Team 6 Graduation Project: Project Documentation

Created by:
Ahmed Ayman
Dina Magdy
Roba Hesham

Track: AI & Machine Learning

Submitted for Konecta

Automated Document Processing Use Case:

1. Introduction and Project Objective

The objective of this module was to design and implement a scalable, intelligent data pipeline capable of automatically processing and classifying critical business documents, specifically In- voices, Purchase Orders (POs), and Approvals. The solution integrates state-of-the-art Large Language Models (LLMs) for classification and data extraction, robust Optical Character Recog- nition (OCR) for handling diverse document formats (scanned images and PDFs), and a persis- tent NoSQL database (MongoDB) for structured storage and data integrity.

2. System Architecture and Technologies

The solution comprises a backend API service (Flask), a dedicated user interface (Streamlit), and a containerized deployment setup (Docker), utilizing a modern Python ecosystem.

1.1 Core Components

- Backend API (Flask): Manages file uploads, data processing, and database interactions. Configured with file size limits (16MB) and secure handling of uploads.
- Frontend Application (Streamlit): Provides an intuitive web interface for document sub- mission and visualization of extracted data.
- Deployment (Docker): Ensures environment consistency across development and pro- duction, bundling the Python runtime, dependencies, and necessary system tools like tesseract-ocr.
- Database (MongoDB): Used for permanent storage, categorized into three distinct col- lections (invoices, purchase_orders, approvals) for efficient querying.

1.2 Technology Stack

- LLM Integration: ChatGoogleGenerativeAI (Gemini-2.5-Flash) via LangChain is utilized for complex tasks: document classification and structured information extraction (JSON out- put).
- Text Extraction/OCR: PyTesseract handles text extraction from image formats (PNG, JPG), while pdfplumber is used for reading text directly from PDF files, ensuring wide document compatibility.

- Data Embedding: The SentenceTransformer (paraphrase-multilingual-MiniLM-L12-v2) generates vector embeddings for the extracted text, enabling future semantic search ca- pabilities and enhancing duplicate detection.

3. Automated Processing Pipeline

The document processing follows a five-step pipeline designed for robustness and high through- put.

1. File Validation and Text Extraction: The API receives a file (PDF or image). It checks the file type and uses the appropriate tool (pdfplumber or PyTesseract) to convert the docu- ment content into raw, searchable text.
2. Document Classification: The raw text is passed to the Gemini LLM with a specific sys- tem prompt to classify the document into one of the three target categories: invoice, purchase_order, or approval.
3. Structured Data Extraction: A second LLM call is made, leveraging the model's ability to structure information. It extracts key-value pairs relevant to the document type (e.g., invoice number, vendor, total amount) and formats them as JSON.
4. Embedding Generation: A high-dimensional vector representation (embedding) of the extracted text is created using the Sentence Transformer model.
5. Storage and Integrity Check: Before saving, the system checks for duplicates based on file name or extracted text. New documents are stored in their corresponding MongoDB collection, complete with the raw text, structured JSON data, and the embedding vector.

4. Addressing Module Requirements and Edge Cases

The implementation directly addresses the key requirements outlined in the module, particu- larly concerning data integrity and pipeline stability.

- ❖ Handling Invoices, POs, and Approvals: Handled via dedicated classification logic and separate collections in MongoDB.
- ❖ Multi-modal LLMs / Cloud OCR (Conceptual): The use of Gemini/LangChain and dedi- cated OCR tools (pytesseract, pdfplumber) mimics the functionality of multi-modal mod- els and cloud OCR services by performing image-to-text conversion and complex reason- ing/structuring.
- ❖ Build Data Pipeline: A complete pipeline is built, including text extraction, LLM-based transformation, validation (duplicate check), and storage.
- ❖ Edge Cases:
  - Low-Quality Scans/Handwriting: Addressed by using image pre-processing (con- version to grayscale) before applying PyTesseract OCR, enhancing text extraction fi- delity.
  - Missing Fields/Complex Layouts: Addressed by relying on the LLM's superior con- textual understanding to parse and extract data from complex, unstructured text, providing resilience against variable document formats.

5. Deployment and Access

The application components have been successfully deployed to the Hugging Face Spaces plat- form, providing public access to both the frontend interface and the backend API service.

- Streamlit Frontend Application: Accessible via the following link, providing the user in- terface for document upload and data visualization: Invoice Reader Streamlit App
- Flask Backend API: The underlying processing service is deployed separately for scalabil- ity and decoupled access: Document Processor API Endpoint

---

ERP Chatbot (RAG + Generative AI) Use Case:

1. Abstract

This report details the implementation of an ERP Chatbot module utilizing a Retrieval-Augmented Generation (RAG) architecture. The project successfully integrates multi-document ingestion, vector embedding storage via MongoDB Atlas Vector Search, and a sophisticated retrieval and generation pipeline using Google's Gemini LLM. A key feature is the ability to query across multiple ERP document collections (Invoices, Purchase Orders, and Approvals), providing grounded and contextually relevant responses. The architecture is designed for future extensibility, supporting features like confidence scoring and structured workflow in-tegrations.

2. Introduction and Project Objective

The primary objective of this module was to design and implement a scalable, intelligent query solution capable of providing accurate, grounded answers based on critical business documents from an Enterprise Resource Planning (ERP) system. The solution uses a RAG pipeline to achieve information retrieval and high-fidelity text generation, effectively turning unstructured ERP documents into a searchable knowledge base.

3. System Architecture and Core Components

The solution comprises a Flask API backend for core RAG functionality and a Streamlit frontend for user interaction, utilizing a modern, containerized deployment strategy.

1.1 Technology Stack

- Backend Framework: Python (Flask) for routing and orchestration.
- Database and Vector Store: MongoDB Atlas (for document storage and Vector Search indexing).
- Vector Embeddings: SentenceTransformer (all-MiniLM-L6-v2) for generating query and document embeddings.
- Generative AI: ChatGoogleGenerativeAI (Gemini 2.5 Flash) via LangChain for contextual generation.
- Deployment: Containerized using Docker (Gunicorn) to ensure environment consistency.

1.2 Data Ingestion and Vector Store

The system is designed to provide answers grounded in Enterprise Resource Planning (ERP) documentation, categorized into multiple data streams.

1. Document Ingestion: Supports and manages data from three distinct ERP collections: invoices, purchase_orders, and approvals.
2. Preprocessing & Embedding: Each document's content is encoded into high-dimensional vectors.
3. Vector Storage: Embeddings are stored and indexed within the respective MongoDB col- lections using MongoDB Atlas Vector Search, enabling efficient semantic retrieval.

4. Retrieval-Augmented Generation (RAG) Pipeline

The RAG pipeline is the central intelligence mechanism, ensuring the LLM's answers are based on factual data retrieved directly from the ERP documents.

1.3 The Query Process

1. **Query Embedding**: The user's natural language question is immediately converted into a vector using the all-MiniLM-L6-v2 model.
2. **Multi-Collection Vector Search (Retrieval):** The backend executes simultaneous Mon- goDB $vectorSearch aggregation pipelines across all specified ERP collections. This en- sures maximum recall across all document types.
3. **Context Aggregation:** The top N most relevant documents from the combined search re- sults are compiled into a unified context block. Each document is labeled with its collection type (e.g., [INVOICE]), serving as a source citation.
4. **Generation (LLM Invocation):** A final, grounded prompt is constructed, instructing the gemini-2.5-flash model to answer *only* based on the provided context, resulting in a factual and traceable response.

5. Addressing Module Requirements and Extensibility

The foundational architecture is designed to support both current requirements and future advanced features.

1.4 Guardrails Implementation

The current RAG implementation directly addresses key guardrail requirements:

- Citation of Sources: Implemented by labeling each context chunk with its originating collection ([INVOICE]), providing a clear path for traceability.
- Context Retention: The API structure facilitates the integration of an external memory store to retain chat history and context across user sessions.
- Confidence Scoring (Planned): The architecture supports augmenting the RAG output with calculated relevance scores from the MongoDB vector search to provide a confidence metric.

1.5 Structured Workflows

The architecture is extensible to integrate structured workflows (e.g., automated leave requests) by enabling the LLM to perform:

- Intent Recognition: Identifying user requests that map to specific ERP actions rather than just information retrieval.
- Function Triggering: Calling specific, non-RAG backend functions or external ERP action APIs based on the recognized intent.

6. Deployment and Access

The application components have been successfully deployed to the Hugging Face Spaces plat- form, providing public access to both the frontend interface and the backend API service.

- **Streamlit Frontend Application**: Accessible via the link below, providing the user inter- face for natural language querying of ERP documents: https://huggingface.co/spaces/ahmed- ayman/erp-chatbot
- **Flask Backend API:** The underlying processing service is deployed separately for scalability and decoupled access: https://huggingface.co/spaces/ahmed-ayman/erp_chatbot_api

7. Conclusion

The ERP Chatbot module delivers a robust, multi-document RAG system utilizing modern embedding models and Google's Gemini LLM. The design provides a scalable foundation for ac- curate, grounded Q&A over ERP data and is ready for the integration of advanced features like workflow automation and improved guardrails.

HR Use Cases:

1. Introduction and Project Objective

The primary objective was to design, develop, and implement an **AI-powered HR API** to leverage company data for predictive and analytical insights. This solution focuses on two critical HR use cases:

1) **Employee Attrition Risk Prediction:** Identifying employees at high risk of leaving the organization.
2) **Training Needs Prediction & Gap Analysis:** Determining which employees require training and performing an in-depth analysis of their specific performance and skill gaps using a Large Language Model (LLM).

2. Data Processing and Predictive Modeling (HR_Usecases.ipynb)

The predictive component involved integrating two datasets (Employee.csv and PerformanceRating.csv), handling class imbalance, and training machine learning models for the two core predictions.

1.1 Data Preparation Highlights

- **Data Integration:** Employee data was merged with the latest performance review data based on EmployeeID and the most recent ReviewDate.
- **Target Encoding:** The categorical Aflrition column was used as the target value (Yes: 1, No: 0).
- **Feature Engineering:** One-Hot Encoding was applied to all remaining high-cardinality categorical features (e.g., BusinessTravel, Department, JobRole, MaritalStatus, Ethnicity, State).
- **Class Imbalance Handling (Attrition Model):** The aflrition data was severely imbalanced (1043:0, 237:1 ratio). To mitigate bias, a compromise was chosen: **Undersampling the Majority** (to 500 samples) and **Oversampling the Minority** (to 250 samples) for the training set, which provided the best-balanced performance.
- **Multicollinearity Check:** Variance Inflation Factor (VIF) analysis was performed to ensure feature independence, leading to the removal of redundant features with high VIF scores.
- **Training Needs Target:** The target variable Needs_Training was explicitly engineered based on the ManagerRating, where a rating of **less than 3 was flagged as 1** (Needs Training), resulting in a well-balanced dataset (3415:1, 3294:0).

1.2 Model Perfiormance Comparison

Three models were trained for Aflrition Prediction: Logistic Regression, a Neural Network, and XGBoost. **XGBoost** demonstrated the best overall balanced performance (Macro F1-Score of 0.81). The **Logistic Regression** model was selected for Training Needs Prediction due to its high and balanced performance metrics on the ready-to-use dataset.

Model	Attrition - Macro Precision	Attrition - Macro Recall	Attrition - Macro F1-Score
XGBoost	**0.83**	**0.79**	**0.81**
Neural Network	0.73	0.72	0.72
Logistic Regression	0.72	0.76	0.74

3. System Architecture and API Endpoints (app.py)

The core application is built using **FastAPI** to serve both the machine learning models and the LLM functionality through a set of structured API endpoints.

a. Core Components and Technologies

i. **Framework:** FastAPI (Python) for robust, async API development.

ii. **ML Models: XGBoost** (for Aflrition) and **Logistic Regression** (for Training Needs), saved using joblib.

iii. **Data Handling:** pandas and numpy for data preprocessing, scaling (StandardScaler), and feature alignment before passing to the models.

iv. **Generative AI: Google's Gemini-2.5-Flash** via the google-generativeai SDK for complex qualitative analysis.

b. Key Endpoints

Endpoint	Method	Functionality	AI Component
/predict_aflrition	POST	Predicts aflrition risk for a single employee payload.	XGBoost Model
/predict_training	POST	Predicts if an employee needs training for a single employee payload.	Logistic Regression Model
/get_high_risk_list	GET	Processes an entire pre-loaded dataset (combined.csv) and returns a filtered list of high-aflrition-risk employees.	XGBoost Model

/get_training_needs_list	GET	Processes an entire pre-loaded dataset (combined.csv) and returns a filtered list of employees who need training.	Logistic Regression Model
/analyze_training_gaps	POST	Accepts employee features and generates a **4-step detailed analysis** of potential skill gaps and actionable management recommendations.	Gemini-2.5-Flash LLM

c. LLM Analysis Prompt Structure

The **Generative AI** is utilized for the **"Training Gap Analysis"** endpoint using a carefully engineered 4-step prompt, acting as an **"Expert HR Analyst"**:

1. **Identify the Gap:** Confirms performance issues (ManagerRating/SelfRating <= 3) and **Attrition Risk** (Aflrition = 1).
2. **Synthesize Context (The "Why"):** Analyzes **Performance vs. Satisfaction** alignment, **Training History** (opportunities vs. taken), and impact of **Workload & Tenure** (OverTime, YearsSinceLastPromotion, etc.) based on the specific JobRole.
3. **Identified Gaps (Summary List):** Provides a clear, bulleted summary of tangible skill/knowledge deficits (e.g., "Time Management," "Client Negotiation").
4. **Actionable Recommendations:** Suggests 2-3 specific actions for the manager and targeted training topics for each gap.

d. LLM Analysis Prompt Structure

The **Generative AI** is utilized for the **"Training Gap Analysis"** endpoint using a carefully engineered 4-step prompt, acting as an **"Expert HR Analyst"**:

5. **Identify the Gap:** Confirms performance issues (ManagerRating/SelfRating <= 3) and **Attrition Risk** (Aflrition = 1).
6. **Synthesize Context (The "Why"):** Analyzes **Performance vs. Satisfaction** alignment, **Training History** (opportunities vs. taken), and impact of **Workload & Tenure** (OverTime, YearsSinceLastPromotion, etc.) based on the specific JobRole.

7. **Identified Gaps (Summary List):** Provides a clear, bulleted summary of tangible skill/knowledge deficits (e.g., "Time Management," "Client Negotiation").
8. **Actionable Recommendations:** Suggests 2-3 specific actions for the manager and targeted training topics for each gap.

e. LLM Analysis Prompt Structure

The **Generative AI** is utilized for the **"Training Gap Analysis"** endpoint using a carefully engineered 4-step prompt, acting as an **"Expert HR Analyst"**:

9. **Identify the Gap:** Confirms performance issues (ManagerRating/SelfRating <= 3) and **Attrition Risk** (Aflrition = 1).
10. **Synthesize Context (The "Why"):** Analyzes **Performance vs. Satisfaction** alignment, **Training History** (opportunities vs. taken), and impact of **Workload & Tenure** (OverTime, YearsSinceLastPromotion, etc.) based on the specific JobRole.
11. **Identified Gaps (Summary List):** Provides a clear, bulleted summary of tangible skill/knowledge deficits (e.g., "Time Management," "Client Negotiation").
12. **Actionable Recommendations:** Suggests 2-3 specific actions for the manager and targeted training topics for each gap.

4. Synthesis of XAI for HR Action

By integrating SHAP and LIME alongside the LLM analysis, the HR AI API delivers a layered explanation:

1. **Macro-Level Validation (SHAP):** HR can globally **validate the model** to ensure that high-level retention strategies are focused on the truly important factors (e.g., compensation, work-life balance).
2. **Case-Level Intervention (LIME):** Managers can use the local explanations to understand **"Why this specific employee is at risk"** or **"Why they need training,"** giving them the necessary context to conduct proactive and personalized retention or performance coaching conversations.

Financial Forecasting with Classical Time Series Models Use case:

1. Introduction

This project focuses on forecasting key financial metrics — Revenue, Expenses, and Cash Flow — for companies over 40 quarters (Q1 2015 to Q4 2024). It uses the Financial Forecasting Dataset (2015–2024) sourced from Kaggle, and implements multiple classical time series forecasting techniques, including Prophet, ARIMA, and SARIMAX. The final deployment is done via a FastAPI-based endpoint hosted on Hugging Face Spaces using Docker.

Dataset Description

- **Source**: Kaggle – Financial Forecasting Dataset (2015–2024)
- **Time Span**: 40 quarters → from Q1 2015 to Q4 2024
- **Frequency**: Quarterly (converted to timestamp using PeriodIndex)
- **Key Variables**: Revenue, Expenses, Cash_Flow, Operating_Income, Net_Income, Company_ID, Year, Quarter.

2. Modeling Approaches

We compared three classical time series forecasting models:

1. ARIMA

- Used as a univariate baseline.
- Simple model but lacked performance due to ignoring external regressors.

2. SARIMAX

- Multivariate, handled seasonality and regressors better.
- Performed better than ARIMA in some financial KPIs.
- Harder to tune with many regressors.

3. Prophet (Final Choice)

- Chosen for because its ease of use and good performance with quarterly data.
- Used with lagged regressors to avoid data leakage.
- Tuned with:
    - changepoint_prior_scale = 0.1
    - seasonality_mode = "multiplicative"
    - Added quarterly Fourier terms for better fit.

Forecasting Strategy

Target Metric	Regressors Used
Revenue	Operating_Income, Cash_Flow, Net_Income
Expenses	Revenue, Operating_Income
Cash Flow	Revenue, Expenses, Net_Income

Anti-Leakage Procedure

To ensure real-world robustness:

- **Lagging**: All regressors were shifted by 1 quarter.
- **Backtesting**: Forecasted the final 4 quarters (Q1–Q4 2024) using prior history.
- **Validation**: Compared actual vs. forecast using:
    - MAE (Mean Absolute Error)
    - RMSE (Root Mean Squared Error)

Forecast Results (CMP_002 Example)

Metric	MAE	RMSE
Revenue	~4708	~6205
Expenses	~34	~35.5
CashFlow	~4126	~5846

3. Deployment: Hugging Face Spaces with FastAPI

The forecasting model was deployed as a RESTful API using FastAPI, allowing real-time access to forecasts via HTTP requests. The backend was built in Python and wrapped in a Docker container for environment consistency and reproducibility. A custom Dockerfile defined the build steps, installing dependencies such as Prophet and scikit-learn from a requirements.txt file. The application was served using Uvicorn on port 7860. This containerized API was then deployed on Hugging Face Spaces, enabling a public-facing interface. Users can interact with the /forecast endpoint by sending a company ID via a POST request, receiving a structured JSON response that includes the actual and forecasted values for Revenue, Expenses, and Cash Flow, along with MAE and RMSE for performance evaluation. The FastAPI interactive documentation is also accessible through a GET request at /docs.

Financial Forecasting with TFT Models Use Case:

1. Introduction

This project aims to build an **AI forecasting service** that integrates with an ERP system to predict future financial performance of companies. The model forecasts quarterly metrics — initially focused on **Revenue**, and later extended to also forecast **Expenses** and **Cash Flow**. The AI service is exposed through a **REST API endpoint**, making it easy for a full-stack/ERP team to integrate it inside dashboards or reporting modules.

The project evolved through several phases:

1. **Dataset preparation and feature engineering**
2. **Experimenting with classical deep learning models (LSTM & RNN)**
3. **Selecting and training a powerful multivariate forecasting model (TFT)**
4. **Building a FastAPI endpoint for real-time predictions**
5. **Containerizing and deploying the model using Docker in a Hugging Face Space**
6. **Providing a final API for ERP integration**

Each step was completed with a focus on accuracy, performance, and clean deployment pipelines.

2. Dataset Structure & Preprocessing

The dataset contains quarterly financial data for multiple companies, covering values such as:

- Revenue
- Expenses
- Operating Income
- Net Income
- Cash Flow
- Assets, Liabilities, Equity
- EPS, ROE, ROA, debt ratios
- Market indicators (Sentiment Score, Buzz, Inflation, etc.)
- Binary indicators (Audit Flag, Fraud Flag, Market Shock Flag, Policy Change Flag)
- **Target_Revenue_Next_Qtr** — the supervised target for forecasting

A synthetic time index (time_idx) was created for each company to ensure temporal ordering.
Only numeric columns are used as **time-varying unknown features**, except ignored fields like *Date* and anomaly classes.

The preprocessing pipeline automatically:

- Sorts data by Company → Year → Quarter
- Generates time_idx using cumulative counts
- Splits the data into training and prediction sets
- Identifies continuous features dynamically
- Removes categorical fields that TFT can't use directly

This ensures the model receives consistent and structured temporal sequences.

3. Baseline Experiments — LSTM and RNN

Before selecting the final model, we experimented with:

LSTM Model

- Used many numerical features
- Learned sequential patterns
- Provided the first stable prediction results
- Worked well but struggled with multi-feature dependencies

Simple RNN

- Faster but less accurate
- Useful as a baseline comparison
- Failed to capture long-term dependencies

These experiments helped confirm that classical models were limited in understanding the complex interactions between financial, economic, and market variables.

The findings led to adopting a model designed specifically for multi-feature time-series forecasting.

4. Final Model — Temporal Fusion Transformer (TFT)

TFT was selected because it is:

- **Designed for multivariate time series**
- Learns relationships between past values, future known values, and static company identity
- Handles attention-based temporal weights
- Performs significantly better than LSTM/RNN models in structured tabular time series

Our TFT model uses:

- **8 past quarters** as the encoder window
- **1 future quarter** as the prediction horizon
- Group Normalizer per company
- Time-varying known feature: **Quarter**
- Time-varying unknown reals: all numeric columns
- Static categorical: Company_ID

TFT was trained in Colab, and the best checkpoint tft-best.ckpt was exported for production.

5. Deployment

To operationalize the TFT forecasting model, we developed a production-ready inference API using FastAPI. The API automatically loads the trained TFT checkpoint, rebuilds the same TimeSeriesDataSet structure used during training, detects the latest available year for each company, and generates quarter-by-quarter predictions. It returns an ERP-ready JSON containing: forecasted revenue, actual revenue (for evaluation), quarter dates, and error metrics (MAE & RMSE). This format supports dashboards, performance tracking, and automated reporting inside the ERP system.

The full service was containerized using Docker. The Dockerfile installs all Python and system dependencies, copies the model, dataset, and API code into the container, and runs the application using uvicorn on port 7860, the Hugging Face standard. Finally, the entire containerized API was deployed on Hugging Face Spaces, providing a free, always-online endpoint that the full-stack team can directly integrate with the ERP system.

The live deployment exposes:

- Interactive API documentation:
  https://dina-tolba-tft-revenue-forecast-api.hf.space/docs
- Main prediction endpoint:
  /predict_tft