AWS re:Invent

DAT369

# Deep dive on Amazon Aurora machine learning integration

**Suprio Pal**
Software Development Manager
Amazon Aurora & Amazon RDS
Amazon Web Services

**Yoav Eilat**
Senior Product Manager
Amazon Aurora & Amazon RDS
Amazon Web Services

AWS re:Invent

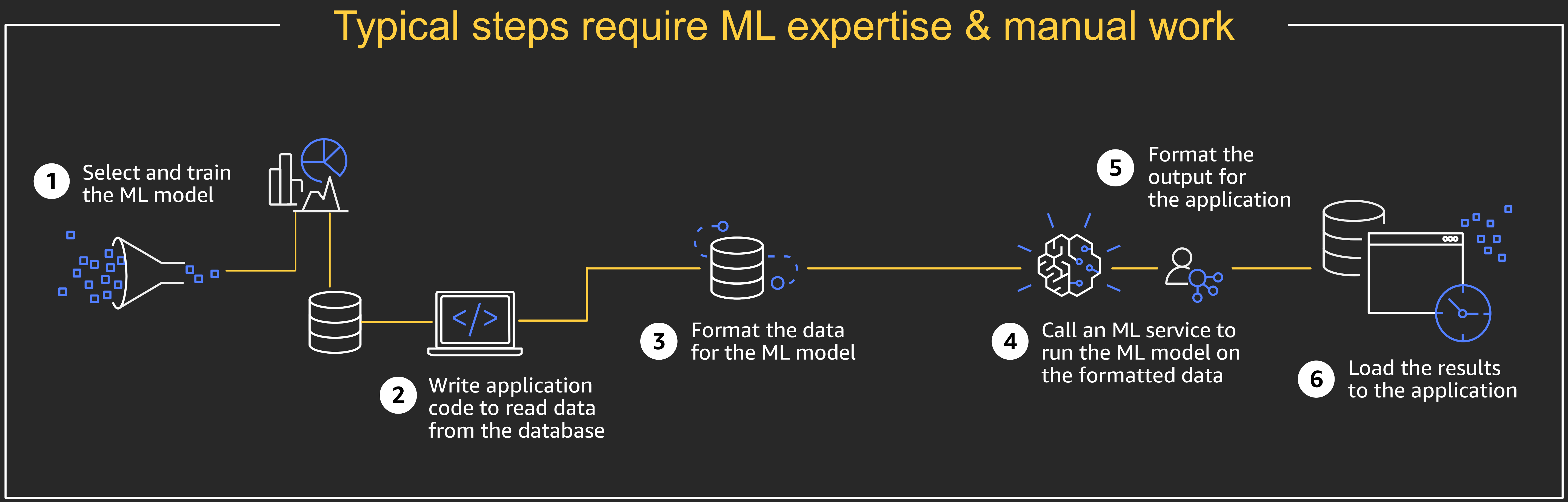
# Agenda

Aurora machine learning overview

Sentiment analysis and churn analysis demos

Open discussion

# Adding ML to an application is challenging



# Amazon Aurora machine learning

Simple, optimized, and secure Aurora, Amazon SageMaker, and Amazon Comprehend (in preview) integration

ML predictions on relational data

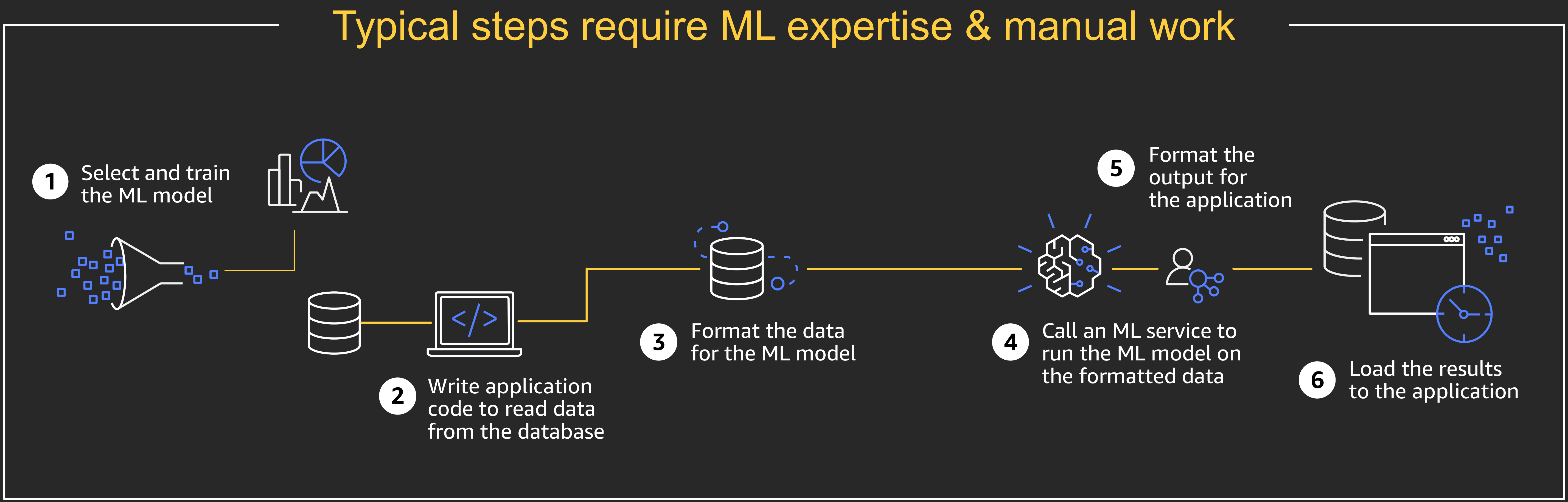
Integration with Amazon SageMaker & Amazon Comprehend

Familiar SQL language, no ML expertise

Low-latency, immediate

Security & governance

# From six steps



# To three steps

## Use the familiar SQL language for training & prediction

1 (Optional) Select and configure the ML model with Amazon SageMaker Autopilot

2 Run a SQL query to invoke the ML service

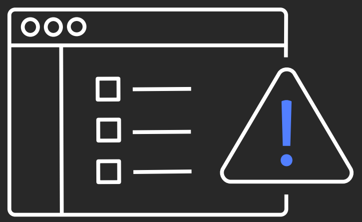
3 Use the results in the application

# From SQL to ML-driven insights

## Find suspected fraudulent transactions

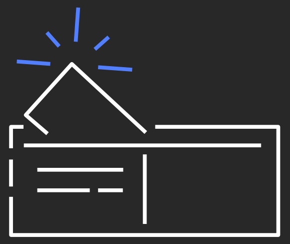
```
CREATE TRIGGER insert_check
BEFORE INSERT ON sales
  FOR EACH ROW
  BEGIN
    IF is_transaction_fraudulent(column1,
      column2, column3 ...) = 'True' THEN
      rollback; END IF;
  END;
```

## Flag comments with negative sentiment



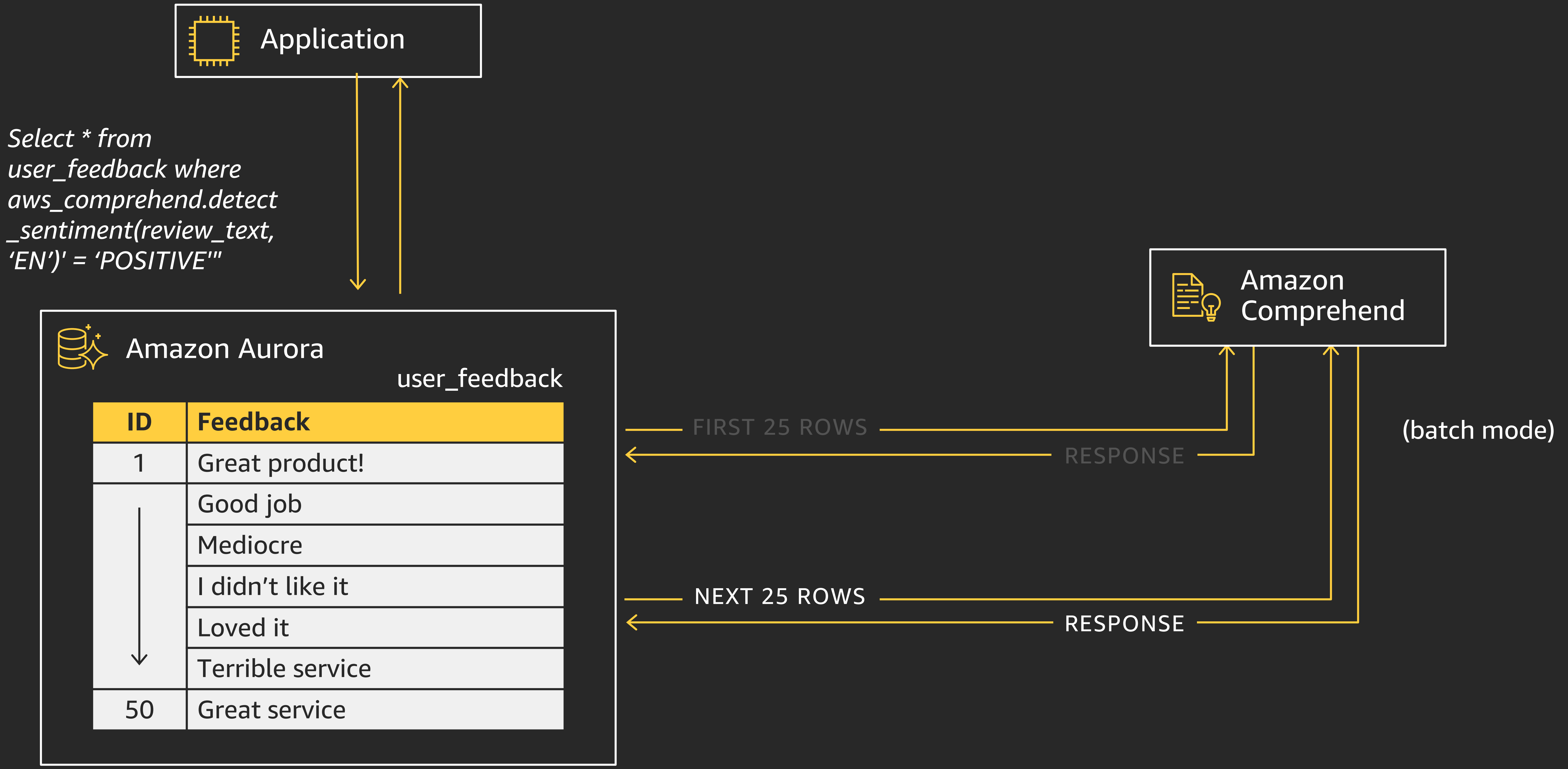
```
SELECT * from product_reviews WHERE
aws_comprehend.detect_sentiment
(review_text, ‘EN’)' = 'NEGATIVE'
```

## Sort customers by predicted future spend



```
SELECT * from customers order by
predicted_future_spend (column1,
column2, ...)
```

# Aurora offers optimized ML query processing




| ID | Feedback |
|---|---|
| 1 | Great product! |
| | Good job |
| | Mediocre |
| | I didn’t like it |
| | Loved it |
| | Terrible service |
| 50 | Great service |

# Demos

# Demo 1: Sentiment analysis with Amazon Comprehend

Amazon Aurora

No machine learning training required

Configure AWS Identity and Access Management (IAM) role for the Aurora database cluster and grant privileges for the users wanting to use Amazon Comprehend in their applications

Invoke Amazon Comprehend functions in SQL queries

Amazon Comprehend

# Demo 1: Sentiment analysis with Amazon Comprehend

Amazon Aurora

## Built-in functions

1) aws_comprehend_detect_sentiment()

2) aws_comprehend_detect_sentiment_confidence()

Amazon Comprehend

## Steps in the demo

- Create a table storing comments for a blogging platform and insert a few sample records
- Execute Aurora MySQL query invoking the Amazon Comprehend built-in functions
- Add a few more records with increasingly negative sentiment and execute the same SQL query

# Demo 2: Customer churn analysis with Amazon SageMaker

```
CREATE FUNCTION churn (
   (in state VARCHAR, in acc_length INT8,in area_code INT8,in int_plan VARCHAR,
    in vmail_plan VARCHAR, in vmail_msg INT8, in day_mins FLOAT8, in day_calls INT8,
    in eve_mins FLOAT8, in eve_calls INT8, in night_mins FLOAT8, in night_calls INT8,
    in int_mins FLOAT8, in int_calls INT8, in cust_service_calls INT8,
    in max_rows_per_batch INT default NULL,
    out expected_to_churn VARCHAR)
AS $$

    SELECT aws_sagemaker.invoke_endpoint('sqlai-scikit-endpoint', max_rows_per_batch,
                state, acc_length, area_code, int_plan, vmail_plan,
                vmail_msg, day_mins, day_calls, eve_mins, eve_calls,
                night_mins, night_calls, int_mins, int_calls, cust_service_calls)

$$ LANGUAGE SQL STABLE PARALLEL SAFE COST 5000;
```

# Predicting customer churn

```
SELECT count(*) as "Predicted to Churn",
       sum(case churn when 'True.' then 1 else 0 end) as "Did Churn",
       sum(case churn when 'False.' then 1 else 0 end) as "Did Not Churn",
       round(100.0 *
             sum(case churn when 'True.' then 1 else 0 end)/count(*), 2) as "Accuracy %"
FROM customer_churn
WHERE 'True.' = churn( state, acc_length, area_code, int_plan, vmail_plan, vmail_msg,
                       day_mins, day_calls, eve_mins, eve_calls, night_mins, night_calls,
                       int_mins, int_calls, cust_service_calls);

 Predicted to Churn | Did Churn | Did Not Churn | Accuracy %
--------------------+-----------+---------------+------------
                403 |       400 |             3 |      99.26
```

# Predicting customer churn

```
explain (analyze, costs false)
SELECT count(*)
FROM customer_churn
WHERE
churn <> churn(state, acc_length, area_code, int_plan, vmail_plan, vmail_msg,
               day_mins, day_calls, eve_mins, eve_calls, night_mins, night_calls,
               int_mins, int_calls, cust_service_calls);

 Aggregate (actual time=118.071..118.072 rows=1 loops=1)
   ->  Nested Loop (actual time=114.494..118.062 rows=86 loops=1)
         ->  Seq Scan on customer_churn (actual time=0.005..0.344 rows=3333 loops=1)
         ->  Function Scan on invoke_endpoint model_output
               (actual time=0.035..0.035 rows=0 loops=3333)
              Batch Processing: num batches=1
                  avg/min/max batch size=3333.000/3333.000/3333.000
                  avg/min/max batch call time=99.656/99.656/99.656
              Filter: ((customer_churn.churn)::text <> (model_output)::text)
              Rows Removed by Filter: 1
 Planning Time: 0.101 ms
 Execution Time: 118.100 ms    35.6 microseconds/row with 1 core, for this model
```

# Thank you!

**Suprio Pal**
suprio@amazon.com

**Yoav Eilat**
yeilat@alumni.has.org

Please complete the session survey in the mobile app.