Multimodal Reasoning AI Agent 🧠

Upload an image and provide a reasoning-based task for the AI Agent. The AI Agent will analyze the image and respond based on your input.

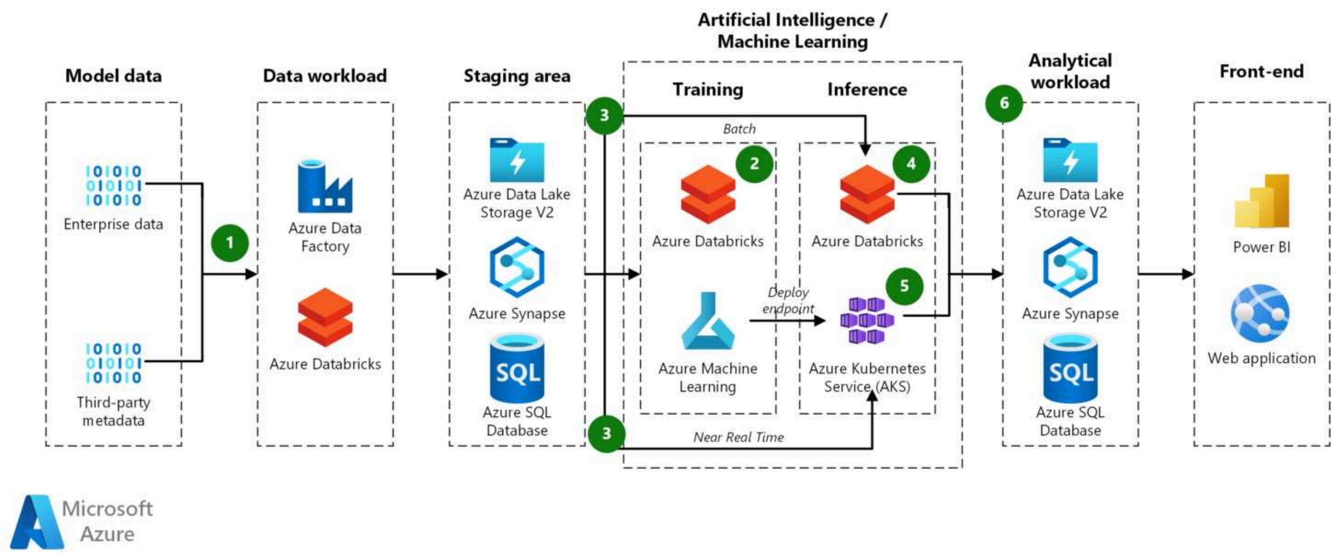
Upload Image

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files

many-models-machine-learning-azure-spark.png 208.8KB



Uploaded Image

Enter your task/question for the AI Agent:

write code for this data flow

Analyze Image

AI Response:

```
## Code Representation of the Data Flow

This outlines the code and configurations involved in each step of the data flow d

**1. Data Ingestion (Model data -> Data workload)**
```

````
This step involves ingesting data from various sources into the data workload. Azu

```python
# Azure Data Factory Pipeline Definition (Conceptual)

from azure.mgmt.datafactory import DataFactoryManagementClient
from azure.mgmt.datafactory.models import (
    LinkedService,
    AzureBlobStorageLinkedService,
    AzureDatabricksLinkedService,
    CopyActivity,
    Dataset,
    AzureBlobDataset,
    Pipeline,
    ActivityDependency,
    DependencyCondition,
)
from azure.common.credentials import ServicePrincipalCredentials

# Replace with your actual credentials and resource group
subscription_id = "your_subscription_id"
resource_group_name = "your_resource_group"
factory_name = "your_data_factory_name"

# Authenticate to Azure
credentials = ServicePrincipalCredentials(
    client_id="your_client_id",
    secret="your_client_secret",
    tenant="your_tenant_id"
)

adf_client = DataFactoryManagementClient(credentials, subscription_id)

# Define Linked Services (connections to data sources and sinks)
blob_storage_ls = LinkedService(
    properties=AzureBlobStorageLinkedService(
        connection_string="your_blob_storage_connection_string"
    )
)
databricks_ls = LinkedService(
    properties=AzureDatabricksLinkedService(
        domain="your_databricks_workspace_url",
        access_token="your_databricks_access_token"
    )
)

# Define Datasets (structures representing data in linked services)
enterprise_data_dataset = Dataset(
````
````

```
    properties=AzureBlobDataset(
        linked_service_name="your_enterprise_data_storage_linked_service", # Repla
        folder_path="your_enterprise_data_folder",
        file_name="your_enterprise_data_file.csv"
    )
)

third_party_metadata_dataset = Dataset(
    properties=AzureBlobDataset(
        linked_service_name="your_third_party_metadata_storage_linked_service", #
        folder_path="your_third_party_metadata_folder",
        file_name="your_third_party_metadata_file.json"
    )
)

# Define Copy Activities (moving data between datasets)
copy_enterprise_data_activity = CopyActivity(
    name="CopyEnterpriseData",
    inputs=[enterprise_data_dataset],
    outputs=[
        Dataset(reference_name="your_staging_data_lake_dataset") # Replace with ac
    ]
)

copy_third_party_metadata_activity = CopyActivity(
    name="CopyThirdPartyMetadata",
    inputs=[third_party_metadata_dataset],
    outputs=[
        Dataset(reference_name="your_staging_data_lake_dataset") # Replace with ac
    ]
)

# Define Pipeline
pipeline = Pipeline(
    activities=[
        copy_enterprise_data_activity,
        copy_third_party_metadata_activity
    ]
)

# Create or update the pipeline
adf_client.pipelines.create_or_update(
    resource_group_name, factory_name, "ingestion_pipeline", pipeline
)
```

2. Training (Data workload → Training)

Azure Databricks is used for batch training of the machine learning model. This involves running Spark code.

```
# Azure Databricks Notebook (Python with Spark) - Training

from pyspark.sql import SparkSession
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
from mlflow import mlflow

# Initialize Spark Session
spark = SparkSession.builder.appName("ModelTraining").getOrCreate()

# Load data from staging area (Azure Data Lake Storage Gen2)
data = spark.read.format("delta").load("wasbs://<container>@<storage_account>.dfs.

# Feature Engineering (example)
feature_cols = ["feature1", "feature2", "feature3"]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

# Train the model (example: Logistic Regression)
lr = LogisticRegression(labelCol="label", featuresCol="features")

# Create a pipeline
pipeline = Pipeline(stages=[assembler, lr])

# Train the model
model = pipeline.fit(data)

# Log the model with MLflow
with mlflow.start_run():
    mlflow.spark.log_model(model, "logistic_regression_model")
    # Log parameters and metrics as needed

# Save the trained model (alternatively, could be saved to Azure ML Model Registry
model.save("dbfs:/mnt/models/trained_model")

spark.stop()
```

3. Staging Area (Data workload → Staging area)

Data is moved and stored in the staging area, which includes Azure Data Lake Storage V2, Azure Synapse, and Azure SQL Database. This can be done via Data Factory or within the Databricks notebooks.

```
# Azure Databricks Notebook (Python with Spark) - Staging
```

```
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("DataStaging").getOrCreate()

# Load processed data
processed_data = spark.read.format("delta").load("wasbs://<container>@<storage_acc

# Write to Azure Data Lake Storage Gen2 (Delta format)
processed_data.write.format("delta").mode("overwrite").save("wasbs://<container>@<

# Write to Azure Synapse Analytics (using Synapse Spark Connector)
processed_data.write \
    .format("com.databricks.spark.sqldw") \
    .option("url", "jdbc:sqlserver://<your_synapse_server>.database.windows.net:14
    .option("dbtable", "<your_synapse_table>") \
    .option("user", "<your_synapse_user>") \
    .option("password", "<your_synapse_password>") \
    .mode("overwrite") \
    .save()

# Write to Azure SQL Database (using JDBC)
processed_data.write \
    .format("jdbc") \
    .option("url", "jdbc:sqlserver://<your_sql_server>.database.windows.net:1433;d
    .option("dbtable", "<your_sql_table>") \
    .option("user", "<your_sql_user>") \
    .option("password", "<your_sql_password>") \
    .mode("overwrite") \
    .save()

spark.stop()
```

4. Inference (Training → Inference)

Azure Databricks is used for batch inference, scoring data based on the trained model.

```
# Azure Databricks Notebook (Python with Spark) - Batch Inference

from pyspark.sql import SparkSession
from pyspark.ml.pipeline import PipelineModel
from mlflow import mlflow

# Initialize Spark Session
spark = SparkSession.builder.appName("BatchInference").getOrCreate()

# Load the trained model (either from DBFS or Azure ML Model Registry)
# Option 1: Load from DBFS
```

```
model = PipelineModel.load("dbfs:/mnt/models/trained_model")

# Option 2: Load from MLflow Model Registry
# Replace with your registry URI and model name
# mlflow.set_registry_uri("your_mlflow_tracking_uri")
# model_uri = f"models/your_model_name/Staging"
# model = mlflow.spark.load_model(model_uri)

# Load data for scoring
scoring_data = spark.read.format("delta").load("wasbs://<container>@<storage_accou

# Make predictions
predictions = model.transform(scoring_data)

# Save the predictions
predictions.write.format("delta").mode("overwrite").save("wasbs://<container>@<sto

spark.stop()
```

5. Model Deployment (Training → Inference)

The trained model is deployed as an endpoint using Azure Machine Learning and Azure Kubernetes Service (AKS) for near real-time inference.

```
# Azure Machine Learning SDK (Python) - Model Deployment

from azureml.core import Workspace, Model
from azureml.core.webservice import AciWebservice, Webservice
from azureml.core.model import InferenceConfig, Environment
from azureml.core.conda_dependencies import CondaDependencies

# Connect to your Azure ML Workspace
ws = Workspace.from_config()

# Register the trained model (if not already done)
# model_path = "dbfs:/mnt/models/trained_model" # Path where the model is saved in
# model = Model.register(workspace=ws,
#                         model_path=model_path,
#                         model_name="your_model_name")

# Get the registered model
model = Model(ws, name="your_model_name")

# Define the inference environment
env = Environment(name="inference_environment")
conda_dep = CondaDependencies()
conda_dep.add_pip_package("scikit-learn")
```

```
conda_dep.add_pip_package("pandas")
conda_dep.add_pip_package("azureml-defaults")
env.python.conda_dependencies = conda_dep

# Define the inference configuration
inference_config = InferenceConfig(
    entry_script="score.py",
    source_directory="./scoring", # Directory containing score.py
    environment=env
)

# Define the deployment configuration (ACI for testing, AKS for production)
deployment_config = AciWebservice.deploy_configuration(
    cpu_cores=1,
    memory_gb=1
)

# Deploy the model as a web service
service_name = "your-realtime-service"
service = Model.deploy(workspace=ws,
                       name=service_name,
                       models=[model],
                       inference_config=inference_config,
                       deployment_config=deployment_config,
                       overwrite=True)
service.wait_for_deployment(show_output=True)

# Example scoring script (score.py)
# import json
# import numpy as np
# import pandas as pd
# from sklearn.externals import joblib  # Or appropriate loading method for your m

# def init():
#     global model
#     model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'your_model_file.p
#     model = joblib.load(model_path)

# def run(raw_data):
#     try:
#         data = json.loads(raw_data)['data']
#         data = np.array(data)
#         predictions = model.predict(data)
#         return json.dumps(predictions.tolist())
#     except Exception as e:
#         error = str(e)
#         return json.dumps({'error': error})
```

6. Analytical Workload (Inference → Analytical workload)

Predictions from both batch and real-time inference are stored in the analytical workload for reporting and analysis.

```
# Azure Databricks Notebook (Python with Spark) - Write Inference Results

from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("WriteInferenceResults").getOrCreate()

# Load batch inference results
batch_predictions = spark.read.format("delta").load("wasbs://<container>@<storage_

# (For real-time inference, results might be written directly by the AKS service o

# Write to Azure Data Lake Storage Gen2 for analytics
batch_predictions.write.format("delta").mode("append").save("wasbs://<container>@<

# Write to Azure Synapse Analytics
batch_predictions.write \
    .format("com.databricks.spark.sqldw") \
    .options(...) # Your Synapse connection options
    .mode("append") \
    .save()

# Write to Azure SQL Database
batch_predictions.write \
    .format("jdbc") \
    .options(...) # Your SQL Database connection options
    .mode("append") \
    .save()

spark.stop()
```

7. Front-end (Analytical workload → Front-end)

Power BI and web applications connect to the analytical workload to visualize and consume the data and model predictions.

Power BI:

- Connect Power BI to Azure Data Lake Storage Gen2, Azure Synapse, or Azure SQL Database using the appropriate connectors.
- Create reports and dashboards based on the data in the analytical workload.

Web Application:

- The web application can interact with the analytical workload in several ways:
    - **Direct database connection:** Connect to Azure Synapse or Azure SQL Database to query data.
    - **API calls:** Call APIs hosted on Azure App Service or other services that expose the analytical data.
    - **Power BI Embedded:** Embed Power BI reports and dashboards within the web application.
    - **Real-time inference API:** Call the deployed Azure Machine Learning endpoint (on AKS) to get real-time predictions.

```
# Python (Flask) example for a web application consuming data from Azure SQL Datab

from flask import Flask, render_template
import pyodbc

app = Flask(__name__)

# Database connection details
server = '<your_sql_server>.database.windows.net'
database = '<your_sql_database>'
username = '<your_sql_user>'
password = '<your_sql_password>'
driver = '{ODBC Driver 17 for SQL Server}' # Or your preferred driver

@app.route('/')
def index():
    conn_str = (
        f'DRIVER={driver};'
        f'SERVER={server};'
        f'DATABASE={database};'
        f'UID={username};'
        f'PWD={password};'
    )
    cnxn = pyodbc.connect(conn_str)
    cursor = cnxn.cursor()
    cursor.execute("SELECT * FROM your_analytics_table")
    rows = cursor.fetchall()
    cnxn.close()
    return render_template('index.html', data=rows)

if __name__ == '__main__':
    app.run(debug=True)
```

This comprehensive outline provides code snippets and configurations relevant to each stage of the data flow. Remember to replace the placeholder values with your actual resource names, connection strings, and credentials.