DATA+AI SUMMIT BY databricks

Video AD Classification Across Millions of Classes

Puneet Jain
Thierry Steenberghs
James Kim

AGENDA

What we will cover today

Ray on Databricks with Spark structured streaming

- Motivation & Challenges
- Sample Code

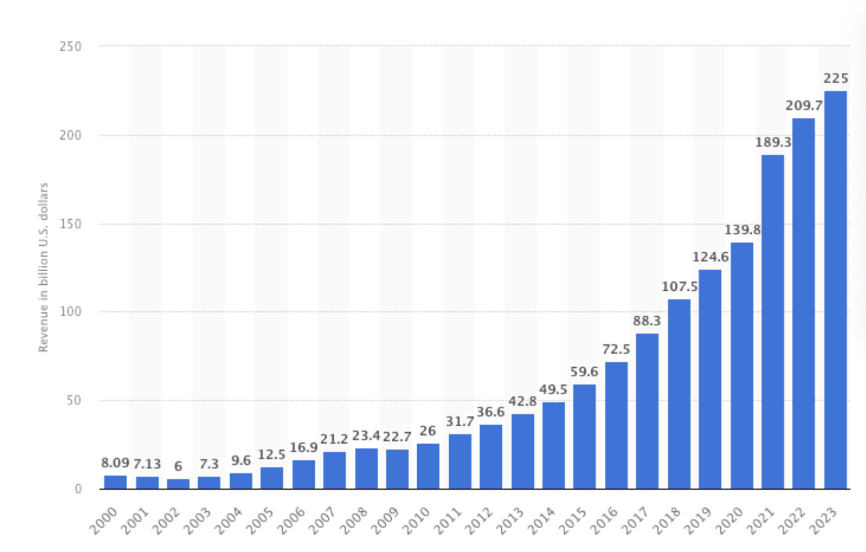
Classification with GenAI across millions of classes

- Motivation & Challenges
- Process of the ML pipeline
- Sample Code
- Q&A

CLASSIFICATION WITH GENAI

Motivation and Challenges

- The amount of ads is increasing exponentially year over year. (Online Advertising Revenue went from 8 to 225 Billion from 2000 to 2023)
    - Almost doubled from 2020 to 2023 from 140 to 225 billion.
- Needed an automated solution to solve the increasing number of ads



CLASSIFICATION WITH GENAI

Motivation and Challenges

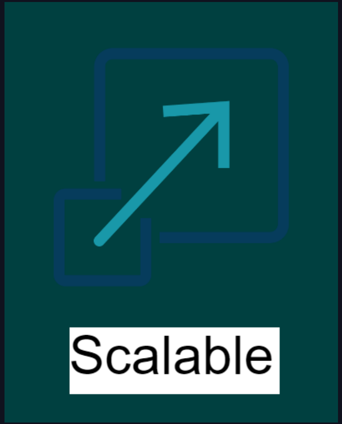
- MediaRadar|Vivvix is a Advertising Intelligence company
- We operate on all Medium:
    - TV (Broadcast/Cable & OnDemand)
    - Print (Newspapers & Magazines)
    - Radio
    - Digital (Online & Mobile)
    - Podcast
    - Outdoors
    - Cinema

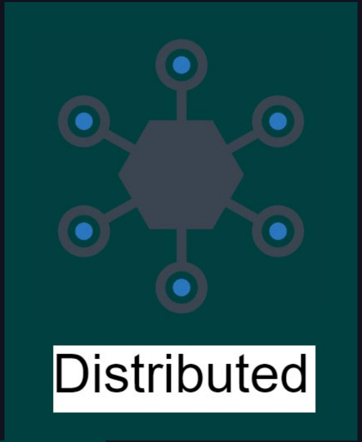
CLASSIFICATION WITH GENAI

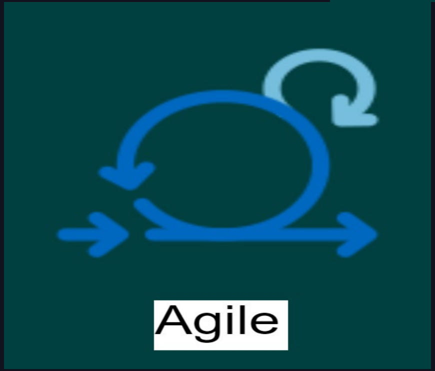
Motivation and Challenges

- Our customers want
  - Accurate Branding (advertised product/service)
  - Accurate Terms (offers)
  - Near real time reporting
  - Representation of multi-lingual creatives
- Need to minimize human classification/attribution

Driving Factors Behind Architecture







Building Component 1

databricks

Security & Governance provides robust security and governance features, including data encryption, access control, compliance certifications, and auditing.	**Data Processing & Management** provides a unified platform for data processing and management, including data ingestion, data cleansing, data transformation, and data integration.
Scalability & Performance built on top of Apache Spark, a distributed computing engine that can process large datasets in parallel. This enables Databricks to handle big data workloads and scale up or down based on demand	**Collaboration & Integrated ML Lib/Framework** provides an interactive workspace with notebooks that support multiple languages such as Python & SQL. integrates well with popular ML libraries and frameworks such as TensorFlow, PyTorch, Scikit

Building Component 2

RAY

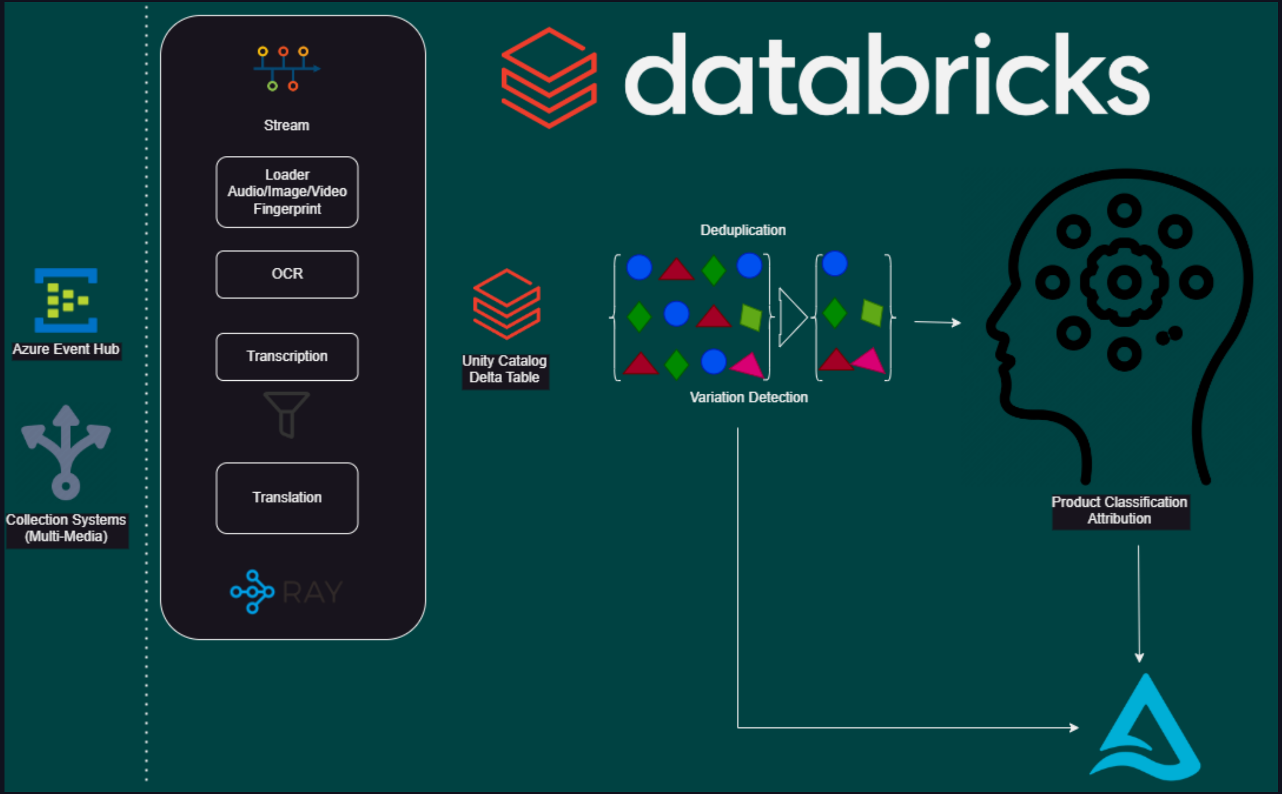
Ray is an open-source unified framework for scaling AI and Python applications like machine learning. It provides the compute layer for parallel processing so that you don't need to be a distributed systems expert. Ray minimizes the complexity of running your distributed individual and end-to-end machine learning workflows

Ease of Use provides a simple and intuitive API for distributed computing, with support for dynamic task parallelism, data parallelism, and actor-based concurrency.	**Flexibility** provides a flexible and extensible architecture, with support for custom schedulers, execution engines, and resource managers.
Scalability & Performance provides efficient and scalable distributed computing, with support for dynamic resource allocation, fault tolerance, and distributed memory management. provides support for fractional GPUs, enabling developers to share GPUs among multiple tasks and optimize GPU utilization	**Integration** provides seamless integration with popular ML libraries and frameworks, such as TensorFlow, PyTorch, and Scikit, making it easier to distribute ML tasks and use fractional GPUs.

https://docs.databricks.com/en/machine-learning/ray-integration.html

CLASSIFICATION WITH GENAI

Basic Architecture



CODE SAMPLE

Configuration

PYTHON

```python
#On the compute config make sure you have the following
#spark.task.resource.gpu.amount 0

# Install what will make the magic a reality
%pip install ray[default,tune,client]==2.10.0

# Let's setup the ray cluster
from ray.util.spark import setup_ray_cluster, shutdown_ray_cluster
ay_conf = setup_ray_cluster(
 min_worker_nodes=2, # this permits scaling of the cluster
 max_worker_nodes=4, # from min to max nodes
 num_cpus_head_node= 3, # all the numbers from here are dependent on
 num_gpus_head_node= 1, # the compute setup.
 num_cpus_per_node= 4,
 Num_gpus_per_node = 1
)
```

CODE SAMPLE

Setting up

```
# Let’s setup the work.Sizing up the number of actors will determine how much performance we can get.And
notice how GPUs can be split. An actor can use a fractional GPU, all depends on how much VRAM is consumed by
the process.
@F.pandas_udf(T.StringType())
def parse_creatives(urls: pd.Series) -> pd.Series:
    start = time.time()
    import ray
    import ray.data

    @ray.remote
    def ray_data_task(ds = None):
        ds = ray.data.from_pandas(pd.DataFrame(urls.to_list(),columns = ['combo']))

        print("shape:",urls.shape[0])
        preds = (
        ds.repartition(urls.shape[0])
        .map(
            FingerprintAudio,
            compute=ray.data.ActorPoolStrategy(min_size=1,max_size=18),
            num_cpus=1,)
```

CODE SAMPLE

Setting up (Cont'd)

```
.map(
            WhisperTranscription,
            compute=ray.data.ActorPoolStrategy(min_size=1,max_size=10),
            num_gpus=.5,
        )
        .map(
            VideoOCR,
            compute=ray.data.ActorPoolStrategy(min_size=1,max_size=18),
            num_cpus=1,
        ))
        end = time.time()
        print("Loaded model dependencies" ,end - start)

        final_df = preds.to_pandas()

        return final_df['final_dict']

    return ray.get(ray_data_task.remote(urls))
```

CLASSIFICATION WITH GENAI

ML PIPELINE PROCESS

DATA PREPROCESSING

pandas

LLM PREDICTED CLASSES

DATA + LLM PREDICTION

SIMILARITY MATCHING (Fuzzywuzzy, Embeddings, etc)

TOP N MATCH PREDICTIONS

CODE SAMPLE

CREATE ENDPOINT

PYTHON

```python
import mlflow.deployments
#Initialize create a Databricks External Model for enhanced governance as it is compatible with OpenAI SDK.
client = mlflow.deployments.get_deploy_client("databricks")

client.create_endpoint(
   name="openai-completions-endpoint",
   config={"served_entities": [
           {"name": "openai-completions-endpoint",
               "external_model": {
                   "name": "gpt-3.5-turbo-0125",
                   "provider": "openai",
                   "task": "llm/v1/completions",
                   "anthropic_config": {
                       "openapi_key": "{{secrets/my_openapi_scope/openai_api_key}}"}}}]}
```

CODE SAMPLE

INITIALIZING OPENAI CLIENT

PYTHON

```python
import os
from openai import OpenAI


api_key = "API_KEY" #your Databricks PAT token


# Initialize the OpenAI client
client = OpenAI(
   api_key="api_key",
   base_url="https://example.staging.cloud.databricks.com/serving-endpoints/openai-completions-endpoint")

INPUT = "RANGEROVER SPORT rangerover sport dynamic air suspension wheel steering configurable terrain response
effortless extreme dynamic air suspension wheel steering configurable terrain response"
```

CODE SAMPLE

MAKING THE API CALL

PYTHON

```python
# Make the API call
response = client.chat.completions.create(
  model="gpt-3.5-turbo-0125",
  messages=[{
          "role": "system",
          "content": "You will be provided with a OCR and audio transcription from a video advertisement. ONLY output the brand or company AND what is being advertised separated by a comma."},
      {"role": "user",
       "content": "INPUT: " + INPUT}],
  temperature=0,
  max_tokens=256,
  top_p=1,
  frequency_penalty=0,
  presence_penalty=0)

# print response
print(response.choices[0].message.content)
```

CODE SAMPLE

SAMPLE SIMILARITY MATCH

PYTHON

```python
from fuzzywuzzy import process

def get_top_matches(query, choices, limit=3):
  results = process.extract(query, choices, limit=limit)
  return results

product_list = ["Range Rover Sport", "Toyota Highlander", "Hyundai Sonata", "Google Pixel 5", "Samsung Galaxy
Buds+", "Apple iPhone 11"]
product_name = "Range Rover"

top_matches = get_top_matches(product_name, product_list)

print("Top 3 similar products:")
for product, score in top_matches:
  print(f"{product} with a similarity score of {score}")
```

Q&A