LLMs for Transaction-Merchant Matching

Rishabh Kaushick
College of Engineering
Northeastern University
Toronto, ON
kaushick.r@northeastern.edu

Abstract

In this report, I explored the performance of traditional supervised learning algorithms and large language models to classify messy financial transaction descriptors to their corresponding merchants. The supervised learning algorithm explored was XG Boost, and the LLMs used were LLaMA 3.2 & Gemma 3 both running locally on Ollama. Various prompt-engineering techniques like zero-shot prompting, 10-shot prompting method have been explored. Additionally, techniques such as Retrieval Augmented Generation (RAG) & Facebook AI Similarity Seach (FAISS) was implemented to add relevant examples to the prompt. Finally, meta-search engine 'SearXNG' was used to see if adding merchant information to rare merchants helped improve the performance. The results show that combining local LLMs with RAG-based prompting and external web augmentation significantly improves classification accuracy and generalization compared to the traditional machine learning approach.

1 Problem Statement

There are millions of credit card transactions that take place every hour, and billions of transactions every year [4]. Each time we swipe our card a messy *merchant descriptor* gets allocated for the corresponding transaction. Large financial organizations like MasterCard or Visa have a difficulty correctly matching these transaction merchant descriptors to merchant names. Consider the examples in the following table:

Table 1:

Messy Merchant Descriptors	Merchant Names	Label
AMZN Mktp CA*UC33G8423	Amazon Marketplace Canada	Match
NETFLX#999-USA	NetFlex Gym USA	Mismatch

Based on the current process (as shown in the Figure 1 below), sometimes the transaction merchant descriptors are mapped to the correct merchant names like in the first Amazon example. This is known as a 'Match'. However, for the second transaction, which belongs to Netflix USA, has been incorrectly matched to 'NetFlex Gym USA', therefore being classified as a 'Mismatch'.

Apart from traditional machine learning approaches, this project seeks to explore whether Large Language Models (LLMs), with their emergent abilities, can solve this classification problem better.

Figure 1: Flow chart of the current process.

Sometimes the very large rule-based algorithm could correctly guess the merchant name from the merchant descriptor - like the Amazon example above. However, sometimes it could make mistakes such as the second example.

Therefore, given a pair of merchant descriptor and merchant name, can we classify whether the current process correctly identified the merchant? (MATCH condition) Or is it possible that it guessed the merchant wrongly? (MISMATCH)

Apart from traditional machine learning approaches, this project seeks to explore whether LLMs, with their emergent abilities, can solve this classification problem better.

2 Datasets

Since the dataset for this project is confidential, other ways to generate a synthetic dataset, like using LLMs, have been explored. The objective of this project is to use LLMs to classify whether the descriptors match the merchants. If we use the same LLMs to generate the data and perform classification, it could potentially have a lot of bias. To solve this, creative approaches had to be followed to create synthetic data which was like real-world data:

1. The LLMs which generate the data will be different from the LLMs which perform the classification task.
2. Industry standard, cloud-based LLMs are used to generate the data. Smaller open source, open weights local models are used to perform the classification. Such LLMs are used to perform classification as no financial sensitive data will be needed to be sent to the servers of large organizations hosting their LLMs on the cloud.
3. The data generation & classification LLMs are as follows:
    a. Data Generation LLMs: OpenAI ChatGPT-4o, ChatGPT-4.1, Gemini 1.5 Pro
    b. Data Classification LLMs: Google Gemma 3, Meta Llama 3.2

The figure shows a flowchart of the data generation pipeline for 'Matched' transactions:

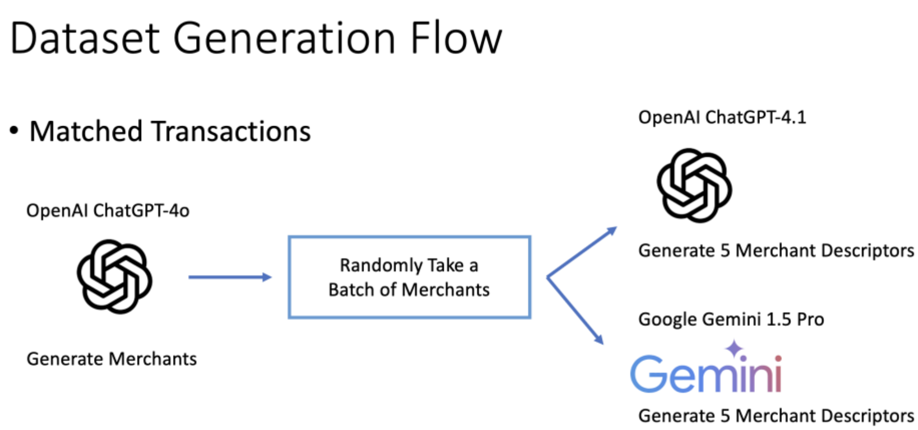


Figure 2: Data generation flow for matched transactions.

```python
def generate_messy_descriptors_openai(merchant_names: List[str]) -> str:
    prompt = f"""You are helping generate synthetic messy merchant descriptors for training a machine learning model.
            For each merchant name below, generate 5 messy descriptors that could realistically appear in transaction data.

            The messy descriptors should:
            - Use abbreviations, typos, and truncations.
            - Randomly include store numbers, city names, or country codes.
            - Add random symbols like *, -, #, etc.
            - Vary the word order sometimes.
            - Maintain overall meaning.

            Output format:
            - Start each merchant block with 'Merchant: <Merchant Name>'
            - List 5 messy descriptors (one per line, no bullets)

            Merchants:
            {chr(10).join(merchant_names)}
            """
    response = openai_client.responses.create(
        model="gpt-4.1",
        input=[{"role": "user", "content": prompt}],
        temperature=0.85,
        max_output_tokens=4000,
    )
    messy_text = response.output_text
    return messy_text
```
[51] ✓ 0.0s Python

Figure 3: Prompt used for generating transaction descriptors for 'Match' transactions.

To generate data for 'Mismatched' transactions, a different approach has been followed as LLMs were not able to generate satisfactory incorrect data.

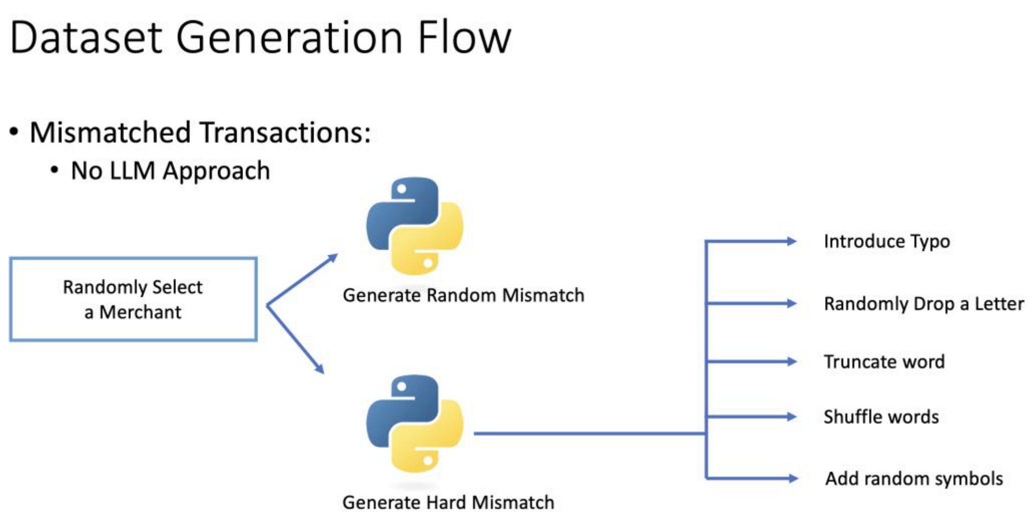


Figure 4: Data generation flow for 'Mismatch' transactions using Python.

First OpenAI ChatGPT-4o has been used to generate over 300 unique merchant names. Next, for each merchant 5 merchant descriptors are generated using the OpenAI and Gemini APIs.

Table 2: The basic feature of both datasets.

	Data Set Characteristics	Attribute Characteristics	Associated Tasks	Number of Instances	Number of Attributes
LLM Generated Transactions	Multivariate	Real	Classification	5350	3

1.1 Data characteristics

Once the dataset was generated, it was divided into the following sets:

1. Training Set: 4280 rows, 80%
2. Validation Set: 535 rows, 10%

3. Testing Set: 535 rows, 10%

The histograms of the data distribution of the labels are shown in Figure 3. We can see that we have the similar number of records for 'Match' & 'Mismatch' records.

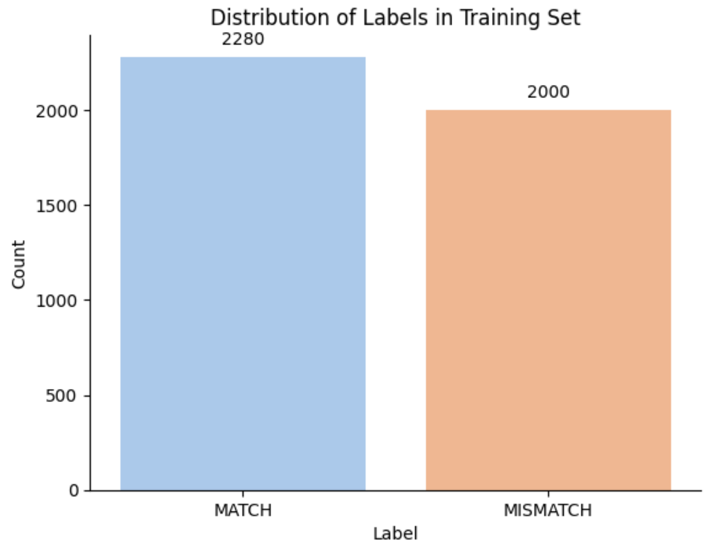


Figure 5: The class frequency of the labels.

Digging deeper, the following graph shows the distribution of how each of the data was generated.

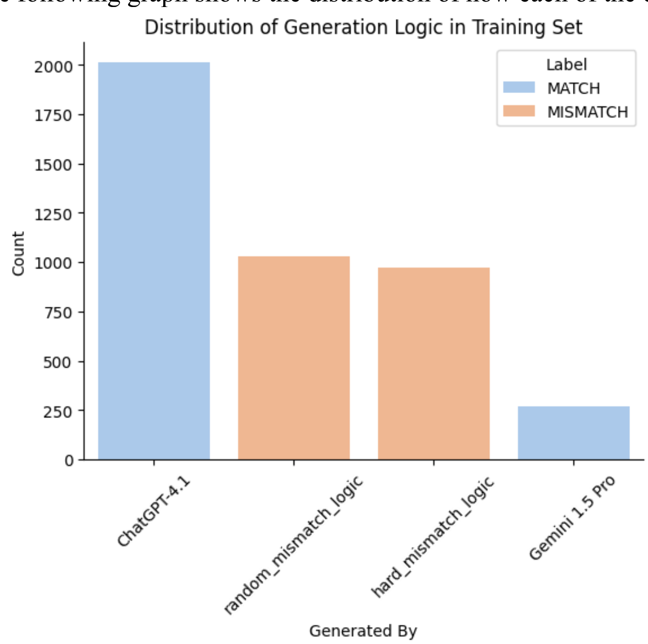


Figure 6: Distribution of how each data point was generated.

3 Supervised Learning: XG Boost Approach

I chose to begin this project with a strong baseline traditional machine learning model. This way once we try different LLMs, we can benchmark them with the baseline to understand if it realty is better or perhaps worse. Extreme Gradient Boost model (XG Boost) was selected as it is an ensemble learning method known as 'boosting' which uses multiple weaker decision trees to collectively form a strong model. This approach is not very simple yet not too complex and provides a good starting point as a baseline. If simpler model is required, we can go back to decision tree or if a more complex model is required, we can experiment with LLMs.

3.1 Term Frequency – Inverse Document Frequency

As most machine learning models, XG Boost cannot understand text data or perform operations on strings. Therefore, the first step before the model is to use an algorithm to convert string into numerical data like vectors. First, we fit and transform the data in the training set and following this we just transform the validation and test set based on the TF-IDF model.

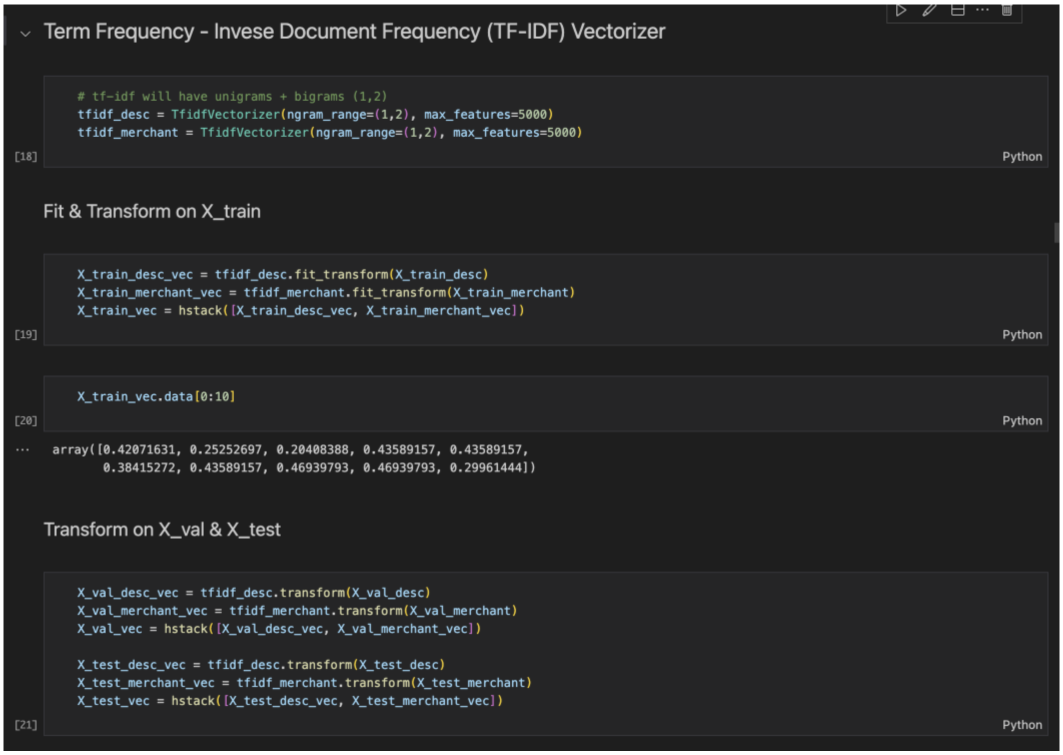


Figure 7: TF-IDF

3.2 XG Boost Model #1

A baseline XG Boost model was initially developed with the following parameters as shown in the figure below:

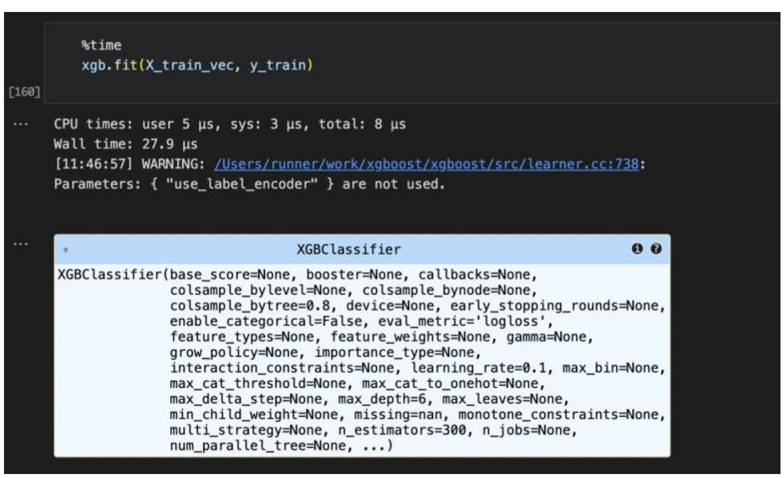


Figure 8: XG Boost model #1

However, this model was overfitting on the training dataset with 86% accuracy but failed to generalize on validation set with 65% accuracy and test set with 66% accuracy. Therefore, hyperparameter tuning approach was explored to improve the XG Boost baseline.

3.2 Hyperparameter Tuned XG Boost Model

The figures below show the different hyper-parameters which were used to re-train the XG Boost model and the final set of parameters which worked the best.

```python
params = {
    'max_depth': [3, 6],
    'learning_rate': [0.01, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'n_estimators': [100, 300]
}
```

[166] ✓ 0.0s Python

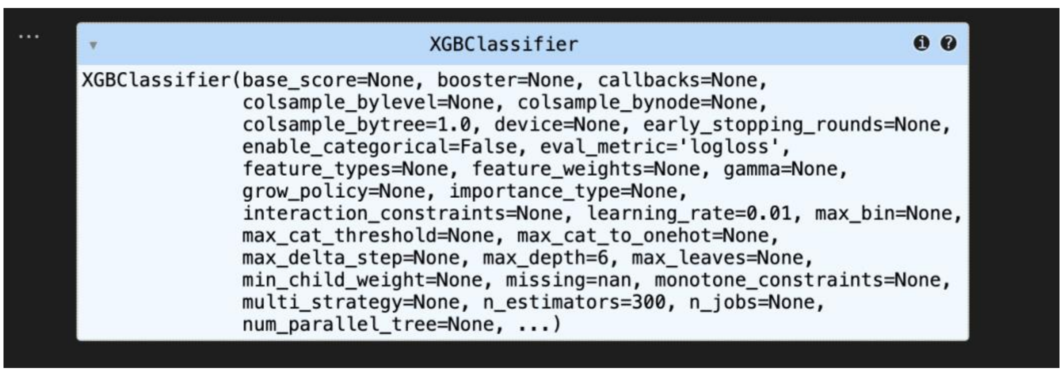
```python
xgb_2 = XGBClassifier(
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
```

[167] ✓ 0.0s Python

```python
grid = GridSearchCV(xgb_2, param_grid=params, scoring='f1', cv=3, verbose=1, n_jobs=1)
grid.fit(X_train_vec, y_train)
print("Best Parameters:", grid.best_params_)
```

[170] ✓ 25.7s Python

Figure 9: XG Boost hyper-parameters



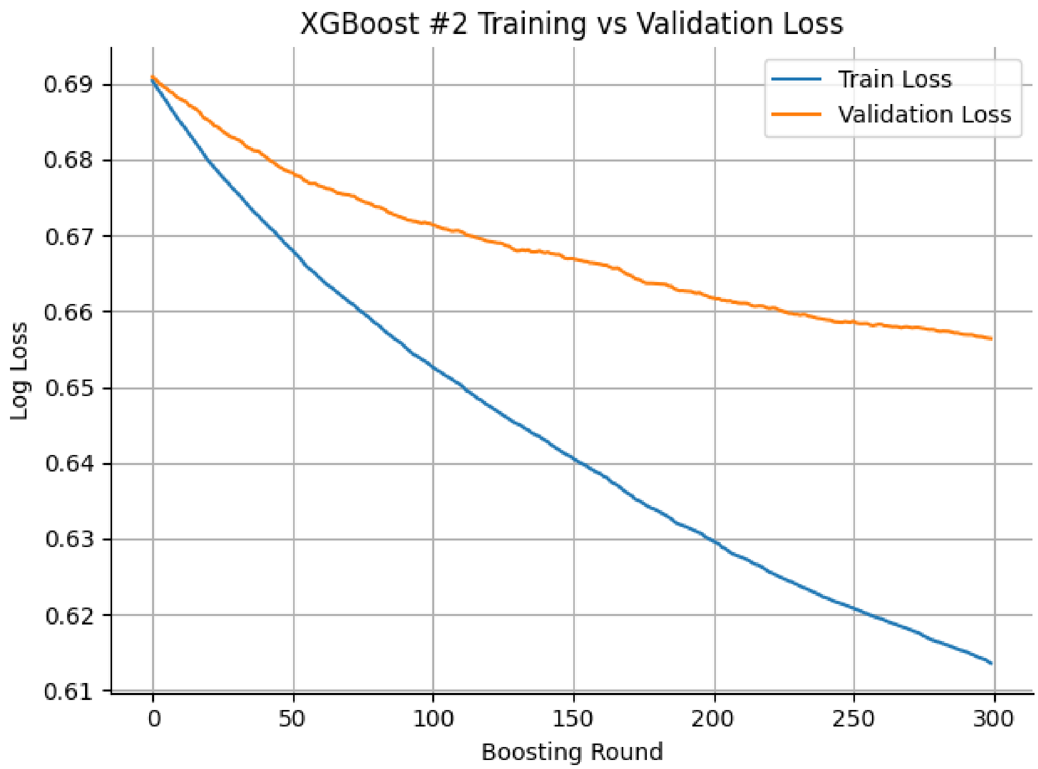
XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=1.0, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=6, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=300, n_jobs=None,
              num_parallel_tree=None, ...)
```

Figure 10: XG Boost Model #2

3.4 Evaluating XG Boost Model

The figure below shows how the loss gradually in the training and validation sets gradually reduce. After around 200 boosting rounds, we still see the training loss reducing drastically, but the validation loss seems to plateau.




```
y_train_pred = best_xgb.predict(X_train_vec)
print("XGBoost Model #2: \nTraining Evaluation Results:\n", classification_report(y_train, y_train_pred))
[179] ✓ 0.0s                                                                                      Python

XGBoost Model #2:
Training Evaluation Results:
               precision    recall  f1-score   support

           0       0.83      0.42      0.56      2000
           1       0.64      0.92      0.76      2280

    accuracy                           0.69      4280
   macro avg       0.74      0.67      0.66      4280
weighted avg       0.73      0.69      0.66      4280
```

```
# Evaluation of validation set
y_val_pred = best_xgb.predict(X_val_vec)
print("Validation Results:\n", classification_report(y_val, y_val_pred))
[180] ✓ 0.0s                                                                                      Python

Validation Results:
               precision    recall  f1-score   support

           0       0.71      0.34      0.46       250
           1       0.60      0.88      0.72       285

    accuracy                           0.63       535
   macro avg       0.66      0.61      0.59       535
weighted avg       0.65      0.63      0.60       535
```

```
# Evaluation of test set
y_test_pred = best_xgb.predict(X_test_vec)
print("Test Results:\n", classification_report(y_test, y_test_pred))
[181] ✓ 0.0s                                                          Python

... Test Results:
               precision    recall  f1-score   support

           0       0.74      0.38      0.51       250
           1       0.62      0.88      0.73       285

    accuracy                           0.65       535
   macro avg       0.68      0.63      0.62       535
weighted avg       0.68      0.65      0.63       535
```

4 Pretrained LLM Approach

4.1 Zero-Shot Prompting

As described in the

```
No Shot Prompting

def classify_with_ollama(desc, merchant, model='gemma3'):
    prompt = f"""
Given the messy transaction descriptor and the merchant name below, predict whether they represent the same entity (MATCH) or not (MISMATC

Messy Descriptor: {desc}
Merchant Name: {merchant}

Respond with only one word: MATCH or MISMATCH.
"""
    try:
        response = ollama.chat(model=model, messages=[{"role": "user", "content": prompt}])
        return response['message']['content'].strip().upper()
    except Exception as e:
        print("Error:", e)
        return "ERROR"
[197] ✓ 0.0s                                                          Python

classify_with_ollama("EBAY HOME-GDS US*", "Subway Store Ottawa", "gemma3")
[198] ✓ 2.0s                                                          Python
... 'MISMATCH'
```

The results are as follows:

```
print("Gemma3 LLM Evaluation on Train Set:")
print(classification_report(y_train, llm_train_preds))
[203] ✓ 0.0s

... Gemma3 LLM Evaluation on Train Set:
               precision    recall  f1-score   support

           0       1.00      0.52      0.68      2000
           1       0.70      1.00      0.83      2280

    accuracy                           0.78      4280
   macro avg       0.85      0.76      0.75      4280
weighted avg       0.84      0.78      0.76      4280
```

Figure: Gemma 3 results on training set.

```
print("Gemma3 LLM Evaluation on Validation Set:")
print(classification_report(y_val, llm_val_preds))
[206] ✓ 0.0s

... Gemma3 LLM Evaluation on Validation Set:
              precision    recall  f1-score   support

           0       0.99      0.54      0.70       250
           1       0.71      1.00      0.83       285

    accuracy                           0.79       535
   macro avg       0.85      0.77      0.77       535
weighted avg       0.84      0.79      0.77       535
```

Figure: Gemma 3 results on validation set.

```
print("Gemma3 LLM Evaluation on Test Set:")
print(classification_report(y_test, llm_test_preds))
208] ✓ 0.0s

... Gemma3 LLM Evaluation on Test Set:
              precision    recall  f1-score   support

           0       1.00      0.48      0.65       250
           1       0.69      1.00      0.82       285

    accuracy                           0.76       535
   macro avg       0.84      0.74      0.73       535
weighted avg       0.83      0.76      0.74       535
```

Figure: Gemma 3 results on test set.

```
print("Llama3.2 LLM Evaluation (w/ Zero Shot Prompting) on Train Set:")
print(classification_report(y_train, llm_train_preds))
[31]

... Llama3.2 LLM Evaluation (w/ Zero Shot Prompting) on Train Set:
              precision    recall  f1-score   support

           0       0.47      0.98      0.63      2000
           1       0.58      0.03      0.05      2280

    accuracy                           0.47      4280
   macro avg       0.52      0.50      0.34      4280
weighted avg       0.53      0.47      0.32      4280
```

```
print("Llama3.2 LLM Evaluation (w/ Zero Shot Prompting) on Validation Set:")
print(classification_report(y_val, llm_train_preds))
[33]
```

```
Llama3.2 LLM Evaluation (w/ Zero Shot Prompting) on Validation Set:
              precision    recall  f1-score   support

           0       0.46      0.97      0.63       250
           1       0.42      0.02      0.03       285

    accuracy                           0.46       535
   macro avg       0.44      0.49      0.33       535
weighted avg       0.44      0.46      0.31       535
```

```
print("Llama3.2 LLM Evaluation (w/ Zero Shot Prompting) on Test Set:")
print(classification_report(y_test, llm_train_preds))
[35]
```

```
Llama3.2 LLM Evaluation (w/ Zero Shot Prompting) on Test Set:
              precision    recall  f1-score   support

           0       0.47      0.98      0.64       250
           1       0.64      0.03      0.06       285

    accuracy                           0.47       535
   macro avg       0.56      0.51      0.35       535
weighted avg       0.56      0.47      0.33       535
```

4.2 Few-Shot Prompting: 20-Shot

```
print("Gemma3 LLM Evaluation (w/ 10-Shot Prompting) on Train Set:")
print(classification_report(y_train, llm_train_preds))
[21]
```

```
Gemma3 LLM Evaluation (w/ 10-Shot Prompting) on Train Set:
              precision    recall  f1-score   support

           0       0.88      0.68      0.77      2000
           1       0.76      0.92      0.84      2280

    accuracy                           0.81      4280
   macro avg       0.82      0.80      0.80      4280
weighted avg       0.82      0.81      0.80      4280
```

```
print("Gemma3 LLM Evaluation (w/ 10-Shot Prompting) on Validation Set:")
print(classification_report(y_val, llm_train_preds))
[25]
```

```
Gemma3 LLM Evaluation (w/ 10-Shot Prompting) on Validation Set:
              precision    recall  f1-score   support

           0       0.88      0.68      0.76       250
           1       0.76      0.92      0.83       285

    accuracy                           0.80       535
   macro avg       0.82      0.80      0.80       535
weighted avg       0.82      0.80      0.80       535
```

```
[27]
print("Gemma3 LLM Evaluation (w/ 10-Shot Prompting) on Test Set:")
print(classification_report(y_test, llm_train_preds))
```

```
... Gemma3 LLM Evaluation (w/ 10-Shot Prompting) on Test Set:
              precision    recall  f1-score   support

           0       0.87      0.63      0.73       250
           1       0.74      0.92      0.82       285

    accuracy                           0.78       535
   macro avg       0.80      0.77      0.77       535
weighted avg       0.80      0.78      0.78       535
```

```
[23]
print("Llama3.2 LLM Evaluation (w/ 10-Shot Prompting) on Train Set:")
print(classification_report(y_train, llm_train_preds))
```

```
... Llama3.2 LLM Evaluation (w/ 10-Shot Prompting) on Train Set:
              precision    recall  f1-score   support

           0       0.48      0.94      0.64      2000
           1       0.69      0.11      0.19      2280

    accuracy                           0.50      4280
   macro avg       0.58      0.53      0.41      4280
weighted avg       0.59      0.50      0.40      4280
```

Since the performance of the Llama3.2 LLM was sub-optimal even on the 10-shot prompting exercise, it was not evaluated on the validation and test sets.

5 Pretrained LLM Approach with RAG & FAISS

The above 10-shot prompting technique could be potentially better than zero-shot prompting in many cases. However, the examples that we provide to the prompt are very important. Sometimes these examples are irrelevant to the current record being classified. This could confuse the model or lead to poor performance. To tackle this problem, I have decided to update each prompt by using the most relevant examples based on the record being classified.

Here is how we can do this using Retrieval Augmented Generation style:

1. store the labeled examples from the training set in a vector store
2. save the vectors in a FAISS index
3. retrieve the K-most similar examples (based on the provided merchant descriptor & merchant)
4. update the prompt with only relevant examples
5. get the classification result from gemma/ llama

5.1 Embeddings Using Sentence Transformer

```
# Combine descriptor + merchant
train_df['Descriptor_Merchant_Combined'] = (
    "Descriptor: " + train_df['Messy Descriptor'] +
    " | Merchant: " + train_df['Merchant Name']
)

# Format for BGE
formatted_corpus = train_df['Descriptor_Merchant_Combined'].apply(format_bge).tolist()

# Generate embeddings
corpus_embeddings = embedder.encode(formatted_corpus, normalize_embeddings=True, show_progress_bar=True)
[26]  ✓ 10.7s
...   Batches: 100% 134/134 [00:10<00:00, 14.75it/s]
```

5.2 Few-Shot Prompting: Relevant 5-Shot Prompts

Once the dataset was generated, it was divided into the following set

5.3 Few-Shot Prompting: Relevant 10-Shot Prompts

Once the dataset was generated, it was divided into the following set

5 Pretrained LLM Approach with RAG, FAISS & SearXNG

This is an experiment to use Search functionality using meta-search engines - like Google, Bing, DuckDuckGo and others to privately browse the web and get results. The architecture of SearXNG is as follows:

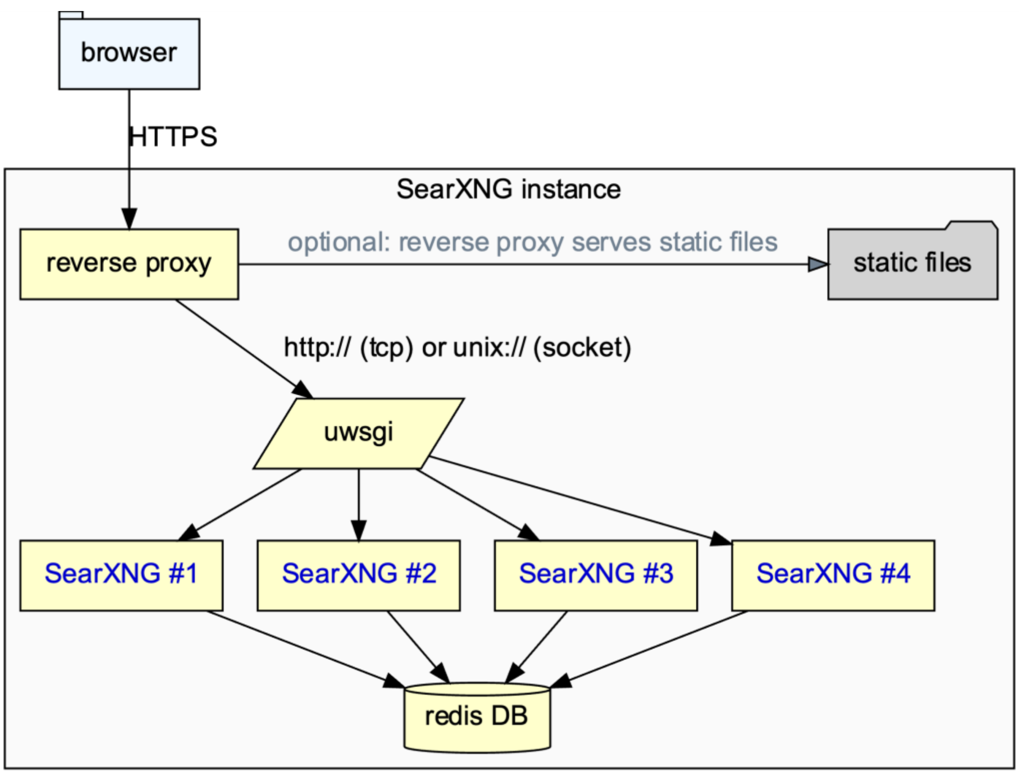


Figure: Architecture of SearXNG [7]

The goal is to utilize this search capabilities to improve the prompts to local models. This allows them to give more context into what each merchant is doing. Below is the proposed flow:

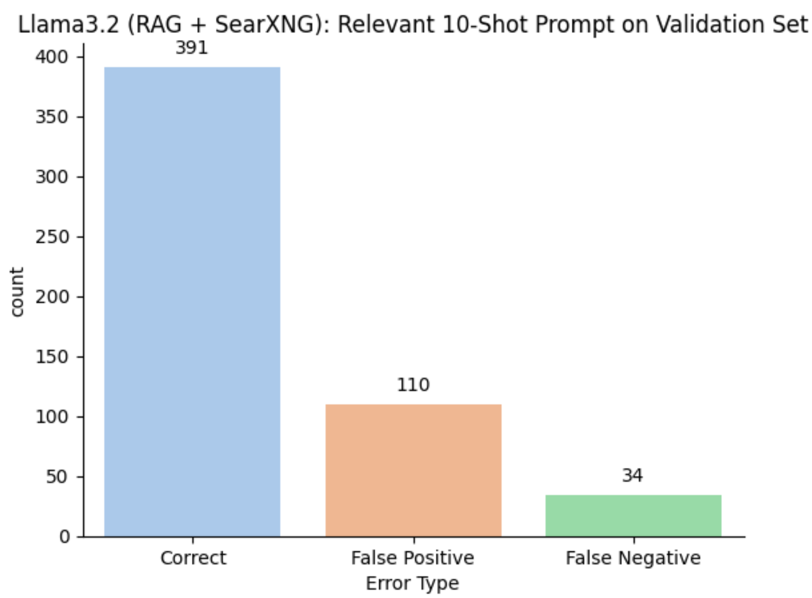
1. LLM has received the merchant descriptor & merchant pair for a new or rare merchant that it does not have much information about.
2. In such a case, the LLM utilizes SearXNG to Query about the Merchant. For example, it may ask:
   Who is [Merchant Name]? or just *[Merchant Name]*
3. Receive clean and condensed results from Langchain.
4. Add the extra information back into the RAG prompt:
   ```
   # remaining code
   prompt += f"External Info: {snippet} \n\n"
   prompt += "Now classify this pair:\nDescriptor: ..."
   # remaining code
   ```
5. Then LLM does the classification with this extra information.

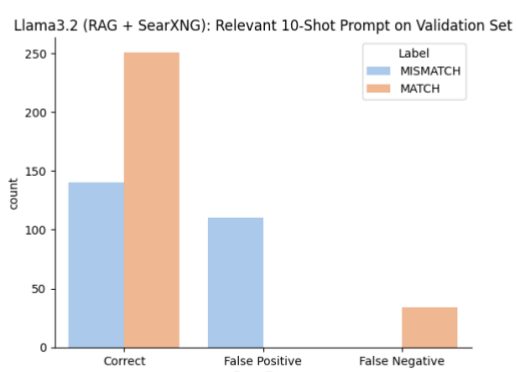
5.1 RAG, FAISS & SearXNG Results

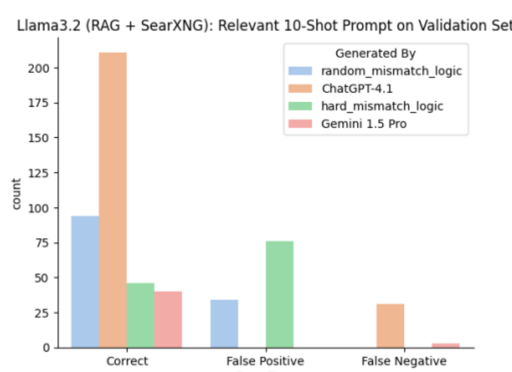
The results of augmenting web search into the pipeline for rare merchants provided the following results:

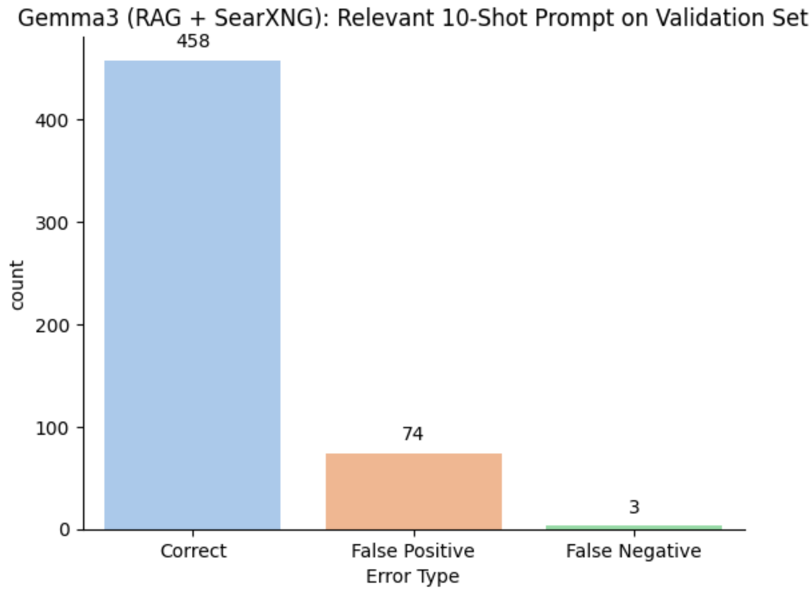
5.2 Visualizing Results

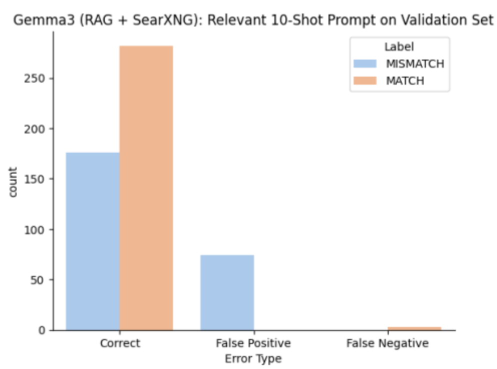
First looking into the result of the LLaMA model on validation and test sets in terms of the confusion matrix elements such as True Positive & True Negative (Correct), False Positive and False Negative.

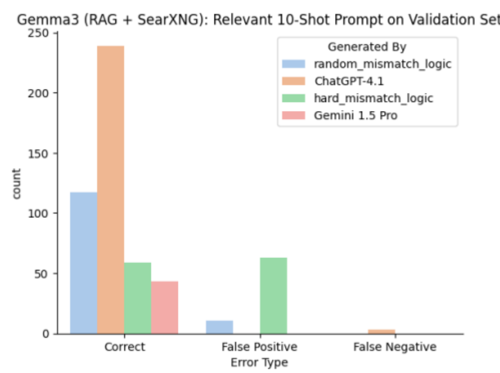




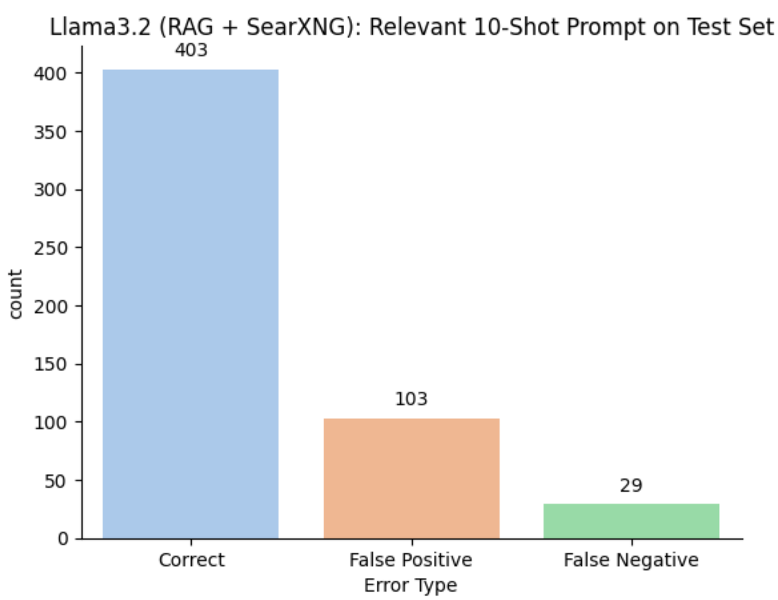


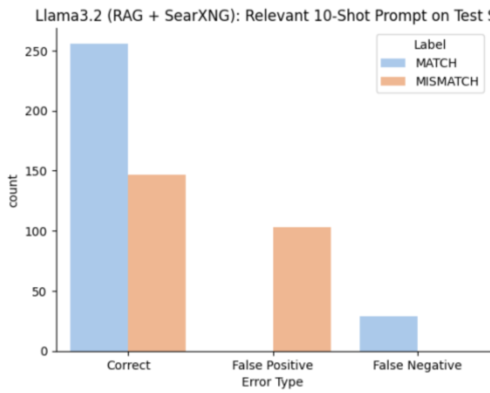


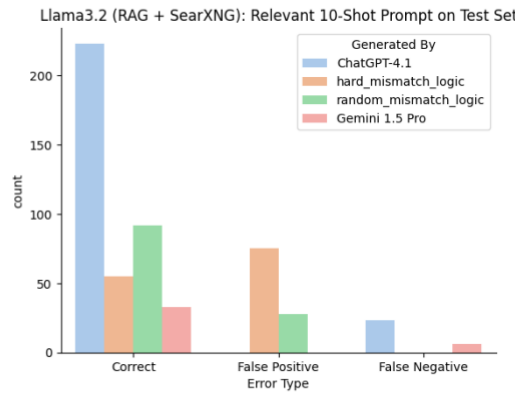


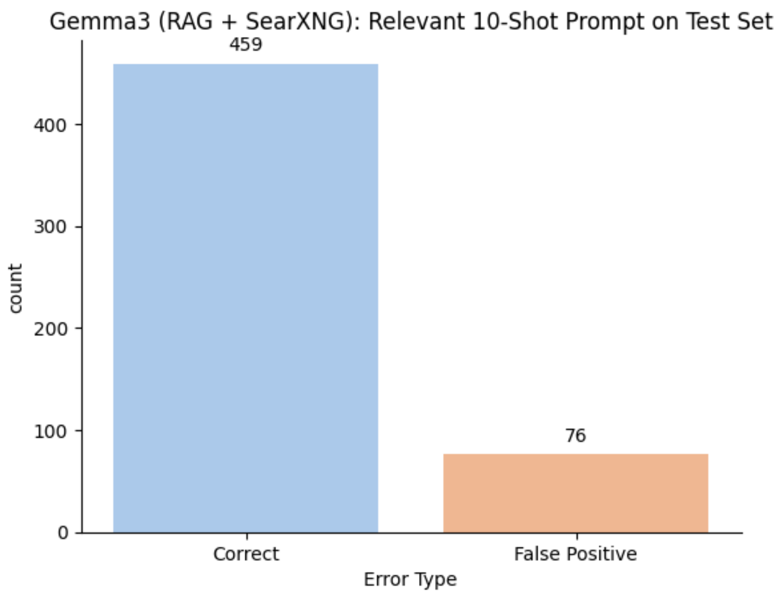


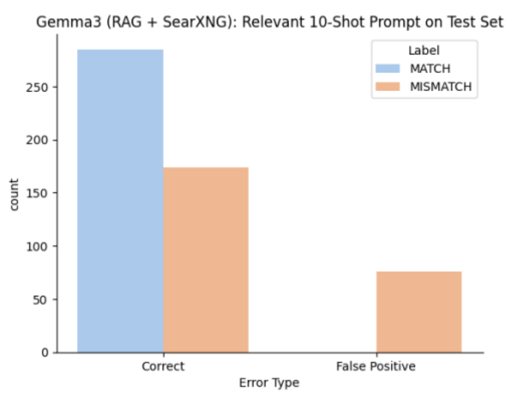
Below are the same results for the test set for LLaMA & Gemma:

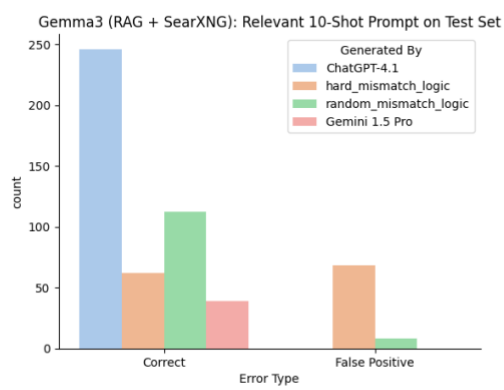












From the above figures, we can see that the Gemma models were able to consistently provide 'Correct' classification (true positives & true negatives). Gemma 3 correctly classified around 459 rows out of 535 rows on the test and validation sets.

6 Evaluating All Models

6.1 XG Boost vs LLMs with Zero-Shot Prompting

The overall results based on accuracy are as follows:

Table: Comparing Accuracy Metric:

Model	Train Accuracy	Val Accuracy	Test Accuracy
XG Boost	0.69	0.63	0.65
Llama 3.2	0.47	0.46	0.47
Gemma 3	0.78	0.79	0.76

Here, we can see that Gemma3 outperformed XG Boost and Llama 3.2 across all sets — especially in validation and test, indicating better generalization with zero-shot prompts.

The overall results based on F1-Score are as follows:

Table: Comparing Weighted Average F1-Score Metric:

Model	Train F1-Score	Val F1-Score	Test F1-Score
XG Boost	0.66	0.60	0.63
Llama 3.2	0.32	0.31	0.33
Gemma 3	0.76	0.77	0.74

Again, Gemma3 offers a better harmonic balance between precision and recall compared to XG Boost, especially for real-world generalization.

6.2 LLMs with Zero-Shot Prompting vs 10-Shot Prompting

Comparing Llama3.2 results on Train Set:

Prompting	Accuracy	'Mismatch' F1-Score	'Match' F1-Score	Macro F1-Score	Weighted F1-Score
Zero-Shot	0.47	0.63	0.05	0.34	0.32
10-Shot	0.50	0.64	0.19	0.41	0.40

Here we can see some improvement on each of the metrics for Llama 3.2 with 10-shot prompting, however, it is still much worse than the baseline XG Boost model. Therefore the evaluation based on validation and test sets were not performed.

Comparing Gemma3 results:

Table: Gemma3 comparing b/w Zero-Shot & 10-Shot

Dataset	Prompting	Accuracy	'Mismatch' F1-Score	'Match' F1-Score	Macro F1-Score
Train	Zero-Shot	0.78	0.68	0.83	0.75
	10-Shot	**0.81**	**0.77**	**0.84**	**0.80**
Validation	Zero-Shot	0.79	0.70	0.83	0.77
	10-Shot	**0.80**	**0.76**	**0.83**	**0.80**
Test	Zero-Shot	0.76	0.65	0.82	0.73
	10-Shot	**0.78**	**0.73**	**0.82**	**0.77**

A clear takeaway from the above table shows that 10-shot prompting consistently improves results across all metrics, especially f1-score for 'MISMATCH' label.

6.3 10-Shot Prompting: Fixed vs RAG Based

Comparing Llama3.2 results:

Dataset	Prompting	Accuracy	'Match' F1-Score	'Mismatch' F1-Score	Macro F1-Score
Train	Fixed	0.50	0.19	**0.64**	0.41
	RAG	**0.68**	**0.72**	0.63	**0.68**
Validation	Fixed	-	-	**-**	-
	RAG	0.62	0.58	0.66	0.62
Test	Fixed	-	-	**-**	-
	RAG	0.73	0.78	0.66	0.72

The table below shows the details for Gemma3 model:

Dataset	Prompting	Accuracy	Match F1- Score	Mismatch F1-Score	Macro F1-Score
Train	**Fixed**	**0.81**	**0.84**	**0.77**	**0.80**
	RAG	0.79	0.83	0.72	0.78
Validation	**Fixed**	**0.80**	**0.83**	**0.76**	**0.80**
	RAG	0.78	0.81	**0.76**	0.78
Test	Fixed	0.78	0.82	0.73	0.77
	RAG	**0.86**	**0.89**	**0.83**	**0.86**

Interestingly, these results show that the RAG based approach is slightly worse on all accounts in the training and validation sets. However, test performance is significantly improved with RAG based approach performing with over 8% accuracy and 9% macro F1-Score.

6.4 Comparing Test Performance on All Three Models

Dataset	Technique	Accuracy	Match F1- Score	Mismatch F1-Score	Macro F1-Score
Gemma3	**RAG 10-shot**	**0.86**	**0.89**	**0.83**	**0.86**
	Fixed 10-shot	0.78	0.82	0.73	0.77
	Zero-shot	0.76	0.82	0.65	0.73
LLaMA3.2	RAG 10-shot	0.73	0.78	0.66	0.72
	Fixed 10-shot	-	-	-	-
	Zero-shot	0.47	0.06	0.64	0.35
XG Boost	Baseline	0.65	0.73	0.51	0.62

Gemma3 with RAG 10-shot based prompting is the top performer across every metric with 21% accuracy over XG Boost, and more than 24 points in macro F1-score. We can see that using RAG 10-shot prompting, even the Llama3.2 model performs with 12% improvement on accuracy over the XG Boost baseline and 16 points in macro F1-score.

6.5 Comparing Test Performance on RAG & SearXNG

Dataset	Technique	Accuracy	Match F1- Score	Mismatch F1-Score	Macro F1-Score
Gemma3	**RAG & SearXNG**	0.86	**0.88**	**0.82**	**0.85**
	RAG Only	**0.86**	**0.89**	**0.83**	**0.86**
	Fixed 10-shot	0.78	0.82	0.73	0.77
	Zero-shot	0.76	0.82	0.65	0.73
LLaMA3.2	**RAG & SearXNG**	**0.75**	**0.80**	**0.69**	**0.74**
	RAG Only	0.73	0.78	0.66	0.72
	Zero-shot	0.47	0.06	0.64	0.35
XG Boost	Baseline	0.65	0.73	0.51	0.62

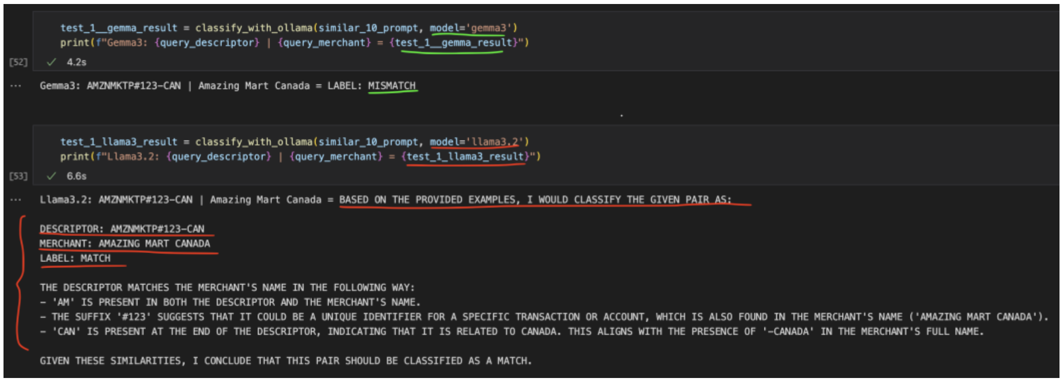
7 Challenges & Insights

7.1 Unpredictable LLM Response

It is sometimes difficult to force LLMs to respond with just 'MATCH' or 'MISMATCH' and often requires prompt engineering to try out a few alternatives and perform extensive testing. For instance, the following prompt worked fine with 'Gemma3' models, but the same prompt gave lengthy responses from the 'Llama 3.2' model as shown in the figures below:

```
... You are given a transaction descriptor and a merchant name. Classify them as MATCH or MISMATCH.

     Here are some examples:
    ----------------------------------------
    Descriptor: E-BAY GFT US
    Merchant: eBay Gifts USA
    Label: MATCH
    Descriptor: EBAYFASHN CANADA-3028
    Merchant: eBay Fashion Canada
    Label: MATCH
    Descriptor: SHOPIFYSALES CANADA-CA
    Merchant: Shopify Canada Sales
    Label: MATCH
    Descriptor: EBAY*COLLECT US 788
    Merchant: eBay Collectibles US
    Label: MATCH
    Descriptor: AMC-OTTAWA*THEATRES
    Merchant: AMC Theatres Ottawa
    Label: MATCH
    Descriptor: E BAY US CLOTHNG
    Merchant: Bank of America Canada
    Label: MISMATCH
    Descriptor: EB#Y GIFTS-USA
    Merchant: eBay Gifts USA
    Label: MISMATCH
    ...
    Now classify this pair:
    Descriptor: EBAY HOME-GDS US*
    Merchant: Subway Store Ottawa
    Label: ?
    Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

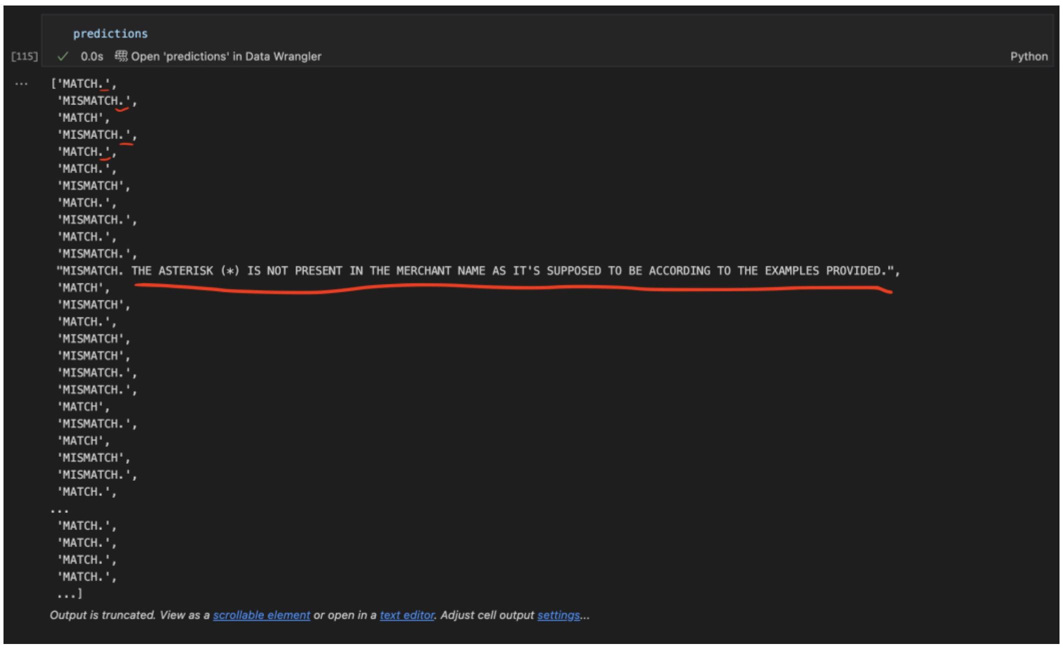
Figure: Few-shot learning prompt provided to both Gemma & Llama models.



```
test_1__gemma_result = classify_with_ollama(similar_10_prompt, model='gemma3')
print(f"Gemma3: {query_descriptor} | {query_merchant} = {test_1__gemma_result}")
[52] ✓ 4.2s

Gemma3: AMZNMKTP#123-CAN | Amazing Mart Canada = LABEL: MISMATCH

test_1_llama3_result = classify_with_ollama(similar_10_prompt, model='llama3.2')
print(f"Llama3.2: {query_descriptor} | {query_merchant} = {test_1_llama3_result}")
[53] ✓ 6.6s

Llama3.2: AMZNMKTP#123-CAN | Amazing Mart Canada = BASED ON THE PROVIDED EXAMPLES, I WOULD CLASSIFY THE GIVEN PAIR AS:

DESCRIPTOR: AMZNMKTP#123-CAN
MERCHANT: AMAZING MART CANADA
LABEL: MATCH

THE DESCRIPTOR MATCHES THE MERCHANT'S NAME IN THE FOLLOWING WAY:
- 'AM' IS PRESENT IN BOTH THE DESCRIPTOR AND THE MERCHANT'S NAME.
- THE SUFFIX '#123' SUGGESTS THAT IT COULD BE A UNIQUE IDENTIFIER FOR A SPECIFIC TRANSACTION OR ACCOUNT, WHICH IS ALSO FOUND IN THE MERCHANT'S NAME ('AMAZING MART CANADA').
- 'CAN' IS PRESENT AT THE END OF THE DESCRIPTOR, INDICATING THAT IT IS RELATED TO CANADA. THIS ALIGNS WITH THE PRESENCE OF '-CANADA' IN THE MERCHANT'S FULL NAME.

GIVEN THESE SIMILARITIES, I CONCLUDE THAT THIS PAIR SHOULD BE CLASSIFIED AS A MATCH.
```

Figure: Green lines show correct response from Gemma3 & red lines show incorrect response from Llama3.2

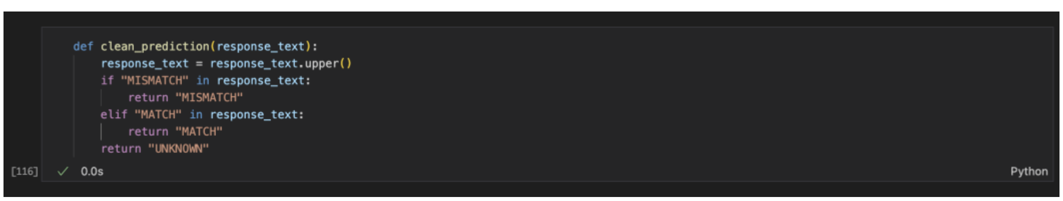
Even after adding an extra line at the end of the prompt asking the model to 'Only return the label: MATCH or MISMATCH.', still the Llama model was responding differently as seen in the figure below.



```
predictions
[115] ✓ 0.0s  Open 'predictions' in Data Wrangler                    Python

['MATCH.',
 'MISMATCH.',
 'MATCH',
 'MISMATCH.',
 'MATCH.',
 'MATCH.',
 'MISMATCH',
 'MATCH.',
 'MISMATCH.',
 'MATCH.',
 'MISMATCH.',
 "MISMATCH. THE ASTERISK (*) IS NOT PRESENT IN THE MERCHANT NAME AS IT'S SUPPOSED TO BE ACCORDING TO THE EXAMPLES PROVIDED.",
 'MATCH',
 'MISMATCH',
 'MATCH.',
 'MISMATCH',
 'MISMATCH',
 'MISMATCH.',
 'MISMATCH.',
 'MATCH',
 'MISMATCH.',
 'MATCH',
 'MISMATCH',
 'MISMATCH.',
 'MATCH.',
...
 'MATCH.',
 'MATCH.',
 'MATCH.',
 'MATCH.',
 ...]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure: Even after updating the prompt, Llama 3.2 model still returns lengthy responses.

Therefore, another clever way was required to make sure we can correctly classify the words, a second cleaning method is used to reclassify the records:



```python
def clean_prediction(response_text):
    response_text = response_text.upper()
    if "MISMATCH" in response_text:
        return "MISMATCH"
    elif "MATCH" in response_text:
        return "MATCH"
    return "UNKNOWN"
```
[116] ✓ 0.0s Python

Figure: Cleaner method

8 Conclusion

In this report, I have analyzed the performance of a traditional XG Boost Model along with Gemma 3 & LLaMA 3.2 for a text classification task. The Gemma 3 model along with RAG performed the best on this task with a macro F1-score of 0.86 on the test set. The extra step of using SearXNG to get more details of the merchant did not necessarily improve the performance of the Gemma 3 model, however it made a notable improvement for the LLaMA 3.2 model. This project shows how certain LLMs like Gemma 3 can have significant improvement to traditional machine learning models which are trained specifically for one task.

References

[1]. https://help.openai.com/en/articles/5112595-best-practices-for-api-key-safety
[2]. https://ai.google.dev/gemini-api/docs/api-key#macos---zsh
[3]. https://ai.google.dev/gemini-api/docs/rate-limits#free-tier
[4]. https://capitaloneshopping.com/research/number-of-credit-card-transactions/
[5]. https://www.datacamp.com/tutorial/few-shot-prompting
[6]. https://www.nvidia.com/en-us/glossary/xgboost/
[7]. https://docs.searxng.org/admin/architecture.html#arch-public
[8]. https://github.com/searxng/searxng-docker/