# STAR Method for ML

*Clear & Effective Way to Explain Your ML Projects*

**Sadum Yeshwanth**

# Contents

# Modified STAR Approach

The **modified START method** helps you clearly and confidently talk about your machine learning projects by breaking them down into four easy parts:

1. **Situation + Task:** Describe the problem you wanted to solve and the data you had.
2. **Action:** Explain what steps you took to build your solution — the models, techniques, and tools you used.
3. **Problems faced**: Share any challenges or issues you encountered and how you fixed them.
4. **Results**: Talk about the outcomes — how well your model worked and the impact it made.

This simple structure makes your explanation easy to follow and shows your problem-solving skills clearly.

# 1. House Price Prediction

## (Situation + Task) Problem Statement and the Data Available

I worked on a house price prediction project where the goal was to predict the sale price of a house based on various features such as location, number of rooms, area, age of the house, etc.
The dataset was from Kaggle's Ames Housing dataset, containing around 80 features and ~1,500 samples. The task was to build a regression model to accurately predict house prices.

## (Action) Approach/Solution Designed and Implemented

I began with EDA to understand relationships between variables and identify missing or skewed data.
Then, I performed feature engineering, including handling missing values, converting categorical features using one-hot encoding, and scaling numerical values.
I tried multiple models like Linear Regression, Random Forest, and XGBoost, and used cross-validation to compare their performance.
Finally, I built a stacked ensemble model using the top 3 performing models to improve prediction accuracy.

## Problems Faced and How I Overcame Them

One major issue was the presence of outliers and skewed distributions, which were affecting model performance. I handled this by applying log transformations on skewed features and filtering extreme outliers during training.
Also, some categorical features had high cardinality. I grouped rare categories under an "Other" label to reduce dimensionality.

## (Results) Evaluation and Impact

The final model achieved an $R^2$ score of 0.92 on the validation set, and the RMSE reduced by 15% compared to the baseline linear regression.
This approach demonstrated how combining models and doing proper preprocessing can lead to better generalization.
The project helped me understand real-world data challenges and how to build robust pipelines for regression tasks.

# 2. Customer Churn Prediction

## (Situation + Task) Problem Statement and the Data Available

I worked on a customer churn prediction project for a telecom company.
The goal was to classify whether a customer is likely to churn (leave the service) based on features like usage data, call drop rate, billing issues, support interaction frequency, etc.
The dataset had around 10,000 records with ~30 features, both numerical and categorical.

## (Action) Approach/Solution Designed and Implemented

I started with exploratory analysis and preprocessing — handling missing values, encoding categorical variables, and scaling where necessary.
For initial modelling, I used a Decision Tree classifier to get a quick sense of feature importance and model interpretability.
The Decision Tree helped identify the top features influencing churn, such as "number of support tickets", "monthly charges", and "contract type".
However, the model overfit quickly and performed poorly on unseen data.

## Problems Faced and How I Overcame Them

The main issues with Decision Trees were:

- Overfitting: The tree memorized the training data due to its depth and branching.
- Bias to dominant features: Some features with many levels (like customer ID or region) dominated splits but weren't truly predictive.

To solve this:

- I switched to a Random Forest, which uses an ensemble of trees and random subsets of features. This reduced variance and improved generalization.
- I also used grid search for hyperparameter tuning (number of trees, max depth, min samples per leaf).
- Finally, I used permutation feature importance on the Random Forest to validate that the important features found earlier were consistent and robust.

## (Results) Evaluation and Impact

The Random Forest model achieved 85% accuracy and an F1-score of 0.83, significantly outperforming the initial decision tree.
The model was also able to rank the most important drivers of churn, helping the business focus on improving customer support and billing clarity.
This project showed me the value of using simple models like Decision Trees for early insights, and ensemble methods like Random Forests for production-grade performance.

# 3. Time Series Anomaly Detection

## (Situation + Task) Problem Statement and the Data Available

I worked on a time series anomaly detection project for an e-commerce platform to monitor payment failure rates in real-time.
The goal was to detect unusual spikes in payment failures that could indicate issues like service outages, third-party gateway failures, or fraud.
The dataset contained minute-level logs of payment success/failure counts across multiple gateways, spanning several weeks.

## (Action) Approach/Solution Designed and Implemented

I began by aggregating the data to a uniform time series and used rolling statistics (mean, std deviation) for baseline anomaly detection.
Then, I implemented more robust models like Seasonal Hybrid Extreme Studentized Deviate (S-H-ESD) and also evaluated Facebook Prophet for seasonality-aware anomaly detection.
Also tested unsupervised models like Isolation Forest on sliding windows of aggregated metrics.

## Problems Faced and How I Overcame Them

One challenge was the natural seasonality in user activity — for example, payment spikes during peak hours or sales events.
Traditional thresholding methods led to many false positives during these expected traffic spikes.
To solve this:

- I incorporated time-of-day and day-of-week seasonality into the model using Prophet.
- For real-time use, I optimized the pipeline using moving z-scores and anomaly scoring, tuned using historical labels of known incidents.

## (Results) Evaluation and Impact

The final model reduced false positives by over 40% compared to the initial heuristic approach.
It detected multiple incidents hours before business teams manually noticed them, enabling proactive mitigation.
This project helped automate observability of payment systems and showed the importance of seasonality-aware modeling in anomaly detection.

# 4. Recommender System

## (Situation + Task) Problem Statement and the Data Available

I worked on building a personalized recommender system for an e-commerce platform to suggest relevant products to users in real time.
The task was to generate top-N product recommendations based on user behaviour, preferences, and item attributes.
The dataset included user clickstreams, purchase history, and product metadata like category, price, and brand.

## (Action) Approach/Solution Designed and Implemented

I implemented a Two-Tower neural network architecture, where one tower encodes user features (e.g., past interactions, demographics) and the other encodes item features (e.g., category, embeddings from metadata).
Both towers output embedding vectors, and cosine similarity between them was used to rank products for a given user.
I trained the model using negative sampling and a contrastive loss function, optimizing it for efficient retrieval instead of full softmax classification.
For serving, I precomputed item embeddings and stored them in a vector search index (like FAISS) to enable fast retrieval.

## Problems Faced and How I Overcame Them

One challenge was cold-start for new users and items, as the model relies on embedding history.
I handled this by:

- Using content-based features (e.g., age, location, item metadata) for cold-start embedding approximation.
- Implementing a hybrid approach: fallback to rule-based or popularity-based recommendations for completely new users.
  Another issue was serving latency at scale, which I reduced by batching requests and optimizing vector search indexing.

## (Results) Evaluation and Impact

The two-tower model achieved a 12% increase in hit rate@10 and 15% higher recall@10 compared to the baseline matrix factorization method.
It also scaled well to millions of users and items and supported real-time personalization.
This project highlighted the effectiveness of representation learning for large-scale retrieval problems.

# 5. RAG Project

## (Situation + Task) Problem Statement and the Data Available

I worked on a RAG-based question answering system to improve internal knowledge access for a customer support team.
The goal was to accurately answer natural language queries using company documentation, knowledge base articles, and structured relationship data.
The data included unstructured documents (PDFs, FAQs), structured data (product metadata), and a knowledge graph built from entity relationships.

## (Action) Approach/Solution Designed and Implemented

I designed a multi-stage RAG pipeline that included:

- Query Rewriting: Used a small language model to normalize and expand user queries, improving retrieval relevance (e.g., rewriting "billing not working" → "issues with billing system failure").
- Retrieval: Combined a dense vector search (using embeddings from a domain-tuned model) with knowledge graph traversal (via a graph database like Neo4j) to surface both document snippets and related entities.
- Reranking: Applied a cross-encoder model to rerank retrieved chunks based on semantic similarity to the query, improving the quality of context passed to the generator.
- RAG Generator: Used a language model hosted on GroqCloud, optimized for ultra-low-latency inference to generate answers from top-ranked context.
- Critic Agent: Post-generation, a lightweight critic LLM agent validated the factual alignment of the generated answer against retrieved chunks and flagged potential hallucinations or gaps.

## Problems Faced and How I Overcame Them

One challenge was poor retrieval recall due to vague or short user queries.
I solved this by:

- Implementing query rewriting to expand ambiguous queries using conversational context or synonyms.
- Leveraging the graph DB to find related entities and inject their relationships into the prompt.

Another issue was latency and cost at inference time.
We optimized this by:

- Batching reranking operations and caching embeddings for high-frequency queries.

### (Results) Evaluation and Impact

The system improved answer accuracy by 27% compared to the previous keyword-based search.
The critic agent reduced hallucination rate by over 35%, and query rewriting boosted top-5 retrieval recall by 20%.
Support agents reported a 40% reduction in time to find answers, and the system was integrated into their workflow for real-time assistance.

# 6. Fine-tuning task

### (Situation + Task) Problem Statement and the Data Available

I worked on fine-tuning a pretrained language model (like BERT) for a customer support ticket classification task.
The goal was to automatically classify incoming tickets into categories like billing issues, technical support, and account management.
The dataset had around 20,000 labelled support tickets with text and category labels, but was imbalanced with some classes underrepresented.

### (Action) Approach/Solution Designed and Implemented

I started by cleaning and preprocessing the text (removing stopwords, tokenization).
Then I fine-tuned a pretrained BERT base model using transfer learning, freezing the lower layers initially and gradually unfreezing during training.
To handle class imbalance, I used weighted loss functions and data augmentation techniques like synonym replacement.
I also experimented with hyperparameter tuning (learning rate, batch size) using validation set performance.
Finally, I deployed the fine-tuned model in a microservice with a REST API for real-time classification.

### Problems Faced and How I Overcame Them

The major challenges were:

- Class imbalance, which was causing poor performance on minority classes.
- Overfitting due to limited labelled data.

To mitigate these:

- I applied class weights in the loss function and used SMOTE to synthetically oversample minority classes.
- I used early stopping and dropout regularization during fine-tuning.
- I also leveraged cross-validation to ensure robust evaluation.

## (Results) Evaluation and Impact

The fine-tuned BERT model achieved an F1-score of 0.87, improving over the baseline traditional ML model (SVM) by 18%.
It increased classification accuracy especially on minority classes, enabling faster and more accurate ticket routing.
This project demonstrated the value of transfer learning and careful handling of imbalanced data in text classification tasks.