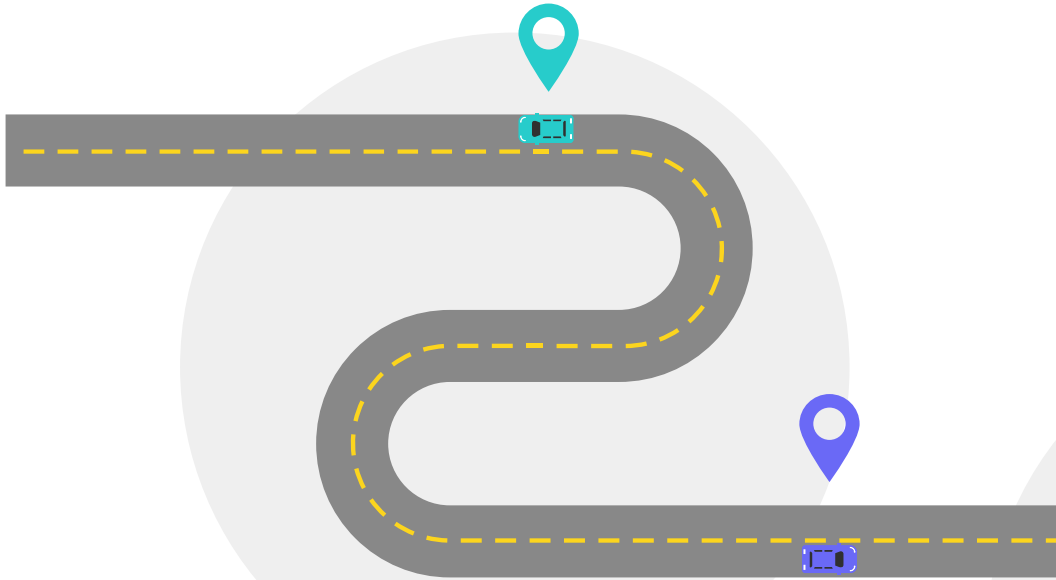
# Investigating Conflation with LLMs

CRWN102 Project C - Ember Lu



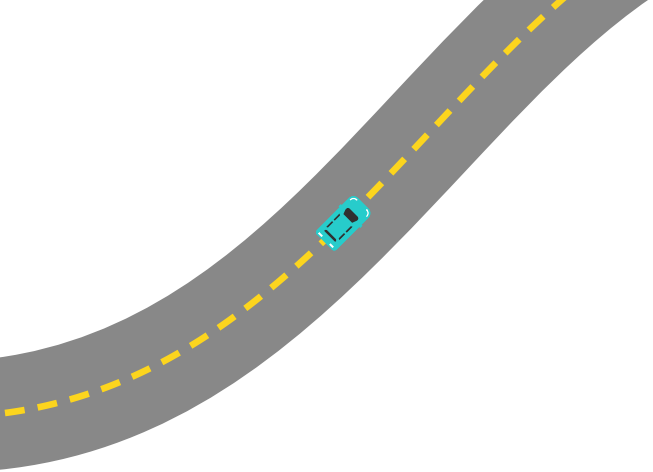
# Project Overview

Problem Statement: Places conflation is “matching records of the same real-world entity from different sources.”

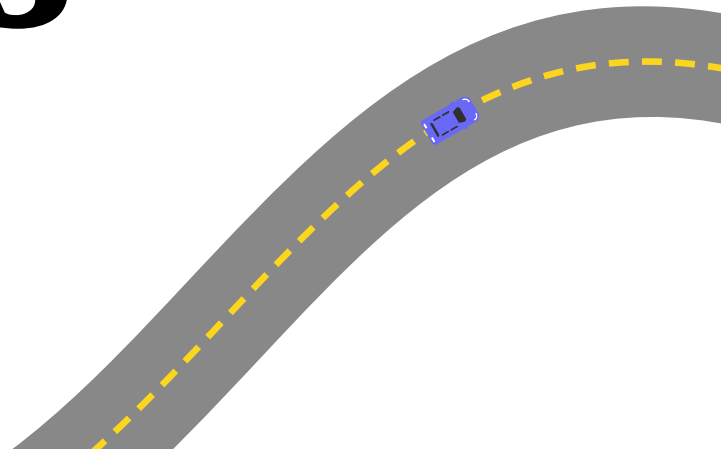
Goal: “Identify a language model that can beat Overture’s in performance while also providing a better price-to-performance ratio.”

**The Bigger Impact**

- Millions of people rely on accurate map data everyday
- From tourists navigating unfamiliar areas to college students accessing their large campuses, accurate map data is the foundation of helping anyone and everyone navigate their daily lives



# OKRs



# Original OKR Draft

## Research 3+ existing online LLM models that can process our JSON geographic data

- Research 5+ of the most widely used small LLM models currently publicly available by reading papers and testing out their demo code if they have any.
- Define 3+ comparable parameters (i.e. accuracy, speed, price etc.) that can be used as benchmarks for each model and its processing of models
- Read up on papers about what static embedding models are

## Evaluate models using test data and real data, and increase their accuracy by at least 20%

- Given the testing datasets from Terraforma, and also publicly available geographic information, continuously train models and read up on how to increase their accuracy
- Analyze the datasets themselves, figure out the best instructions specific to our JSON data that will help the LLM make its decision

# Current OKRs

Identify and benchmark a minimum of three existing LLM models best suited to process JSON/Parquet geographic data

- Research 3+ small and fast LLM/embedding models (Google's Electra, Microsoft's Phi-3 Mini, and Meta's Llama 3.2), currently publicly available by reading papers and testing out their demo code, if they have any.
- Determine and visualize model performance using measurable metrics such as <1K tokens per request, >50% ratio of true positive/negative to false positive/negative place matches (matplotlib), target > 50% Precision and > 50% Recall accuracy calculations, time per request (around or <1sec), pricing per request (<$1.25/1M tokens), etc.

Fine-tune and optimize the most efficient models found to perform better than Overture's existing places matcher in terms of accuracy and efficiency

- Train each of the selected models on existing places datasets provided (3K data points), and places data cleaned to increase the overall model accuracy by at least 10%
- Analyze model performance across 4+ key data segments (starting with dependence on categories, emails, websites, brands, and addresses) to identify strengths and weaknesses and how much accuracy changes with the data available for each feature (i.e. base_email available vs base_email not available).

# OKR Changes

01

**Contextual Performance Metrics**

Additional Precision, Recall, F1 Score, and more!

02

**Model Selections**

1. Narrowing down models
2. Qwen 3 -> Google Electra

03

**System Prompting**

Focusing on LLM prompt simplicity

# Approach & Methods

Libraries: Torch, Pandas, Hugging Face, scikit-learn, matplotlib

```
model_name = "microsoft/phi-3-mini-4k-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)

device = "cuda" if torch.cuda.is_available() else "cpu"

model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    torch_dtype=torch.bfloat16,
    device_map=None,
    num_labels=2
)
```

Load LLM as a Classifier using AutoModelForSequenceClassification()

```
df.drop(['emails', 'base_emails', 'brand', 'base_brand'], axis=1)

train_df, test_df = train_test_split(df, test_size=0.3, random_state=42)
```

Split Dataset into 70% for training data and 30% for testing data

# Approach & Methods

Tokenize dataset and train selected model using Hugging Face Trainer

```python
def tokenize_data(data):
    prompt = (f"Record A: {data['names']} | {data['websites']} | {data['socials']} | {data['emails']} | {data['phones']} | {data['addresses']}\n"
        f"Record B: {data['base_names']} | {data['base_websites']} | {data['base_socials']} | {data['base_emails']} | {data['base_phones']} | {data['base_addresses']}\n"
    )
    tokenized = tokenizer(prompt, max_length=256, truncation=True)
    tokenized["labels"] = int(data["label"])
    tokenized["id"] = data["id"]
    return tokenized
```

```python
training_args = TrainingArguments(
    "test-trainer",
    report_to="none",
    bf16=True,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=test_data,
    tokenizer=tokenizer,
)

trainer.train()
```

Challenge #1: GPU Limitations

- Solution #1: Training models partially using LORA Configs
- Solution #2: Google Colab's Hours using the A100 GPU

Results:

- Training time decreased from 2+ hours to <2 minutes
- Training entire models instead of just parts

# Approach & Methods

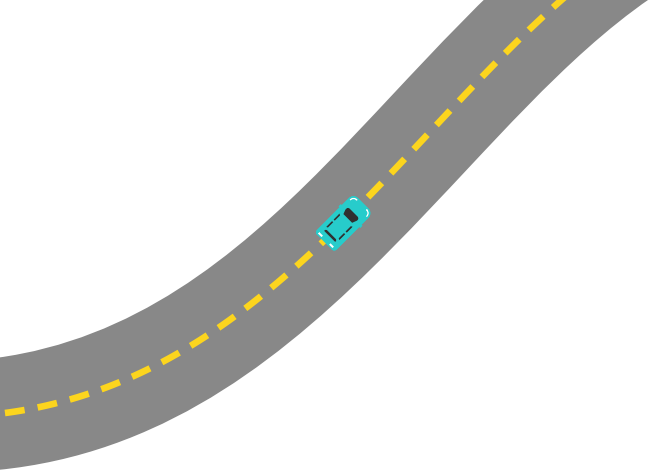
```python
import numpy as np
import time
# Testing Data, Overall Accuracy Metric
start_time = time.time()
test_data = test_data.map(tokenize_data)
results = trainer.predict(test_data)
argmax_preds = torch.argmax(torch.tensor(results.predictions), dim=-1).numpy()
end_time = time.time()
# Output Labels of First 20 Points Tested
print(argmax_preds[:20])

# Output Overall Accuracy
print(f"Accuracy: {np.mean(results.label_ids == argmax_preds)}")

# Output Time 900 Samples
print(f"\nTime: {end_time - start_time:.2f} seconds")
```

1. Testing Dataset using previously allocated 30% of 3K data points

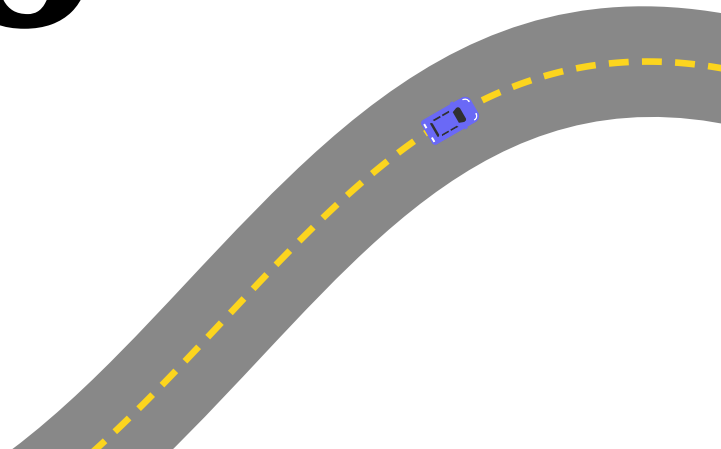
2. Models's metrics are evaluated on 900 data points

3. Visualize Metrics using matplotlib

```python
def plot_metric(title, phi, llama, electra, save_name=None):
    plt.figure(figsize=(8,5))
    plt.plot(scenarios, phi, marker='o', label="PHI-3")
    plt.plot(scenarios, llama, marker='o', label="Llama 3.2")
    plt.plot(scenarios, electra, marker='o', label="ELECTRA")
    plt.ylabel(title)
    plt.title(title)
    plt.grid(True)
    plt.legend()
    plt.xticks(rotation=25)
    plt.tight_layout()

    if save_name:
        plt.savefig(save_name, dpi=300, bbox_inches="tight")

    plt.show()
```



# Demo



# Demo Snapshots (Phi-3)

```python
import numpy as np
import time
# Testing Data, Overall Accuracy Metric
start_time = time.time()
test_data = test_data.map(tokenize_data)
results = trainer.predict(test_data)
argmax_preds = torch.argmax(torch.tensor(results.predictions), dim=-1).numpy()
end_time = time.time()
# Output Labels of First 20 Points Tested
print(argmax_preds[:20])

# Output Overall Accuracy
print(f"Accuracy: {np.mean(results.label_ids == argmax_preds)}")

# Output Time 900 Samples
print(f"\nTime: {end_time - start_time:.2f} seconds")
```

```
Map: 100% 900/900 [00:04<00:00, 582.94 examples/s]
[1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 0]
Accuracy: 0.9066666666666666

Time: 17.20 seconds
```

Accuracy Metrics

```python
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(
    "test-trainer",
    report_to="none",
    bf16=True,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=test_data,
    tokenizer=tokenizer,
)

trainer.train()
```

```
/tmp/ipython-input-2036784160.py:11: FutureWarning: `tokenizer` is
  trainer = Trainer(
[789/789 06:39, Epoch 3/3]
```

| Step | Training Loss |
|---|---|
| 500 | 0.318400 |

Training the Model

Tokens for 900 Requests

```python
# Token Counts
import numpy as np
num_tokens = sum(len(example["input_ids"]) for example in train_data)
print("Training Tokens: ", num_tokens)
total_eval_tokens = sum(len(ex["input_ids"]) for ex in test_data)
print("Testing Tokens: ", total_eval_tokens)
```

```
Training Tokens:  446392
Testing Tokens:  193399
```

Metrics Calculations (F1, Precision, Recall, etc.)

```python
# True/False Positives/Negatives Counts, Precision, Recall, and F1 Score Metrics
tp_count = 0
fp_count = 0
tn_count = 0
fn_count = 0
for x in range(len(results.label_ids)):
  if (results.label_ids[x] == 1 and argmax_preds[x] == 1):
    tp_count += 1
  elif (results.label_ids[x] == 0 and argmax_preds[x] == 0):
    tn_count += 1
  elif (results.label_ids[x] == 0 and argmax_preds[x] == 1):
    fn_count += 1
  else:
    fp_count += 1

print("True Positive: ", tp_count)
print("True Negative: ", tn_count)
print("TP:FP - ", float(tp_count)/fp_count)
print("False Positive: ", fp_count)
print("False Negative: ", fn_count)
print("TN:FN - ", float(tn_count)/fn_count)
precision = tp_count/(tp_count + fp_count)
print("Precision: ", precision)
recall = tp_count/(tp_count + fn_count)
print("Recall: ", recall)
print("F1 Score: ", 2*float(precision)*recall/(precision+recall))
```

```
True Positive:  515
True Negative:  301
TP:FP -  13.205128205128204
False Positive:  39
False Negative:  45
TN:FN -  6.688888888888889
Precision:  0.9296028808866426
Recall:  0.9196428571428571
F1 Score:  0.9245960502692998
```

# Demo Snapshots (Llama 3.2)

```python
import numpy as np
import time
# Testing Data, Overall Accuracy Metric
start_time = time.time()
test_data = test_data.map(tokenize_data)
results = trainer.predict(test_data)
argmax_preds = torch.argmax(torch.tensor(results.predictions), dim=-1).numpy()
end_time = time.time()
# Output Labels of First 20 Points Tested
print(argmax_preds[:20])

# Output Overall Accuracy
print(f"Accuracy: {np.mean(results.label_ids == argmax_preds)}")

# Output Time 900 Samples
print(f"\nTime: {end_time - start_time:.2f} seconds")
```

```
Map: 100% 900/900 [00:01<00:00, 547.90 examples/s]
[1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1]
Accuracy: 0.6822222222222222

Time: 5.80 seconds
```

Accuracy Metrics

```python
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(
    "test-trainer",
    report_to="none",
    bf16=True,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=test_data,
    tokenizer=tokenizer,
)

trainer.train()
```

```
/tmp/ipython-input-1297736554.py:11: FutureWarning: `tokenizer`
  trainer = Trainer(
[789/789 01:55, Epoch 3/3]

Step  Training Loss
500   0.872000
```

Training the Model

Tokens for 900 Requests

```python
# Token Counts
import numpy as np
num_tokens = sum(len(example["input_ids"]) for example in train_data)
print("Training Tokens: ", num_tokens)
total_eval_tokens = sum(len(ex["input_ids"]) for ex in test_data)
print("Testing Tokens: ", total_eval_tokens)
```

```
Training Tokens:  342519
Testing Tokens:  148510
```

Metrics Calculations (F1, Precision, Recall, etc.)

```python
# True/False Positives/Negatives Counts, Precision, Recall, and F1 Score Metrics
tp_count = 0
fp_count = 0
tn_count = 0
fn_count = 0
for x in range(len(results.label_ids)):
  if (results.label_ids[x] == 1 and argmax_preds[x] == 1):
    tp_count += 1
  elif (results.label_ids[x] == 0 and argmax_preds[x] == 0):
    tn_count += 1
  elif (results.label_ids[x] == 0 and argmax_preds[x] == 1):
    fn_count += 1
  else:
    fp_count += 1

print("True Positive: ", tp_count)
print("True Negative: ", tn_count)
print("TP:FP - ", float(tp_count)/fp_count)
print("False Positive: ", fp_count)
print("False Negative: ", fn_count)
print("TN:FN - ", float(tn_count)/fn_count)
precision = tp_count/(tp_count + fp_count)
print("Precision: ", precision)
recall = tp_count/(tp_count + fn_count)
print("Recall: ", recall)
print("F1 Score: ", 2*float(precision)*recall/(precision+recall))
```

```
True Positive:  437
True Negative:  177
TP:FP -  3.735042735042735
False Positive:  117
False Negative:  169
TN:FN -  1.047337278106509
Precision:  0.7888086642599278
Recall:  0.7211221122112211
F1 Score:  0.753448275862069
```

# Demo Snapshots (Electra)

```python
import numpy as np
import time
# Testing Data, Overall Accuracy Metric
start_time = time.time()
test_data = test_data.map(tokenize_data)
results = trainer.predict(test_data)
argmax_preds = torch.argmax(torch.tensor(results.predictions), dim=-1).numpy()
end_time = time.time()
# Output Labels of First 20 Points Tested
print(argmax_preds[:20])

# Output Overall Accuracy
print(f"Accuracy: {np.mean(results.label_ids == argmax_preds)}")

# Output Time 900 Samples
print(f"\nTime: {end_time - start_time:.2f} seconds")
```

```
Map: 100% 900/900 [00:01<00:00, 508.72 examples/s]
[1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0]
Accuracy: 0.8277777777777777

Time: 4.04 seconds
```

Accuracy Metrics

```python
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(
    "test-trainer",
    report_to="none",
    bf16=True,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=test_data,
    tokenizer=tokenizer,
)

trainer.train()
```

```
/tmp/ipython-input-2036784160.py:11: FutureWarning: `tokenizer`
  trainer = Trainer(
[789/789 00:42, Epoch 3/3]
```

| Step | Training Loss |
|---|---|
| 500 | 0.537200 |

Training the Model

Tokens for 900 Requests

```python
# Token Counts
import numpy as np
num_tokens = sum(len(example["input_ids"]) for example in train_data)
print("Training Tokens: ", num_tokens)
total_eval_tokens = sum(len(ex["input_ids"]) for ex in test_data)
print("Testing Tokens: ", total_eval_tokens)
```

```
Training Tokens:  484349
Testing Tokens:  209231
```

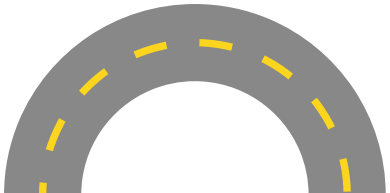
Metrics Calculations (F1, Precision, Recall, etc.)

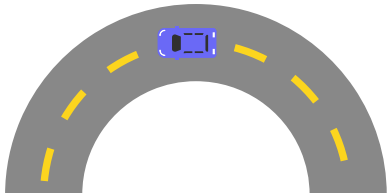
```python
# True/False Positives/Negatives Counts, Precision, Recall, and F1 Score Metrics
tp_count = 0
fp_count = 0
tn_count = 0
fn_count = 0
for x in range(len(results.label_ids)):
  if (results.label_ids[x] == 1 and argmax_preds[x] == 1):
    tp_count += 1
  elif (results.label_ids[x] == 0 and argmax_preds[x] == 0):
    tn_count += 1
  elif (results.label_ids[x] == 0 and argmax_preds[x] == 1):
    fn_count += 1
  else:
    fp_count += 1

print("True Positive: ", tp_count)
print("True Negative: ", tn_count)
print("TP:FP - ", float(tp_count)/fp_count)
print("False Positive: ", fp_count)
print("False Negative: ", fn_count)
print("TN:FN - ", float(tn_count)/fn_count)
precision = tp_count/(tp_count + fp_count)
print("Precision: ", precision)
recall = tp_count/(tp_count + fn_count)
print("Recall: ", recall)
print("F1 Score: ", 2*float(precision)*recall/(precision+recall))
```

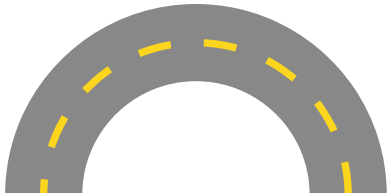
```
True Positive:  475
True Negative:  270
TP:FP -  6.012658227848101
False Positive:  79
False Negative:  76
TN:FN -  3.5526315789473686
Precision:  0.8574007220216606
Recall:  0.8620689655172413
F1 Score:  0.8597285067873303
```

# Outcomes (with Contextual Performance)

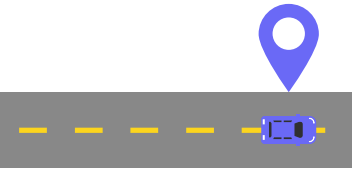
| Microsoft Phi-3 | Meta Llama 3.2 | Google Electra |
|---|---|---|
| • Accuracy: 90.4%<br>• Precision: 0.93<br>• Recall: 0.92<br>• F1 Score: 0.92<br>• $0.13 per 1M tokens<br>• 15.5 ms / request | • Accuracy: 89%<br>• Precision: 0.90<br>• Recall: 0.92<br>• F1 Score: 0.91<br>• $0.06 per 1M tokens<br>• 6.34 ms / request | • Accuracy: 85.7%<br>• Precision: 0.90<br>• Recall: 0.87<br>• F1 Score: 0.88<br>• Open Source<br>• 4.57 ms / request |
| Phi-3 performs best when the categories and brands features are excluded<br>• F1 Score: 0.941 | Llama 3.2 performs best when the no features are excluded | Electra performs best when the emails and brands features are excluded<br>• F1 Score: 0.892 |







# OKRs Met!



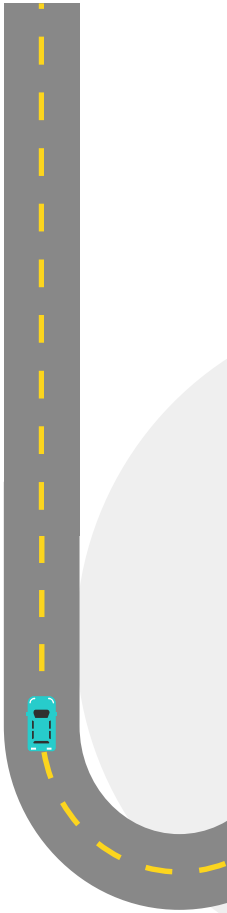
✅ 50%+ ratio of true positive/negative to false positive/negative place matches (matplotlib)

✅ Target > 50% Precision and > 50% Recall accuracy calculations

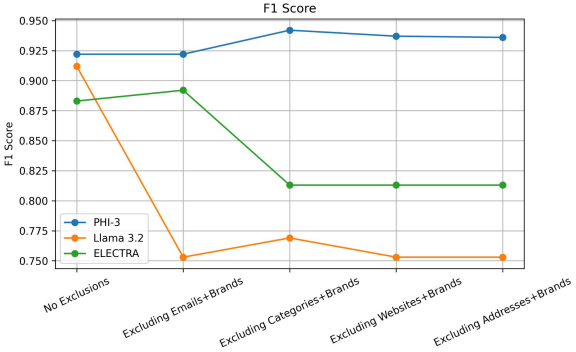
✅ F1 Score > 80%

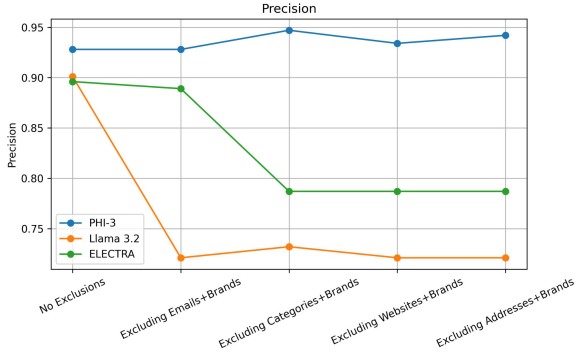
✅ Pricing per request (<$1.25/1M tokens)

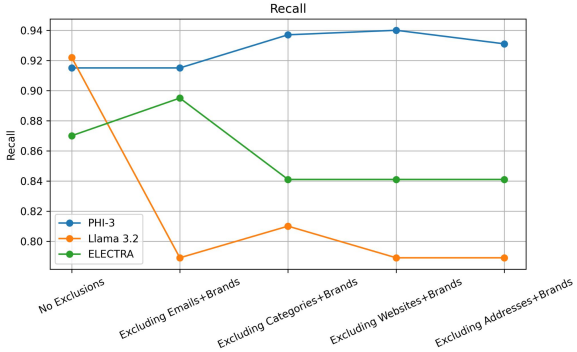
✅ Time per request (around or <1sec)

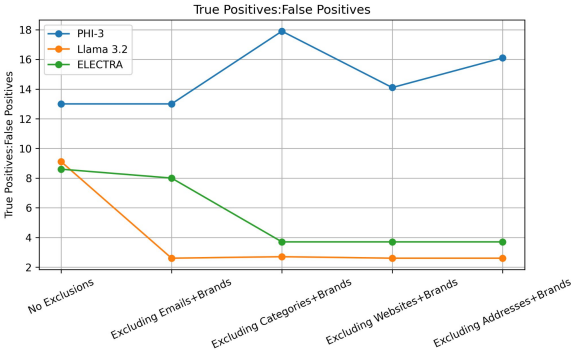


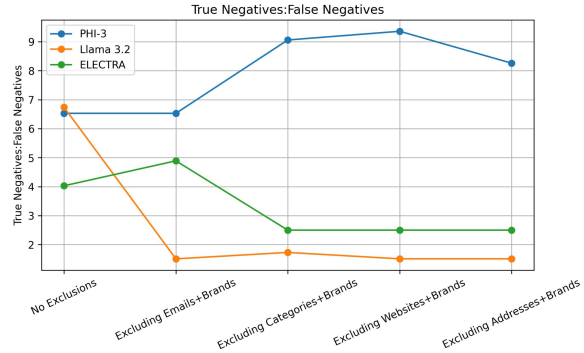
# Metric Visualization Plots

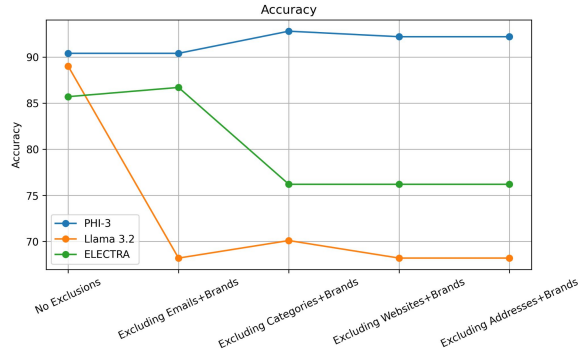


# Insights

## Best Performance

Phi-3

Phi-3 has the highest F1 score and accuracy overall, but is also heavy in token use and pricing

## Best Price-to-Performance

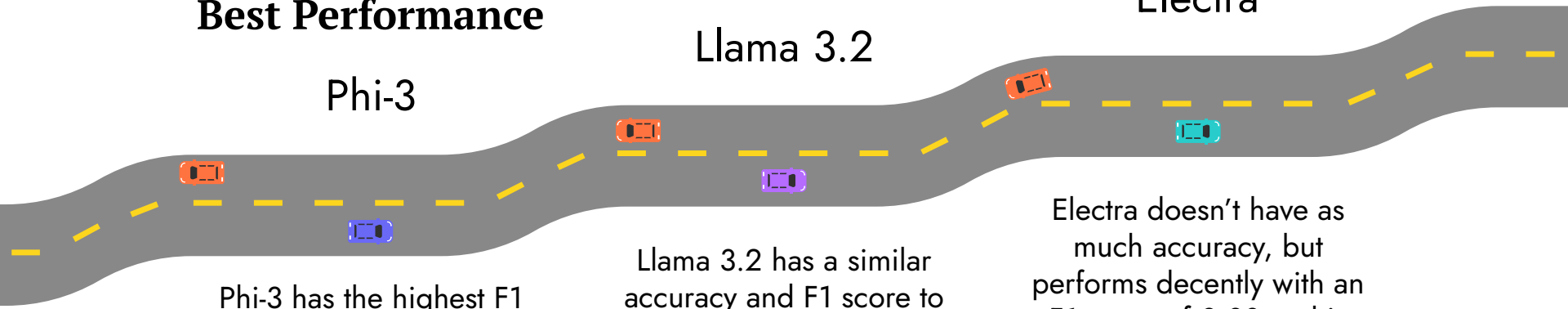
Llama 3.2

Llama 3.2 has a similar accuracy and F1 score to the Phi-3, but is half the price and token usage

## Best Resources

Electra

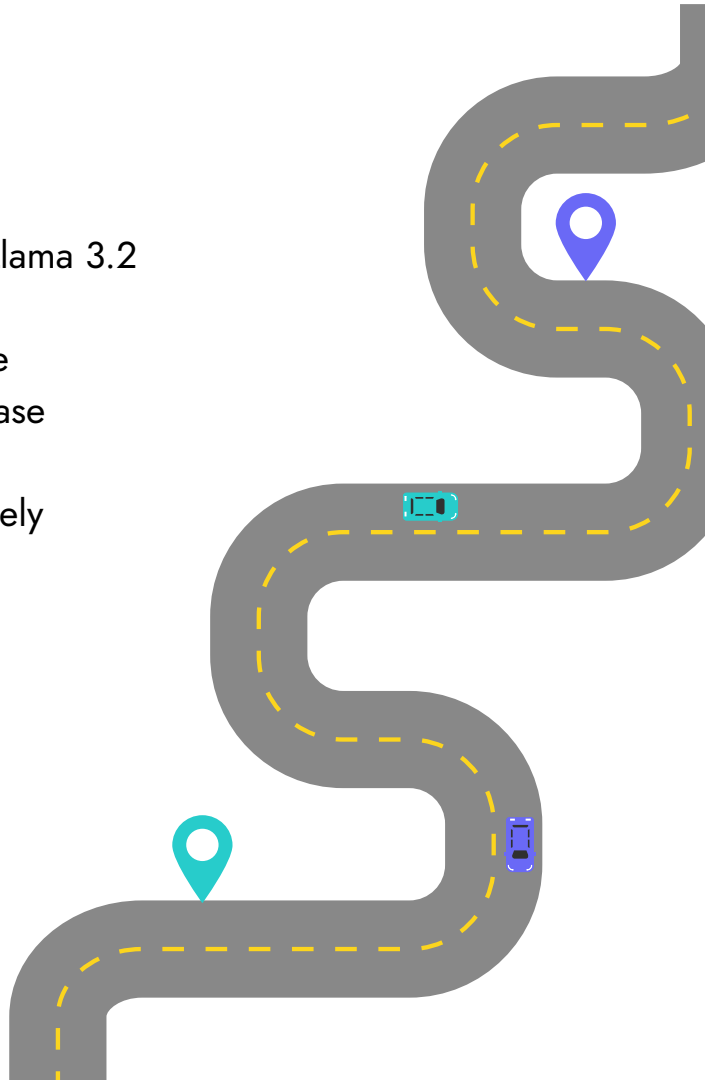
Electra doesn't have as much accuracy, but performs decently with an F1 score of 0.88 and is fastest and cheapest with the same token usage as the Phi-3

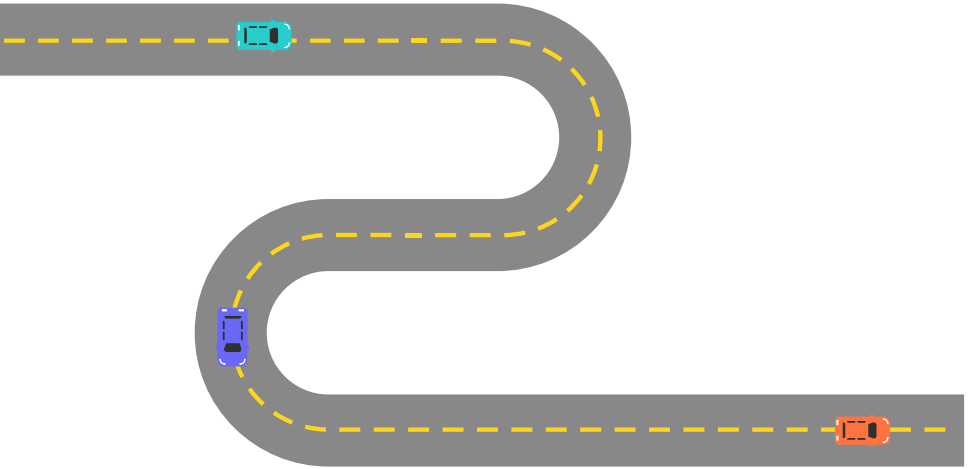


# Reflection & Next Steps

- The best price-to-performance model evaluated is the Meta Llama 3.2 with 1B parameters
  - With more and more Llama 4 versions coming out, the performance of Llama models with geodata may increase
- Should this replace OMF's current matching model? No.
  - Only matches 94% of the places OMF's model accurately matches at best.

- Next…
  - Put more of the dozens of models available on Hugging Face to the test
  - Train on advanced GPUs from the start, decreasing training time and resources
  - Full-stack GUI for inputting data into selected previously-trained LLMs & comparing results live

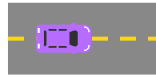




# Thank you!

Ember Lu
emrlu@ucsc.edu / emberlu@gmail.com
Thank you to Professor Rao and OMF for their mentorship & guidance through this project!

github.com/project-terraforma/conflation-with-llm



For a list of all metrics with contextual performance, visit the QR code or https://tinyurl.com/projectc-metrics