplainer` (suitable for tree-based models like XGBoost), are applied to the trained models.

Output

SHAP values are generated for each feature and each prediction. These values quantify the contribution of each feature to pushing the model's output away from a baseline value. This helps identify the key drivers behind an individual employee's predicted score or classification.

Aggregation

Since insights come from multiple datasets, anomaly detectors, and potentially multiple models, this module combines these signals into a unified view.

Focus

Primarily aggregates the feature importance scores (SHAP values) obtained from the Explainer module.

Techniques

An ensemble method is used to combine feature rankings from various sources. This involves the following rank aggregation methods (inspired by techniques from [1]):

- **Borda Count Aggregation:** Assigns points based on positions in rankings.
- **Geometric Mean Aggregation:** Combines ranks by computing the geometric mean of their SHAP values.
- **Rank Position Aggregation:** Averages the actual ranks across rankings.
- **Robust Rank Aggregation (RRA):** Statistically models rankings to identify consistently high-ranking features.

Finally, a meta feature list that combines the feature rank lists across various datasets is created for each employee.

High-Level Design

Conceptual Workflow

1. **Ingestion:** Load diverse employee datasets.
2. **Transformation:** Clean, standardize, and engineer features using the Preprocessor.
3. **Detection:** Identify anomalies and rule-based flags using the Detector.
4. **Prediction:** Train and apply machine learning models (Models) to predict engagement-related outcomes.
5. **Interpretation:** Generate feature importance scores using SHAP (Explainer).
6. **Synthesis:** Combine anomaly flags, model predictions, and feature importance scores using rank aggregation (Aggregator).
7. **Output:** Generate final reports (e.g., `final_results_aggregated.csv`).

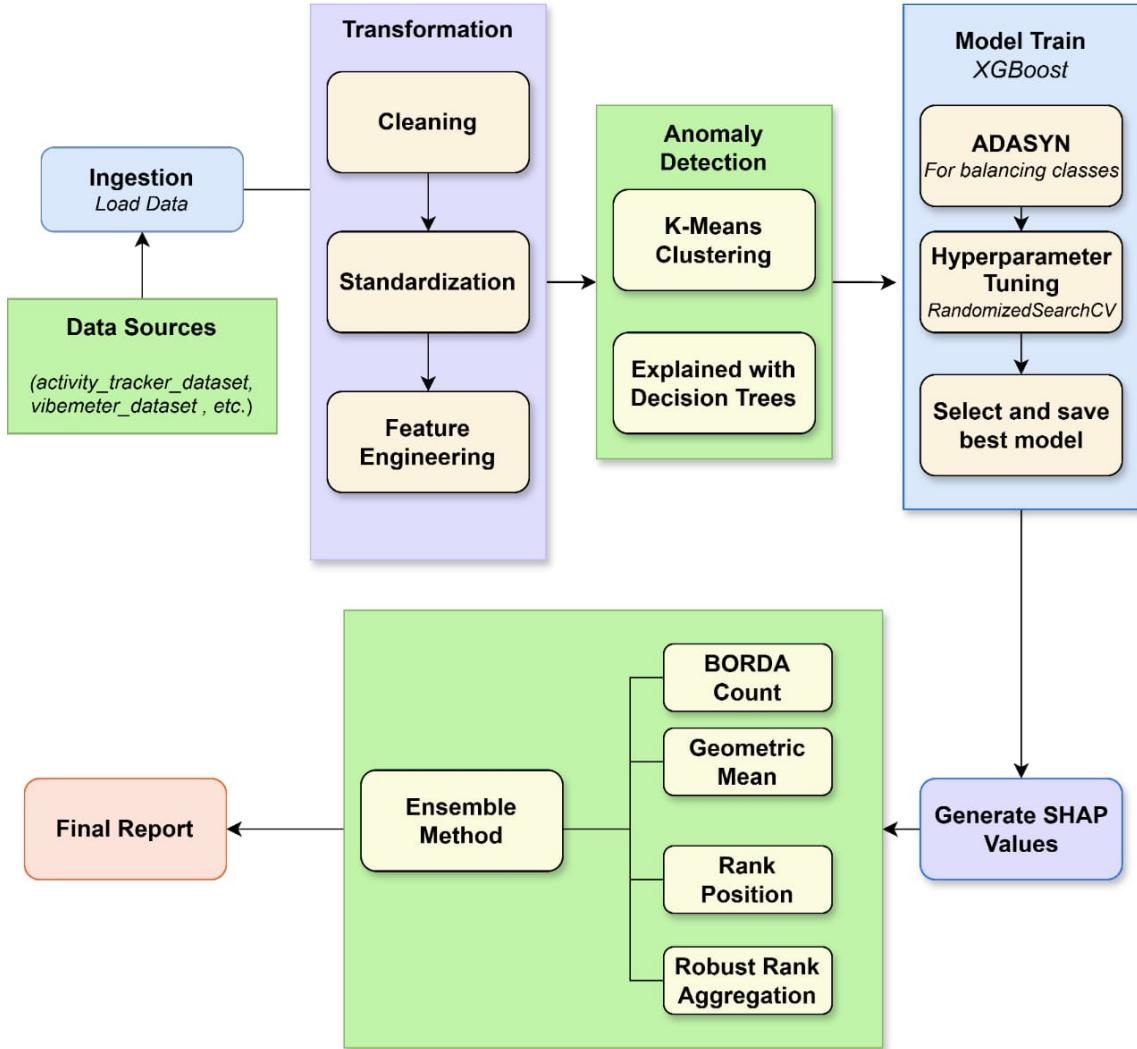


Figure 1: Overflow

Chatbot Functionality

The chatbot enables intelligent, context-aware conversations with employees by combining structured data and conversational AI. It uses sentence embeddings to recommend personalized questions based on each employee's profile and dynamically adapts its flow with follow-ups, sentiment tracking, and contradiction detection. All conversations are securely stored in a PostgreSQL database for reporting. A transformer-based sentiment model flags emotional concerns early, ensuring timely and data-driven engagement support.

Pipeline Architecture

Data Extraction and Classification (SHAP-Based Feature Interpretation)

- **Input:** SHAP feature importance values generated from an employee performance or engagement prediction model.
- **Process:** Each feature is compared against the global mean. Based on whether it is higher or lower than the mean, it is categorized as either positive (strength) or negative (concern).
- **Output:** A structured JSON object representing the categorized features for a specific employee

Summary Generation and Embedding

- **Input:** JSON file from the SHAP classification step.
- **Process:** A natural language summary is generated using a language model that concisely describes the employee's strengths and challenges.
- **Embedding:** The summary is transformed into a vector using the sentence-transformers/all-MiniLM-L6-v2 embedding model.
- **Output:** A high-dimensional contextual embedding vector representing the employee profile.

Intelligent Question Selection

- **Input:** A JSON-based question bank containing categorized objective and descriptive questions aligned with SHAP features.
- **Transformation:** Questions are flattened into an array of strings and embedded using the same all-MiniLM-L6-v2 embedding model.

- **Similarity Search:** Using cosine similarity, the top 10 most relevant questions are retrieved by comparing their embeddings with the employee summary embedding.
- **Output:** A curated list of 10 personalized questions that guide the chatbot's conversation.

Conversational Chatbot Layer

LLM-Driven Dialogue System

- **Model:** The GPT-4.0 model is accessed via OpenAI API to manage conversations.
- **System Prompts:** Predefined prompts structure the behavior of the LLM to ensure conversations are goal-oriented, empathetic, and guided by relevant questions.
- **Memory Management:** A buffer memory mechanism stores ongoing conversation history, enabling context awareness, follow-ups, and contradiction detection.

Functional Components

`generate_follow_up()`

- **Purpose:** Deepens understanding of employee responses.
- **Mechanism:**
 - Randomly selects one of the 10 relevant questions.
 - Based on the user's answer, it generates two targeted follow-up questions using GPT-4.0.
- **Goal:** Extract context-rich insights that support sentiment analysis and HR decision-making.

`detect_contradiction()`

- **Purpose:** Identifies logical or emotional inconsistencies in user responses over time.
- **Mechanism:**
 - Compares recent and past responses stored in memory.
 - Uses GPT-4.0 to flag potential contradictions.
 - Generates clarifying follow-up questions to resolve discrepancies.
- **Outcome:** Ensures conversational integrity and a deeper understanding of employee sentiment.

Data Storage and Reporting

PostgreSQL Database Integration

- **Purpose:** Long-term storage and analysis of all employee interaction data.
- **Stored Data Includes:**
 - Full conversation logs.
 - Sentiment scores across emotional dimensions.
 - Generated summaries and question sets.
 - Contradiction detection flags and follow-up logs.

Report Generation and HR Integration

Sentiment Analysis:

- Conducted using the `bhadresh-savani/distilbert-base-uncased-emotion` transformer model.
- Extracts sentiment across dimensions such as joy, anger, sadness, fear, surprise, and love.

Threshold Detection:

- Aggregated sentiment scores are compared to learned thresholds.
- Employees with scores indicating potential concern are automatically flagged for HR review.

Reporting:

- Structured employee reports are generated from the stored data.
- Reports include performance insights, emotional well-being trends, engagement patterns, and feedback history.
- These reports support data-driven HR interventions and ongoing employee support.

Scalability and Reliability

- Embedding and LLM inference are modular and can be parallelized for large-scale deployment.
- PostgreSQL provides scalable, ACID-compliant storage ensuring reliable querying, indexing, and data integrity.
- Security measures such as encrypted storage and role-based access control ensure data privacy and compliance.

Conclusion

This pipeline combines explainable AI, intelligent conversation design, and robust data infrastructure to create a dynamic employee engagement system. It allows organizations to proactively identify concerns, drive personalized dialogue, and generate actionable insights for HR — ultimately enhancing workplace well-being and retention.

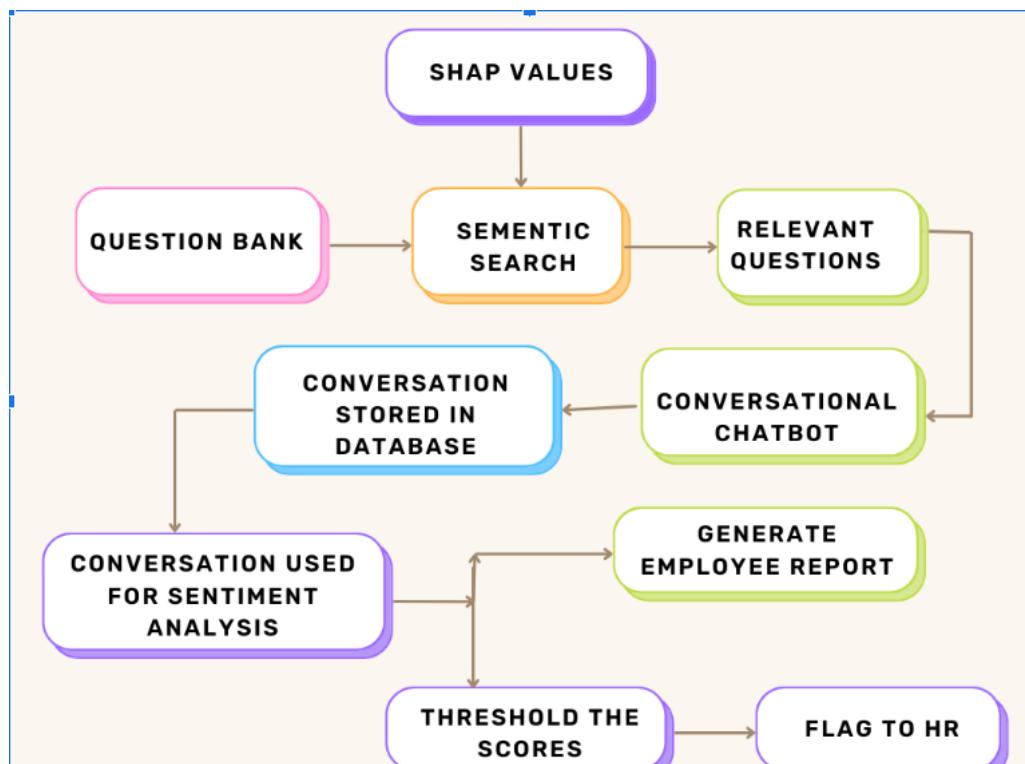


Figure 2: Employee Engagement System Architecture

System Workflow

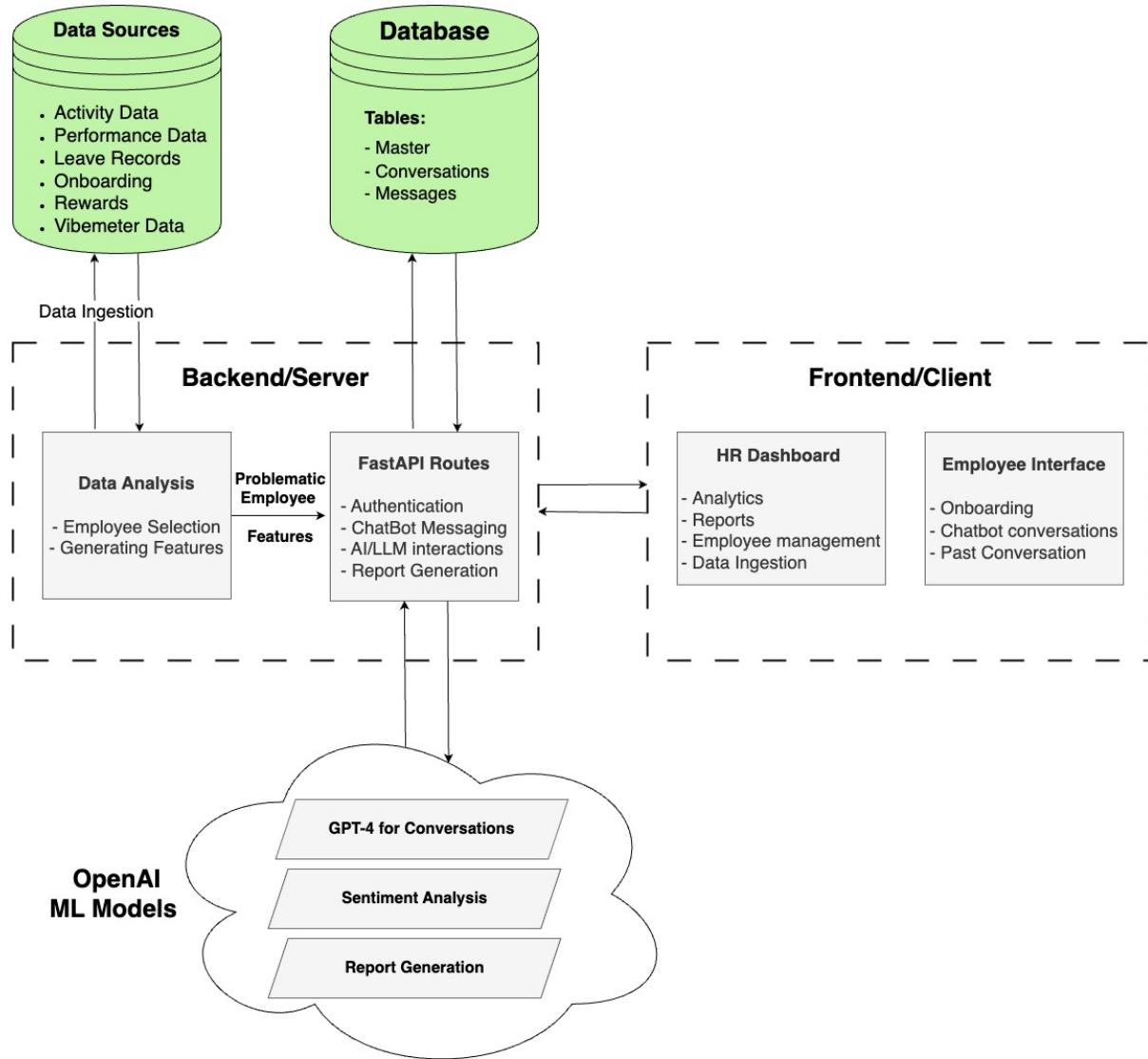


Figure 3: System Flow

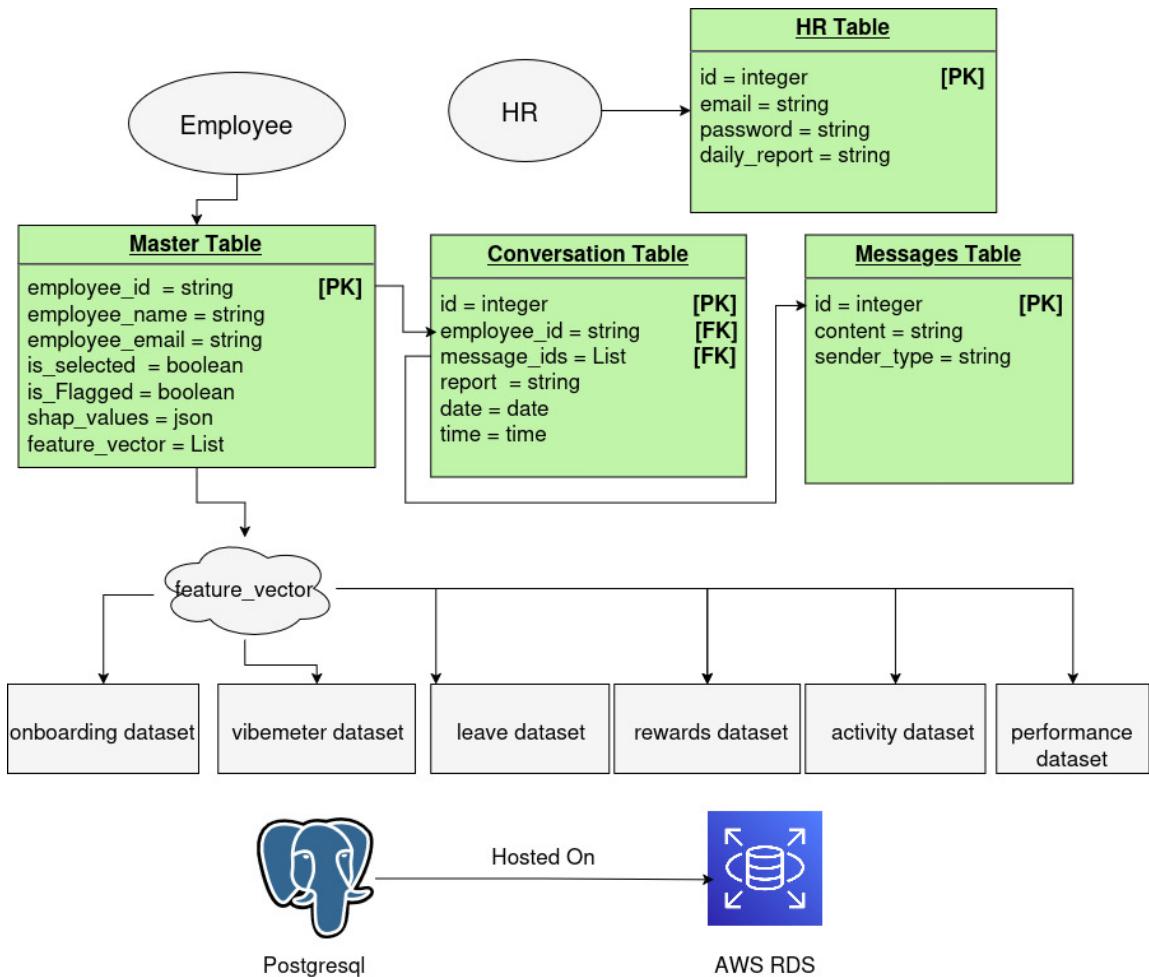


Figure 4: ER Diagram of the Database

API Workflow

Endpoint	Method	Request Body	Response Body	Description
data/ingest	POST	FormData with file and table	{"message": "string", "records_added": integer}	Ingests CSV data for a specific table.
data/ingest_shap	POST	FormData with file	{"message": "string", "updated_count": integer}	Processes SHAP values from CSV.
data/employees	GET	N/A	{"employees": [{"Employee_ID": "string", "Employee_Name": "string", ...}]}]	Retrieves all employee records.

Table 3: Data Ingestion and Retrieval Endpoints

Endpoint	Method	Request Body	Response Body	Description
conversation/start	POST	N/A	{ "conversation_id": integer, "welcome_message": "string" }	Initiates a new conversation.
conversation/message	POST	{ "message": "string", ... }	{ "response": "string", "message_type": "string" }	Processes messages.
conversation/history	GET	N/A	{ "conversations": [{ "id": integer, ... }] }	Retrieves conversation history.

Table 4: Conversation Management API Endpoints

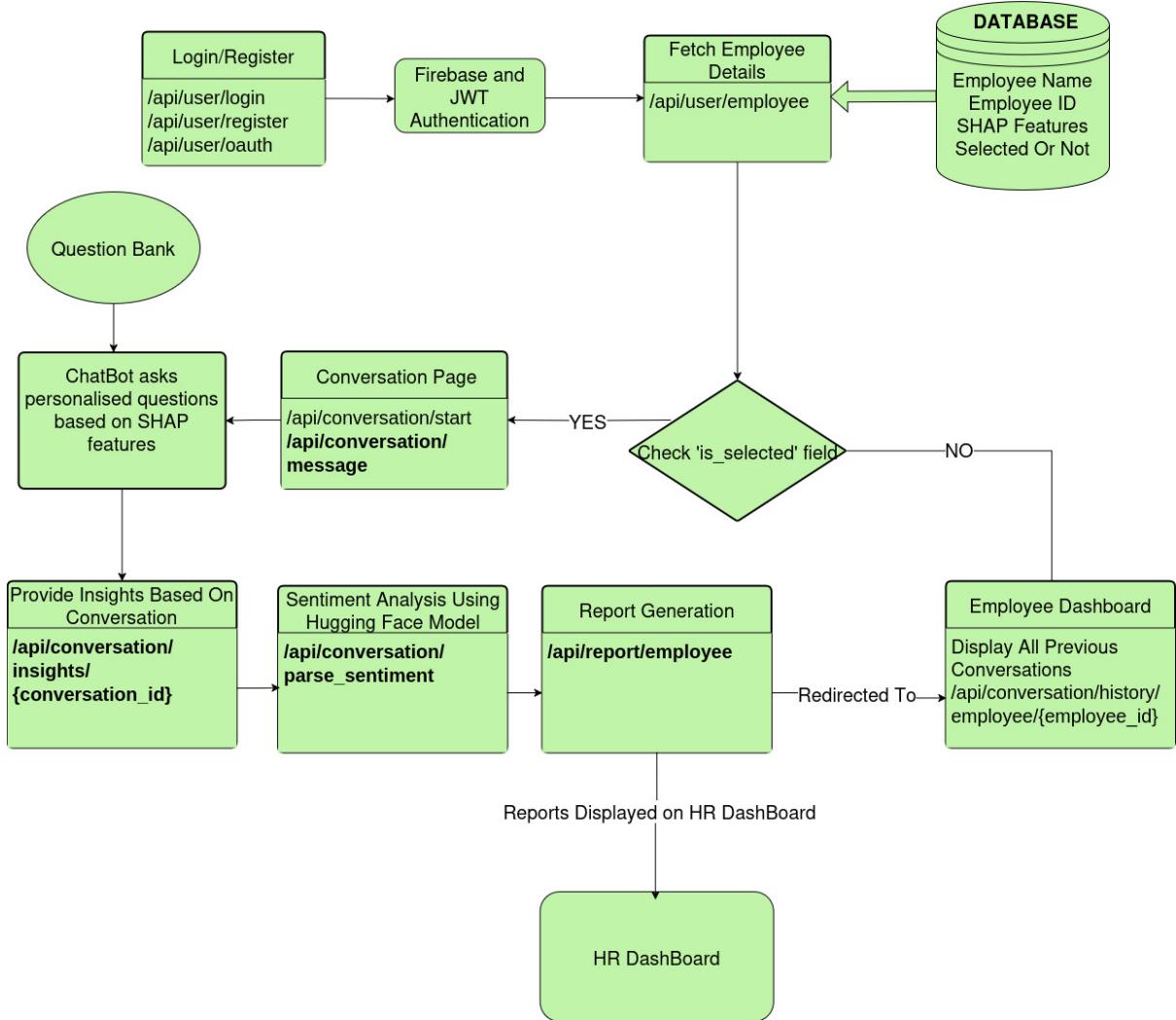


Figure 5: Overall Workflow With APIs

Voice Transcription and Text-to-Speech Integration

Objective: Enable seamless voice-based interaction within the chatbot using Deepgram's transcription and synthesis APIs. This adds natural conversation flow and accessibility for diverse users.

Voice Transcription (Speech-to-Text):

- **Input:** Voice audio from the user, typically in `audio/wav` format, captured through the web interface.
- **Processing:** Audio is streamed in real-time to Deepgram's `nova-2` model using a combination of `httpx` and `websockets` for low-latency, high-accuracy transcription.
- **Features:** Automatic punctuation, casing, and numeral formatting improve transcript readability and structure.
- **Output:** A clean JSON response containing the transcribed text, which is then passed to the GPT model for response generation.
- **Fallback:** If audio quality is poor or transcription fails, the system gracefully defaults to manual text input.

Text-to-Speech (Speech Synthesis):

- **Input:** Text response generated by the GPT model.
- **Model:** Deepgram's `aura-asteria-en` neural voice synthesizer is used to convert text into natural-sounding speech.
- **Output:** Audio in `.mp3` format, ready to be streamed directly to the user or downloaded for offline playback.
- **Reliability:** If synthesis fails, the chatbot delivers the text response directly to maintain conversational continuity.

Key Benefits:

- **Accessibility:** Facilitates use by employees with visual impairments or reading difficulties.
- **Engagement:** Adds emotional tone and human-like nuance to chatbot interactions.
- **Scalability:** Fully modular and can be integrated with both web and mobile frontends.

Question Bank

```
question_bank = {
    "Average_Vibe_Score": {
        "description": "Explores how employees perceive their overall emotional well-being and satisfaction at work.",
        "questions": [
            "How do you think your overall mood compares to your general sense of well-being at work?",
            "How do you feel about the trend in your mood at work-has it been improving or declining?",
            "Does your level of mood align with your overall job satisfaction? Why or why not?",
            "What aspects of your job contribute most to your mood, either positively or negatively?",
            "Do you think your mood at work affects how engaged or productive you feel?"
        ]
    },
    "Latest_Vibe_Score": {
        "description": "Focuses on current mood and short-term factors influencing emotional state at work.",
        "questions": [
            "Does your current mood at work match how you generally feel during the day?",
            "What are some factors you believe have contributed to your mood recently?",
            "Have you noticed any patterns in your mood based on workload or deadlines?",
            "Do you feel more supported by your colleagues when your mood is better?",
            "Would you say your mood accurately represents your day to day experience at work?"
        ]
    },
    "Total_Rewards_Received": {
        "description": "Evaluates employee satisfaction with recognition and rewards for contributions.",
        "questions": [
            "Do you feel adequately rewarded for your contributions, considering your Total Rewards Received?",
            "How do you perceive the balance between the rewards you have received and the effort you've put in?",
            "What type of rewards or recognition do you find most meaningful?",
            "Do you feel that the reward system is fair and transparent across the organization?",
            "Have you ever felt that your contributions deserved more recognition than you received?"
        ]
    }
}
```

Figure 6: Question Bank Part 1

```
"Sick_Leaves": {
    "description": "Explores employee comfort and organizational support around health-related absences.",
    "questions": [
        "Do you believe your current leave allowance supports a healthy work-life balance?",
        "Have you ever hesitated to take a leave due to workload or team pressure?",
        "Do you feel that taking regular breaks helps you maintain long-term motivation at work?",
        "Do you feel comfortable taking leave when you need it, or do you feel guilty about it?"
    ]
},
"Annual_Leaves": {
    "description": "Assesses perceptions of annual leave policy and its effectiveness in supporting well-being.",
    "questions": [
        "How do you feel about the frequency and availability of your leave options?",
        "Do you feel that the current leave policy supports your personal and professional needs adequately?",
        "How do you usually decide when to take a leave, based on workload, personal needs, or other factors?",
        "Does the process of requesting and approving leave feel smooth and fair to you?",
        "Have you ever felt the need for additional leave types that aren't currently available?"
    ]
},
"Casual_Leaves": {
    "description": "Focuses on usage patterns, flexibility, and impact of casual leave on work-life balance.",
    "questions": [
        "Is there a specific type of leave that you tend to use more frequently, and why?",
        "Would you take more leave if you felt it wouldn't impact your career progression?",
        "How does taking leave affect your workload when you return? Do you feel overwhelmed catching up?",
        "Do you believe your company's leave policy is competitive compared to industry standards?",
        "If you could improve one aspect of the leave policy, what would it be?"
    ]
},
"Unpaid_Leaves": {
    "description": "Evaluates how financial and policy factors influence the use of unpaid leave.",
    "questions": [
        "Would you take more unpaid leaves if financial constraints weren't a factor?",
        "Have you ever had to work while on leave due to urgent tasks? If so, how did it impact you?",
        "Do you feel comfortable requesting unpaid leave when needed?"
    ]
}
```

Figure 7: Question Bank Part 2

Annexure

References

- Lin, S. (2010). Rank aggregation methods. *WIREs Computational Statistics*, 2(5), 555–570.
- Moshkovitz, Michal, et al. "Explainable k-means and k-medians clustering." *International Conference on Machine Learning*. PMLR, 2020.
- He, H., et al. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. *2008 IEEE International Joint Conference on Neural Networks*, pp. 1322–1328.
- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference*, pp. 785–794.
- Sharma, P., et al. (2020). Evaluating Tree Explanation Methods for Anomaly Reasoning: A Case Study of SHAP TreeExplainer and TreeInterpreter. *arXiv preprint arXiv:2010.06734*.

System Specifications

Environment & Dependencies

- **Language:** Python 3.x
- **Core Libraries:**
 - Pandas – for data manipulation
 - NumPy – for numerical operations
 - Scikit-Learn – for preprocessing, modeling, and evaluation
 - XGBoost – for model training
 - SHAP – for explainability
 - JSON – for configuration and data handling
- **Specific Versions Used:**
 - imblearn==0.0
 - matplotlib==3.8.2
 - numpy==2.2.4
 - pandas==2.2.3

- scikit_learn==1.3.2
- scipy==1.15.2
- seaborn==0.13.2
- xgboost==3.0.0
- shap

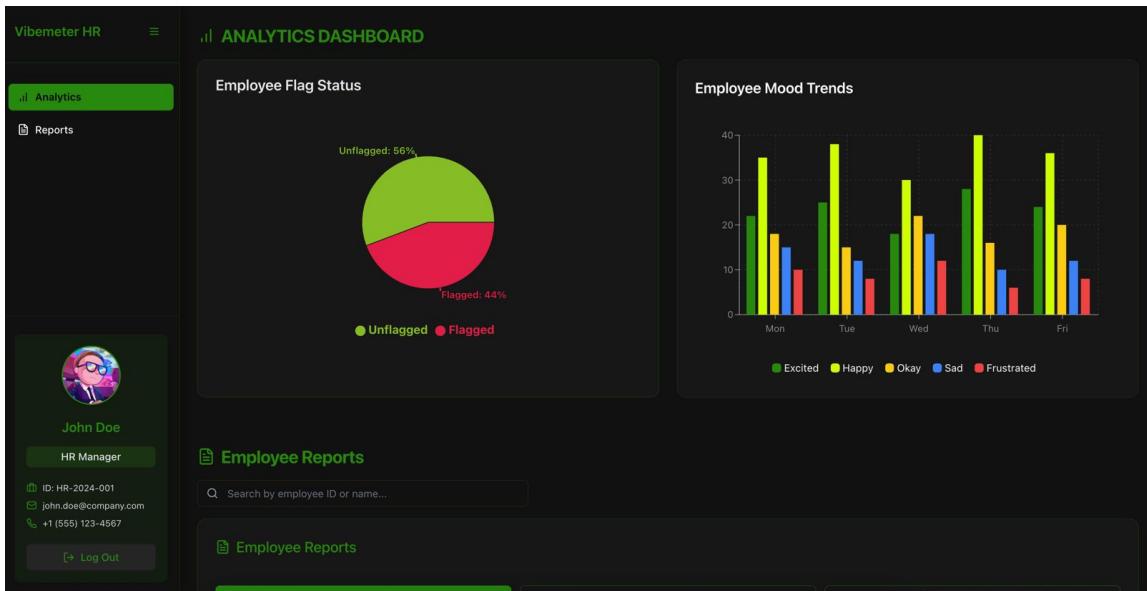
Input Data (Datasets/ folder)

- activity_tracker_dataset.csv
- leave_dataset.csv
- onboarding_dataset.csv
- performance_dataset.csv
- rewards_dataset.csv
- vibemeter_dataset.csv

Output Data (Output_files/ folder)

- feature_thresholds.csv
- final_results_aggregated.csv
- models_summary.csv

HR Dashboard



Chatting Section

Deloitte Well-being Check-in

Your responses are confidential and will help us provide better support.

DC: Hello Random! This is just a quick check-in to see how you're doing today. I hope all is well!!
11:58 pm

ME: Hi, I am fine.
02:14 am

DC: Do you feel that the current leave system supports proactive leave-taking rather than reactive leave-taking?
02:14 am

ME: I feel the current system supports reactive leave-taking
02:15 am

DC: What do you think could be done to shift from a reactive to a more proactive approach in terms of leave-taking policies?
02:15 am

Chat-Bot Conversation Function

```

def chatbot_conversation(shap_values, chat_history, user_response, message_type, question_set):
    """Handles the chatbot conversation logic."""

    try:
        if message_type == "welcome":
            first_shap_value = next(iter(shap_values))
            current_shap_questions = question_bank[first_shap_value].get(
                "questions", [])
            current_question = random.choice(current_shap_questions)
            asked_questions.add(current_question)
            return current_question, "normal_question"

        if message_type == "followup_1" or message_type == "normal_question":
            print(f"User Response1: {user_response}")
            contradiction_follow_up = detect_contradiction(
                user_response, chat_history)

            if contradiction_follow_up != "" and contradiction_follow_up != None and contradiction_follow_up != "None.":

                asked_questions.add(contradiction_follow_up)
                if message_type == "normal_question":
                    return contradiction_follow_up, "followup_1"
                elif message_type == "followup_1":
                    return contradiction_follow_up, "followup_2"
            else:
                print(f"User Response2: {user_response}")
                follow_up_question = generate_follow_up(user_response)
                if message_type == "normal_question":
                    return follow_up_question, "followup_1"
                elif message_type == "followup_1":
                    return follow_up_question, "followup_2"

        next_question = select_next_question(chat_history, question_set)
        asked_questions.add(next_question)
        # chat_history.append(AMessage(content=next_question))
        return next_question, "normal_question"
    
```

Text to Speech Function

```

headers = {
    "Authorization": f"Token {DEEPCGRAM_API_KEY}",
    "Content-Type": "application/json"
}
# Make the request to Deepgram
async with httpx.AsyncClient() as client:
    response = await client.post(
        f"https://api.deepgram.com/v1/speak?model=aura-asteria-en&gender=female",
        headers=headers,
        json=payload,
        timeout=30.0
    )

    # Check for successful response
    if response.status_code != 200:
        error_message = f"Deepgram TTS API error: {response.status_code}"
        try:
            error_detail = response.json()
            error_message += f" - {error_detail}"
        except:
            pass

        raise HTTPException(
            status_code=response.status_code,
            detail=error_message
        )

    # # Return the audio stream
    return StreamingResponse(
        io.BytesIO(response.content),
        media_type="audio/mp3",
        headers={
            "Content-Disposition": f"attachment; filename=speech.mp3"
        }
    )

```

Transcribe (Speech to Text) Function

```
@router.post("/transcribe")
async def transcribe_audio(audio: UploadFile = File(...)):
    """Convert speech audio to text using Deepgram"""
    if not DEEPGRAM_API_KEY:
        raise HTTPException(status_code=500, detail="Deepgram API key not configured")

    try:
        audio_content = await audio.read()
        # print("Audio content length:", len(audio_content))
        # Send to Deepgram
        async with httpx.AsyncClient() as client:
            response = await client.post(
                "https://api.deepgram.com/v1/listen?model=nova-2&smart_format=true",
                headers={
                    "Authorization": f"Token {DEEPGRAM_API_KEY}",
                    "Content-Type": "audio/wav" # Adjust based on your audio format
                },
                content=audio_content
            )

            if response.status_code != 200:
                raise HTTPException(status_code=response.status_code, detail="Failed to transcribe audio")

            result = response.json()

            transcript = result["results"]["channels"][0]["alternatives"][0]["transcript"]
            # print("Transcript:", transcript)

            return {"transcript": transcript}
    except httpx.RequestError as e:
        raise HTTPException(status_code=500, detail=f"HTTP error: {str(e)}")
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error in speech-to-text: {str(e)}")
```