Spotify Churn Prediction, Design and Implementation of an MLOps System

Rick de Rijk, Thom Verzantvoort, Tycho van Rooij, Stefan Vonk, Gilbert Laanen

Group 5

Table of contents

List of Figures

1 Overview of the ML Application

1.1 Use Case Description

The Spotify Churn Prediction application aims to identify users who are likely to cancel their Spotify Premium subscription. Churn prediction is a typical classification task in customer-retention analytics, where the goal is to estimate the probability that a current subscriber will stop using the service within a defined time horizon.

The project focuses on transforming user-level behavioral and demographic data into actionable features that can be used by machine-learning models. Examples of such features include user age, average listening time, number of songs played per day, skip rate, and advertisements listened to per week. By analyzing these patterns, the model learns behavioral signals that differentiate loyal users from those at risk of canceling their subscription at Spotify.

1.2 Business Goal and ML Objective

The main objective of the application is to predict whether a user will churn (cancel their subscription) or remain an active subscriber. This is modeled as a binary classification problem, where the output variable represents churn versus non-churn. The probability generated by the model provides actionable insights into user behavior that can support targeted retention campaigns and personalized marketing strategies.

For this purpose, a dataset named spotify_churn_dataset.csv was used. The dataset contains historical user activity with labeled churn outcomes and includes both continuous and categorical variables relevant to user engagement. The dataset was obtained from Kaggle Lauraalonso (2025), which also provided an example notebook that served as inspiration for the data preprocessing and model-training stages.

The ML model expects structured tabular data as input and produces a binary churn label with an associated probability score. The performance of the model is evaluated on accuracy, precision, and recall to ensure a balanced identification of potential churners without excessive false positives.

1.3 MLOps Requirements

From an operational perspective, the Spotify churn prediction system must support a fully automated MLOps workflow to ensure scalability, reproducibility, and maintainability.

- Retraining requirement: The model automatically retrains when new data is uploaded to the Google Cloud Storage data bucket. This guarantees that the model remains accurate and reflective of the most recent user behavior.

- Deployment requirement: The model must be deployed as a REST API (prediction-api) accessible through a user interface (prediction-ui). This setup allows real-time churn predictions through API requests from external applications.

- Continuous integration and delivery requirement: Whenever code is updated or new model versions are committed to the GitHub repository, Google Cloud Build automatically rebuilds and redeploys the Docker images for both the API and UI, ensuring consistent and version-controlled deployment.

- Deployment constraints: The system must run efficiently within the limits of Google Cloud Run, which provides scalability but restricts memory and CPU usage. The inference service should maintain low latency and high availability to be suitable for real-time applications.

- Monitoring and maintenance: The MLOps environment tracks API uptime, model versions, and possible data or concept drift. These metrics are used to determine when retraining or redeployment is required.

2 Design and Implementation of the MLOps System

The MLOps architecture for the Spotify churn prediction project automates the entire machine learning lifecycle, from data ingestion and model training to deployment and prediction. It is fully implemented on Google Cloud Platform (GCP) using Vertex AI, Cloud Build, Cloud Run, and Cloud Storage to ensure scalability, automation, and reproducibility.

Data and artifacts are stored in dedicated GCS buckets: the data bucket for raw and processed datasets, the model bucket for trained models, and the temporary bucket for intermediate files.

2.1 System Architecture

The MLOps architecture for the Spotify churn prediction project automates the entire machine learning lifecycle, from data ingestion and model training to deployment and prediction. It is fully implemented on Google Cloud Platform (GCP) using Vertex AI, Cloud Build, Cloud Run, and Cloud Storage to ensure scalability, automation, and reproducibility.

Data and artifacts are stored in dedicated GCS buckets: the data bucket for raw and processed datasets, the model bucket for trained models, and the temporary bucket for intermediate files.

2.2 Machine Learning Pipeline (Vertex AI)

Our machine learning pipeline is implemented using Google Cloud's Vertex AI Pipelines, a managed service based on Kubeflow that orchestrates containerized machine learning workflows. The pipeline automates the entire model development lifecycle from data preprocessing through conditional model deployment, implementing industry-standard MLOps practices to ensure production system reliability and performance.

The pipeline begins with data preparation (download_data), downloading the Spotify churn dataset from Google Cloud Storage and performing feature engineering. The preprocessing phase (preprocess_and_split) removes irrelevant columns and selects five key predictors: age, listening time, songs played per day, skip rate, and advertisements listened per week. The data is then split into training (80%) and test (20%) sets using stratified sampling to maintain class distribution. A StandardScaler is fitted exclusively on the training data to normalize features, preventing data leakage while ensuring consistent preprocessing for both training and inference.

Following data preparation, the pipeline enters the model training and selection phase through parallel execution of the train_rf and train_lr components. These components train Random Forest and Logistic Regression classifiers respectively, maximizing computational efficiency through concurrent execution. The predict_rf and predict_lr components then evaluate both models on the held-out test set using F1-score as the primary metric, with accuracy as a secondary measure. The compare_models component

automatically selects the best-performing model based on test F1-score, implementing an objective model selection process without manual intervention.

The final phase implements a champion/challenger validation pattern through three critical components. First, the download_production_model component retrieves the current production model from Cloud Storage. Second, the evaluate_production_model component evaluates this baseline model on the same test set used for the newly trained models, ensuring fair comparison. Third, the compare_with_baseline component compares the new model (challenger) against the production model (champion). Models with better performance than the current model in production trigger the upload_to_gcs component, which uploads both the model and preprocessing artifacts (the fitted scaler) to Cloud Storage, ensuring complete reproducibility. If no production baseline exists, the compare_with_baseline component automatically approves the first model for deployment, handling the initial deployment scenario gracefully.

This automated approach ensures continuous model improvement while maintaining production system quality through objective validation gates and complete artifact versioning, aligning with modern MLOps principles for reliable machine learning systems in production environments.

2.3 Prediction and Serving Components

The serving layer comprises two Flask-based components that enable end-to-end prediction delivery. The Prediction API loads the most recent model from the /models directory, exposes a /predict endpoint for JSON requests, and returns churn predictions in real time. The Prediction UI, illustrated in Figure 3 and Figure 4, provides a user-friendly web interface where input features can be entered and results are visualized. It communicates with the API via the PREDICTOR_API environment variable. Both components are containerized and deployed to Google Cloud Run as spotify-prediction-api:latest and spotify-prediction-ui:latest, ensuring scalable and consistent serving across environments.

2.4 CI/CD Pipelines (Cloud Build)

Continuous integration and deployment are managed through Cloud Build. The configuration files (pipeline_executor_cloudbuild.json, spotify_churn_pipeline_execution_cloudbuild.json, and cicd-automated.json) automate three key stages:

1. Building and pushing the **pipeline-executor** image to Artifact Registry.

2. Executing the Vertex AI pipeline using parameters.json for runtime settings.

3. Downloading trained models, rebuilding Docker images for the **Prediction API** and **Prediction UI**, and deploying both to Cloud Run.

Each configuration uses substitution variables (_REPOSITORY, _LOCATION, _MODEL_REPO, _CONF_REPO, _TEMP_REPO) to define repository paths, region, and storage locations. Triggers run automatically on pushes to the main branch or can be executed manually.

This integrated setup ensures that any change in data or code automatically triggers model retraining and redeployment. The workflow combines modular pipelines, automated testing, and scalable serving, fulfilling the MLOps principles of automation, reproducibility, and continuous improvement.

2.5 Pipeline Executor Container

To complete the automated workflow described in Sections 2.1-2.4, a pipeline-executor container was developed to programmatically trigger Vertex AI Pipelines from Cloud Build. Its function is to execute the training pipeline defined in spotify_churn_pipeline.yaml without manual notebook interaction.

The main script, pipeline_executor.py, uses the google-cloud-aiplatform library to create and run a PipelineJob in Vertex AI. It reads command-line arguments for the pipeline name, definition path, root directory, and parameter file, loads these settings from JSON, and submits the job with the appropriate runtime configuration.

Dependencies are listed in requirements.txt (google-cloud-aiplatform[pipelines], google-cloud-storage, google-auth), while the Dockerfile defines /workspace as the working directory and runs pipeline_executor.py as the entrypoint. The container image is built and pushed to Artifact Registry through pipeline_executor_tool_cloudbuild.json, ensuring it can be reused across CI/CD workflows.

This component forms the operational link between Cloud Build automation and Vertex AI Pipelines, enabling consistent, reproducible, and fully automated retraining within the overall MLOps architecture.

3 Reflection on Design and Implementation

3.1 Alternative Designs

Several alternative designs could have been applied to improve modularity and scalability. One option would be to serve the trained model through Vertex AI Endpoints instead of a containerized Flask API on Cloud Run. Vertex AI Endpoints provide a managed serving environment that handles scaling, versioning, and monitoring automatically, which reduces operational overhead (Ranasinghe, 2021). However, this approach limits flexibility in defining custom preprocessing, authentication, or multi-model routing logic. By contrast, the Flask API on Cloud Run offers full control over request handling and integration with the Prediction UI, at the cost of higher maintenance effort.

Another potential improvement would be to separate the training and deployment triggers in Cloud Build. This strategy would shorten build time and allow independent retraining or redeployment of individual components. It also aligns with recommended MLOps best practices, where conditional deployment is performed only when the newly trained model outperforms the existing one (Google Cloud, 2024).

3.2 Improvements and Scalability

While the current system is fully functional, several improvements could further enhance performance and maintainability. From a model perspective, including categorical user features such as country, device type, or subscription type could capture more behavioral variance and improve churn prediction accuracy. From an operational perspective, implementing model versioning in Google Cloud Storage would support traceability and rollback functionality. Furthermore, introducing drift detection and basic endpoint testing within the CI/CD pipeline would strengthen reliability and early error detection. Finally, integrating environment variables and Secret Manager for secure configuration would align the system with best practices for production-grade MLOps. Finally, incorporating systematic unit testing for components such as the Prediction API and the pipeline executor would strengthen the system's robustness and support production-grade reliability, ensuring early detection of logic errors and integration issues.

3.3 Reflection on Implementation Process

Building the end-to-end MLOps system provided valuable insights into automating the machine learning lifecycle. The main technical challenges occurred during Docker image creation and Cloud Build integration. Initial builds frequently failed due to missing dependencies and path mismatches between local and cloud environments, which were resolved by restructuring the Dockerfile and testing builds incrementally. Configuring Vertex AI Pipeline execution required additional troubleshooting, particularly in managing file paths and permissions when the executor accessed the pipeline YAML. Adjusting the workspace structure and verifying service-account roles eventually stabilized execution. Another challenge involved aligning substitution variables across Cloud Build steps; consistency in variable naming and scope solved recurring trigger errors. Overall, the project deepened the team's understanding of

4 Individual Contributions

The Spotify churn-prediction MLOps project was completed collaboratively by five group members, each responsible for specific technical and organizational components of the system. Clear task allocation ensured efficient workflow management and consistent integration across all modules.

Rick de Rijk: led the development of the Vertex AI pipeline, including the design of data preprocessing, training, and evaluation components. He implemented parameter tuning within the pipeline configuration and ensured that the pipeline artifacts and outputs were correctly managed in Google Cloud Storage.

Thom Verzantvoort: was responsible for the continuous integration and deployment setup. He configured the Cloud Build workflows, managed trigger automation, and integrated the pipeline-executor container to enable fully automated retraining and redeployment of the models.

Tycho van Rooij: focused on the implementation and refinement of the Prediction API and UI. He developed the RESTful interface, ensured end-to-end communication between the API and the front-end, and validated the correct model serving on Cloud Run.

Stefan Vonk: contributed to system testing, deployment verification, and the documentation of the overall MLOps workflow. He ensured that all services functioned correctly within the integrated cloud environment and supported debugging during the deployment phase.

Gilbert Laanen: coordinated report structuring and contributed to the technical writing, integrating system descriptions, pipeline explanations, and architectural visualizations into a coherent final document.

Through this structured division of responsibilities, the team achieved an automated, reproducible, and scalable MLOps solution that successfully integrates data processing, model training, continuous deployment, and user-facing prediction capabilities.

5 Technology Statement

During the preparation of this assignment, the group used **ChatGPT (GPT-5) & Claude** as a support tool to improve documentation quality, resolve technical issues, and enhance writing clarity. The tool was specifically applied to assist in:

- explaining and documenting the system design and CI/CD architecture,

- refining the structure and clarity of written report sections, and

- providing feedback on code documentation and debugging of Cloud Build and Vertex AI configurations.

All AI-assisted outputs were critically reviewed, verified, and edited by the group members, **Rick de Rijk, Thom Verzantvoort, Tycho van Rooij, Stefan Vonk, and Gilbert Laanen**, to ensure correctness, academic integrity, and compliance with the course's AI-Index Level 4 guidelines.

ChatGPT was used strictly as a supportive assistant, not as a content generator. The final analyses, designs, and interpretations were produced, validated, and approved by the team. Consequently, the authors take full responsibility for the accuracy and originality of all content included in this report.

6 References

Google Cloud. (2024). MLOps: Continuous delivery and automation pipelines in machine learning. Retrieved from https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Lauraalonso. (2025, 28 september). Spotify 2025 - EDA + prediction models. Kaggle. https://www.kaggle.com/code/2025-eda-prediction-models/input

Ranasinghe, H. (2021). Deploying powerful machine learning models on Google Cloud. Datatonic. Retrieved from https://datatonic.com/insights/deploying-machine-learning-models-google-cloud/

7 Appendix
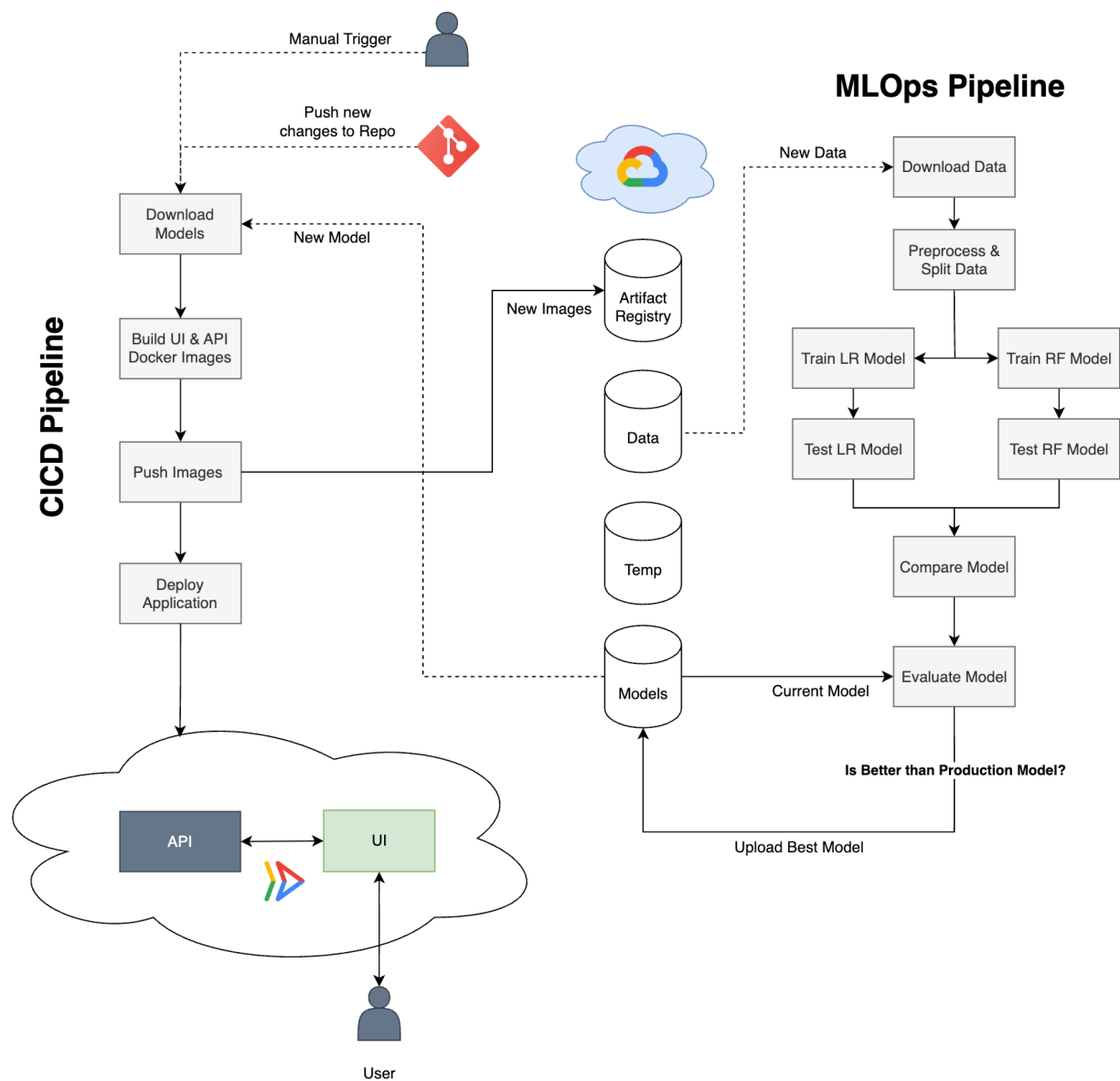
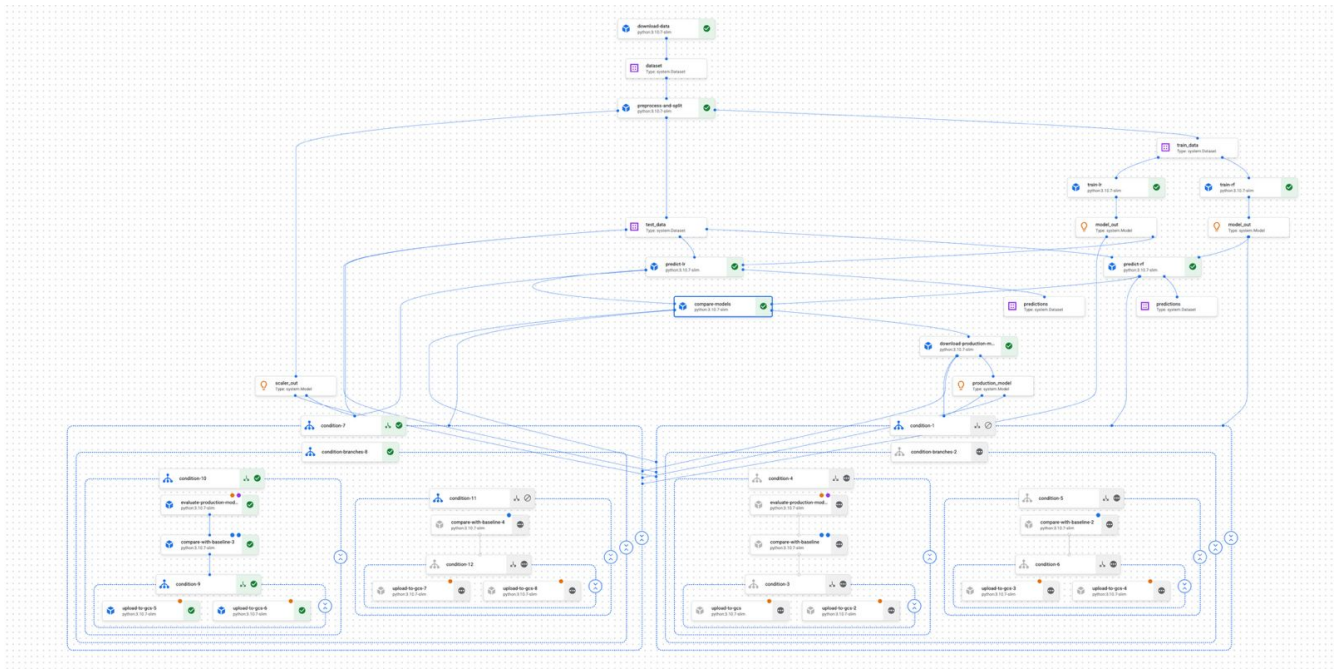Figure 1: The complete workflow of the Full MLOps System working in Production
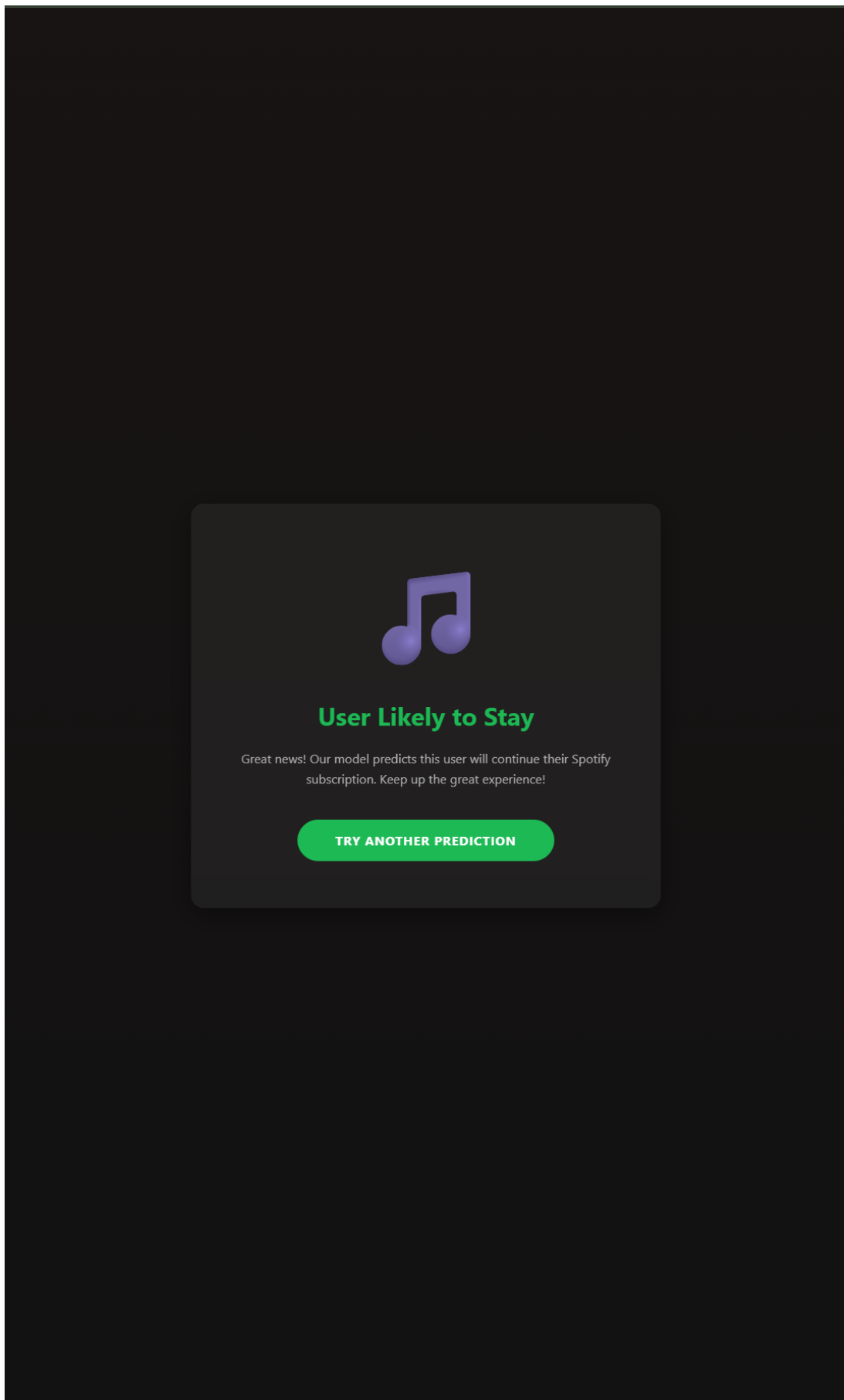
Figure 2: Vertex AI Pipeline

Figure 3: Prediction UI Input Spotify

Figure 4: Prediction-UI-Response-Spotify