Scaling Recommendation Systems to AWS

This presentation outlines our journey from a local prototype to a scalable, cloud-based recommendation system on AWS. We'll explore how we transitioned data handling, model storage, training, and real-time inference from laptop-based operations to robust AWS services, ensuring efficiency and scalability for final deployment.

Step 1: Data Migration - From Laptop to Cloud

Before: Local Data Handling

Our prototype relied on local data storage (e.g.,

```
pd.read_csv('users.dat', ...)
```

), limiting scalability and performance. Large files could crash the system, making it unsuitable for production.

```python
app = FastAPI(
    title="Hybrid Recommender Service",
    description="FastAPI wrapper around HybridRecommenderV3",
    version="1.0.0"
)

# Instantiate the recommender model (adjust paths as needed)
model = HybridRecommenderV3(
    data_paths={
        'users': './data/users.dat',
        'movies': './data/movies.dat',
        'ratings': './data/ratings.dat'
    },
    artifacts_path='./recommender_artifacts_v3'
)
```

After: AWS S3 & Glue Crawler

We transition to AWS S3 for central, secure, and infinitely scalable data storage. Files are uploaded using

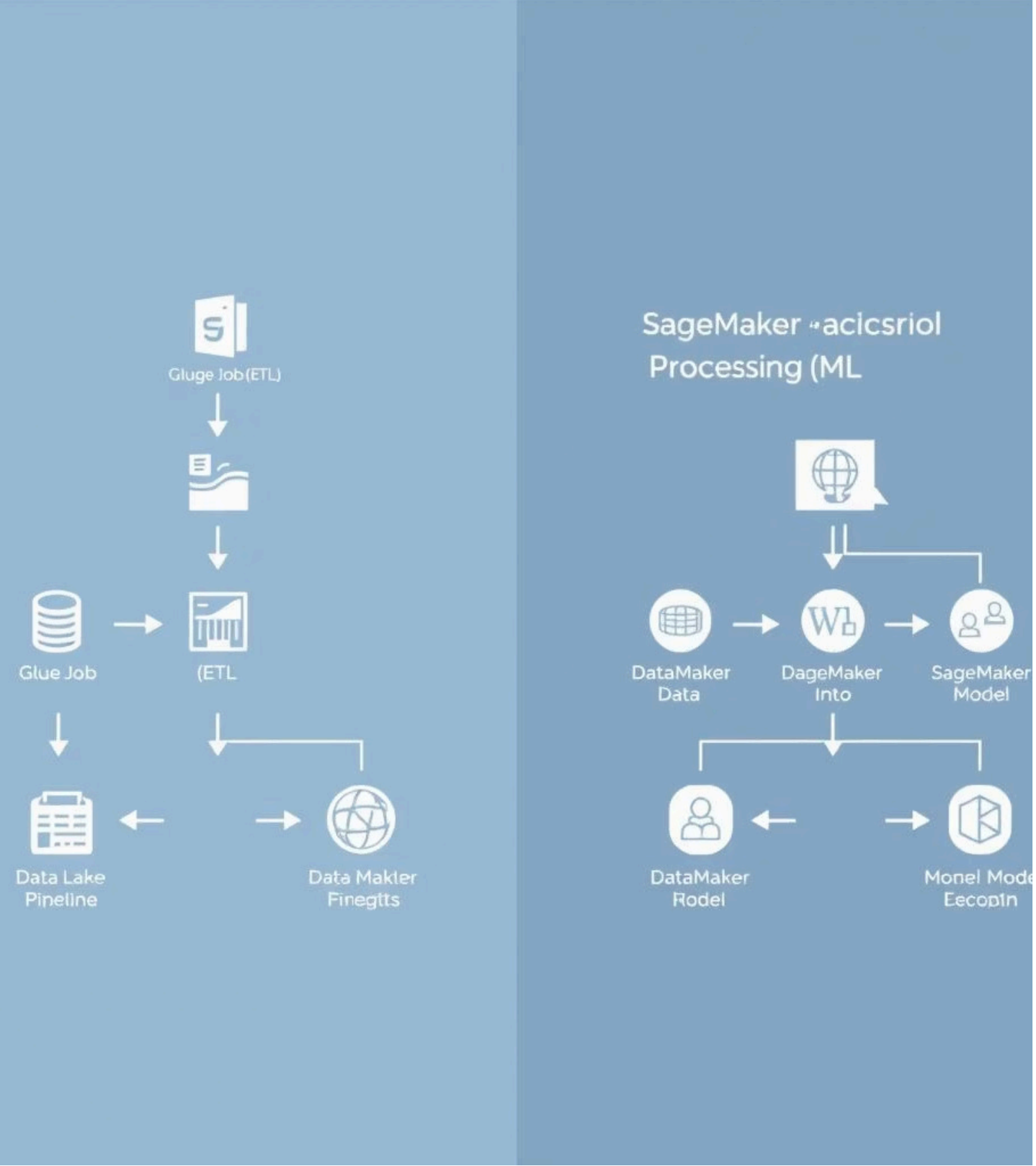
```
aws s3 cp users.dat s3://my-bucket/raw/
```

. AWS Glue Crawler then automatically discovers data structure, transforming raw data into queryable tables for tools like Athena.

Step 2: Storing & Loading Model Artifacts

Save Model	joblib.dump(..., './cf_model.joblib')	Save to S3 with boto3.put_object(...)
Load Model	joblib.load('./cf_model.joblib')	Download from S3, then load with joblib.load('/tmp/...')

In production, model files are stored in S3 for backup, versioning, and reusability across AWS services like Lambda and SageMaker, ensuring robust and accessible model management.



Step 3: Glue Job vs. SageMaker Processing

Choosing the right tool for data processing and ML logic is crucial for efficiency and scalability.

Glue Job (ETL)

Best for cleaning large CSVs, joins, and general ETL. Uses PySpark and auto-scales, making it serverless. Ideal for raw data transformations.

SageMaker Processing (ML)

Better for ML preprocessing, TF-IDF, and artifact generation. Supports Python libraries (pandas, sklearn) directly, avoiding Spark ML rewrites. Requires manual instance selection.

We use Glue for raw ETL and SageMaker for ML logic to avoid rewriting our code.

Step 4: Collaborative Filtering Training (ALS)

Local Prototype Training

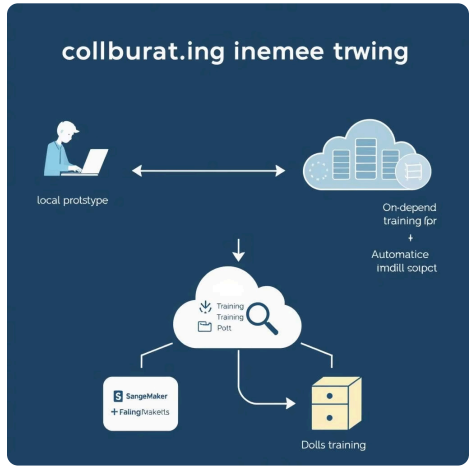
Our initial prototype involved training implicit.ALS models directly on a laptop. Models were saved locally using

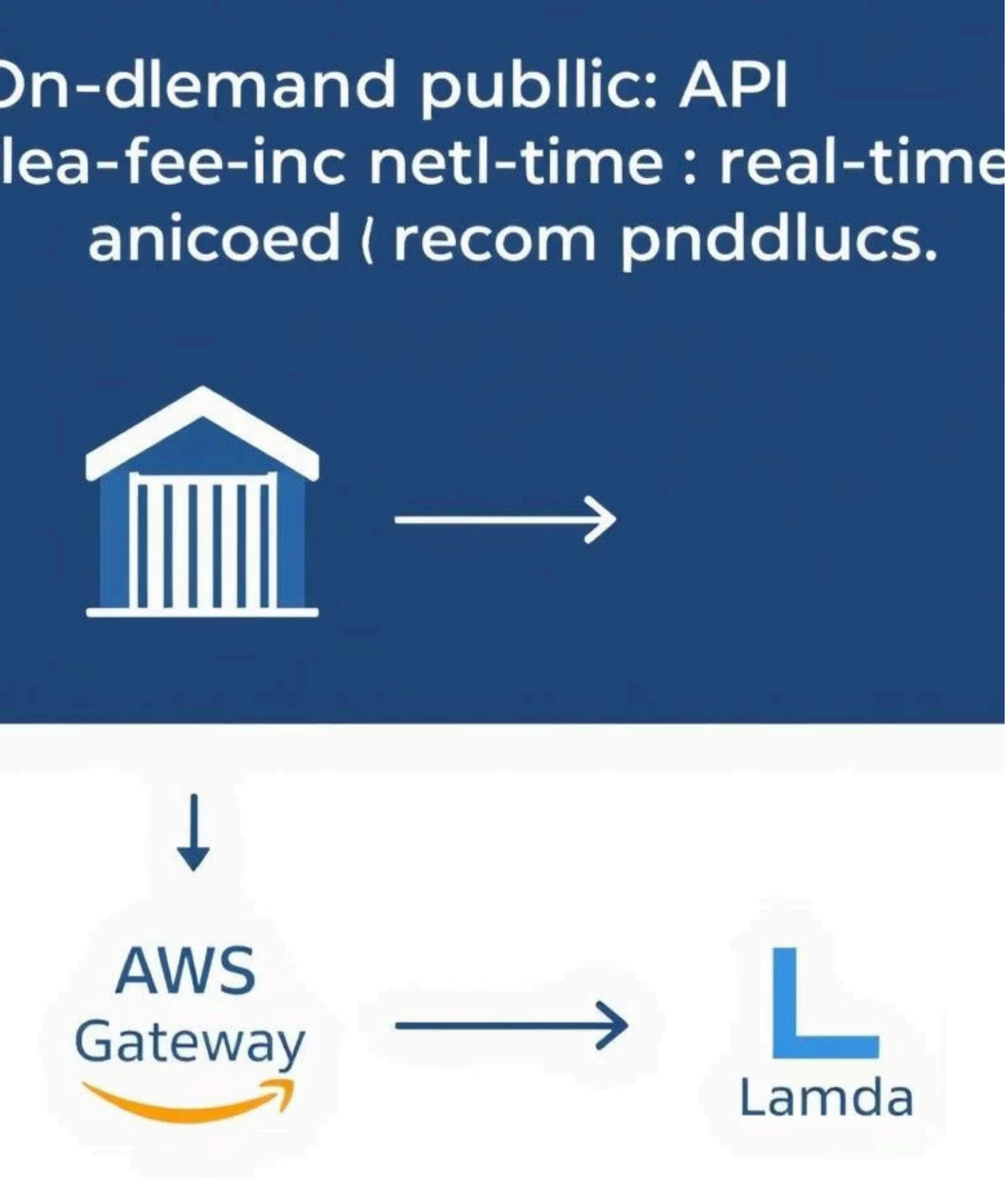
```
joblib.dump()
```

, which was sufficient for development but lacked scalability and robust management features.

AWS Equivalent: SageMaker Training Job

For production, we scale our ALS model training using SageMaker Training Jobs. This service runs our code on-demand, logs everything, automatically saves models to S3, and shuts down after training, providing a scalable and efficient solution.





Step 5: Real-Time Recommendations with Lambda & API Gateway

Transitioning from local script calls to a real-time public API for recommendations.

Local Script Call

Prototype recommendations were generated via a Python function call:

```
get_recommendations(user_id)
```

.

AWS Production Flow

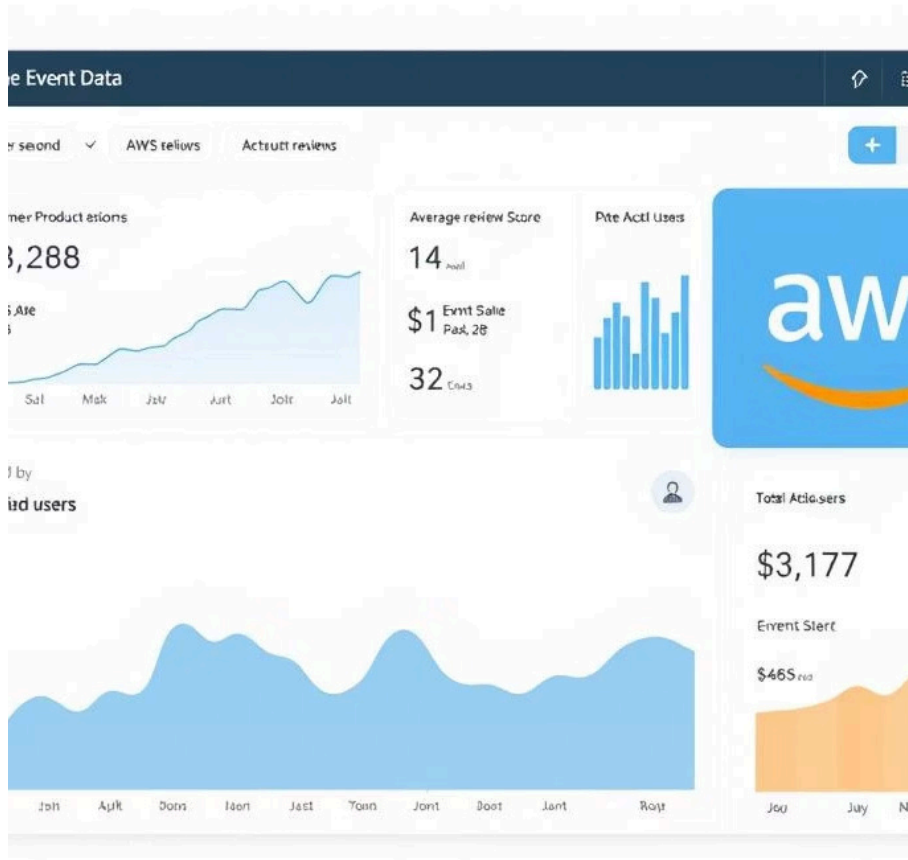
In production, API Gateway triggers a Lambda function to return real-time recommendation results, transforming our logic into a serverless, on-demand public API.

This setup ensures our recommendation logic is highly available and scalable without managing any servers.

Real-Time Events & Review Data with AWS

We leverage AWS services for capturing and accessing real-time event data and review information.

- **Amazon Kinesis:** Used to capture real-time mood clicks, ensuring immediate data ingestion.
- **Kinesis Firehose:** Stores the captured real-time data in S3 for subsequent analysis and training.
- **Athena:** Provides instant access to official Amazon reviews from the AWS Open Data Registry. We used public alternatives like MovieLens for rapid prototyping due to limited access to the full Amazon review dataset.



Lambda Functions & DynamoDB for Recommendation Flow

Our recommendation flow is powered by three key AWS Lambda functions, optimized with DynamoDB for efficient data access.

Enrichment Lambda

Cleans and tags raw mood/click data (e.g., normalizes emojis, extracts geo/device info), storing enriched logs in S3 for training.

RecHandler Lambda

Runs our main CF + CB model logic on-demand when a user hits "Get Recommendations," exposed via API Gateway.

WatchParty Lambda

Handles multiplayer sync, allowing users to join rooms, chat, and watch content together in real-time (uses WebSocket + DynamoDB).

Why DynamoDB? Instead of loading full `.joblib` models, we store user/item embeddings in DynamoDB, giving each Lambda instant access to only what it needs, without cold starts or huge files.