Problem Statement: Auto-Complete Query Retrieval System

1. Introduction

In the world of e-commerce and search, the auto-complete system (ACS) is a critical component for enhancing user experience. A powerful ACS anticipates user intent by suggesting relevant and popular queries as the user types a prefix. This not only reduces user effort but also guides them toward better search outcomes, significantly improving engagement and conversion rates. The challenge is to retrieve the most relevant query completions from a massive pool of potential queries, balancing accuracy, relevance, and performance.

2. The Challenge

Your mission is to design and build a sophisticated retrieval system for an Auto-Complete Service. Given a user's input prefix, your system must retrieve the **top 150 most likely query completions** from a large candidate pool. The goal is to maximize the chances that the user's intended final query is present within your retrieved list.

Important Context: In a real-world search system, what a user types (Original Search Term or OST) often undergoes several transformations—such as font normalization and spell correction—to become a Processed Search Term (PST). The prefixes you will receive in the training and test data are derived from raw user OSTs. However, the candidate query_pool is composed entirely of cleaned PSTs. For instance, a user's typo-ridden OST like "ifon 16" might correspond to a final PST of "iphone 16" in the pool. Similarly, a misspelled query like "blaek blaijer" could map to "black blazer" after spell correction.

3. Dataset & Resources

You will be provided with the following datasets in Parquet format to train and build your solution:

- **Training Data (train.parquet):**
  - This file contains pairs of <prefix, query>, representing historical user sessions where a given prefix ultimately resulted in the user searching for the final query.
  - **Schema:** prefix (string), query (string)
- **Candidate Query Pool (query_pool.parquet):**
  - A comprehensive list of all unique queries that your system should retrieve from. For any given prefix during evaluation, your retrieved candidates must be a subset of this pool.
  - **Schema:** query (string)
- **Query Feature Table (query_features.parquet):**
  - A supplementary table containing aggregated performance metrics for queries over

the past 30 days. This data can be leveraged to enhance your retrieval logic with popularity and engagement signals.
    - **Schema:** query (string), orders (int), volume (int), catalog_clicks (int), catalog_views (int)
- **Test Prefixes (test_prefixes.parquet):**
    - This file contains the set of prefixes for which you must generate predictions for your final submission.
    - **Schema:** prefix (string)

4. Getting Started: Loading the Data

Here is a Python code snippet to help you load the datasets and convert them into pandas DataFrames using the datasets library.

```
from datasets import load_dataset

# Loading the Data from Hugging Face Hub
train_data = load_dataset("123tushar/Dice_Challenge_2025",
data_files="train_data/*.parquet")
test_prefixes_data = load_dataset("123tushar/Dice_Challenge_2025",
data_files="test_prefixes_data/*.parquet")
query_features = load_dataset("123tushar/Dice_Challenge_2025",
data_files="query_features/*.parquet")
pool = load_dataset("123tushar/Dice_Challenge_2025",
data_files="pool/*.parquet")

# Reading the Data to pandas DF
df_train = train_data['train'].to_pandas()
df_test = test_prefixes_data['train'].to_pandas()
df_query_features = query_features['train'].to_pandas()
df_pool = pool['train'].to_pandas()
```

5. Task

Your task is to build a model that, for every prefix in test_prefixes.parquet, can retrieve a list of 150 candidate queries from the query_pool.parquet. You are free to use any open-source models, embeddings (e.g., Word2Vec, BERT, Sentence-BERT), or traditional IR techniques (e.g., TF-IDF, BM25) to build your retrieval model. Combining the provided datasets to create powerful features is highly encouraged.

6. Model Evaluation Metric

Submissions will be evaluated based on the **Hit Rate** at different cutoffs on the provided test prefixes. The primary metric will be **HitRate@150**, with **HitRate@9** as a secondary, more stringent metric.

- **Hit Rate@k:** This metric measures the percentage of test prefixes for which the user's intended query is captured within the top k retrieved candidates. A prediction is counted as a "hit" if your list of k queries contains a query that is an exact match OR a close semantic equivalent to the ground-truth query.
    - **What is a "Semantic Equivalent"?** It's a query that expresses the same user intent, even if the phrasing is different. For your own modeling and validation, you are encouraged to use open-source embedding models to define and measure semantic similarity. However, for the final judging, semantic equivalence will be determined using our proprietary in-house models. For example:
        - If the ground-truth is "latest iphone model", a retrieved query of "iphone new model" would be a hit.
        - If the ground-truth is "running shoes for men", a retrieved query of "men's running shoes" would be a hit.
    - HitRate@150: The number of times the correct or most similar query is in your top 150 predictions, divided by the total number of test instances.
    - HitRate@9: The number of times the correct or most similar query is in your top 9 predictions, divided by the total number of test instances.

A higher Hit Rate indicates a more accurate and effective retrieval system.

7. Submission Format

Your submission should be a **CSV or Parquet file** with two columns: prefix and retrieved_queries. The file must contain a prediction for every prefix listed in the test_prefixes.parquet file. The retrieved_queries column should contain an array of 150 strings representing your predicted queries.

Example (CSV format):

```
prefix,retrieved_queries
"sho","[""shoes for men"", ""casual shoes"", ""sneakers"", ""shoes for
women"", ...]"
"red dr","[""red dress for women"", ""party wear red dress"", ""red
dress"", ""red dragon"", ...]"
...
```

Also create a short presentation for the panel members clearly explaining your solution and the results.

8. Team Evaluation Criteria

Criterion	What we're looking for
Technical Depth & Rigor	Quality of modeling, feature engineering, evaluation frameworks
Innovation & Creativity	Originality and thoughtfulness of the solution
Feasibility & Practical Impact	Applicability, performance, and scalability of the approach
Communication & Presentation	Clarity of reasoning, documentation, code quality and storytelling

Good luck, and we look forward to seeing your innovative solutions!