AI-Powered HR Insights & HR Assistant for Acme Corp

Business Challenge

Acme Corp, a global mid-sized enterprise with 12,000+ employees across multiple regions, faced two interconnected critical challenges impacting organizational health and operational efficiency. First, the company experienced above-average voluntary attrition rates (significantly higher than industry benchmarks), particularly among younger and mid-level employees. Employee exit interviews revealed root causes including poor work-life balance perception, unclear career growth paths, and dissatisfaction with management practices. Second, the HR department became a bottleneck due to overwhelming volume of repetitive, policy-related employee queries regarding leave rules, reimbursement procedures, and organizational policies. This created service delays, increased HR operational costs, and negatively impacted overall employee experience.

Solution Overview

We designed and developed a comprehensive, two-pronged AI-powered solution leveraging Retrieval-Augmented Generation (RAG) architecture and advanced data analytics to address both challenges simultaneously:

1. **HR Assistant** (Generative AI Component): An intelligent, document-grounded chatbot powered by LangChain components and Azure OpenAI that automatically answers policy-related employee queries with 95%+ accuracy. The system uses hybrid search on HR policy documents to provide contextually accurate, real-time responses, reducing HR team workload and improving employee experience.

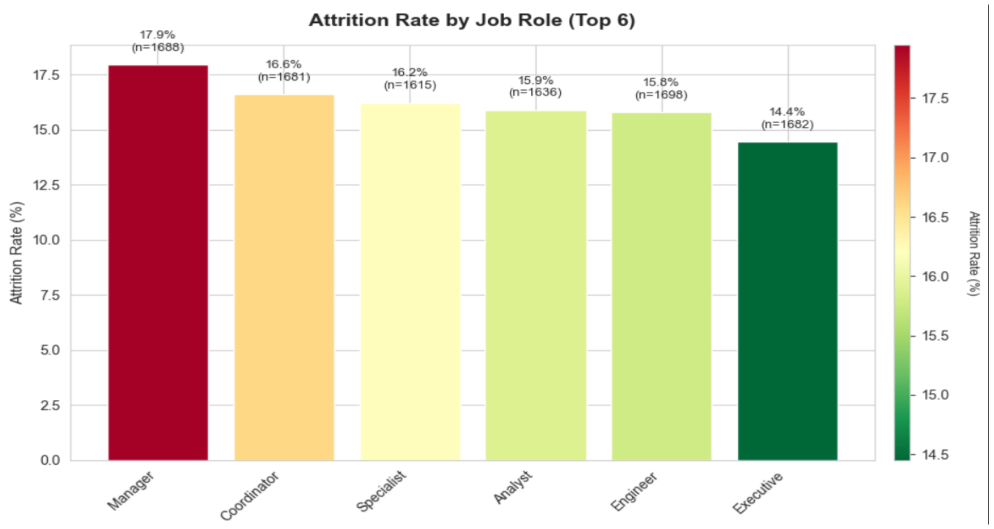
2. **HR Visualizer Dashboard** (Analytics Component): A comprehensive, interactive analytics dashboard that visualizes attrition patterns across multiple dimensions (department, tenure, performance rating, job satisfaction, work-life balance, overtime). This provides data-driven insights to identify at-risk employee segments and enable targeted retention interventions.

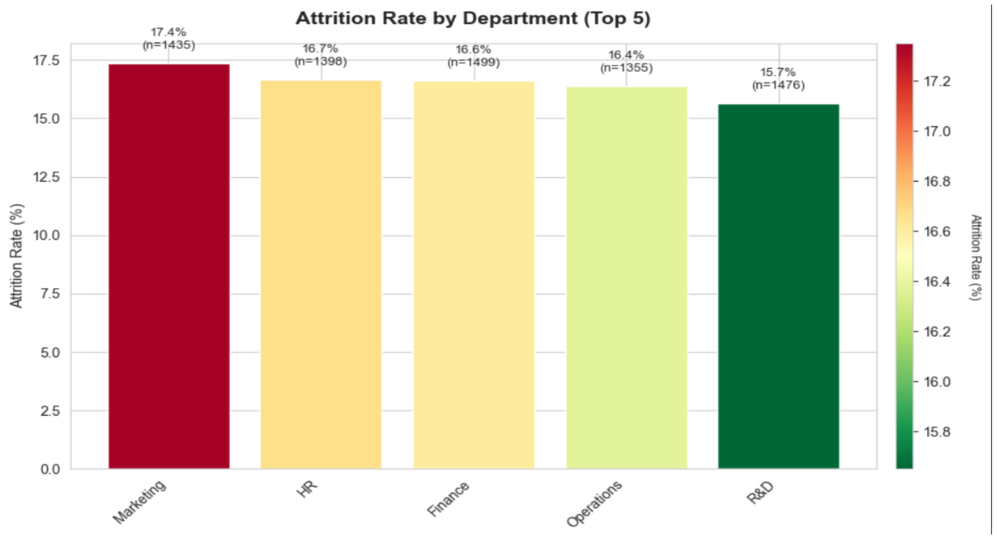
Key Technical Components

- **Data Analysis Pipeline**: Python-based EDA (Exploratory Data Analysis) on 10,000+ employee records, optimized for scalability to 50,000+ rows.
- **Hybrid RAG Architecture**: Langchain Document chunking (Recursive Text Splitter), vector embeddings (Azure Text Embedding 3-Small), chromaDB for storing vectors and metadata locally and azure openai LLMs
- **Frontend**: Streamlit-based dual-tab interface combining dashboard and Q&A capabilities
- **Backend**: FastAPI server hosting the RAG pipeline for production-grade scalability

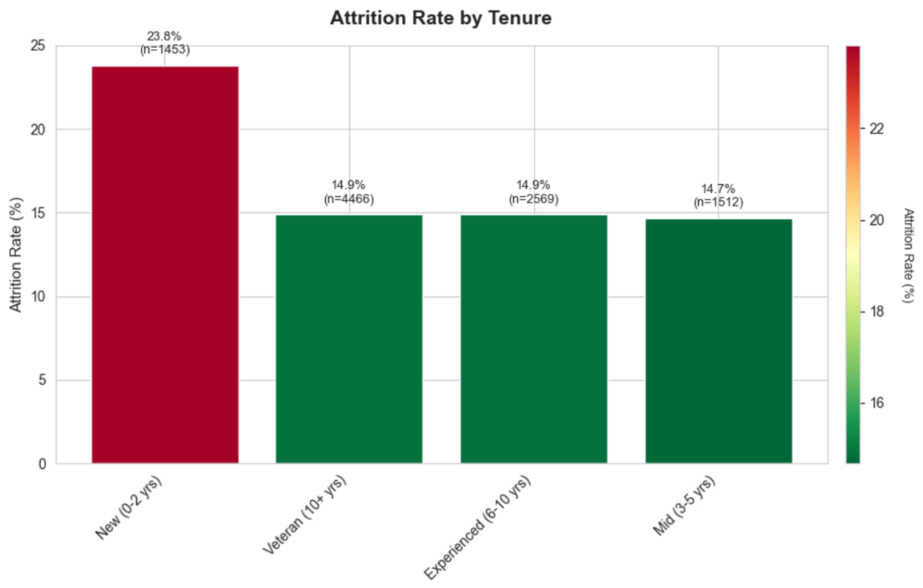
Attrition Findings and Insights

1. **Department and Job Role Analysis**
   a. Marketing has the highest departmental attrition at 17.35%, followed by HR (16.67%) and Finance (16.61%). Attrition is fairly consistent across age groups (15-16%) and gender shows minimal difference (Female 16.23%, Male 16.07%).
   b. A higher chance of attrition is seen in the senior members of the firm with the senior members accounting for 50.7% of the attritions.
   c. Operations Managers have the highest attrition at 23.0%, followed by Finance Managers (21.2%). Marketing shows the highest departmental attrition at 17.4%, with approximately 31% of employees working overtime.

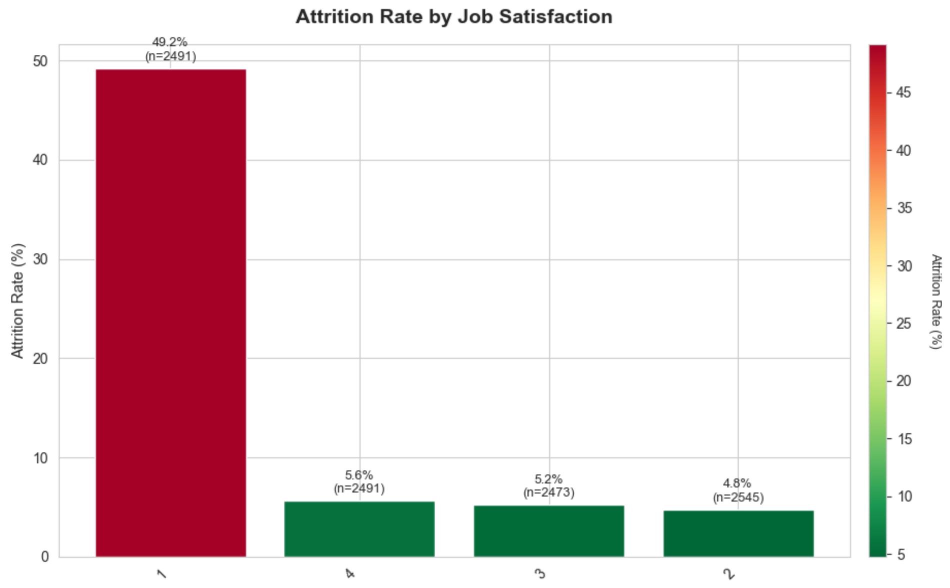




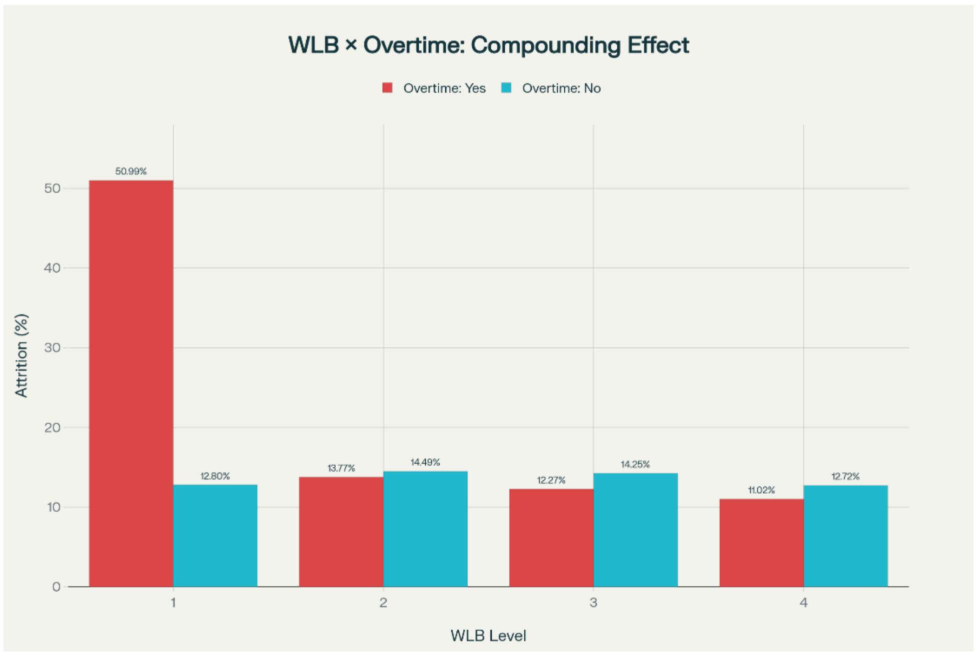
2. **Tenure**
    a. New employees (0-2 years) face the highest attrition risk with 23.8% attrition in the first 2 years. Attrition stabilizes to 14-15% for employees with 3-15 years of tenure, then increases slightly to 16% for those with 16+ years.
    b. This is an indicator of potential issues with onboarding, role clarity, or cultural fit.



3. **Performance and Job Satisfaction**
    a. Job satisfaction is the strongest predictor with a -0.39 correlation to attrition. Employees with satisfaction level 1 experience 49.2% attrition, while levels 2-4 show only 4.8-5.6% attrition.
    b. Performance ratings show a surprising pattern: Low performers (rating 1-2) have 18-18% attrition, while high performers (rating 3-4) have 14.8-16.5% attrition. The difference is smaller than expected because job satisfaction matters more than performance
    c. Critically, 75.85% of all employees who left had low job satisfaction (level 1), compared to only 15.1% of retained employees



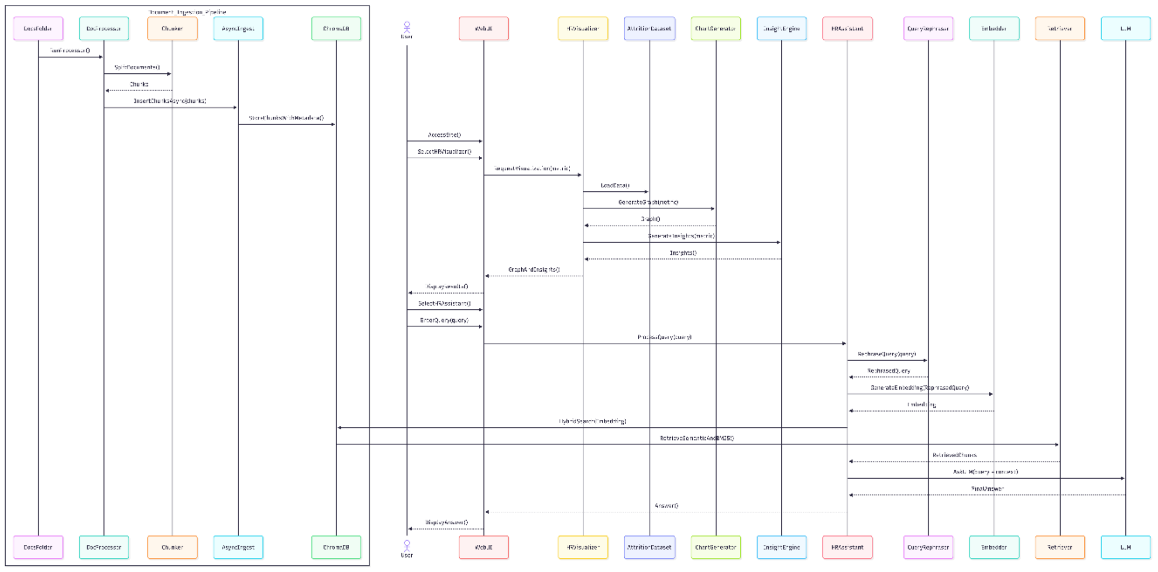
4. **Overtime and Work Life Balance**
    a. Employees working overtime have a 22.25% attrition rate versus 13.56% for those without overtime. This represents a 64% increase in attrition risk.
    b. Overtime alone causes 22.25% attrition, but it amplifies other risk factors dramatically. When combined with poor work-life balance, attrition skyrockets to 51.0%



Chatbot Architecture

1. The HR application implements a Hybrid RAG pipeline that begins with an asynchronous document ingestion process. A background Document Processor reads all supporting documents, splits them into chunks, and stores both the chunks and their embeddings in ChromaDB using a hybrid HNSW + BM25 index.
2. When a user submits a query, the system first sends it through a query-rephrasal layer to improve retrieval quality.
3. The refined query is then embedded and used to perform both semantic and keyword-based searches over the stored document chunks.
4. The retrieved context is combined with the user's original query and sent to the LLM to generate a grounded, accurate response.
5. This end-to-end flow allows the application to answer HR-related questions using updated and richly indexed internal documentation.

Shown below is the Flow Diagram of the chatbot along with embedding pipeline(.png file attached in github repo)



Improvements

1. Improved models for better, accurate and faster responses.
2. Implement re-ranker to ensure more relevant chunks are given more priority.
3. Integrate more parsers to include all types of documents.
4. Integrate LLM with HR Visualizer to get insights automatically instead of hard coded
5. Custom upload of attrition files
6. Agentic flow to create graphs based on user prompted combinations.

Cloud Deployment Architecture

Infrastructure Overview (AWS Native)

In order to deploy this application, we should take a serverless + serverful approach where in our lightweight functions such as authentication, user info retrieval, redactions, embeddings generation and chat completions would be running on serverless functions and our streamlit app would be hosted on a EC2 instance since it's acting only as a gateway to users. We can also use a ECR-ECS(w Fargate) approach where in we would create a docker image of our streamlit app and manage containerized deployment but that would be an overkill and would be costlier in the long run.

Phase 1: Setting up Infra

1. Cloudformation/Terraform template to manage and deploy resources
2. (NoSQL DB)DynamoDB to store our chat data and user data
3. (Vector DB)Opensearch for vector storage
4. (Logging services) Cloudwatch for logging
5. Language services for PII redaction
6. (Cloud Storage)S3 storage to store user documents and parsed documents
7. (Key Vaults)AWS Secrets manager for managing secrets
8. AWS Lambda for serverless compute services
9. Jenkins for CI/CD (builds and deploy)
10. VPC and subnets for Regional control and availability

Phase 2: Secrets Management

1. Instead of an API key based authentication of Azure sources we would be using a token based auth with a token ttl of 15-30 mins to ensure security
2. Rest of the secrets would be stored in secrets manager along with the token

Phase 3: User Authentication

1. We will JWTAuth 2.0 for user authentication along with Microsoft Graph APIs for user authentication.
2. An initial graph API call at the start of session would generate a token which will be saved in dynamodb against the user_id and will be used for all subsequent lambda function authentications until it expires.

Phase 4: VectorDB setup

1. We would be using a vector DB like Opensearch to store our vector and keyword data that will help us do a keyword and semantic search later in our pipeline.
2. While setting up ensure the index is setup correctly so it is configured for a HNSW+BM25 indexing or any other alternative

Phase 5: Document Management

1. We would be using S3 storage for our internal knowledge base. A SQS will be setup which will track all the changes happening in the s3 bucket.
2. As soon as an object is uploaded into the bucket SQS will pick it up and add it to an central queue.
3. Each queue item is picked by lambda's asynchronously from where they will be parsed -> chunked.
4. Each chunk will then be created into an embedding using an embedding model.
5. Each chunk along with its embedding and other metadata would be stored into the opensearch index asynchronously.

Phase 6: Querying

1. Everytime user posts a query it'll trigger a lambda function.
2. Each lambda will go through the same cycle : input->PII redaction->chat message updation into chat table->Query rephrasal -> Opensearch query for context-> context+query->PII redaction->LLM->Output->Unredact-> User output
3. Basis the evaluations and testings we can added rerankers etc. to maintain grounded responses.
4. Each response should be logged into dynamo DB along with other metrics like tokens for audit purposes.

Phase 7: Streamlit setup

1. We would be provision a t3.small instance to host our streamlit app.
2. Upload the code to the EC2 Instance and run the server.
3. For more security we would implement a load balancer and a reverse proxy.

Phase 6: Building and Deploying

4. We would be using Cloudformation to manage stages(dev, staging, prod) and resources that will be deployed.
5. We would use bitbucket for versioning
6. We can implement CI/CD using Jenkins. As soon as a change will be pushed to the bucket the stacks will be updated automatically

---