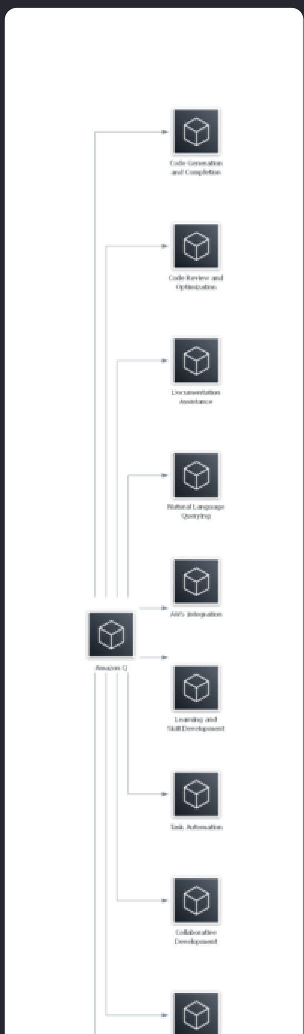
Session Info

Messages: 8

Clear Chat

Latest Diagram



AWS DevOps Agent

Your intelligent AWS assistant

CONTEXT: i have file in local folder file_audit.json, which contain file audit data on the data files we receive daily 2 times a day o
(each time 10 files max at a batch time each file has a unique (ex: "fil_id":"2006052061539") , each batch is a cycle1 and cycle2 ,
CNTL.C3.20250915.132541 in local) and data files , the control file has records of 2Xnumber of files including "fil_id" in the head
and each files footer(FTR) and header deatiling record count. (if 5 files coming in acycle, the CNTL file has 10 entries Model endp
deploy a model endpoint using sage maker studio which can predict how many files can be expected in a futire particular day b

Thank you for providing the preview of the file_audit.json. This file contains detailed information about file processing, includin
processing statuses. Now, let's discuss how to use this data to create a SageMaker endpoint for predicting the number of files ex

To create a model endpoint using Amazon SageMaker that can predict how many files can be expected on a future particular da
to follow these steps:

1. Data Preparation
2. Model Development
3. Model Deployment

Let's go through each step:

1. Data Preparation:

First, we need to preprocess the data and extract relevant features. Here's what we should do:

a. Extract the date and aggregate the data by date:

- Use the "fil_creatn_dt" field
- Count the number of files per day
- Sum the "in_fil_pkg_hdr_cnt" for each day (this represents the total number of files received)

b. Create additional features that might be useful for prediction:

- Day of the week
- Month
- Is it a weekday or weekend

- Is it a holiday (you may need to incorporate a holiday calendar)

c. Create a time series dataset with these features

2. Model Development:

For this type of prediction, we can use a time series forecasting model. Amazon SageMaker provides several algorithms that can

a. DeepAR: This is a supervised learning algorithm for forecasting scalar time series using recurrent neural networks (RNN).

b. Prophet: This is an open-source forecasting procedure implemented by Facebook, which works well with time series that hav seasons of historical data.

c. XGBoost: While not specifically designed for time series, it can be effective when you engineer appropriate features.

For this example, let's use the DeepAR algorithm, as it's well-suited for time series forecasting and is natively supported by Sage

3. Model Deployment:

After training the model, we'll deploy it as a SageMaker endpoint for real-time predictions.

Here's a high-level Python script that outlines the process:

```
import boto3
import sagemaker
from sagemaker import get_execution_role
from sagemaker.predictor import json_serializer, json_deserializer
from sagemaker.amazon.amazon_estimator import get_image_uri

# Set up the SageMaker session
sagemaker_session = sagemaker.Session()
role = get_execution_role()

# Prepare your data and upload to S3
# This step involves the data preparation described earlier
# Assume we have training_data and test_data ready and uploaded to S3

# Set up the DeepAR estimator
region = boto3.Session().region_name
container = get_image_uri(region, "forecasting-deepar")

estimator = sagemaker.estimator.Estimator(
    container,
    role,
    train_instance_count=1,
```

```python
    train_instance_type='ml.c4.xlarge',
    output_path='s3://your-bucket/output',
    sagemaker_session=sagemaker_session
)

# Set hyperparameters
estimator.set_hyperparameters(
    time_freq='D',  # Daily data
    context_length=30,  # Use 30 days of context
    prediction_length=7,  # Predict for the next 7 days
    epochs=100
)

# Train the model
estimator.fit({'train': training_data, 'test': test_data})

# Deploy the model
predictor = estimator.deploy(
    initial_instance_count=1,
    instance_type='ml.t2.medium'
)

# Set the correct serializer and deserializer for the predictor
predictor.serializer = json_serializer
predictor.deserializer = json_deserializer

# Now you can use the predictor to make predictions
# Example:
# result = predictor.predict({
#     "instances": [
#         {"start": "2023-05-01", "target": [10, 20, 30]}  # historical data
#     ],
#     "configuration": {"num_samples": 50}
# })
```

To implement this solution, you'll need to:

1. Prepare your data as described in the Data Preparation step. This will involve writing a script to process the file_audit.json c
2. Set up an AWS environment with the necessary permissions to use SageMaker.
3. Implement the Python script above, filling in the details for your specific data and S3 bucket locations.
4. Run the script to train the model and deploy the endpoint.

5. Create a separate script or application that can send requests to the deployed endpoint for predictions.

Remember to monitor the performance of your model over time and retrain it periodically with new data to ensure its predictio

Would you like me to help you with any specific part of this process, such as data preparation or setting up the SageMaker envir

can you help me deploy this model endpoint. i already have a sage maker domain and a jupyterlab notebook created, and dom
v4nprd-s301-tmp for put and get

Certainly! I'd be happy to help you deploy the model endpoint using your existing SageMaker domain and JupyterLab noteboo
necessary setup and S3 bucket access, we'll focus on creating and deploying the model endpoint.

Let's create a Jupyter notebook with the necessary code to prepare the data, train the model, and deploy the endpoint. Here's a

1. Open your JupyterLab notebook in your SageMaker domain.
2. Create a new notebook and name it something like "file_prediction_model.ipynb".
3. Copy and paste the following code into the notebook, and we'll go through it section by section:

```
import boto3
import sagemaker
import pandas as pd
import numpy as np
from sagemaker import get_execution_role
from sagemaker.amazon.amazon_estimator import get_image_uri
import json
from datetime import datetime, timedelta

# Set up the SageMaker session and role
sagemaker_session = sagemaker.Session()
role = get_execution_role()

# S3 bucket
bucket = 'ddps-v4nprd-s301-tmp'
prefix = 'file_prediction_model'

# Step 1: Data Preparation
def prepare_data(file_path):
    # Read the JSON file
    with open(file_path, 'r') as f:
```

```python
        data = [json.loads(line) for line in f]

    # Convert to DataFrame
    df = pd.DataFrame(data)

    # Convert date string to datetime
    df['fil_creatn_dt'] = pd.to_datetime(df['fil_creatn_dt'])

    # Group by date and aggregate
    daily_data = df.groupby('fil_creatn_dt').agg({
        'fil_id': 'count',
        'in_fil_pkg_hdr_cnt': 'sum'
    }).reset_index()

    # Rename columns
    daily_data.columns = ['date', 'file_count', 'total_packages']

    # Add features
    daily_data['day_of_week'] = daily_data['date'].dt.dayofweek
    daily_data['month'] = daily_data['date'].dt.month
    daily_data['is_weekday'] = (daily_data['day_of_week'] < 5).astype(int)

    # Sort by date
    daily_data = daily_data.sort_values('date')

    return daily_data

# Prepare the data
data = prepare_data('file_audit.json')

# Step 2: Split the data into training and test sets
train_data = data[data['date'] < '2006-04-01']
test_data = data[data['date'] >= '2006-04-01']

# Step 3: Prepare data for DeepAR
def prepare_deepar_data(df, file_name):
    deepar_data = df[['date', 'file_count']].rename(columns={'date': 'start', 'file_count': 'target
    deepar_data['start'] = deepar_data['start'].dt.strftime('%Y-%m-%d')
    deepar_data['target'] = deepar_data['target'].apply(lambda x: [float(x)])

    # Save to JSON Lines format
    with open(file_name, 'w') as f:
        for _, row in deepar_data.iterrows():
            json.dump(row.to_dict(), f)
```

```python
            f.write('\n')

    # Upload to S3
    s3_path = f's3://{bucket}/{prefix}/{file_name}'
    boto3.Session().resource('s3').Bucket(bucket).upload_file(file_name, f'{prefix}/{file_name}')

    return s3_path

train_path = prepare_deepar_data(train_data, 'train.json')
test_path = prepare_deepar_data(test_data, 'test.json')

# Step 4: Set up the DeepAR estimator
region = boto3.Session().region_name
container = get_image_uri(region, "forecasting-deepar")

estimator = sagemaker.estimator.Estimator(
    container,
    role,
    train_instance_count=1,
    train_instance_type='ml.c4.xlarge',
    output_path=f's3://{bucket}/{prefix}/output',
    sagemaker_session=sagemaker_session
)

# Set hyperparameters
estimator.set_hyperparameters(
    time_freq='D',  # Daily data
    context_length=30,  # Use 30 days of context
    prediction_length=7,  # Predict for the next 7 days
    epochs=100
)

# Step 5: Train the model
estimator.fit({'train': train_path, 'test': test_path})

# Step 6: Deploy the model
predictor = estimator.deploy(
    initial_instance_count=1,
    instance_type='ml.t2.medium'
)

# Print the endpoint name
print(f"Model endpoint name: {predictor.endpoint_name}")
```

```python
# Example of how to use the predictor
latest_date = data['date'].max()
start_date = latest_date + timedelta(days=1)

request_data = {
    "instances": [
        {
            "start": start_date.strftime('%Y-%m-%d'),
            "target": data['file_count'].tail(30).tolist()
        }
    ],
    "configuration": {"num_samples": 50}
}

result = predictor.predict(request_data)
print(f"Prediction for the next 7 days: {result['predictions'][0]['mean']}")
```

Now, let's go through each section of the code:

1. We import the necessary libraries and set up the SageMaker session and role.
2. In the `prepare_data` function, we read the JSON file, process it, and create daily aggregates with additional features.
3. We split the data into training and test sets.
4. The `prepare_deepar_data` function formats the data for the DeepAR algorithm and uploads it to your S3 bucket.
5. We set up the DeepAR estimator with the specified hyperparameters.
6. We train the model using the prepared data.
7. After training, we deploy the model as an endpoint.
8. Finally, we print the endpoint name and show an example of how to use the predictor.

To run this notebook:

1. Make sure the `file_audit.json` file is in the same directory as your notebook.
2. Run each cell in the notebook sequentially.
3. The final cell will print the model endpoint name and a sample prediction for the next 7 days.

After running this notebook, you'll have a deployed model endpoint that you can use for predictions. The endpoint name will be

To use this endpoint in other applications or scripts, you can use the AWS SDK (boto3) to send requests to the endpoint. Here's a

```
import boto3
import json

def predict_file_count(endpoint_name, start_date, historical_data):
    client = boto3.client('runtime.sagemaker')

    request_data = {
        "instances": [
            {
                "start": start_date,
                "target": historical_data
            }
        ],
        "configuration": {"num_samples": 50}
    }

    response = client.invoke_endpoint(
        EndpointName=endpoint_name,
        ContentType='application/json',
        Body=json.dumps(request_data)
    )

    result = json.loads(response['Body'].read().decode())
    return result['predictions'][0]['mean']

# Example usage
endpoint_name = "your-endpoint-name"  # Replace with the actual endpoint name
start_date = "2023-05-01"
historical_data = [10, 15, 20, 18, 22, 25, 30]  # Last 7 days of file counts

prediction = predict_file_count(endpoint_name, start_date, historical_data)
print(f"Predicted file counts for the next 7 days: {prediction}")
```

Remember to replace `"your-endpoint-name"` with the actual endpoint name that was printed when you deployed the model.

Is there anything specific you'd like me to explain further or any part of the process you need more help with?

please note main important fields from the json file are "fil_id": which unique number for each file received on a day & in_tot_re
our trend is date and time (am/pm) and fil_id , in_tot_rec_cnt gives us total picture on how many files for each morningam/pm

and total record count in each file.

Thank you for providing that important clarification. You're right, we should focus on the key fields that give us the most relevar
Let's modify our approach to better utilize the "fil_id" and "in_tot_rec_cnt" fields, and incorporate the AM/PM cycle information
and model input to reflect this.

Here's an updated version of the code that focuses on these important fields:

```
import boto3
import sagemaker
import pandas as pd
import numpy as np
from sagemaker import get_execution_role
from sagemaker.amazon.amazon_estimator import get_image_uri
import json
from datetime import datetime, timedelta

# Set up the SageMaker session and role
sagemaker_session = sagemaker.Session()
role = get_execution_role()

# S3 bucket
bucket = 'ddps-v4nprd-s301-tmp'
prefix = 'file_prediction_model'

# Step 1: Data Preparation
def prepare_data(file_path):
    # Read the JSON file
    with open(file_path, 'r') as f:
        data = [json.loads(line) for line in f]

    # Convert to DataFrame
    df = pd.DataFrame(data)

    # Convert date and time strings to datetime
    df['datetime'] = pd.to_datetime(df['fil_creatn_dt'] + ' ' + df['fil_creatn_time'])

    # Add AM/PM cycle
    df['cycle'] = df['datetime'].dt.strftime('%p')

    # Group by date and cycle
    daily_data = df.groupby([df['datetime'].dt.date, 'cycle']).agg({
```

```
        'fil_id': 'count',
        'in_tot_rec_cnt': 'sum'
    }).reset_index()

    # Rename columns
    daily_data.columns = ['date', 'cycle', 'file_count', 'total_records']

    # Add features
    daily_data['day_of_week'] = pd.to_datetime(daily_data['date']).dt.dayofweek
    daily_data['month'] = pd.to_datetime(daily_data['date']).dt.month
    daily_data['is_weekday'] = (daily_data['day_of_week'] < 5).astype(int)

    # Sort by date and cycle
    daily_data = daily_data.sort_values(['date', 'cycle'])

    return daily_data

# Prepare the data
data = prepare_data('file_audit.json')

# Step 2: Split the data into training and test sets
split_date = '2006-04-01'
train_data = data[data['date'] < split_date]
test_data = data[data['date'] >= split_date]

# Step 3: Prepare data for DeepAR
def prepare_deepar_data(df, file_name):
    # Prepare the data for DeepAR
    deepar_data = df.sort_values(['date', 'cycle'])

    # Create a continuous datetime index
    date_cycle = pd.to_datetime(deepar_data['date'].astype(str) + ' ' + deepar_data['cycle'].map({'
    deepar_data.index = date_cycle

    # Resample to ensure all AM/PM cycles are represented
    deepar_data = deepar_data.resample('12H').asfreq()

    # Fill missing values (you might want to use a more sophisticated method)
    deepar_data['file_count'] = deepar_data['file_count'].fillna(0)
    deepar_data['total_records'] = deepar_data['total_records'].fillna(0)

    # Prepare the data in the format DeepAR expects
    deepar_json = []
    for name, group in deepar_data.groupby(deepar_data.index.date):
```

```
        item = {
            "start": group.index[0].strftime('%Y-%m-%d %H:%M:%S'),
            "target": group['file_count'].tolist(),
            "dynamic_feat": [group['total_records'].tolist()]
        }
        deepar_json.append(item)

    # Save to JSON Lines format
    with open(file_name, 'w') as f:
        for item in deepar_json:
            json.dump(item, f)
            f.write('\n')

    # Upload to S3
    s3_path = f's3://{bucket}/{prefix}/{file_name}'
    boto3.Session().resource('s3').Bucket(bucket).upload_file(file_name, f'{prefix}/{file_name}')

    return s3_path

train_path = prepare_deepar_data(train_data, 'train.json')
test_path = prepare_deepar_data(test_data, 'test.json')

# Step 4: Set up the DeepAR estimator
region = boto3.Session().region_name
container = get_image_uri(region, "forecasting-deepar")

estimator = sagemaker.estimator.Estimator(
    container,
    role,
    train_instance_count=1,
    train_instance_type='ml.c4.xlarge',
    output_path=f's3://{bucket}/{prefix}/output',
    sagemaker_session=sagemaker_session
)

# Set hyperparameters
estimator.set_hyperparameters(
    time_freq='12H',  # Twice daily data
    context_length=14,  # Use 7 days (14 periods) of context
    prediction_length=14,  # Predict for the next 7 days (14 periods)
    epochs=100,
    num_dynamic_feat=1  # We're using total_records as a dynamic feature
)
```

```python
# Step 5: Train the model
estimator.fit({'train': train_path, 'test': test_path})

# Step 6: Deploy the model
predictor = estimator.deploy(
    initial_instance_count=1,
    instance_type='ml.t2.medium'
)

# Print the endpoint name
print(f"Model endpoint name: {predictor.endpoint_name}")

# Example of how to use the predictor
latest_date = data['date'].max()
start_date = pd.to_datetime(latest_date) + timedelta(days=1)

# Get the last 14 periods (7 days) of data for context
context_data = data.sort_values(['date', 'cycle']).tail(14)

request_data = {
    "instances": [
        {
            "start": start_date.strftime('%Y-%m-%d 00:00:00'),
            "target": context_data['file_count'].tolist(),
            "dynamic_feat": [context_data['total_records'].tolist()]
        }
    ],
    "configuration": {"num_samples": 50}
}

result = predictor.predict(request_data)
print(f"Prediction for the next 7 days (AM and PM cycles):")
for i, prediction in enumerate(result['predictions'][0]['mean']):
    cycle = 'AM' if i % 2 == 0 else 'PM'
    print(f"Day {i // 2 + 1} {cycle}: {prediction:.2f} files")
```

Key changes in this updated version:

1. We now consider both the date and time of file creation, separating data into AM and PM cycles.
2. We use both 'fil_id' count (as file_count) and 'in_tot_rec_cnt' (as total_records) in our analysis.
3. The DeepAR data preparation now creates a time series with two data points per day (AM and PM cycles).

4. We use the 'total_records' as a dynamic feature in our DeepAR model, which can help improve predictions.
5. The model now predicts at 12-hour intervals, giving separate predictions for AM and PM cycles.
6. We've adjusted the context_length and prediction_length to account for the twice-daily data points.

To use this model:

1. Run the notebook as before, ensuring 'file_audit.json' is in the same directory.
2. The model will train on the data up to April 1, 2006, and use the rest for testing.
3. After deployment, you can use the predictor to forecast the number of files expected in each AM and PM cycle for the next 7

This approach should give you more detailed and accurate predictions, taking into account the AM/PM cycles and using the tota information for the model.

Is there anything else you'd like me to explain or modify in this approach?

the data should contain 2006 to 2025 , better use data from last 5 years (2020 and forward) for training

I apologize for the confusion. You're absolutely right that we should use more recent and relevant data for training. Let's modify onwards for training. Here's an updated version of the code that takes this into account:

```
import boto3
import sagemaker
import pandas as pd
import numpy as np
from sagemaker import get_execution_role
from sagemaker.amazon.amazon_estimator import get_image_uri
import json
from datetime import datetime, timedelta

# Set up the SageMaker session and role
sagemaker_session = sagemaker.Session()
role = get_execution_role()

# S3 bucket
bucket = 'ddps-v4nprd-s301-tmp'
prefix = 'file_prediction_model'
```

```python
# Step 1: Data Preparation
def prepare_data(file_path):
    # Read the JSON file
    with open(file_path, 'r') as f:
        data = [json.loads(line) for line in f]

    # Convert to DataFrame
    df = pd.DataFrame(data)

    # Convert date and time strings to datetime
    df['datetime'] = pd.to_datetime(df['fil_creatn_dt'] + ' ' + df['fil_creatn_time'])

    # Filter data from 2020 onwards
    df = df[df['datetime'].dt.year >= 2020]

    # Add AM/PM cycle
    df['cycle'] = df['datetime'].dt.strftime('%p')

    # Group by date and cycle
    daily_data = df.groupby([df['datetime'].dt.date, 'cycle']).agg({
        'fil_id': 'count',
        'in_tot_rec_cnt': 'sum'
    }).reset_index()

    # Rename columns
    daily_data.columns = ['date', 'cycle', 'file_count', 'total_records']

    # Add features
    daily_data['day_of_week'] = pd.to_datetime(daily_data['date']).dt.dayofweek
    daily_data['month'] = pd.to_datetime(daily_data['date']).dt.month
    daily_data['is_weekday'] = (daily_data['day_of_week'] < 5).astype(int)

    # Sort by date and cycle
    daily_data = daily_data.sort_values(['date', 'cycle'])

    return daily_data

# Prepare the data
data = prepare_data('file_audit.json')

# Step 2: Split the data into training and test sets
# Use the last 6 months of data for testing, the rest for training
split_date = pd.to_datetime(data['date'].max()) - pd.DateOffset(months=6)
train_data = data[data['date'] < split_date]
```

```
test_data = data[data['date'] >= split_date]

# Step 3: Prepare data for DeepAR
def prepare_deepar_data(df, file_name):
    # Prepare the data for DeepAR
    deepar_data = df.sort_values(['date', 'cycle'])

    # Create a continuous datetime index
    date_cycle = pd.to_datetime(deepar_data['date'].astype(str) + ' ' + deepar_data['cycle'].map({'
    deepar_data.index = date_cycle

    # Resample to ensure all AM/PM cycles are represented
    deepar_data = deepar_data.resample('12H').asfreq()

    # Fill missing values (you might want to use a more sophisticated method)
    deepar_data['file_count'] = deepar_data['file_count'].fillna(0)
    deepar_data['total_records'] = deepar_data['total_records'].fillna(0)

    # Prepare the data in the format DeepAR expects
    deepar_json = []
    for name, group in deepar_data.groupby(deepar_data.index.date):
        item = {
            "start": group.index[0].strftime('%Y-%m-%d %H:%M:%S'),
            "target": group['file_count'].tolist(),
            "dynamic_feat": [group['total_records'].tolist()]
        }
        deepar_json.append(item)

    # Save to JSON Lines format
    with open(file_name, 'w') as f:
        for item in deepar_json:
            json.dump(item, f)
            f.write('\n')

    # Upload to S3
    s3_path = f's3://{bucket}/{prefix}/{file_name}'
    boto3.Session().resource('s3').Bucket(bucket).upload_file(file_name, f'{prefix}/{file_name}')

    return s3_path

train_path = prepare_deepar_data(train_data, 'train.json')
test_path = prepare_deepar_data(test_data, 'test.json')

# Step 4: Set up the DeepAR estimator
```

```python
region = boto3.Session().region_name
container = get_image_uri(region, "forecasting-deepar")

estimator = sagemaker.estimator.Estimator(
    container,
    role,
    train_instance_count=1,
    train_instance_type='ml.c4.xlarge',
    output_path=f's3://{bucket}/{prefix}/output',
    sagemaker_session=sagemaker_session
)

# Set hyperparameters
estimator.set_hyperparameters(
    time_freq='12H',  # Twice daily data
    context_length=28,  # Use 14 days (28 periods) of context
    prediction_length=14,  # Predict for the next 7 days (14 periods)
    epochs=100,
    num_dynamic_feat=1  # We're using total_records as a dynamic feature
)

# Step 5: Train the model
estimator.fit({'train': train_path, 'test': test_path})

# Step 6: Deploy the model
predictor = estimator.deploy(
    initial_instance_count=1,
    instance_type='ml.t2.medium'
)

# Print the endpoint name
print(f"Model endpoint name: {predictor.endpoint_name}")

# Example of how to use the predictor
latest_date = data['date'].max()
start_date = pd.to_datetime(latest_date) + timedelta(days=1)

# Get the last 28 periods (14 days) of data for context
context_data = data.sort_values(['date', 'cycle']).tail(28)

request_data = {
    "instances": [
        {
            "start": start_date.strftime('%Y-%m-%d 00:00:00'),
```

```
            "target": context_data['file_count'].tolist(),
            "dynamic_feat": [context_data['total_records'].tolist()]
        }
    ],
    "configuration": {"num_samples": 50}
}

result = predictor.predict(request_data)
print(f"Prediction for the next 7 days (AM and PM cycles):")
for i, prediction in enumerate(result['predictions'][0]['mean']):
    cycle = 'AM' if i % 2 == 0 else 'PM'
    print(f"Day {i // 2 + 1} {cycle}: {prediction:.2f} files")
```

Key changes in this updated version:

1. In the `prepare_data` function, we now filter the data to include only records from 2020 onwards:

   ```
   df = df[df['datetime'].dt.year >= 2020]
   ```

2. We've changed the train-test split to use the last 6 months of data for testing and the rest for training. This ensures we're usi evaluation:

   ```
   split_date = pd.to_datetime(data['date'].max()) - pd.DateOffset(months=6)
   train_data = data[data['date'] < split_date]
   test_data = data[data['date'] >= split_date]
   ```

3. We've increased the `context_length` in the DeepAR hyperparameters to 28 (14 days), which may help capture more extend

   ```
   context_length=28,  # Use 14 days (28 periods) of context
   ```

4. In the prediction example, we're now using the last 28 periods (14 days) of data for context, matching our model's context le

This approach ensures that we're using the most recent and relevant data (from 2020 onwards) for our model, which should lea current and future file counts.

To use this updated model:

1. Make sure your `file_audit.json` contains data from 2006 to 2025.
2. Run the notebook in your SageMaker JupyterLab environment.
3. The model will train on data from 2020 to 6 months before the most recent date, and use the last 6 months for testing.

4. After deployment, you can use the predictor to forecast the number of files expected in each AM and PM cycle for the next 7

Is there anything else you'd like me to adjust or explain further?

give me step by step notes on how to do this endpoint deployment

Certainly! I'll provide you with a step-by-step guide on how to deploy this endpoint using your SageMaker domain and JupyterL walkthrough:

1. Prepare Your Environment:
   - Log in to your AWS Console.
   - Navigate to Amazon SageMaker.
   - Open your SageMaker domain and launch your JupyterLab environment.
2. Create a New Notebook:
   - In JupyterLab, click on the "+" button in the top left corner.
   - Choose "Python 3 (Data Science)" as your kernel.
   - Name your notebook (e.g., "file_prediction_endpoint.ipynb").
3. Install Required Libraries:
   - In the first cell of your notebook, add and run the following:

```
!pip install sagemaker pandas numpy
```

4. Import Libraries and Set Up SageMaker Session:
   - In a new cell, add and run:

```
import boto3
import sagemaker
import pandas as pd
import numpy as np
from sagemaker import get_execution_role
from sagemaker.amazon.amazon_estimator import get_image_uri
import json
from datetime import datetime, timedelta

sagemaker_session = sagemaker.Session()
role = get_execution_role()
```

```
# S3 bucket
bucket = 'ddps-v4nprd-s301-tmp'
prefix = 'file_prediction_model'
```

5. Data Preparation Function:
   - Copy and paste the `prepare_data` function from the previous message into a new cell.
6. Load and Prepare Data:
   - Ensure your `file_audit.json` is in the same directory as your notebook.
   - In a new cell, add:

```
data = prepare_data('file_audit.json')
```

7. Split Data into Training and Test Sets:
   - In a new cell, add:

```
split_date = pd.to_datetime(data['date'].max()) - pd.DateOffset(months=6)
train_data = data[data['date'] < split_date]
test_data = data[data['date'] >= split_date]
```

8. Prepare Data for DeepAR:
   - Copy and paste the `prepare_deepar_data` function from the previous message.
   - In a new cell, add:

```
train_path = prepare_deepar_data(train_data, 'train.json')
test_path = prepare_deepar_data(test_data, 'test.json')
```

9. Set Up DeepAR Estimator:
   - In a new cell, add:

```
region = boto3.Session().region_name
container = get_image_uri(region, "forecasting-deepar")

estimator = sagemaker.estimator.Estimator(
    container,
    role,
    train_instance_count=1,
    train_instance_type='ml.c4.xlarge',
```

```
    output_path=f's3://{bucket}/{prefix}/output',
    sagemaker_session=sagemaker_session
)

estimator.set_hyperparameters(
    time_freq='12H',
    context_length=28,
    prediction_length=14,
    epochs=100,
    num_dynamic_feat=1
)
```

10. Train the Model:
    - In a new cell, add:

    ```
    estimator.fit({'train': train_path, 'test': test_path})
    ```

    - This step may take some time to complete.
11. Deploy the Model:
    - After training is complete, in a new cell, add:

    ```
    predictor = estimator.deploy(
        initial_instance_count=1,
        instance_type='ml.t2.medium'
    )
    print(f"Model endpoint name: {predictor.endpoint_name}")
    ```

    - Note down the endpoint name that is printed.
12. Test the Deployed Endpoint:
    - In a new cell, add:

    ```
    latest_date = data['date'].max()
    start_date = pd.to_datetime(latest_date) + timedelta(days=1)

    context_data = data.sort_values(['date', 'cycle']).tail(28)

    request_data = {
        "instances": [
            {
                "start": start_date.strftime('%Y-%m-%d 00:00:00'),
    ```

```
            "target": context_data['file_count'].tolist(),
            "dynamic_feat": [context_data['total_records'].tolist()]
        }
    ],
    "configuration": {"num_samples": 50}
}

result = predictor.predict(request_data)
print(f"Prediction for the next 7 days (AM and PM cycles):")
for i, prediction in enumerate(result['predictions'][0]['mean']):
    cycle = 'AM' if i % 2 == 0 else 'PM'
    print(f"Day {i // 2 + 1} {cycle}: {prediction:.2f} files")
```

13. Clean Up (Optional):
    - If you want to delete the endpoint after testing, add in a new cell:

    ```
    predictor.delete_endpoint()
    ```

14. Save Your Notebook:
    - Click on "File" > "Save Notebook" to save your work.
15. Monitor Your Endpoint:
    - In the AWS Console, go to Amazon SageMaker > Inference > Endpoints to see your deployed endpoint.
    - You can monitor metrics, logs, and manage the endpoint from here.

Remember to shut down your notebook when you're done to avoid unnecessary charges. Also, keep in mind that the deployed charges as long as it's running, so delete it if you're not actively using it.

Is there any specific part of this process you'd like me to explain in more detail?

Ask me anything about AWS...