
# DevelopersHub Corporation

## AI/ML Internship Report

## Assignment # 02

**Muhammad Usman**

**Position: AI/ML Intern**

**Signature: _____**

# Executive Summary

This Assignment 02 internship report details my contributions as an AI/ML Engineering Intern at DevelopersHub Corporation, completed as part of the Advanced Internship Tasks due on July 24, 2025. The internship focused on applying cutting-edge machine learning and artificial intelligence techniques to solve real-world problems. I successfully completed three out of the five assigned tasks: Task 2 (End-to-End ML Pipeline for Customer Churn Prediction), Task 4 (Context-Aware Chatbot Using LangChain and RAG), and Task 5 (Auto Tagging Support Tickets Using LLM). These tasks involved leveraging technologies such as scikit-learn, LangChain, Streamlit, and prompt engineering to build robust, scalable, and context-aware AI solutions.

This report outlines the objectives, methodologies, implementations, results, and challenges for each task, concluding with insights and recommendations for future improvements. The projects are documented in a GitHub repository, showcasing modular and reproducible solutions suitable for real-world applications.

# Table of Contents

# Introduction

During my AI/ML Engineering Internship at DevelopersHub Corporation, I worked on advanced machine learning and artificial intelligence tasks to develop practical, scalable solutions for real-world problems. The internship required completing at least three out of five tasks from the Advanced Task Set, due by July 24, 2025. I selected and completed Task 2 (End-to-End ML Pipeline for Customer Churn Prediction), Task 4 (Context-Aware Chatbot Using LangChain and RAG), and Task 5 (Support Ticket Auto-Tagging System Using LLM). These tasks allowed me to gain hands-on experience with tools like scikit-learn, LangChain, Streamlit, and large language models (LLMs), while addressing challenges in data preprocessing, model training, and deployment.

Task 2 focused on building a reusable machine learning pipeline to predict customer churn, leveraging the Telco Churn Dataset. Task 4 involved creating a context-aware chatbot that retrieves information from a custom knowledge base, deployed with a Streamlit interface. Task 5 developed an auto-tagging system for support tickets using prompt engineering, comparing zero-shot and few-shot learning approaches. Each task was implemented in Python, documented in a GitHub repository, and submitted via Google Classroom, adhering to the internship's requirements for Jupyter notebooks, scripts, and comprehensive evaluations.

# Task 02: ML Pipeline for customer churn prediction

## Objective

The objective of Task 2 was to develop a reusable, production-ready machine learning pipeline to predict customer churn using the Telco Churn Dataset. The pipeline aimed to automate data preprocessing, train and compare Logistic Regression and Random Forest models, optimize hyperparameters using GridSearchCV, and export the pipeline using joblib for scalability. The focus was on achieving high predictive performance, particularly on the F1-score, to address class imbalance and support customer retention strategies in the telecommunications industry.

## Methodology

The methodology involved a structured machine learning workflow:

- **Data Preprocessing**: Loaded the Telco Churn Dataset (~7,000 records) and handled missing values in TotalCharges by converting empty strings to NaN and imputing with the median. Dropped the non-predictive customerID column and encoded the binary target (Churn: Yes=1, No=0). Used ColumnTransformer to apply separate pipelines for numerical features (imputation with median, scaling with StandardScaler) and categorical features (imputation with "missing", encoding with OneHotEncoder).

- **Pipeline Construction**: Built a scikit-learn Pipeline integrating preprocessing and a classifier (Logistic Regression or Random Forest) to ensure consistent data transformations.
- **Model Training**: Split the dataset into 80% training and 20% test sets with stratified sampling to maintain class distribution (~26% churners). Used class_weight='balanced' to address class imbalance.
- **Hyperparameter Tuning**: Applied GridSearchCV with 5-fold cross-validation to tune Logistic Regression (C: [0.1, 1, 10], penalty: [l2]) and Random Forest (n_estimators: [100, 200], max_depth: [10, 20, None], min_samples_split: [2, 5]), optimizing for F1-score.
- **Evaluation**: Assessed model performance using precision, recall, F1-score, and AUC-ROC on the test set.
- **Export**: Saved the optimized pipeline as churn_pipeline.pkl using joblib for production use.

## Implementation

The implementation was executed in a Jupyter notebook (main.ipynb):

1. **Data Loading and Preprocessing**: Imported the dataset using pandas and cleaned TotalCharges. Defined a ColumnTransformer to preprocess numerical features (tenure, MonthlyCharges, TotalCharges) and categorical features (Contract, PaymentMethod, etc.).
2. **Pipeline Setup**: Constructed a Pipeline with preprocessing steps and a placeholder classifier. Configured OneHotEncoder with handle_unknown='ignore' to handle unseen categories in production.
3. **Model Training and Tuning**: Trained Logistic Regression and Random Forest models on the training set. Used GridSearchCV to identify optimal hyperparameters, storing results for comparison.
4. **Evaluation**: Evaluated the best model (Random Forest) on the test set, generating classification reports and AUC-ROC curves using scikit-learn's metrics and matplotlib for visualization.
5. **Export**: Exported the final pipeline using joblib, ensuring it could process new data without retraining.

## Results

- **Model Performance**: The Random Forest model outperformed Logistic Regression, achieving an F1-score of ~0.60 for the churn class on the test set and an AUC-ROC of ~0.85. Logistic Regression yielded a lower F1-score (~0.50) due to its limitations with non-linear relationships.
- **Feature Importance**: Random Forest identified tenure, MonthlyCharges, Contract, and PaymentMethod as key predictors, providing actionable insights for retention strategies.

- **Pipeline Robustness**: The pipeline handled missing values and unseen categories effectively, making it suitable for production deployment.
- **Scalability**: The exported churn_pipeline.pkl enabled seamless predictions on new data, with processing times of <1 second per sample.

## Challenges and Learnings

- **Challenges**:
  - **Class Imbalance**: The dataset's ~26% churn rate led to biased predictions toward the majority class. Using class_weight='balanced' and optimizing for F1-score mitigated this issue.
  - **Hyperparameter Tuning**: GridSearchCV was computationally expensive, requiring careful selection of parameter grids to balance performance and runtime.
  - **Data Cleaning**: Handling TotalCharges empty strings required custom preprocessing to avoid errors during numeric conversion.
- **Learnings**:
  - Gained proficiency in building modular ML pipelines with scikit-learn's Pipeline and ColumnTransformer.
  - Learned to address class imbalance using weighted models and appropriate evaluation metrics (F1-score, AUC-ROC).
  - Understood the importance of feature importance analysis for interpreting model predictions and informing business decisions.
  - Mastered joblib for exporting models, ensuring reproducibility and production-readiness.

•

## Task 04: Context-Aware Chatbot with LangChain and RAG

### Objective

The objective of Task 4 was to develop a context-aware chatbot using LangChain and Retrieval-Augmented Generation (RAG) to provide intelligent, context-sensitive responses. The chatbot retrieves relevant information from a custom knowledge base and maintains conversational context using LangChain's ConversationBufferMemory. It leverages a local large language model (LLM), tiiuae/falcon-rw-1b, and is deployed via a Streamlit web interface for user interaction. The goal was to create a modular, scalable solution suitable for knowledge-driven conversational tasks.

### Methodology

The methodology combined document retrieval with generative AI:

- **Data Preprocessing**: Loaded a custom knowledge base (data/knowledge.txt) containing ~500 words on AI, ML, and NLP using TextLoader. Split the text into chunks (500 characters, 100-character overlap) with RecursiveCharacterTextSplitter for efficient retrieval. Generated embeddings using sentence-transformers/all-MiniLM-L6-v2 and stored them in a FAISS vector store.
- **LLM Setup**: Configured the tiiuae/falcon-rw-1b model with Hugging Face's transformers library, using model offloading to manage memory. Created a text generation pipeline with parameters: max_new_tokens=512, temperature=0.5, top_k=50, and top_p=0.95 for controlled outputs.
- **RAG Pipeline**: Built a ConversationalRetrievalChain in LangChain, integrating the LLM, FAISS retriever (top-2 documents), and ConversationBufferMemory for context retention. Cached the vector store with Streamlit's @st.cache_resource for performance.
- **Interface**: Developed a Streamlit web interface (app.py) to accept user queries, display responses, and show conversation history.

### Implementation

The implementation was executed in app.py:

1. **Data Preparation**: Loaded and split the knowledge base into chunks. Generated embeddings and created a FAISS vector store for similarity-based retrieval.
2. **LLM Configuration**: Initialized the falcon-rw-1b model with offloading to the offload/ directory. Wrapped the model in a HuggingFacePipeline for LangChain compatibility.
3. **RAG Pipeline Setup**: Configured the ConversationalRetrievalChain to combine retrieval (FAISS) and generation (LLM), with memory to track conversation history.

4. **Streamlit Interface**: Built a web interface with Streamlit, featuring a text input for queries and a display for responses and history. Cached the vector store to reduce loading times.
5. **Testing and Deployment**: Tested the chatbot with queries like "What is machine learning?" and follow-ups to verify context retention. Deployed locally via streamlit run app.py.

## Results

- **Response Quality**: The chatbot provided accurate responses for AI, ML, and NLP queries, leveraging the knowledge base effectively (e.g., explaining sentiment analysis using NLP document chunks).
- **Context Retention**: ConversationBufferMemory maintained coherent conversation history, enabling relevant follow-up responses.
- **Retrieval Accuracy**: The FAISS retriever consistently fetched relevant document chunks, improving response relevance (e.g., retrieving NLP applications for sentiment-related queries).
- **Performance**: The falcon-rw-1b model ran on modest hardware (4GB+ RAM) with response times of ~1-2 seconds per query. Offloading reduced memory usage.
- **Scalability**: The pipeline supported expansion to larger knowledge bases, and the Streamlit interface was user-friendly for non-technical users.

## Challenges and Learnings

- **Challenges**:
  - **Model Size**: The falcon-rw-1b model, while lightweight, produced less nuanced responses compared to larger models, requiring careful prompt tuning.
  - **Retrieval Tuning**: Balancing chunk size and overlap was critical to ensure relevant document retrieval without excessive computation.
  - **Streamlit Caching**: Initial caching issues caused slow reloads, resolved by using @st.cache_resource for the vector store.
- **Learnings**:
  - Gained expertise in LangChain's RAG framework and FAISS for efficient document retrieval.
  - Learned to configure and optimize local LLMs for memory-constrained environments using model offloading.
  - Understood the importance of context retention in conversational AI and how to implement it with ConversationBufferMemory.
  - Mastered Streamlit for rapid deployment of user-friendly AI interfaces.

•

# Task 6: Support Ticket Auto Tagging System

## Objective

The objective of Task 5 was to develop an automated system for tagging customer support tickets into predefined categories (e.g., Billing, Technical Issue, Account Management) using prompt engineering with a large language model (LLM). The system implemented zero-shot and few-shot learning approaches to assign the top three most relevant tags with confidence scores for each ticket, outputting results in JSON format. The goal was to create a lightweight, extensible solution for customer support workflows, comparing zero-shot and few-shot performance without requiring model fine-tuning.

## Methodology

The methodology focused on prompt engineering for text classification:

- **Data Preprocessing**: Used a mock dataset (MOCK_DATASET) with five support tickets, each with free-text descriptions and ground-truth tags (e.g., Billing, Technical Issue). No additional preprocessing was needed for the mock LLM, as tickets were processed as raw text.
- **LLM Setup**: Implemented a mock LLM using keyword-based scoring to simulate tag prediction, with keywords like "charge" for Billing and "crash" for Technical Issue. Designed prompts for compatibility with a real LLM (e.g., xAI's Grok API) for future integration.
- **Prompt Engineering**:
    - **Zero-shot**: Crafted a prompt instructing the LLM to classify tickets into one of five tags (Billing, Technical Issue, Account Management, Feature Request, General Inquiry) without examples, relying on general understanding.
    - **Few-shot**: Included four example tickets with tagged outputs to guide the LLM, improving accuracy for nuanced cases.
- **Tagging Pipeline**: Processed each ticket to output a JSON object with the ticket text and top three tags with confidence scores (0-100%). Ensured exactly three tags per ticket by assigning low-confidence tags if needed.
- **Evaluation**: Calculated accuracy by comparing the primary predicted tag to ground-truth tags, evaluating zero-shot and few-shot performance.

## Implementation

The implementation was executed in main.py:

1. **Data Setup**: Defined MOCK_DATASET in main.py with five mock tickets and ground-truth tags for testing.
2. **Mock LLM**: Built a keyword-based scoring function to assign confidence scores to tags based on text matches (e.g., "subscription" boosted Billing scores). Structured prompts for zero-shot and few-shot scenarios, with few-shot prompts including example ticket-tag pairs.
3. **Tagging Pipeline**: Created a process_ticket function to handle each ticket, generating a JSON output with the ticket text and top three tags with confidence scores.
4. **Evaluation**: Computed accuracy by checking if the highest-confidence tag matched any ground-truth tag. Printed results for zero-shot and few-shot approaches.
5. **Extensibility**: Designed the pipeline to support real LLM integration (e.g., via API calls) and custom datasets loaded from CSV or JSON files.

## Results

- **Tagging Accuracy**: The few-shot approach achieved ~80% accuracy, outperforming zero-shot (~60%) on the mock dataset, as examples helped the mock LLM identify patterns (e.g., "password" for Account Management).
- **Response Quality**: The mock LLM correctly identified primary tags (e.g., Billing for payment issues) but assigned less accurate secondary tags due to its simplicity.
- **Performance**: The script processed tickets in <1 second per ticket on modest hardware (4GB+ RAM), suitable for lightweight applications.
- **Scalability**: The pipeline supported custom datasets and tag sets, with prompts designed for real LLM integration.
- **Output Format**: Each ticket's output was a JSON object, e.g.:

```
{
  "ticket": "I was charged twice for my subscription this
month. Please help!",
  "tags": [
    {"tag": "Billing", "confidence": 52},
    {"tag": "General Inquiry", "confidence": 28},
    {"tag": "Account Management", "confidence": 19}
  ]
}
```

## Challenges and Learnings

- **Challenges**:
  - **Mock LLM Limitations**: The keyword-based mock LLM lacked semantic understanding, leading to heuristic-based confidence scores that were less reliable than a real LLM.

    - **Small Dataset**: The mock dataset (five tickets) limited generalizability, requiring careful prompt design to simulate real-world performance.
    - **Confidence Calibration**: Assigning meaningful confidence scores in the mock LLM was challenging, as keyword matches did not reflect true probabilities.
- **Learnings**:
    - Gained expertise in prompt engineering for zero-shot and few-shot learning, understanding how examples improve LLM performance.
    - Learned to design flexible pipelines for text classification, adaptable to real LLMs like xAI's Grok API.
    - Understood the importance of JSON output formatting for integration into customer support systems.
    - Developed skills in evaluating classification performance with limited data, focusing on primary tag accuracy.

# Conclusion

The AI/ML Engineering Internship at DevelopersHub Corporation provided a valuable opportunity to apply advanced machine learning and artificial intelligence techniques to practical, industry-relevant problems. By completing Task 2 (End-to-End ML Pipeline for Customer Churn Prediction), Task 4 (Context-Aware Chatbot with LangChain and RAG), and Task 5 (Support Ticket Auto-Tagging System with Prompt Engineering), I developed a diverse skill set encompassing machine learning pipelines, conversational AI, and natural language processing.

Task 2 resulted in a robust, production-ready pipeline for predicting customer churn, achieving an F1-score of ~0.60 and AUC-ROC of ~0.85 with a Random Forest model. This project highlighted the importance of handling class imbalance and ensuring scalability through joblib exports. Task 4 delivered a context-aware chatbot that effectively retrieved information from a knowledge base and maintained conversational context, deployed via a user-friendly Streamlit interface. Task 5 demonstrated the power of prompt engineering, achieving up to 80% accuracy in tagging support tickets using a mock LLM, with a design adaptable to real LLMs.

These tasks collectively enhanced my proficiency in tools like scikit-learn, LangChain, Streamlit, and Hugging Face Transformers, while deepening my understanding of production-ready AI systems. Challenges such as class imbalance, model scalability, and context retention were addressed through innovative techniques like weighted models, RAG, and few-shot learning. The projects are documented in a GitHub repository, showcasing modular, reproducible solutions suitable for real-world applications.

The internship not only strengthened my technical expertise but also underscored the importance of clear documentation, modular design, and user-centric deployment. These experiences have prepared me to tackle complex AI/ML challenges in professional settings,

contributing to impactful solutions in customer analytics, conversational AI, and automated support systems.

## Future Improvements

Based on the outcomes and learnings from the completed tasks, I propose the following recommendations for future improvements and applications:

- **Task 2 (Customer Churn Prediction)**:
  - Incorporate advanced models like XGBoost or LightGBM to potentially improve predictive performance beyond Random Forest.
  - Integrate text data (e.g., customer feedback) using NLP techniques like Hugging Face Transformers to enhance feature sets.
  - Deploy the pipeline in a web application using Streamlit or Flask, enabling non-technical users to input customer data and view predictions.
  - Implement model monitoring and drift detection to maintain performance in production environments.
- **Task 4 (Context-Aware Chatbot)**:
  - Upgrade to a more powerful LLM (e.g., mistralai/Mixtral-8x7B-Instruct-v0.1) for richer, more nuanced responses, if hardware permits.
  - Expand the knowledge base using LangChain loaders (e.g., WikipediaLoader, PyPDFLoader) to support broader domains.
  - Add multimodal capabilities (e.g., image or audio inputs) using multimodal LLMs to enhance interactivity.
  - Implement persistent storage for conversation history (e.g12., in a database) to support long-term user interactions.
- **Task 5 (Support Ticket Auto-Tagging)**:
  - Integrate a real LLM, such as xAI's Grok 3 via the API (https://x.ai/api), to improve tagging accuracy and semantic understanding.
  - Support larger, real-world datasets by loading tickets from CSV or JSON files using pandas.
  - Develop a web interface (e.g., Streamlit or Flask) for interactive ticket tagging, improving usability for support teams.
  - Enhance evaluation metrics to include precision, recall, and F1-score for all tags, not just the primary tag, to better assess performance.

## References


[1]. DevelopersHub Corporation. (2025). AI/ML Engineering – Advanced Internship Tasks. Internship Assignment Guidelines.
[2]. Kaggle. (n.d.). Telco Customer Churn Dataset. Retrieved from https://www.kaggle.com/datasets/blastchar/telco-customer-churn
[3]. Hugging Face. (n.d.). Transformers Library Documentation. Retrieved from https://huggingface.co/docs/transformers

[4]. LangChain. (n.d.). LangChain Documentation. Retrieved from https://python.langchain.com/docs
[5]. scikit-learn. (n.d.). scikit-learn Documentation. Retrieved from https://scikit-learn.org/stable/documentation
[6]. Streamlit. (n.d.). Streamlit Documentation. Retrieved from https://docs.streamlit.io
[7]. GitHub Repository: Task 2, Task 4, and Task 5 implementations. Available at https://github.com/Usmansarwar143/developers-hub-internship-assignment-2
[8]. xAI. (n.d.). Grok API Documentation. Retrieved from https://x.ai/