SAMSUNG

AI Course

Capstone Project Final Report

For students (instructor review required)

Churn Prediction System for SaaS Products

05/12/2025

Mohammed Alabdulmuhsin
Wassayef Alkherb
Abdulrahman Alkhaldi
Mira Alfuraih
Hisham Alnabulsiyyah
Abdulazaz Aamri

Table Of Content

1. Introduction

1.1. Background Information

Our project, Stay Ahead, is centered on developing an intelligent churn prediction system for SaaS products. Subscription-based platforms often face significant revenue loss when users cancel their plans unexpectedly [1]. By analyzing customer demographics, account details, service usage trends, and billing behavior, we aim to understand the factors that contribute to churn. The project is built using the Telco Customer Churn dataset, which provides a structured foundation for training and evaluating machine learning models capable of estimating the likelihood of customer cancellation

1.2. Motivation and Objective

Churn is one of the most critical challenges for SaaS companies, as retaining existing customers is far more cost-effective than acquiring new ones [2]. This motivated our team to design a system that can identify high-risk users early, allowing product owners to take proactive actions such as offering personalized discounts or extending subscriptions. The main objective of the project is to build a reliable machine learning system that assigns a churn-risk probability to each subscriber and ranks them, accordingly, enabling better decision-making and more effective retention strategies.

1.3. Members and Role Assignments

Our team is composed of six members, each contributing a clearly defined role to ensure smooth progress throughout the project. Hisham Alnabulsiyyah leads the frontend development and is responsible for building the dashboard interface. Abdulaziz Aamri handles backend development and model integration. Abdulrahman Alkhaldi serves as the Data Analyst, managing exploration, cleaning, and feature engineering. Wassayef Alkherb, Mohammed Alabdulmuhsin, and Mira Alfuraih work as Machine Learning Engineers, focusing on model development, hyperparameter tuning, and performance optimization. This task distribution ensures full coverage of the project's technical and operational requirements.

1.4. Schedule and Milestones

The project follows a structured three-week timeline. During Week 1, the team focuses on defining the scope, conducting exploratory data analysis, cleaning the dataset, and performing feature engineering. Week 2 centers on model development -training baseline and advanced models, tuning hyperparameters, optimizing thresholds, and comparing performance results. Week 3 is dedicated to deployment, where the selected model is integrated into a functional dashboard, followed by final testing, documentation preparation, and presentation development. This schedule ensures balanced progress across all stages of the project. A more detailed timeline can be found in the project schedule document:

https://1drv.ms/x/c/063A949A68992A91/IQAr5KhYY4nyTqlx_4CJbQ6uATzvrThUz3zMiJ1kkBZS9vQ

2. Project Execution

2.1. Data Acquisition

The dataset used in this project is the **Telco Customer Churn Dataset**, originally published on Kaggle by IBM Watson Analytics. It is publicly available at: https://www.kaggle.com/datasets/blastchar/telco-customer-churn

This dataset contains 7,043 customer records, each describing a telecom subscriber's demographic information, account details, billing type, service usage, and whether the customer churned or remained active. The data was acquired directly through the Kaggle API using the kagglehub Python library for reproducibility and automated access in the analysis pipeline. The dataset is widely used for churn prediction studies due to its balanced combination of categorical, numerical, and service-related attributes, making it suitable for evaluating the performance of different machine learning models.

2.2. Training Methodology

The training methodology followed a structured machine-learning pipeline designed to ensure fair evaluation and model robustness. The dataset was first split into 80% training and 20% testing using stratified sampling to preserve the original churn distribution. All experiments were performed on the training portion, while the test set was kept strictly unseen until final evaluation. To prevent data leakage and ensure consistent preprocessing, a unified pipeline was implemented consisting of numerical standardization and categorical one-hot encoding, managed through a ColumnTransformer. Training mainly relied on 5-fold Stratified Cross-Validation, allowing every model to be trained and validated across multiple data partitions. This approach reduces overfitting, provides more reliable performance estimates, and ensures that results are not dependent on a single split. Hyperparameter tuning for advanced models (e.g., XGBoost) was performed using Bayesian optimization sweeps, while baseline models (Logistic Regression, Random Forest) used their standard or grid-searched configurations. The final performance metrics for each model were reported on the untouched test set to ensure unbiased comparison.

2.3. Workflow

The overall workflow of the project followed a sequential and systematic machine-learning pipeline to ensure reproducibility, consistency, and fair model comparison. The process began with data acquisition from the Kaggle Telco Customer Churn dataset, followed by preprocessing, which included handling missing values, feature renaming, encoding categorical features, and scaling numeric ones. Afterward, the dataset was split into training and testing sets using stratified sampling to preserve churn distribution. The training phase relied on a unified preprocessing pipeline combined with 5-fold Stratified Cross-Validation, allowing each model to be trained and validated multiple times for robust performance estimation. Baseline models were trained first, followed by more advanced architectures such as Random Forest, XGBoost, and NODE. For XGBoost, a dedicated hyperparameter sweep workflow was applied to identify optimal configurations. Once training was complete, all models were evaluated on the untouched test set

using consistent metrics (ROC-AUC, Recall, Precision, F1-score, and Accuracy). Finally, results were visualized through ROC curves, confusion matrices, and summary tables, forming the basis for comparison in the modeling section.

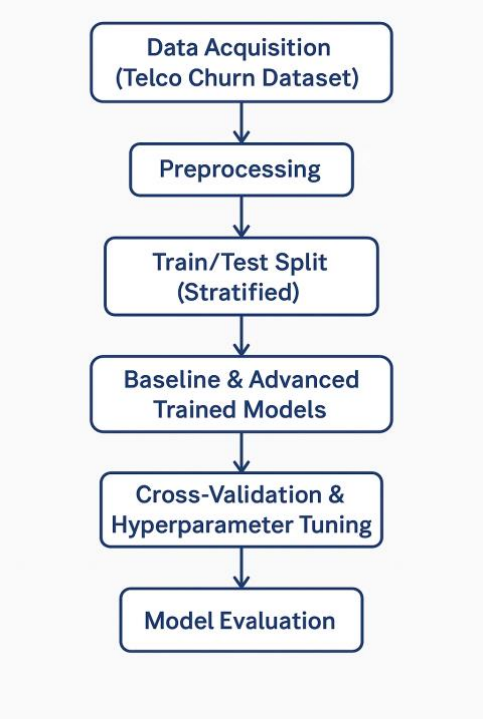


Figure 2.1: General Workflow

2.4. System Diagram

The system is designed as a structured end-to-end pipeline that transforms raw customer data into actionable churn-risk insights. As shown in the diagram, the workflow begins with data ingestion, followed by preprocessing and feature engineering to clean, encode, and prepare the dataset. The processed data then flows into the model-training stage, where multiple algorithms are evaluated and optimized. Afterward, the system performs prediction and evaluation, generating churn probabilities and ranking users by risk. Finally, the outputs integrate into a web-application dashboard, enabling stakeholders to monitor high-risk customers and take proactive retention actions.

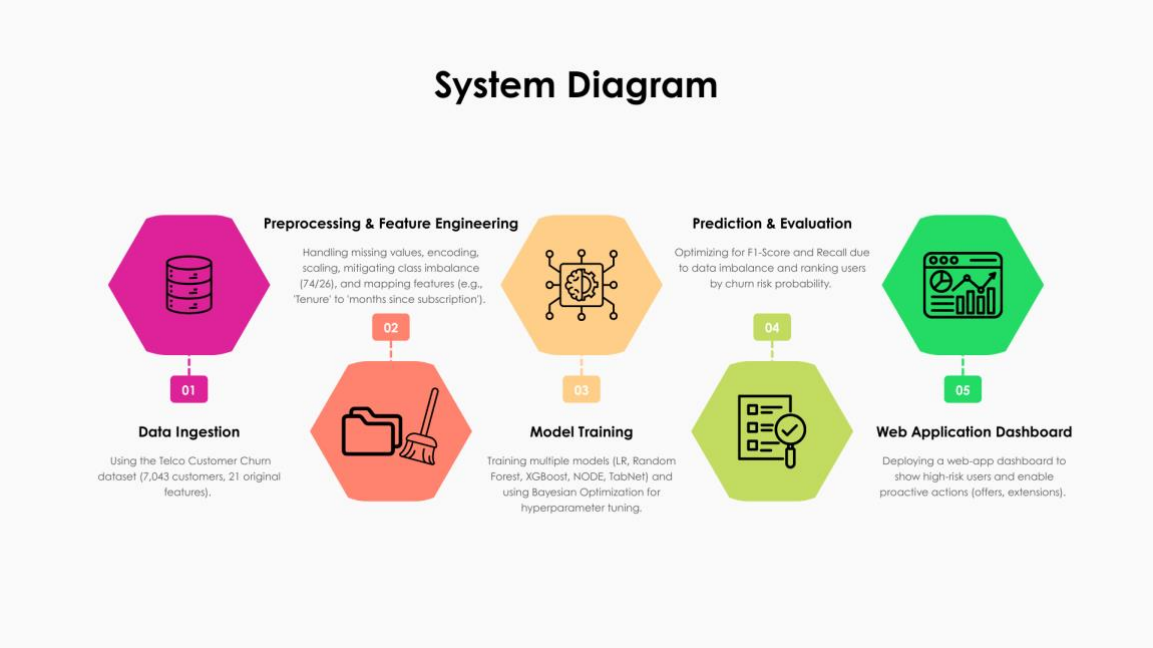


Figure 2.2: General System Diagram

3. Results

3.1. Data Preprocessing

We first created a cleaned version of the data (df_clean) and converted the TotalCharges column from string to numeric. Rows with invalid or blank TotalCharges values (11 customers) were removed, and we confirmed that there are no duplicate rows and that each customerID is unique.

Next, we built a modeling dataset (df_model), encoded the target variable Churn as 0/1, and dropped the customerID column since it is only an identifier and does not carry predictive information. Finally, we separated features and target (X and y), identified numeric and categorical variables, and applied one-hot encoding to the categorical features to obtain a fully numeric dataset ready for modeling. Figure 3.1 below shows a general data preprocessing pipeline.

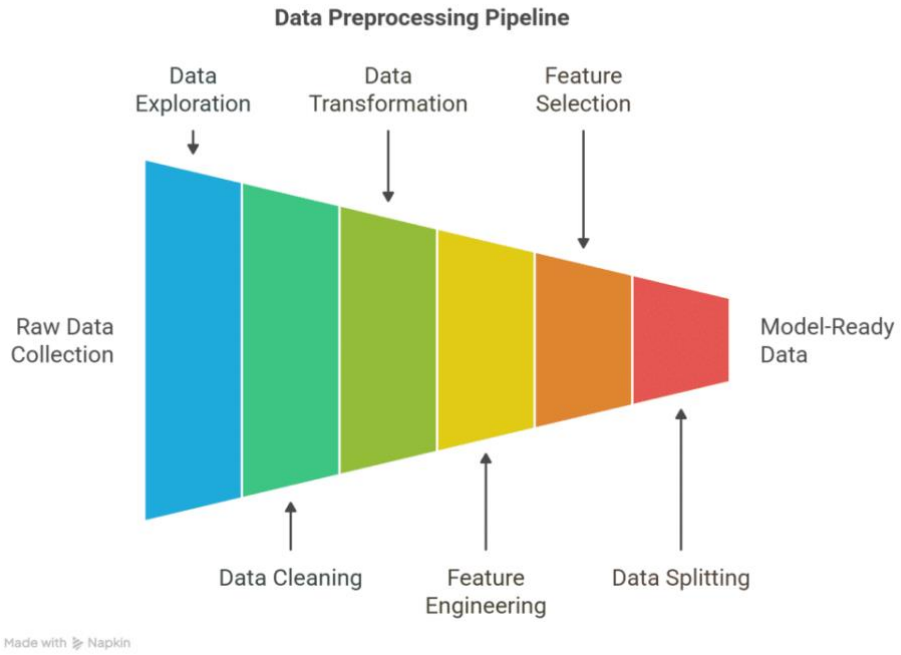


Figure 3.1: Data preprocessing pipeline visualization [4]

3.2. Exploratory Data Analysis (EDA)

The exploratory data analysis focused on understanding churn behaviour and identifying the main factors associated with customer attrition.

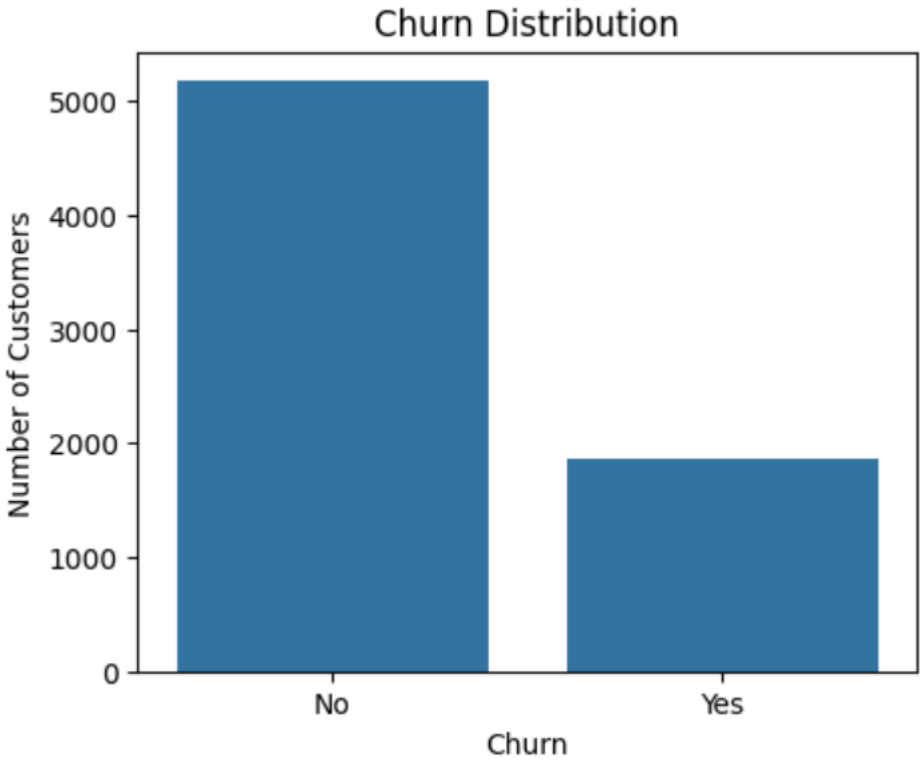


Figure 3.2: Churn Distribution

The overall churn distribution in Figure 3.2 shows that roughly one quarter of the customers churn, confirming that churn is a relevant business problem for this subscription setting. Furthermore, this plot shows a clear unbalance in the target variable, which requires careful handling.

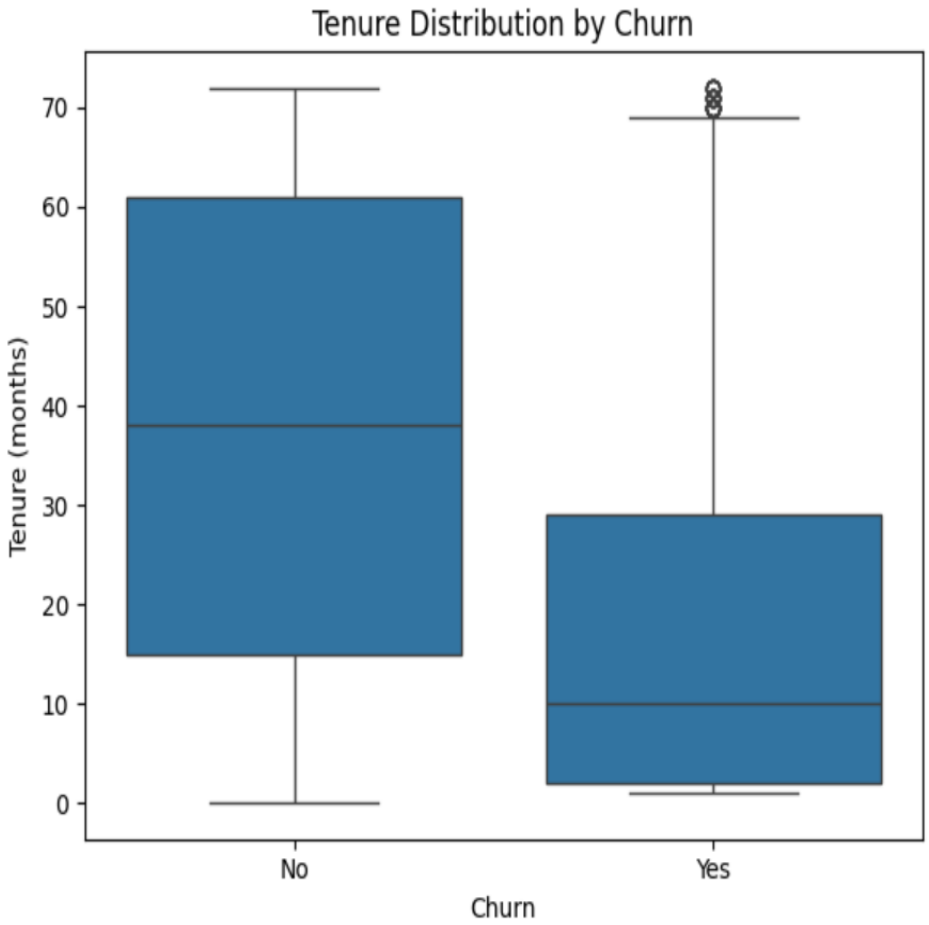


Figure 3.3: Tenure Distribution by Churn (boxplot)

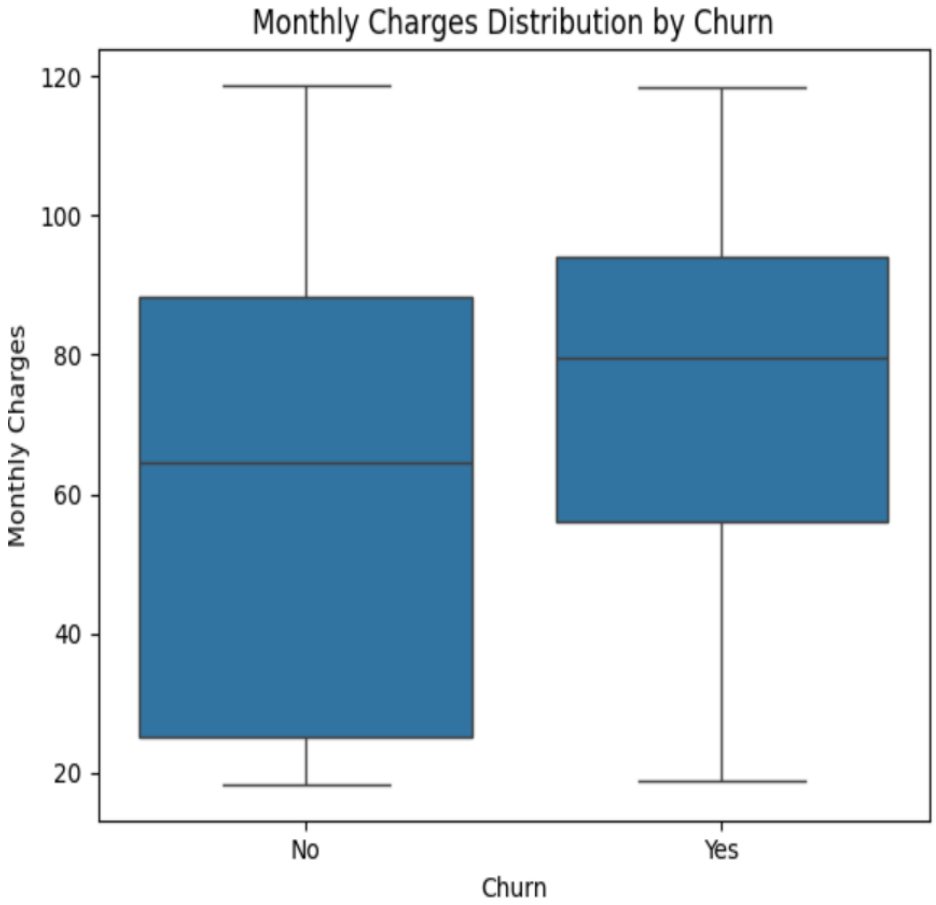


Figure 3.4: Monthly Charges Distribution by Churn (boxplot)

We first examined the main numerical features related to customer lifetime and billing. The boxplots of tenure and MonthlyCharges by churn status in Figures 3.3 and 3.4, respectively, reveal clear differences between churned and non-churned customers. Churned customers tend to have lower tenure, indicating that users are more likely to cancel during the early months of their subscription. At the same time, churned customers generally pay higher monthly charges, suggesting that expensive plans are more sensitive to churn and may require special attention (e.g., discounts or plan adjustments) to retain users.

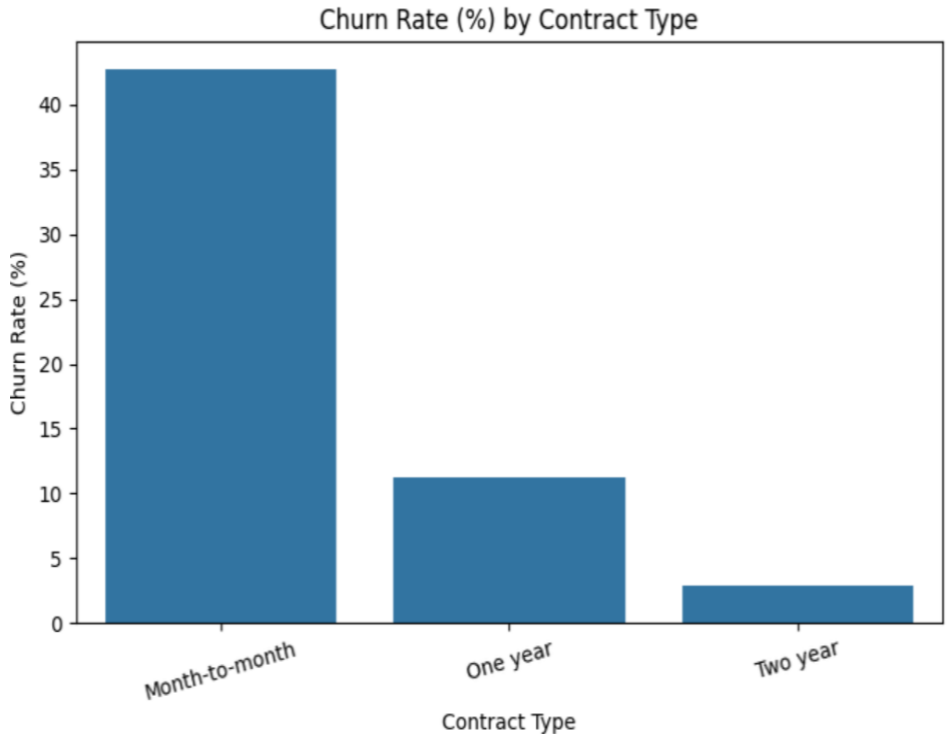


Figure 3.5: Churn by Contract Type

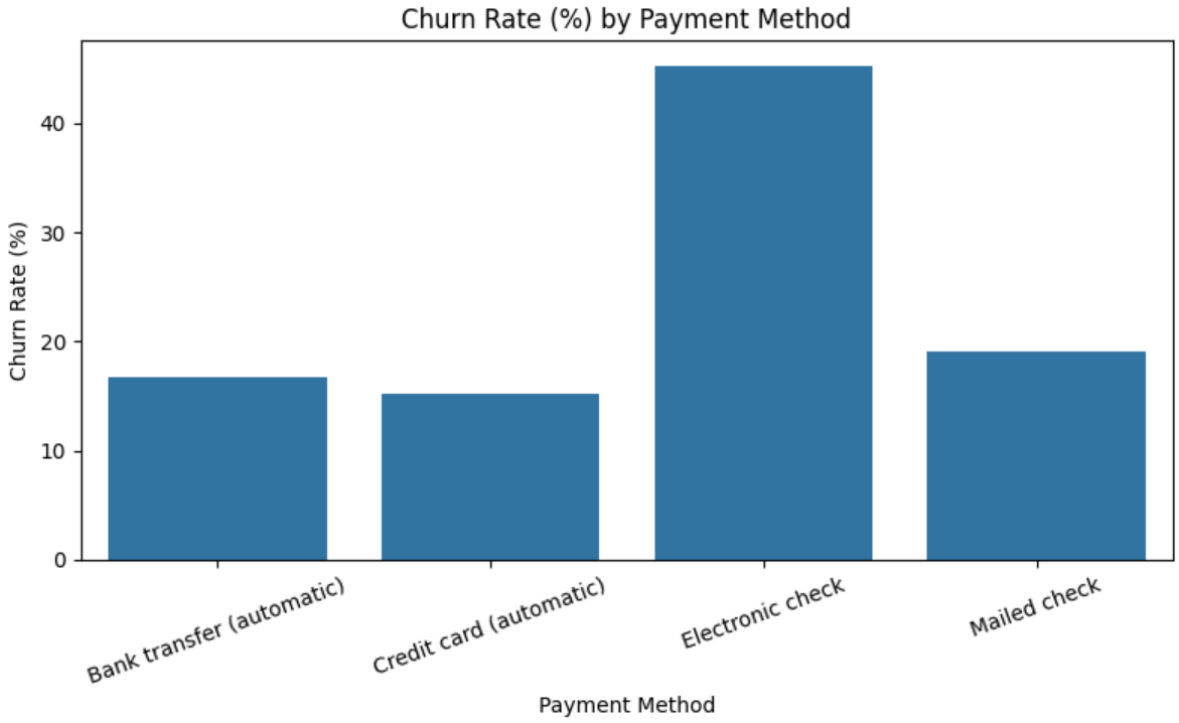


Figure 3.6: Churn by Payment Method

Next, we analyzed key categorical features that can be directly acted upon by a SaaS product owner. The plots for Contract and PaymentMethod (Figures 3.5 and 3.6) show that customers on month-to-month contracts and those paying via Electronic check exhibit noticeably higher churn rates, whereas customers with longer contracts (one or two-year) are much more stable.

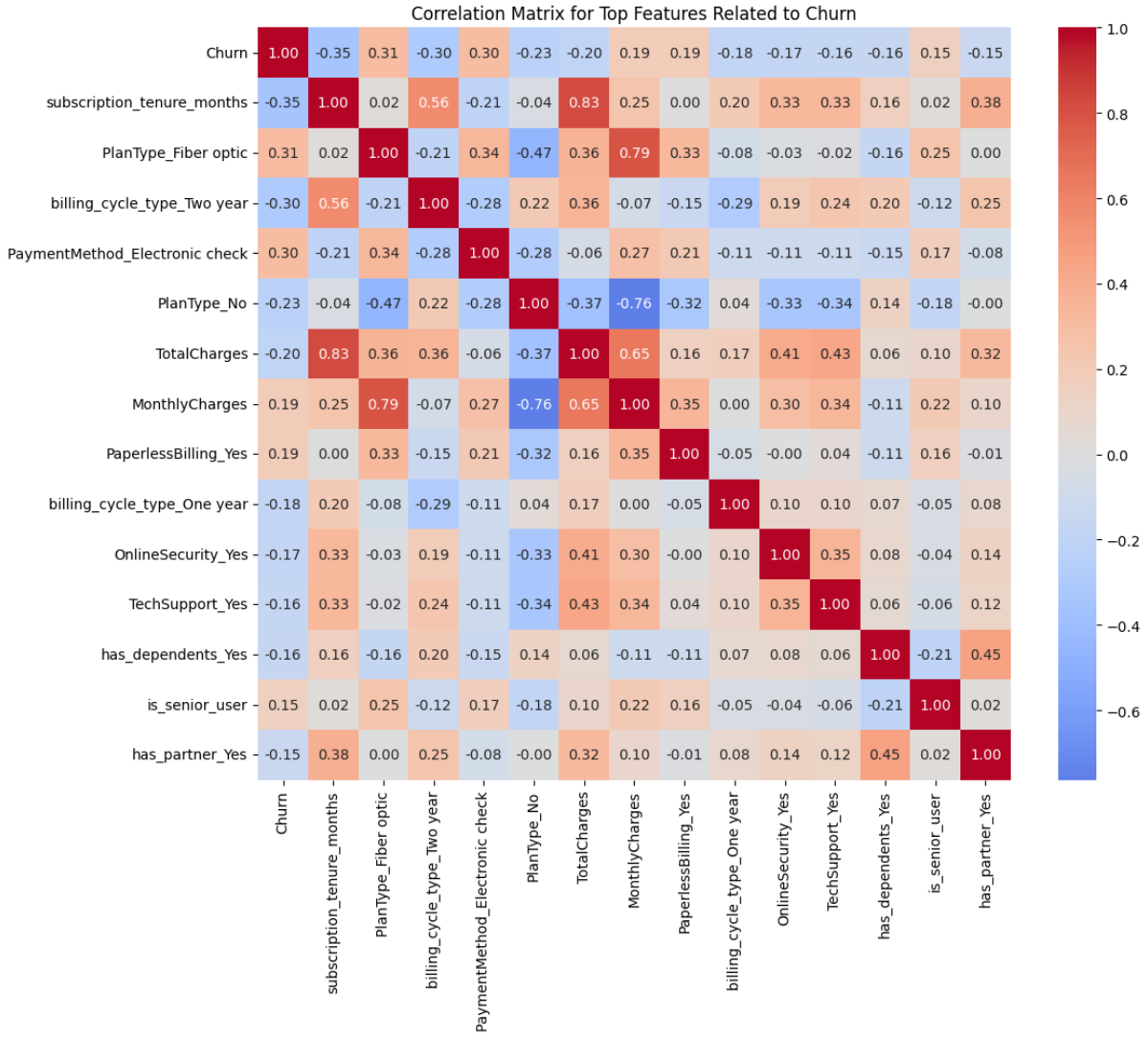


Figure 3.7: Correlation Matrix for Top Features Related to Churn

To summarize these relationships numerically, we computed a correlation matrix on the one-hot encoded dataset and visualized the top features most related to churn (Figure 3.7). The heatmap shows that churn is negatively correlated with tenure, TotalCharges, and two-year contracts, reinforcing the idea that long-term, committed customers are less likely to leave. In contrast, churn is positively correlated with Fiber optic internet, higher monthly charges, and Electronic check payments, highlighting higher-risk segments. Overall, the EDA provides a clear picture of which customer groups are most vulnerable to churn and offers concrete guidance on where a SaaS owner should focus retention efforts.

3.3. Modeling

We will experiment with five models, from different modelling families:

- Logistic Regression and Random Forest as a simple baseline
- XGBoost as a boosting model
- NODE (Neural Oblivious Decision Ensembles) and TabNet as a modern deep tabular model

3.3.1 Logistic Regression

This model was used to predict customer churn by estimating the probability of churn for each customer.

It relies on a linear relationship between the input features, which is then transformed into a probability used for classification.

Logistic Regression allows for a clear and interpretable understanding of the factors influencing customer churn.

- Default model

The model was first trained using its default configuration to establish an initial performance level. These baseline results were used as a point of comparison to evaluate the impact of the optimization steps applied later.

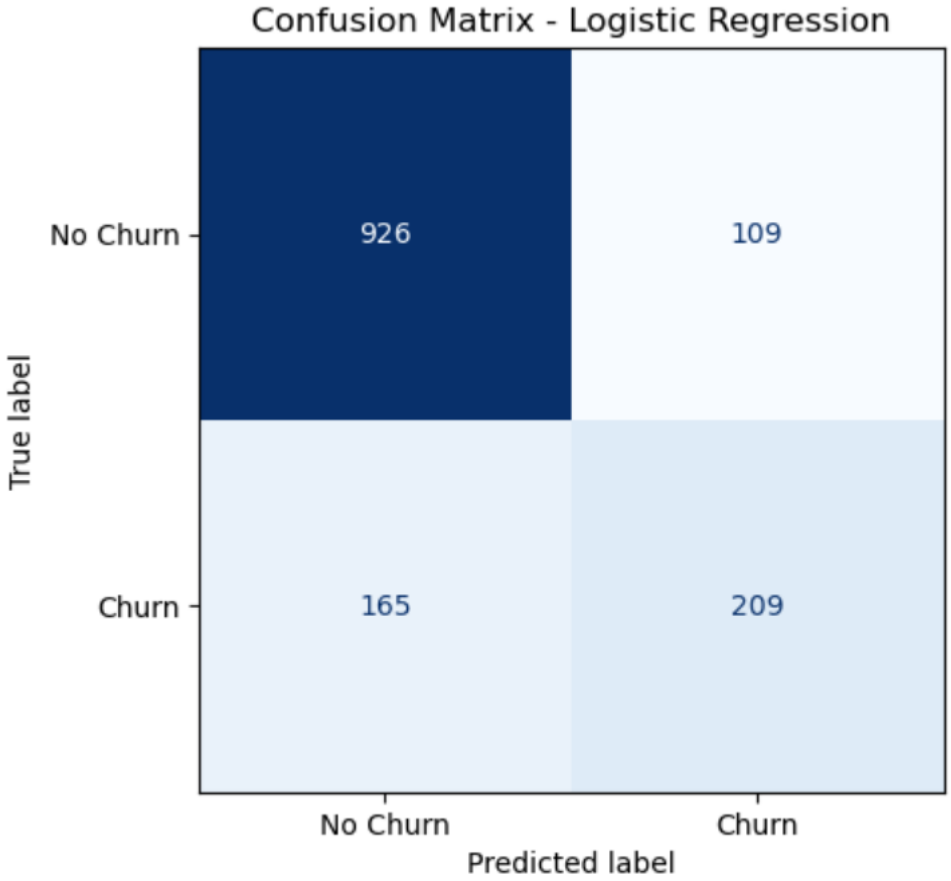


Figure 18: Confusion matrix - Default Logistic Regression model

The confusion matrix in Figure 3.8 above indicates that the model correctly predicts most non-churn cases (926), but misclassifies 109 of them as churn. On the churn side, 209 cases were correctly detected, while 165 churn customers were incorrectly predicted as non-churn, showing room for improvement in capturing churn behavior.

Table 11: report - Default Logistic Regression model

Recall	Precision	F1-score	Accuracy
0.56	0.66	0.60	0.81

These results indicate that the model, before applying any optimization, maintains a reasonable balance between detecting churn cases and preserving overall accuracy.

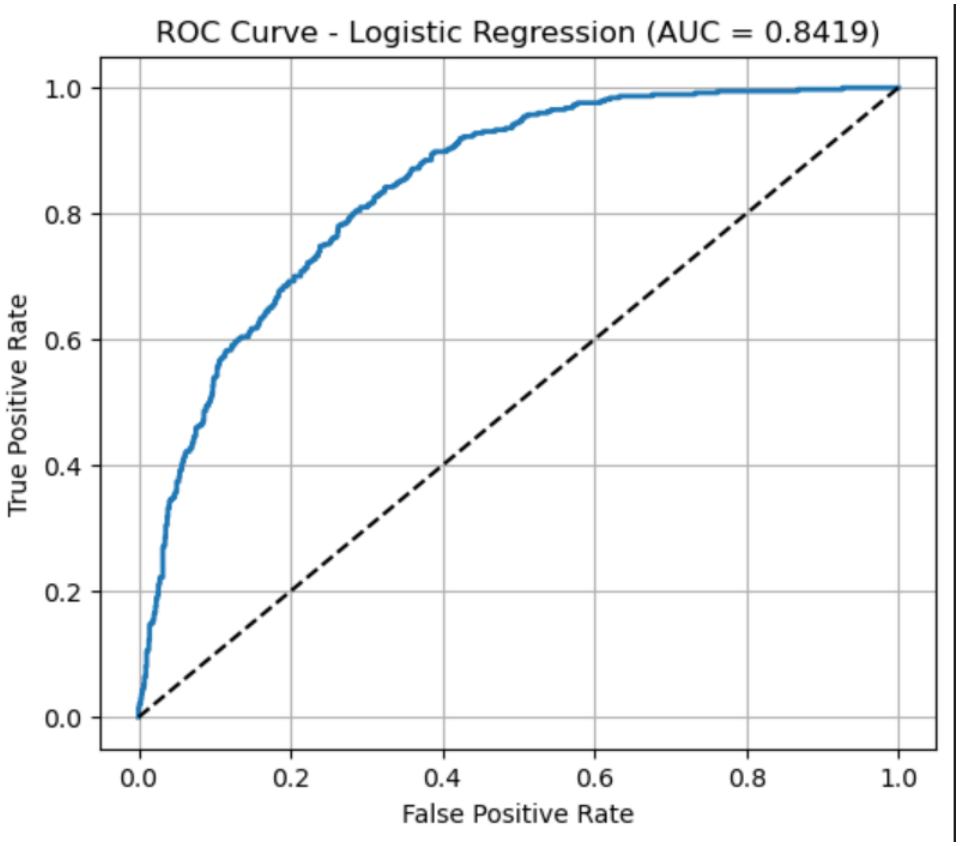


Figure 29: ROC curve - Default Logistic Regression model

The ROC curve in Figure 3.9 shows strong discriminative ability, achieving an AUC of 0.84. This indicates that the Logistic Regression model is able to separate churn and non-churn customers across different threshold values.

- Optimized model

After evaluating the default configuration, a set of improvements was applied to enhance the model's ability to detect churn cases.

These improvements included the use of Stratified K-Fold Cross-Validation, handling class imbalance through class weighting, performing hyperparameter tuning, and adjusting the classification threshold.

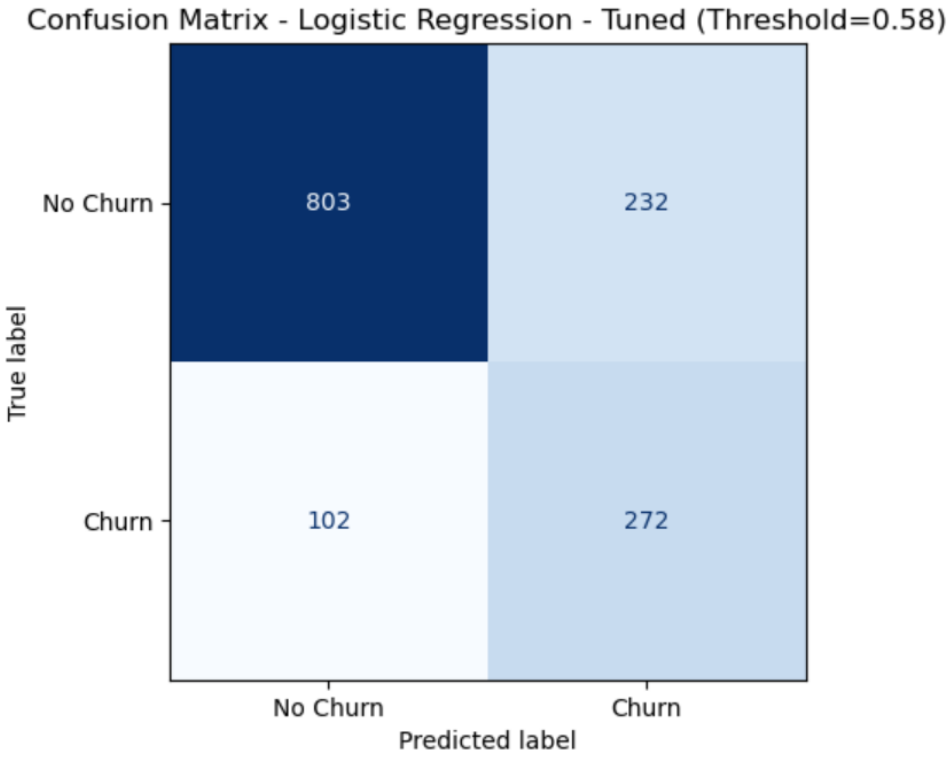


Figure 3.10: Confusion matrix - Tuned Logistic Regression model

The confusion matrix shows that the tuned model correctly identifies a higher number of churn customers compared to the default configuration, with 272 true churn predictions.
Although some non-churn customers are still misclassified, the model demonstrates improved sensitivity toward churn detection after optimization.

Table 2: report - Tuned Logistic Regression model

Recall	Precision	F1-score	Accuracy
0.73	0.54	0.62	0.76

After optimization, the model achieves a higher recall of 0.73 for churn customers, indicating improved detection of churn cases.
This improvement comes with a slight decrease in accuracy (0.76), reflecting an expected trade-off when prioritizing churn identification in imbalanced data.

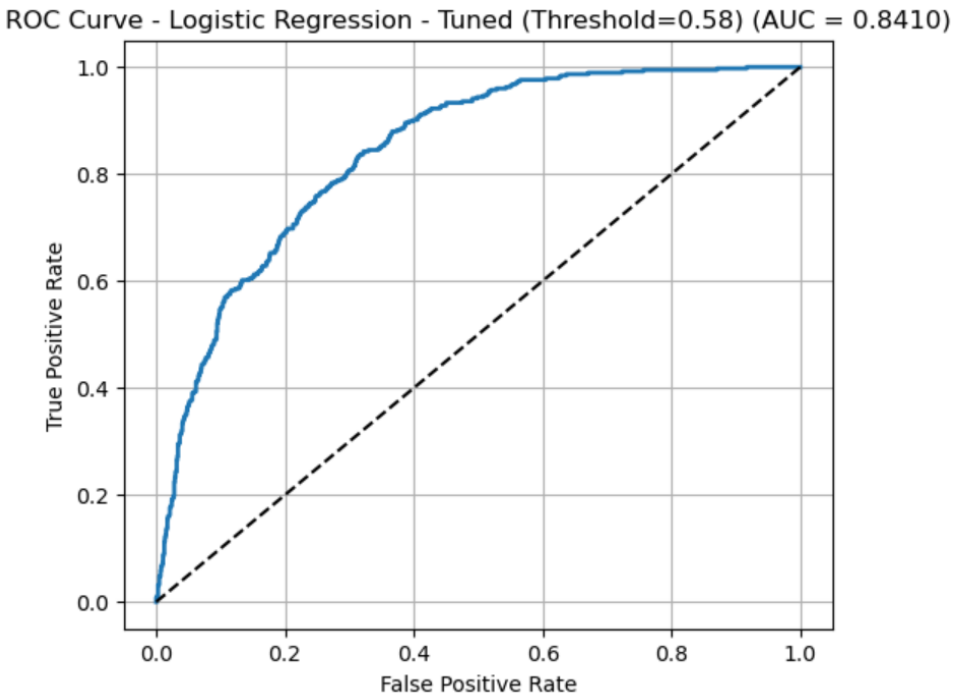


Figure 3.11: ROC curve - Tuned Logistic Regression model

The ROC curve indicates strong discriminative performance, with an AUC value of 0.84, showing that the optimized model maintains good class separation.

Compared to the default configuration, the model sustains reliable overall performance while focusing more on detecting churn customers.

3.3.2 Random Forest

To build a stronger baseline model, we used Random Forest. This ensemble method combines multiple decision trees, which helps reduce overfitting and capture non-linear patterns in the data. Random Forest is generally more robust than a single tree and often performs well on tabular datasets like ours.

- Default model

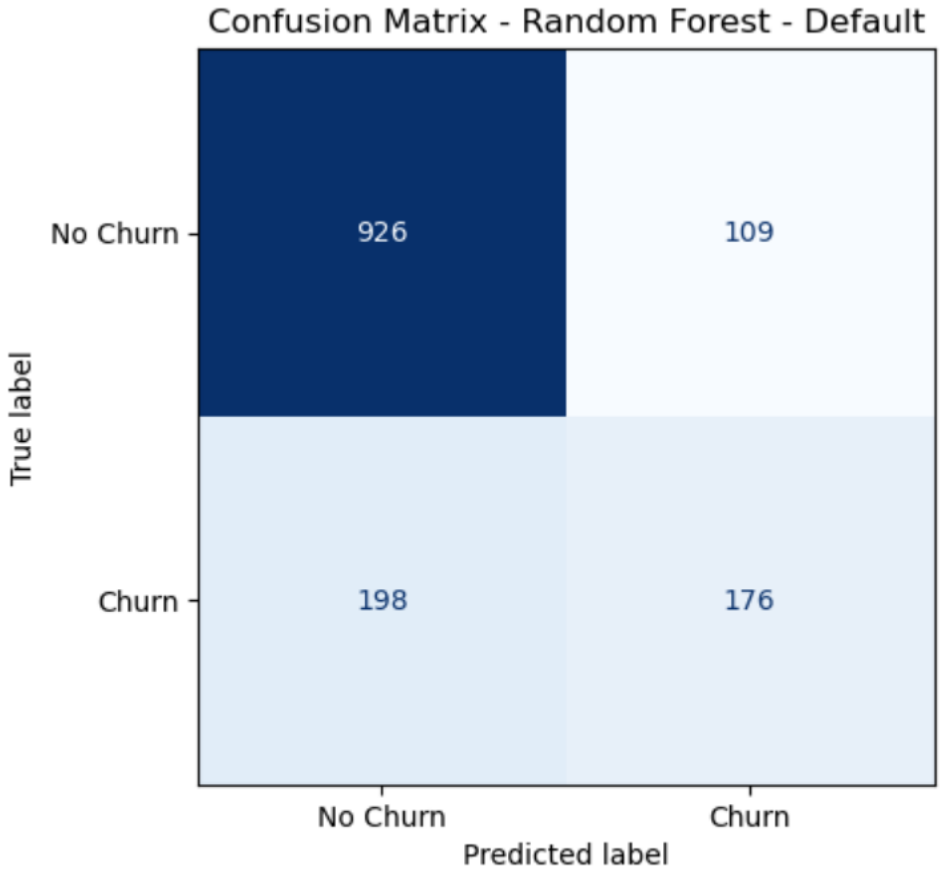


Figure 4: Confusion matrix - Default Random Forest model

The confusion matrix shows that default Random Forest correctly classified 926 non-churn customers and 176 churn customers, while 198 churn cases were misclassified as non-churn.

Table 3: report - Default Random Forest model

Recall	Precision	F1-score	Accuracy
0.47	0.62	0.53	0.78

The default Random Forest model shows moderate performance, achieving an accuracy of 0.78 and an F1-score of 0.53. The relatively low recall (0.47) indicates that many churn customers are still missed, highlighting the need for further optimization.

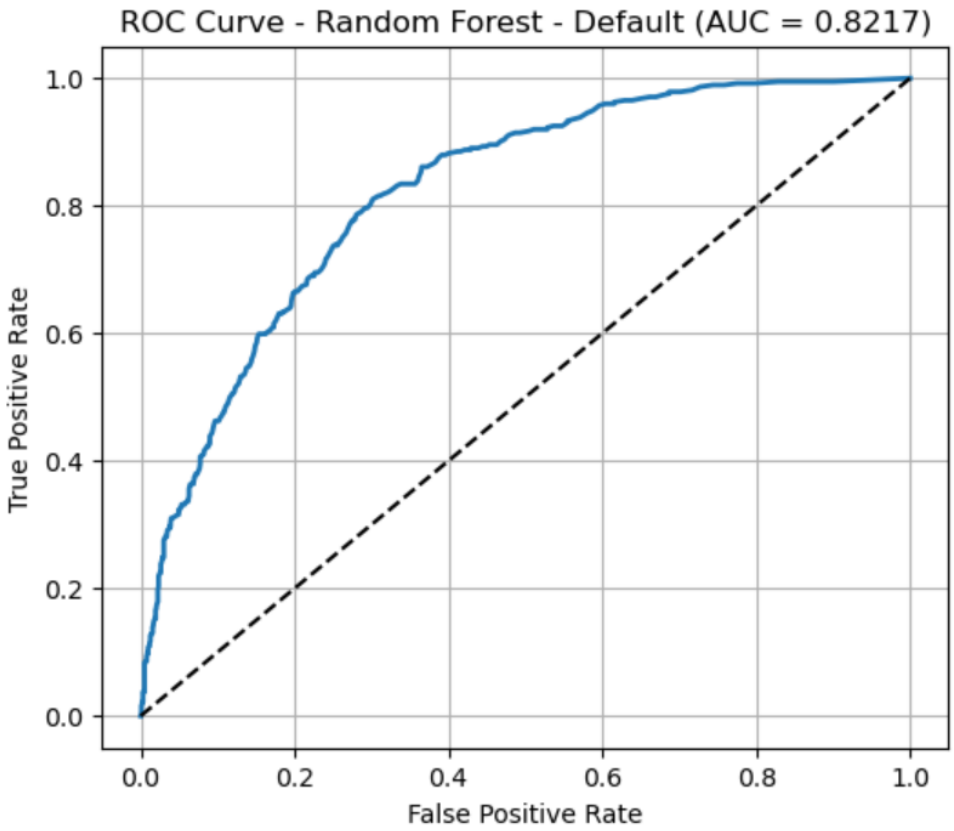


Figure 5.13: ROC curve - Default Random Forest model

The ROC curve shows that the default Random Forest achieves an AUC of 0.8217, reflecting good class separation performance before applying any optimization.

- Optimized model

To **improve** the model performance, we applied hyperparameter tuning using RandomizedSearchCV. The tuning process focused on optimizing key Random Forest parameters such as n_estimators, max_depth, min_samples_split, min_samples_leaf, max_features, and bootstrap, using F1-score as the evaluation metric. The following results present the performance of the tuned Random Forest model on the test set.

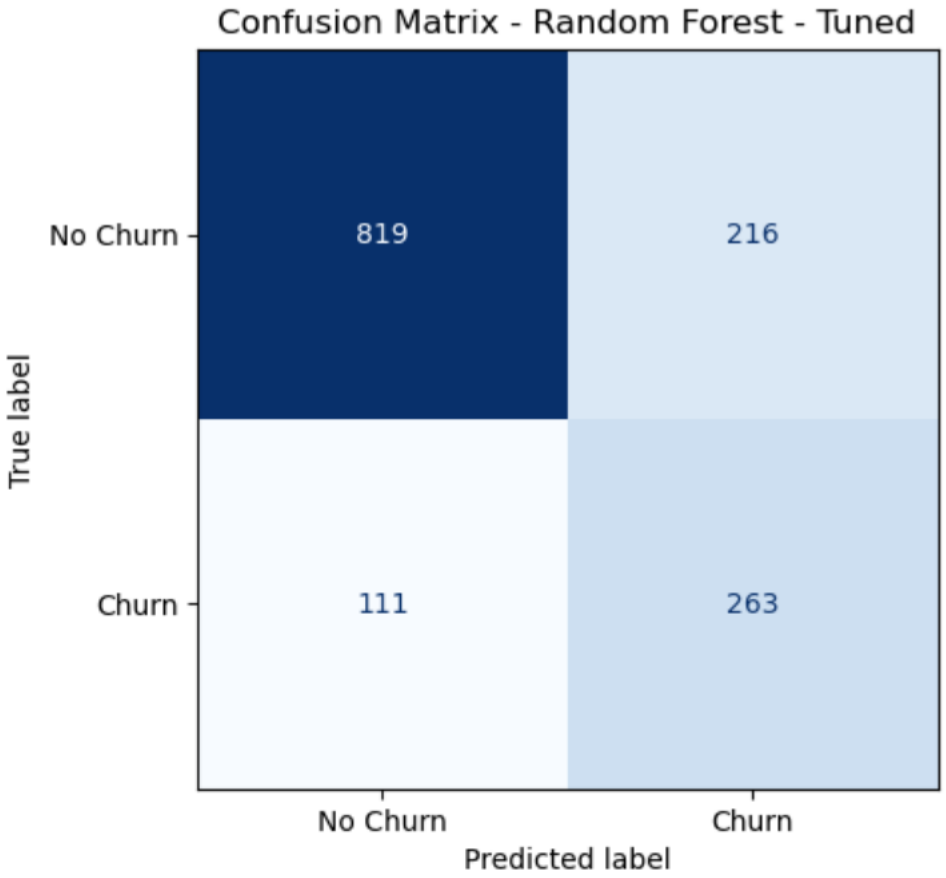


Figure 64: Confusion matrix - Tuned Random Forest model

The confusion matrix in Figure 3.14 shows that the tuned Random Forest correctly identified 263 churn customers and 819 non-churn customers.
Compared to the default model, false negatives were reduced, indicating an improvement in detecting actual churn cases.

Table 4.4: report - Tuned Random Forest model

Recall	Precision	F1-score	Accuracy
0.70	0.55	0.62	0.77

After hyperparameter tuning, the Random Forest model showed improved overall balance, cas shown in Table 3.4.
The model achieved 77% accuracy, with an F1-score of 0.62 for the churn class, reflecting better trade-off between precision and recall compared to the default configuration.

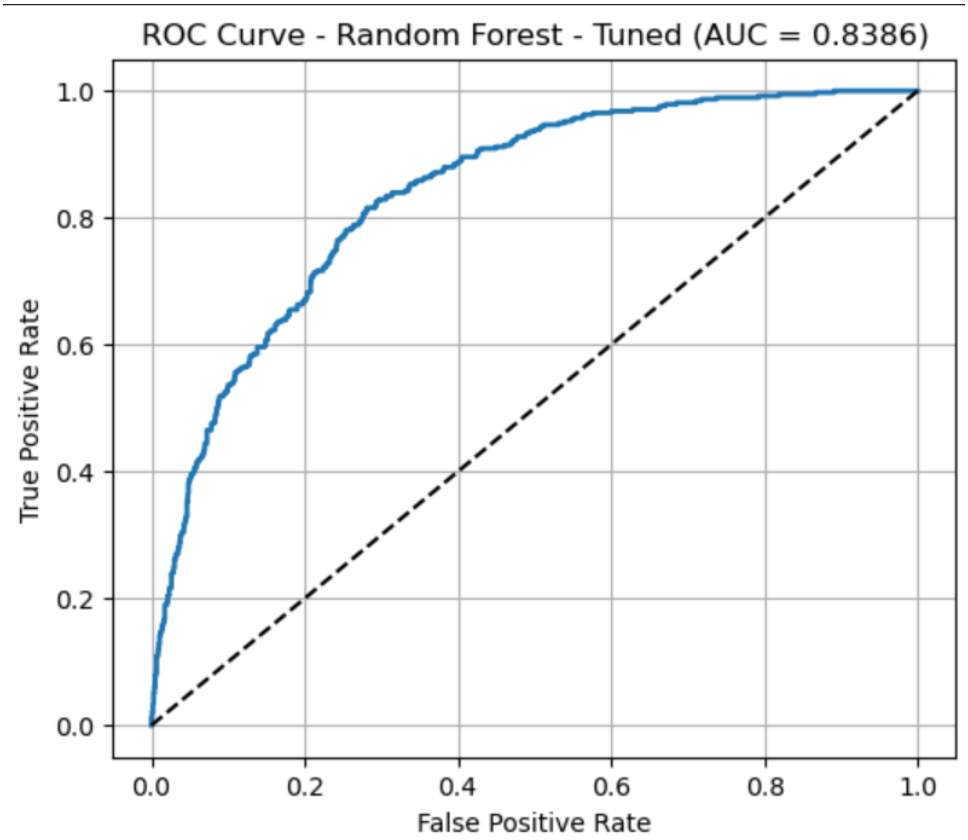


Figure 3.15: ROC curve - Tuned Random Forest model

The ROC curve of the tuned Random Forest model achieved an AUC of 0.8386, indicating strong discriminative ability.

This reflects an improvement over the default model in separating churn and non-churn customers across different thresholds.

3.3.3 XGBoost

XGBoost was selected as a boosting-based model due to its strong performance on tabular datasets and its ability to capture complex non-linear interactions between features. Compared to the baseline models, XGBoost enhances learning by iteratively correcting previous errors, making it highly effective for imbalanced problems such as churn prediction.

- Optimized model

We applied Bayesian hyperparameter optimization to find an optimal configuration, followed by full model retraining. This allowed the model to improve recall while maintaining balanced predictive behavior across classes. The selected configuration (max_depth = 2, γ = 3.88, n_estimators = 300, learning_rate = 0.207) represents the best trade-off discovered in the 5577th hyperparameter sweep using Weights & Biases for monitoring and sweeping.

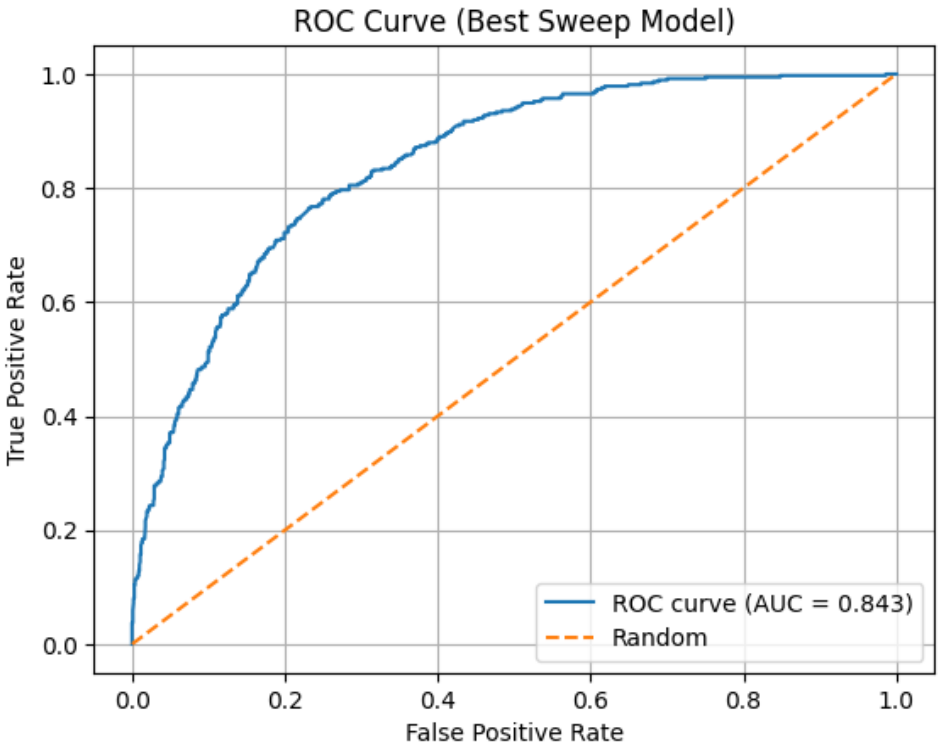


Figure 3.16: ROC Curve – XGBoost

The ROC curve in Figure 3.16 clearly demonstrates a good separability between churn and non-churn customers, achieving an AUC of **0.8433**, which is slightly higher than both Logistic Regression (AUC ≈ 0.84) and Random Forest (AUC ≈ 0.8386) as shown in the report.
This indicates that XGBoost maintains a consistently high true-positive rate across a wide range of thresholds, confirming its robustness and reliability as a classifier. The smooth upward curve also suggests stable probability estimates, which aligns with the model's boosting-based learning strategy.

Table 3.5: Report - XGBoost

Recall	Precision	F1-score	Accuracy
0.7968	0.5059	0.6189	0.7395

Overall, XGBoost delivers strong performance with an excellent recall of **79.68%,** which is close to Random Forest's recall while maintaining comparable stability. Its precision **(50.59%)** is moderate, reflecting a trade-off where the model prioritizes identifying as many churn customers as possible. The F1-score **(0.6189)** shows that the model achieves a balanced compromise between capturing churners and controlling false positives. Although the accuracy **(73.95%)** is slightly lower than Logistic Regression, the improvement in recall makes XGBoost more suitable for churn detection, where missing a churn customer is more costly than predicting a few additional false positives.

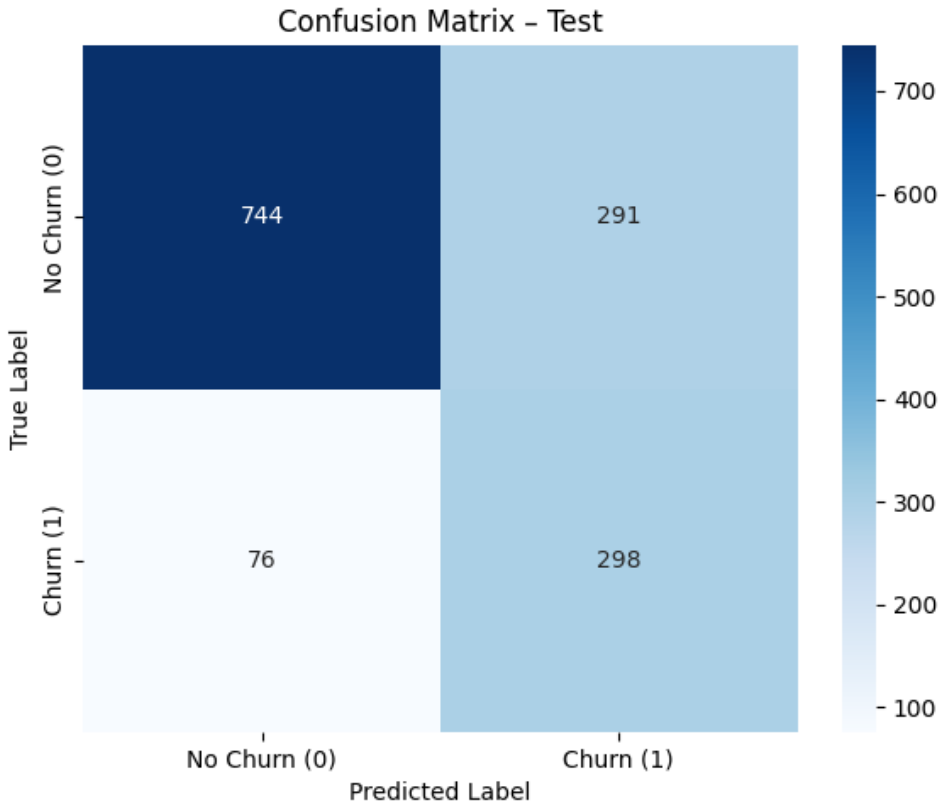


Figure 3.17: Confusion Matrix - XGBoost

The confusion matrix in Figure 3.17 reveals that the XGBoost model correctly identifies **298 churn customers**, reducing false negatives compared to the baseline Logistic Regression model (FN = 114).

This improvement directly contributes to its higher recall. However, the model produces **291 false positives**, which explains its moderate precision. This trade-off is expected in churn prediction, where the priority is minimizing false negatives to avoid losing customers. The matrix confirms that XGBoost maintains a strong balance between detecting churners and preserving overall classification stability, outperforming earlier baseline models while remaining competitive with more advanced architectures like NODE.

3.3.4 NODE (Neural Oblivious Decision Ensembles)

To test with a modern tabular model, we decided to use NODE because it blends the interpretability of tree structures with the flexibility of neural networks.

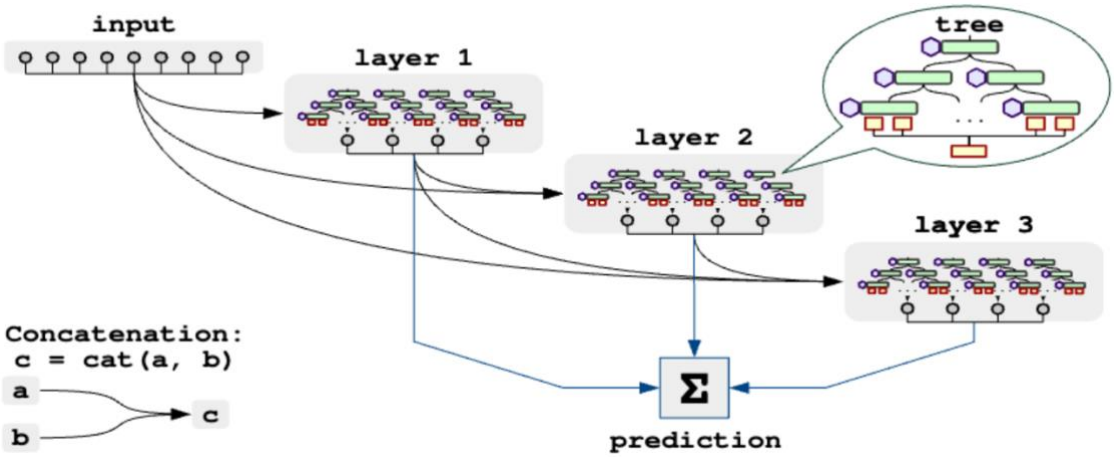


Figure 718: NODE Model Architecture [3]

As shown in Figure 3.18, the NODE architecture is composed of multiple stacked layers of oblivious decision trees. Each layer receives the original input features along with the outputs from previous layers, allowing the model to gradually build richer, more expressive representations of the data.

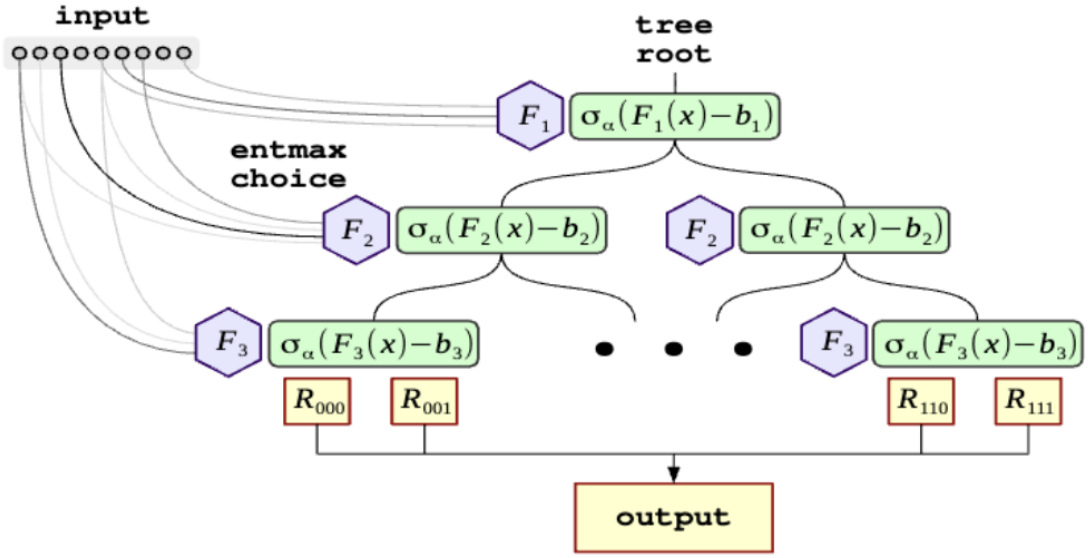


Figure 8: Internal Structure of a NODE Oblivious Decision Tree [3]

Each individual tree, illustrated in Figure 3.19, uses feature-selection functions and entmax gating to softly route the input through all tree paths. Unlike classical decision trees that make hard splits, NODE trees evaluate all branches simultaneously, generating weighted leaf outputs. These outputs are then combined across all trees and layers, and finally aggregated to produce the model's prediction.

- Default model

At first, we started with a baseline for NODE by training it on our dataset, without applying any balancing techniques. Our goal is not to achieve high performance, but to understand how well the model will perform.

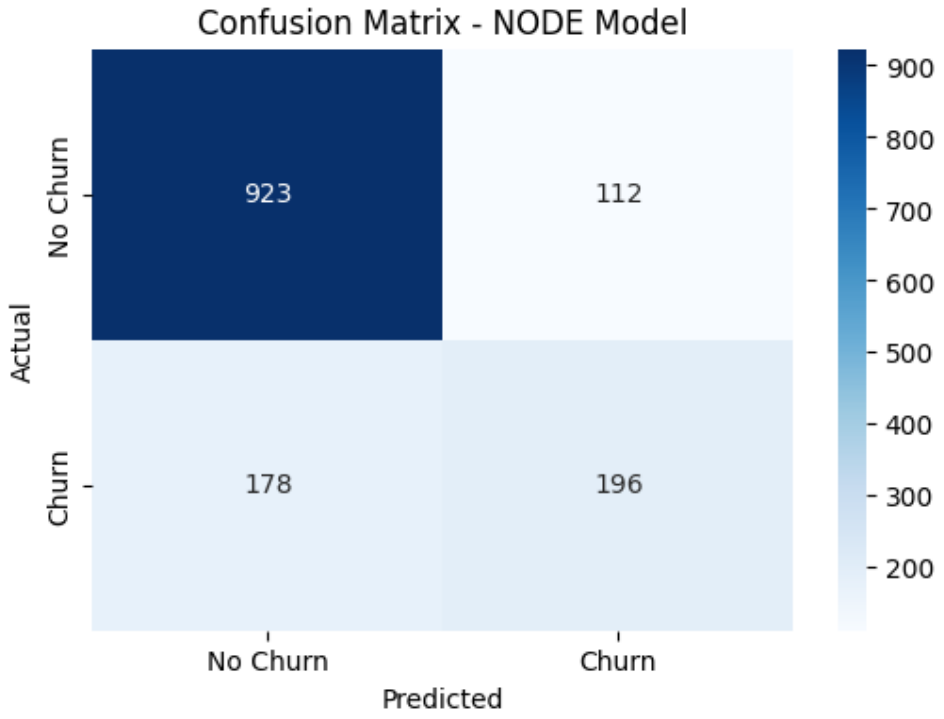


Figure 3.20: Confusion matrix - Default NODE model

The results showed a slight limitation, especially for the minority class as shown in Figure 3.20, since we had an imbalanced dataset.

Table 56: report - Default NODE model

Recall	Precision	F1-score	Accuracy
52%	64%	57%	79%

Moreover, for the recall, having a 52% indicates that the model missed a noticeable portion of actual churn customers. Also, the F1 score reflected the trade-off between the low recall and high precision (See table X)

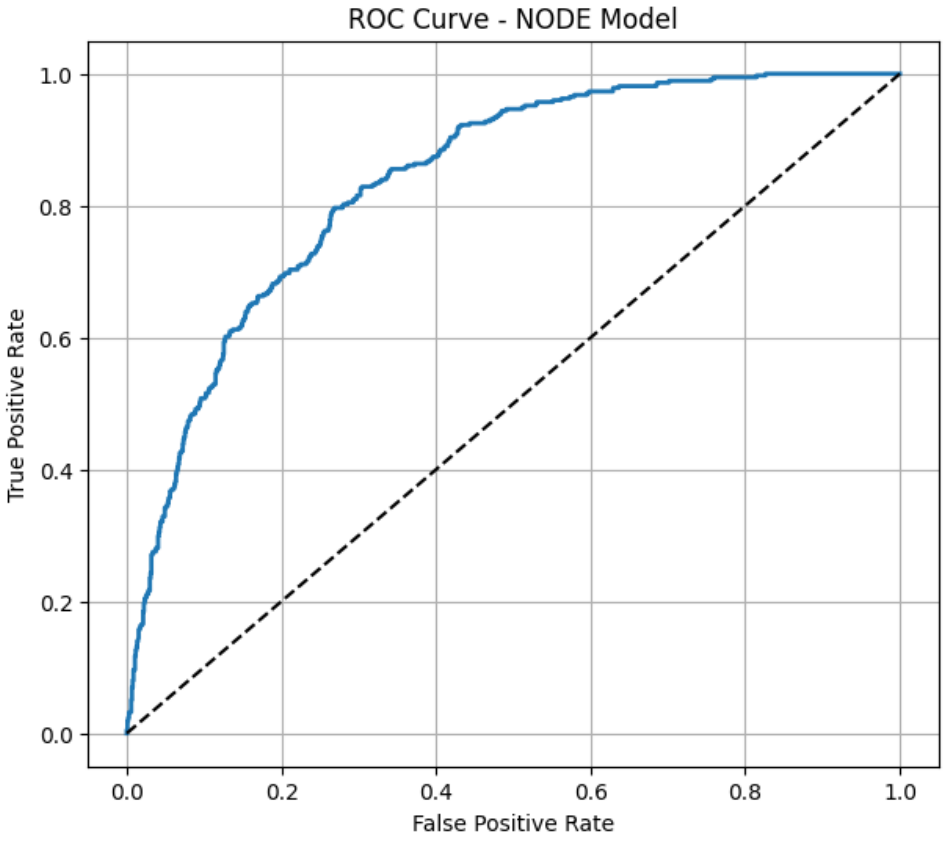


Figure 3.21: ROC curve - Default NODE model

The ROC curve in Figure 3.21 demonstrates strong discriminative performance, with an ROC-AUC score of **83.8%**. The curve rises sharply toward the top-left corner, indicating high true positive rates at relatively low false positive rates. Which confirms that the model effectively separates churners from non-churners across a wide range of thresholds.

- Using the Class weight

After evaluating the baseline model and understanding the limitations, we trained the same model, but this time by applying class weight to penalize the misclassification of the minority class more heavily.

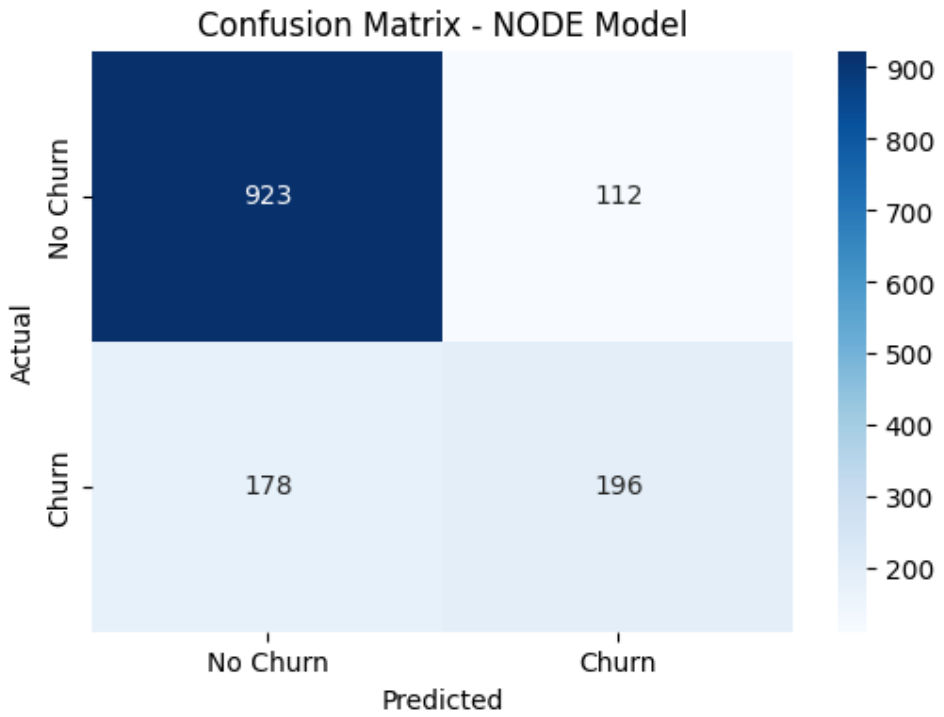


Figure 3.22: Confusion matrix - Default NODE model with class weight

When applying the class weight, the model became more sensitive to the minority class and produced more balanced predictions. See Figure 3.22

Table 6.7: report – Default NODE model with class weight

Recall	Precision	F1-score	Accuracy
78%	51%	62%	75%

As Table 3.7 shows, the recall improved compared to the baseline model, which means the model captured more churn customers. Also, the F1 score increased because of the boost in the recall, even if the precision decreased a little, which is acceptable because we are focusing on the churn customers more

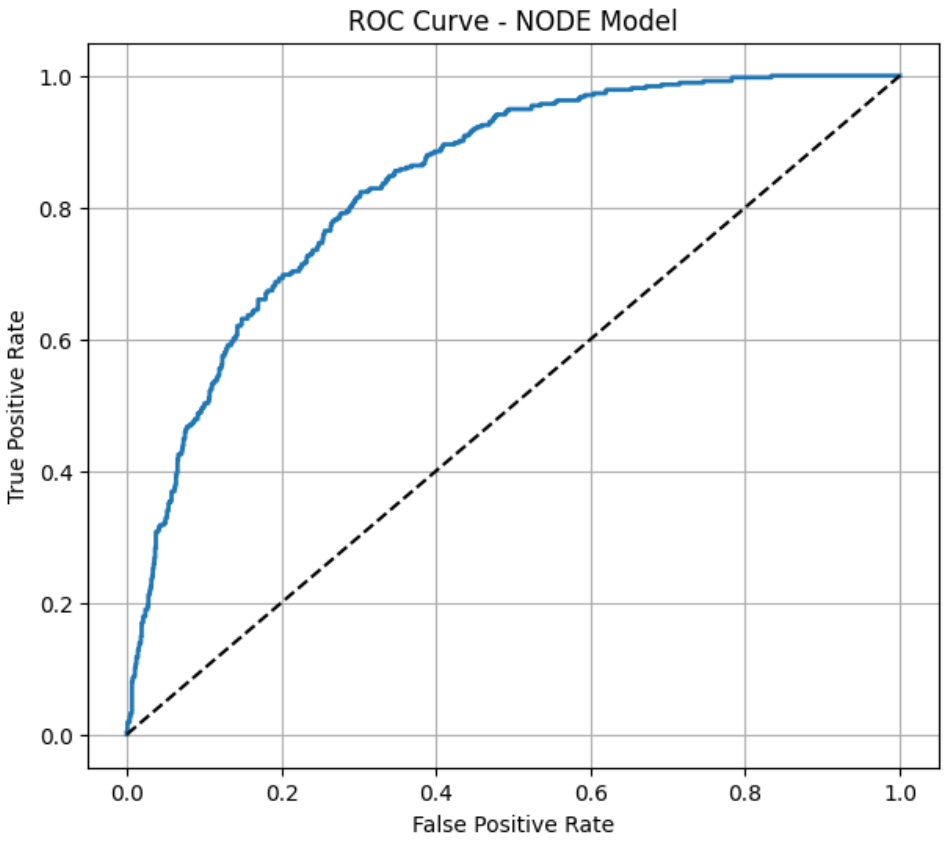


Figure 3.23: ROC curve - Default NODE model with class weight

Compared to the baseline model, the class-weighted version achieves very similar performance, and the AUC values are nearly identical (This achieved **83.6%**). However, class weighting is still beneficial because it generally improves sensitivity toward the minority class, helping the model avoid missing true churn customers.

- Threshold tuning

One more step to solve the imbalanced, is to try threshold tuning. We tested a range of thresholds (0.10 to 0.89) to analyze the trade-off between Recall, Precision, and F1-score. The best threshold is 0.58. So, we applied it when in inference.

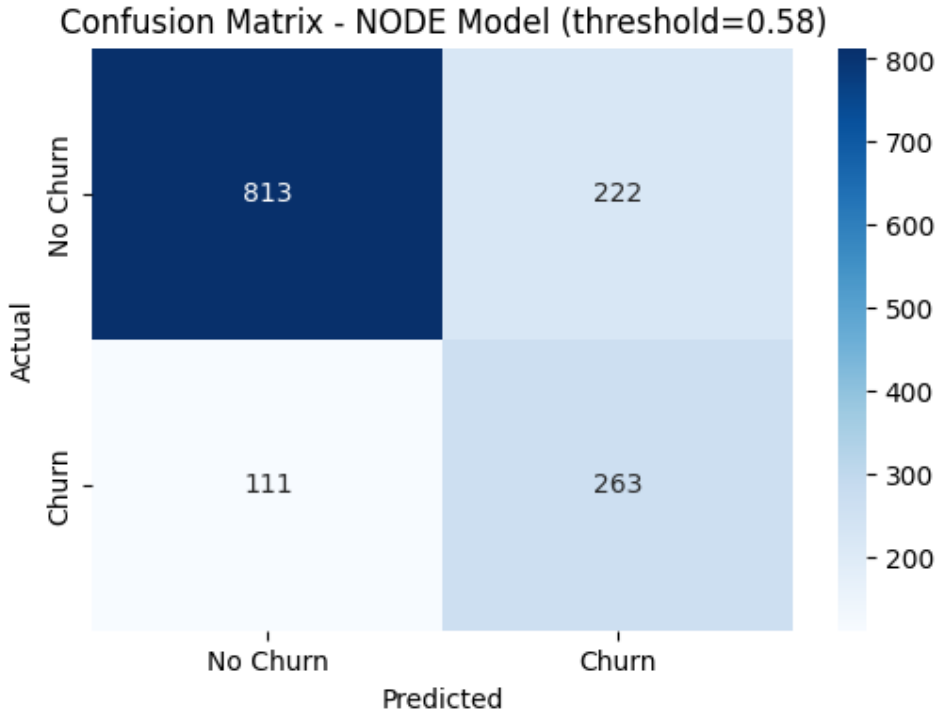


Figure 924: Confusion matrix - Threshold tuning

After adjusting the threshold, the model correctly identified more churn customers (TP = 263) and reduced false negatives (FN = 111) compared to both the baseline model and the class-weighted model. See Figure 3.24.

Table 7.8: report - Threshold tuning

Recall	Precision	F1-score	Accuracy
70%	54%	61%	76%

By specifying the threshold, the model correctly identified more churn cases than the baseline model and maintained strong detection similar to the class-weighted model. However, threshold tuning provides a middle ground between both models, reducing false negatives compared to the baseline and improving stability compared to the class-weighted version. See table 3.8.

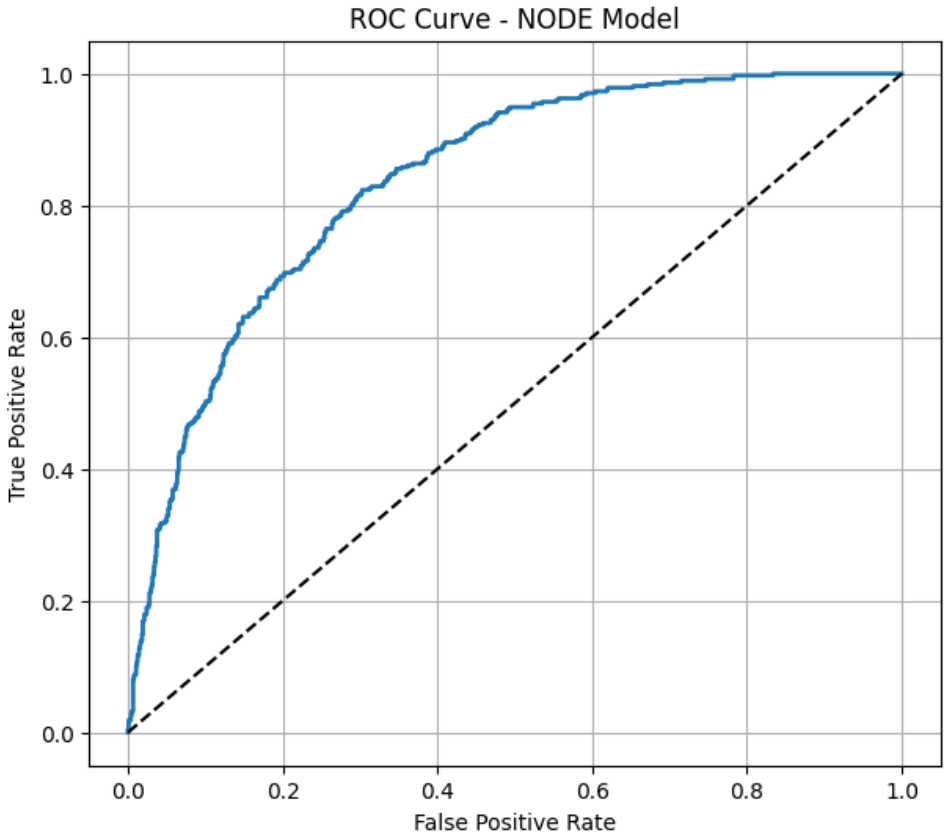


Figure 3.25: ROC curve - Threshold tuning

The ROC-AUC for the threshold-tuned model is 83.67%, which is nearly identical to the previous models. This confirms that tuning the threshold does not change the model's inherent discriminative power but adjusts the decision point to better capture churn customers.

- Hyperparameter tuning

To further improve the performance, we applied Bayesian hyperparameter optimization using Optuna. We searched across a wide range of architectural and training hyperparameters, including the number of layers, number of trees, depth, dropout, learning rate, and batch size.
Initially, some trials failed due to memory issues (especially large tree counts or batch sizes), so we refined the search space and stabilized the tuning process.
The key parameters explored were:

- Layers and depth: Larger values improved learning but caused overfitting, so we need a balanced combination
- Number of trees: Higher counts increased capacity, but sometimes caused memory errors
- Input dropout: Moderate dropout worked best for reducing overfitting
- Learning rate: The chosen rate stabilized training and avoided divergence
- Layer dimension: Moderate sizes improved learning without adding extra complexity
- Max epochs: Mid-range epochs performed best without overfitting
- Batch size: Very large batches were unstable, a smaller batch size improved generalization

The best parameters are: (lr: 0.0009047246536415638, num_layers: 3, depth: 7, num_trees: 192, layer_dim: 48, input_dropout: 0.08873321842598679, batch_size: 256, max_epochs: 80)

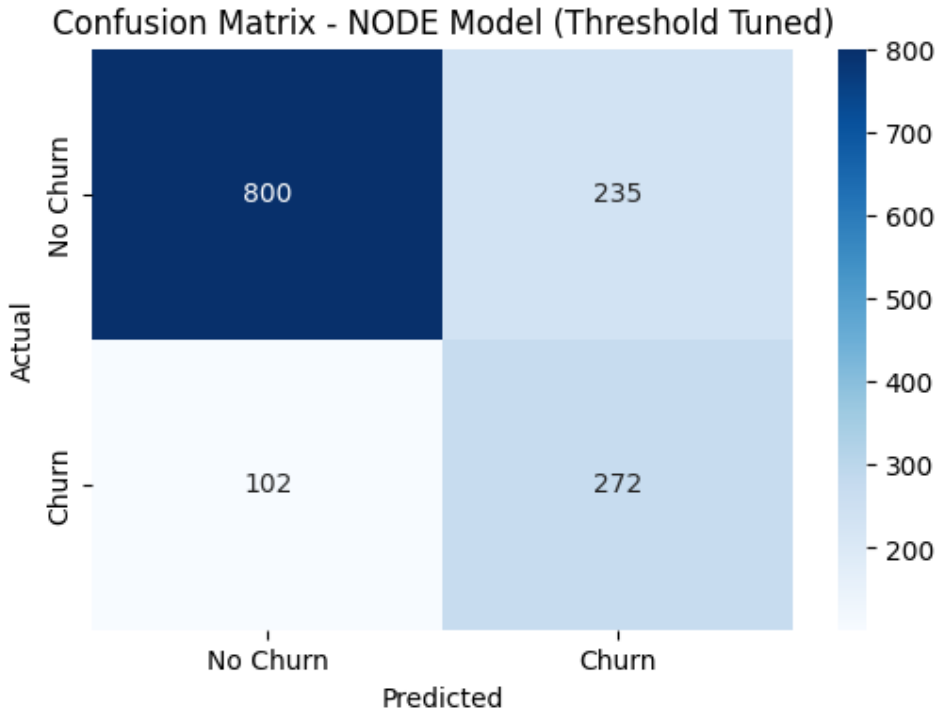


Figure 3.26: Confusion matrix - Optimized model

The optimized model reduced false negatives compared to the baseline and produced more stable results than the class-weighted and threshold-tuned models.

Table 8: report - Optimized model

Recall	Precision	F1-score	Accuracy
73%	54%	62%	76%

As shown in Table 3.9 above, this model achieved the most balanced performance across all metrics.

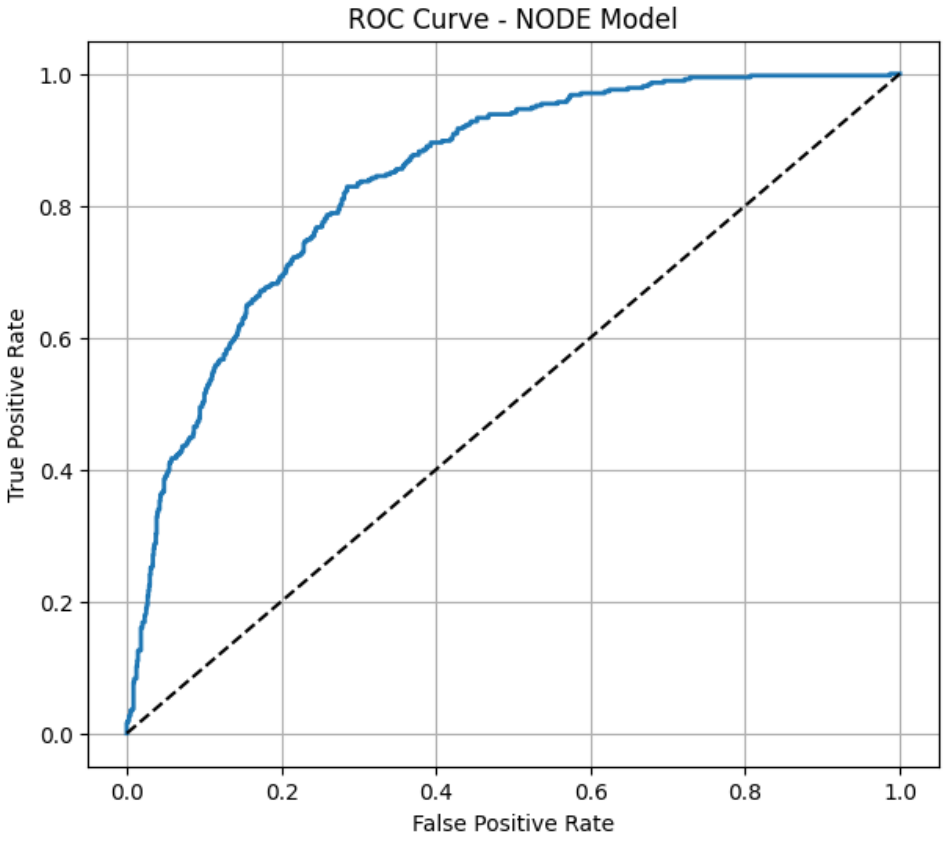


Figure 3.27: ROC curve - Optimized model

The ROC curve in Figure 3.27 shows a strong AUC, with a value 84%, consistent with previous models, confirming that the tuned architecture improves performance without increasing overfitting.

3.3.5 TabNet

To explore a modern deep-learning approach for tabular data, we trained an ensemble of TabNet models using 5-fold Stratified Cross-Validation and averaged their predictions on the test set. TabNet uses sequential attention to focus on the most informative features at each step, which might make it a good candidate for capturing complex interactions in our churn dataset.

- Optimized model

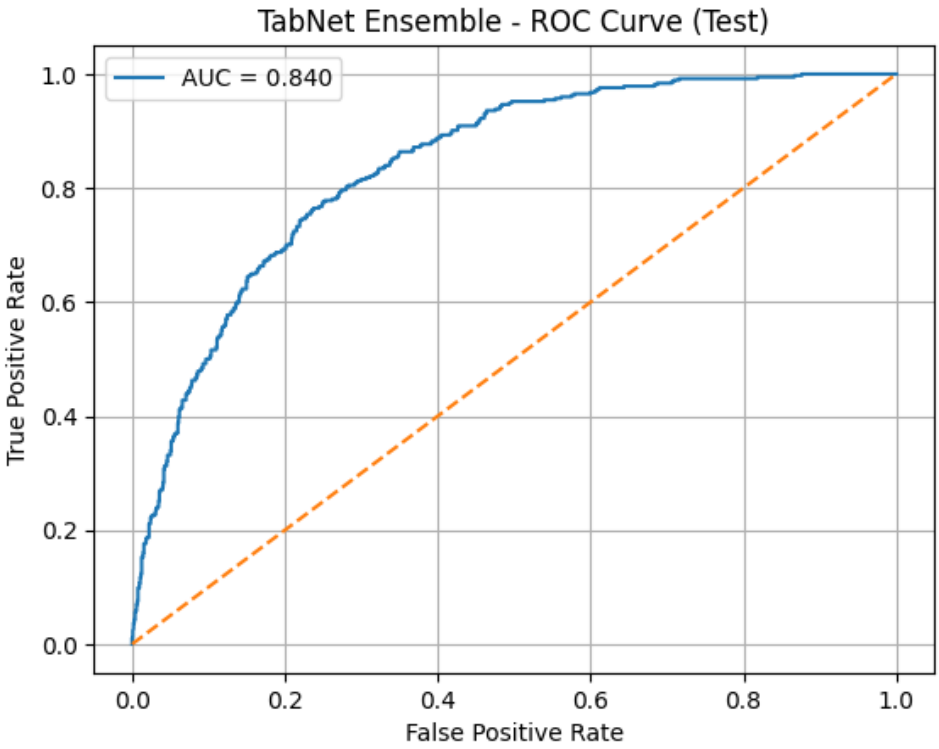


Figure 3.28: ROC curve - TabNet

The ROC curve in Figure 3.28 shows that the TabNet ensemble achieves an AUC of about **0.84**, which is in the same range as Logistic Regression, Random Forest, XGBoost, and the optimized NODE model. This means TabNet is able to rank churn vs. non-churn customers reasonably well across different thresholds, even though its classification behavior at the default threshold is different from the other models.

Table 3.10: report – TabNet model

Recall	Precision	F1-score	Accuracy
0.50	0.6448	0.5633	0.7942

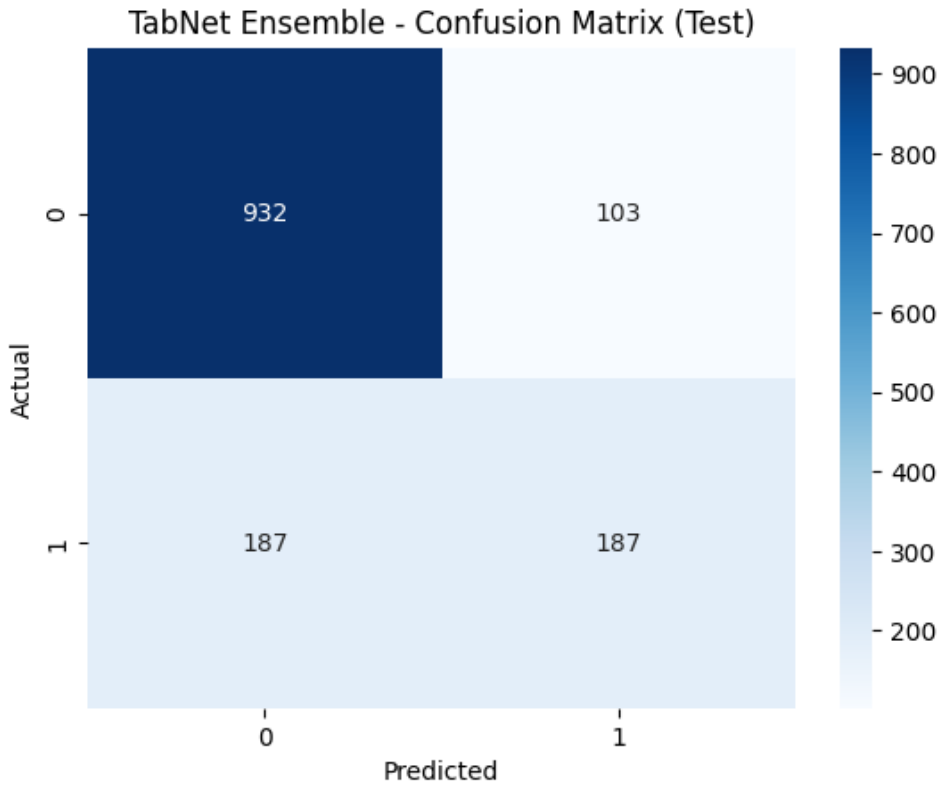


Figure 3.29: Confusion Matrix - TabNet

The confusion matrix in Figure 3.29 above shows that TabNet correctly classifies most non-churn customers (TN = 932) and is more conservative in predicting churn (TP = 187, FN = 187). This explains the **high precision (0.64) and highest accuracy (~79%),** but also the **lowest recall (0.50)** among all models. Compared to XGBoost, Random Forest, and NODE, TabNet misses more actual churners but makes fewer false churn alarms. In a churn-prediction setting where catching as many churners as possible is more important than overall accuracy, this configuration of TabNet is less suitable; however, with threshold tuning it could be shifted toward higher recall while keeping its strong probability estimates.

After Grid Search over the threshold, we get a better overall performance, as shown below.
After applying Grid Search over 100 threshold values (0.10 to 1.00), the TabNet ensemble achieved a noticeably better balance across all metrics. Compared to the original TabNet (threshold = 0.5), the tuned version dramatically increased recall from **50% to 75.9%,** meaning the model now captures far more churn customers. Precision naturally dropped from 0.64 to 0.54, which is expected because detecting more churners usually increases false positives. Overall performance improved, with the F1-score rising from **0.56 to 0.63**, showing a healthier trade-off between catching churners and limiting mistakes. Although accuracy decreased slightly (from ~79% to ~76%), this is acceptable since churn prediction prioritizes recall over strict accuracy.
When compared with the other models, the tuned TabNet now behaves similarly to Random Forest, XGBoost, and the optimized NODE model in terms of recall and F1-score, while still maintaining the same ROC-AUC (~0.84). This confirms that TabNet already produces reliable probability estimates, and the main limitation was the decision threshold, not the model itself.

After tuning, TabNet becomes a competitive deep-learning alternative with strong recall and a more practical prediction behavior for churn-sensitive applications.

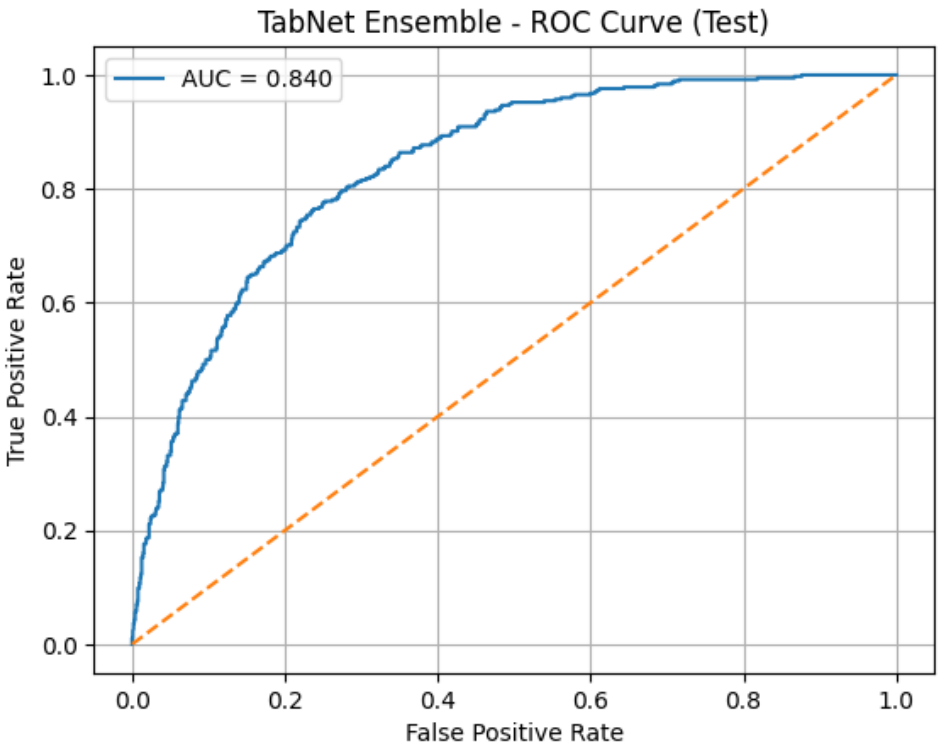


Figure 3.30: ROC curve – TabNet After Grid Search

Tab 3.11: report – TabNet after Grid Search

Recall	Precision	F1-score	Accuracy
0.7594	0.5399	0.6311	0.7644

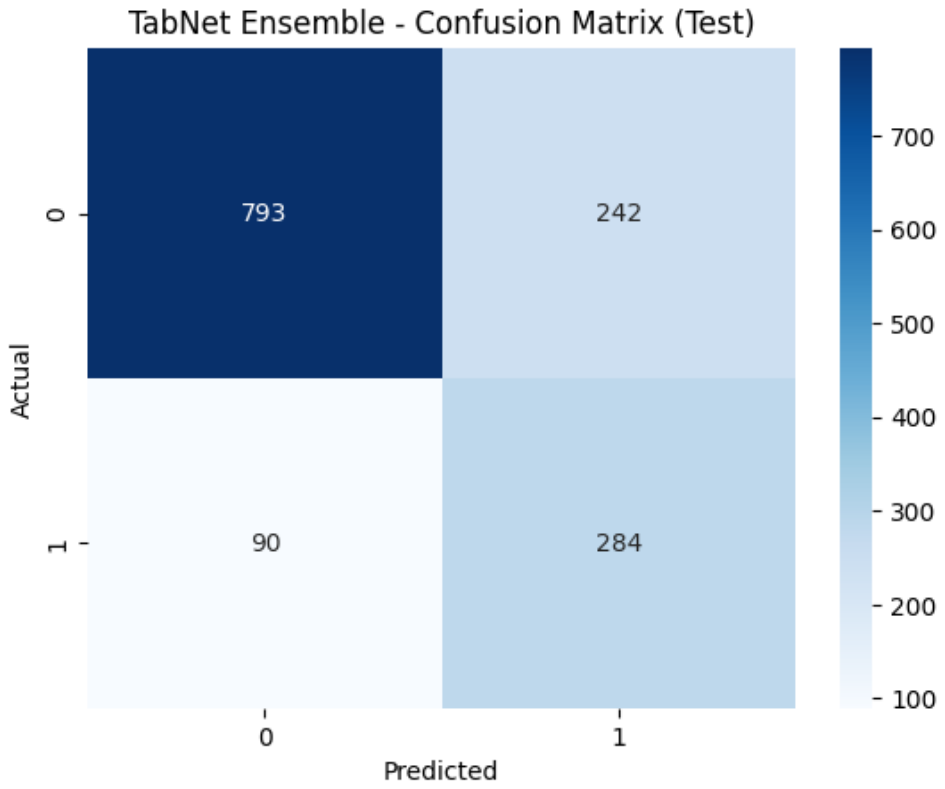


Figure 3.31: Confusion Matrix – TabNet after Grid Search

3.4 Model selection

For the final model, we selected the **Bayesian-tuned XGBoost classifier**. Across all candidates (Logistic Regression, Random Forest, NODE, TabNet), XGBoost consistently achieved one of the **highest recalls (~80%)** on the churn class and a **strong F1-score**, while also delivering the **best ROC-AUC (~0.84)**, indicating excellent ranking ability between churners and non-churners. At the same time, XGBoost remains relatively **lightweight, fast, and easy to deploy** compared to deep models and large ensembles, with mature support in standard ML pipelines and clear feature-importance tools. This combination of high recall on the minority churn class, competitive overall performance, robustness, and practical deployability makes XGBoost the most suitable choice for the final churn prediction model.

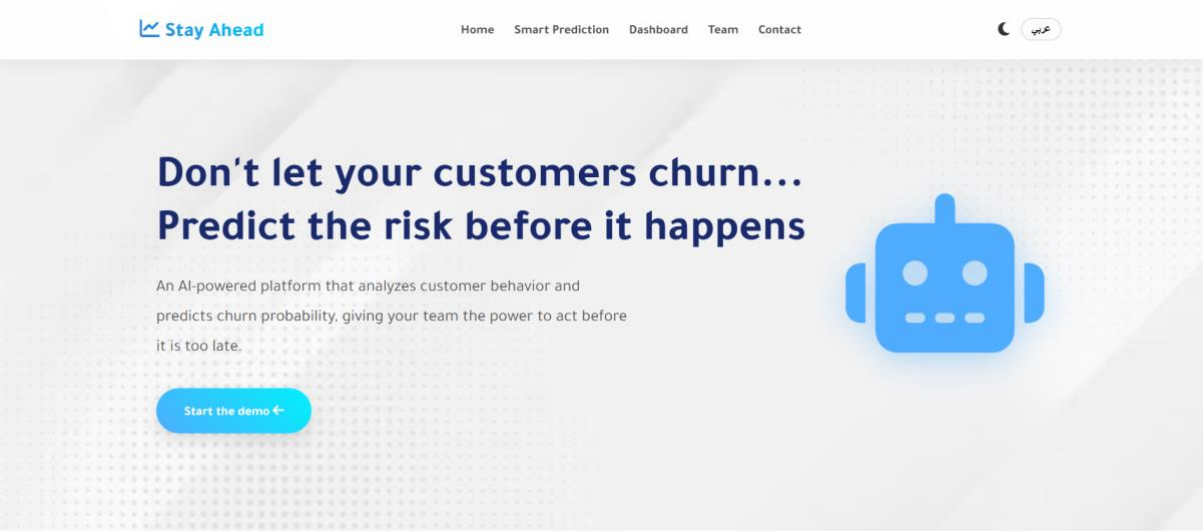
3.5. User Interface & Integration

"Stay Ahead" system is built on a fully integrated architecture that combines a modern, user-friendly frontend, a high-performance AI backend, and a seamless integration layer connecting both sides. This section outlines the technologies and mechanisms used in each component.

3.5.1 Frontend (User Interface)

The frontend was developed using standard web technologies including HTML5, CSS3, and JavaScript (Vanilla ES6). This ensures fast performance, lightweight loading, and compatibility across all devices without relying on heavy frameworks. The interface includes all main pages such as the Home page, Prediction page, Dashboard, Team page, and Contact page.

A dynamic translation engine was implemented inside the file main.js, enabling real-time switching between Arabic and English. The system automatically adjusts page text and layout direction (RTL/LTR) whenever the user changes the language. The frontend also uses localStorage to store user preferences such as language and theme (light/dark), and SweetAlert2 to present clear interactive alerts and messages.
Communication with the backend is carried out using fetch requests, allowing the frontend to send and receive structured JSON data directly from the API.



3.5.2 Backend (Prediction Engine)
The backend was built using Python 3.11 together with the FastAPI framework to create a fast, scalable REST API. The application runs on Uvicorn, providing stable performance and low-latency responses. The backend receives customer data, validates it through Pydantic models, and processes it using a pre-trained machine learning model responsible for churn prediction and risk classification.
The backend exposes several key endpoints, including: POST /predict — receives customer data and returns the churn probability and risk level. GET /high-risk-users — retrieves a list of customers classified as “High Risk” for display in the dashboard.
CORS settings are enabled to ensure safe and consistent communication between the frontend and backend. The backend service is deployed on Render Web Service, making it accessible and ready to integrate with any external system.
3.5.3 Integration Layer (Frontend ↔ Backend Communication)
The integration between the frontend and backend is implemented through REST API communication using standard HTTP requests. The frontend sends data via fetch, encoded in JSON format, and processes the results returned by the backend in real time.
When a user submits customer data on the Prediction page, the frontend sends a request to the endpoint POST /predict. The backend processes the input, generates the churn prediction and risk classification, and sends the results back to the interface, which then displays them using interactive UI elements.

On the Dashboard page, the frontend continuously requests data from the endpoint GET /high-risk-users to fetch updated high-risk customer records and populate the table dynamically, with full language support applied to all labels and text elements.
The frontend is deployed on Render Static Site, while the backend operates on Render Web Service. Both are connected through a single unified API URL, ensuring stable and reliable communication between the two layers.

3.6. Testing and Improvements

After developing all models, we evaluated them on the untouched test set to ensure fair and unbiased comparison. The testing phase confirmed the behavior observed during cross-validation: models with higher recall (Random Forest, XGBoost, tuned NODE, tuned TabNet) were more effective at capturing churn customers, while models with higher precision (Logistic Regression, original TabNet) produced fewer false alarms.
Based on these test results, several improvements were applied to strengthen model performance. First, **class balancing techniques** (class weights and threshold tuning) were tested on NODE and proved effective in increasing recall without harming AUC. Second, **Bayesian hyperparameter optimization** was applied to XGBoost and NODE, improving the stability and overall F1-score of both models. Third, **threshold grid search** was introduced for TabNet, which significantly improved its recall and made its behavior more suitable for churn prediction.
These improvements collectively increased the system's ability to detect churn customers while maintaining reliable probability estimates. The final selection of models and thresholds ensures that the deployed system favors recall, aligning with the real-world objective of minimizing lost customers, while still preserving good precision and overall accuracy.

4. Projected Impact

4.1. Accomplishments and Benefits

Throughout this project, we successfully developed a complete churn prediction pipeline, from data preprocessing and exploratory analysis to model training, optimization, and evaluation. We experimented with a diverse set of machine learning models, including Logistic Regression, Random Forest, XGBoost, NODE, and TabNet, and compared them using recall and F1-score metrics.
Key accomplishments include improving recall on the minority churn class through class weighting, threshold tuning, and hyperparameter optimization. These steps significantly enhanced the system's ability to detect high risk customers, aligning the model's behavior with real-world retention goals.
The most notable achievement is achieving a high recall of ~80% in the final churn-prediction model, which is considered a strong result in churn systems. Overall, the project delivers a functional, data-driven solution that can help SaaS businesses reduce churn and improve customer retention.

4.2. Future Improvements

Although the system performs well, several enhancements could further strengthen its performance:

1- Expanding the dataset with a more versatile customers, and incorporating additional features, such as customer support logs or user activity histories, could provide richer signals for the model
2- Explore deeper NODE and TabNet configurations, which may yield even higher performance
3- Experiment with additional techniques that improve minority-class prediction, such as cost-sensitive learning
4- Refine the decision threshold using methods such as Gridsearch or Bayesian optimization to maximize recall; as threshold tuning proved highly impactful in our experiments
5- Enhance the dashboard with explainability tools such as SHAP values to give product teams clearer insight into why each customer is predicted to churn

These improvements would contribute to transforming the current prototype into a fully operational churn-prediction system suitable for production environments

5. References

[1] "Subscription churn 101: What businesses need to know | Stripe." Accessed: Nov. 29, 2025. [Online]. Available: https://stripe.com/resources/more/subscription-churn-101?

[2] "SaaS Churn: Challenges and Solutions." Accessed: Nov. 29, 2025. [Online]. Available: https://blog.revpartners.io/en/revops-articles/churn-saas-challenges-solutions

[3] S. P. Yandex, S. Morozov, and A. Babenko, "Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data," Sep. 2019, Accessed: Nov. 29, 2025. [Online]. Available: https://export.arxiv.org/pdf/1909.06312v2.pdf

[4] "Data Preprocessing in Machine Learning," Couchbase Blog. Accessed: Nov. 29, 2025. [Online]. Available: https://www.couchbase.com/blog/data-preprocessing-in-machine-learning/