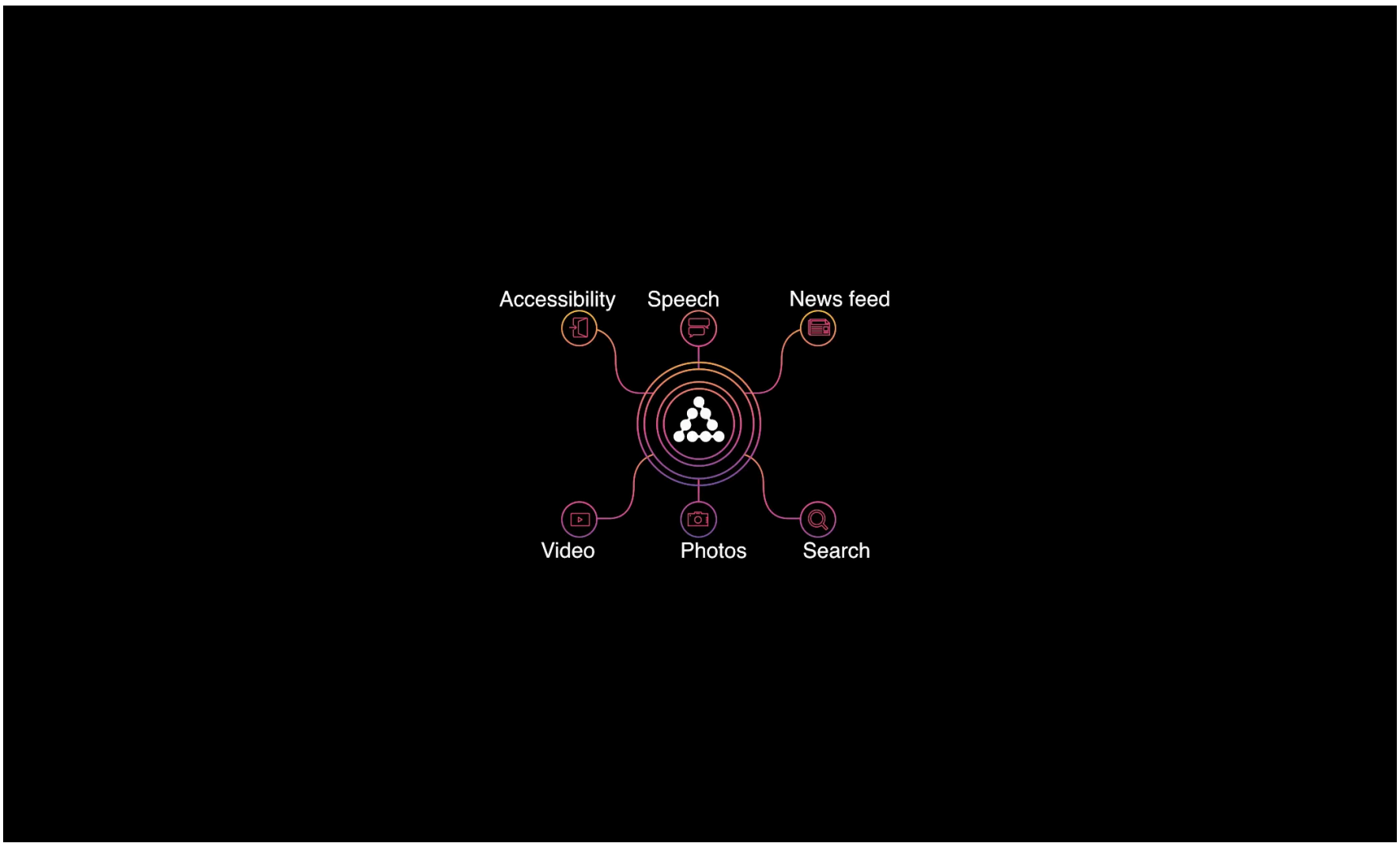
POSTED ON MAY 9, 2016 TO AI RESEARCH, CORE INFRA, ML APPLICATIONS

Introducing FBLearner Flow: Facebook’s AI backbone



By Jeffrey Dunn

Many of the experiences and interactions people have on Facebook today are made possible with AI. When you log in to Facebook, we use the power of machine learning to provide you with unique, personalized experiences. Machine learning models are part of ranking and personalizing News Feed stories, filtering out offensive content, highlighting trending topics, ranking search results, and much more. There are numerous other experiences on Facebook that could benefit from machine learning models, but until recently it's been challenging for engineers without a strong machine learning background to take advantage of our ML infrastructure. In late 2014, we set out to redefine machine learning platforms at Facebook from the ground up, and to put state-of-the-art algorithms in AI and ML at the fingertips of every Facebook engineer.

In some of our earliest work to leverage AI and ML — such as delivering the most relevant content to each person — we noticed that the largest improvements in accuracy often came from quick experiments, feature engineering, and model tuning rather than applying fundamentally different algorithms. An engineer may need to attempt hundreds of experiments before finding a successful new feature or set of hyperparameters. Traditional pipeline systems that we evaluated did not appear to be a good fit for our uses — most solutions did not provide a way to rerun pipelines with different inputs, mechanisms to explicitly capture outputs and/or side effects, visualization of outputs, and conditional steps for tasks like parameter sweeps.

To address these points, we wanted a platform with the following properties:

- Each machine learning algorithm should be implemented once in a reusable manner.
- Engineers should be able to write a training pipeline that parallelizes over many machines and can be reused by many engineers.

- Training a model should be easy for engineers of varying ML experience, and nearly every step should be fully automated.
- Everybody should be able to easily search past experiments, view results, share with others, and start new variants of a given experiment.

We decided to build a brand-new platform, FBLearner Flow, capable of easily reusing algorithms in different products, scaling to run thousands of simultaneous custom experiments, and managing experiments with ease. This platform provides innovative functionality, like automatic generation of UI experiences from pipeline definitions and automatic parallelization of Python code using futures. FBLearner Flow is used by more than 25 percent of Facebook's engineering team. Since its inception, more than a million models have been trained, and our prediction service has grown to make more than 6 million predictions per second.

Eliminating manual work required for experimentation allows machine learning engineers to spend more time on feature engineering, which in turn can produce greater accuracy improvements. Engineers are able to have an impact at scale — there are fewer than 150 workflow authors, and their efforts have an impact beyond themselves and their team. FBLearner Flow provides the platform and tools to enable engineers to run thousands of experiments every day.

Core concepts and components

Before we take a closer look at the system, there are a few key concepts to understand:

Workflows: A workflow is a single pipeline defined within FBLearner Flow and is the entry point for all machine learning tasks. Each workflow performs a specific job, such as training and evaluation of a specific model. Workflows are defined in terms of operators and can be parallelized.

Operators: Operators are the building blocks of workflows. Conceptually, you can think of an operator like a function within a program. In FBLearner Flow, operators are the smallest unit of execution and run on a single machine.

Channels: Channels represent inputs and outputs, which flow between operators within a workflow. All channels are typed using a custom type system that we have defined.

The platform consists of three core components: an authorship and execution environment for custom distributed workflows, an experimentation management UI for launching experiments and viewing results, and numerous predefined pipelines for training the most commonly used machine learning algorithms at Facebook.

Authorship and execution environment

All workflows and operators in FBLearner Flow are defined as functions in Python and apply special decorators to integrate with the platform. Let's examine a simple scenario where we want to train a decision tree on the classic Iris data set to predict a flower's species based on its petal and sepal sizes. Suppose the data set available to us is in Hive and has five columns, which represent petal width, petal length, sepal width, sepal length, and species for a sample of flowers. In this workflow, we will evaluate the performance of the model in terms of log loss and predict the species for an unlabeled data set.

A sample workflow to handle this task might look like:

```
# The typed schema of the Hive table containing the input data
feature_columns = (
    ('petal_width', types.INTEGER),
    ('petal_height', types.INTEGER),
    ('sepal_width', types.INTEGER),
    ('sepal_height', types.INTEGER),
)
label_column = ('species', types.TEXT)
all_columns = feature_columns + (label_column,)

# This decorator annotates that the following function is a workflow within
# FBLearner Flow
@workflow(
    # Workflows have typed inputs and outputs declared using the FBLearner type
    # system
    input_schema=types.Schema(
        labeled_data=types.DATASET(schema=all_columns),
        unlabeled_data=types.DATASET(schema=feature_columns),
    ),
    returns=types.Schema(
        model=types.MODEL,
        mse=types.DOUBLE,
        predictions=types.DATASET(schema=all_columns),
    ),
)
def iris(labeled_data, unlabeled_data):
    # Divide the dataset into separate training and evaluation dataset by random
    # sampling.
    split = SplitDatasetOperator(labeled_data, train=0.8, evaluation=0.2)

    # Train a decision tree with the default settings then evaluate it on the
    # labeled evaluation dataset.
    dt = TrainDecisionTreeOperator(
        dataset=split.train,
        features=[name for name, type in feature_columns],
        label=label_column[0],
    )
    metrics = ComputeMetricsOperator(
        dataset=split.evaluation,
        model=dt.model,
        label=label_column[0],
        metrics=[Metrics.LOGLOSS],
    )

    # Perform predictions on the unlabeled dataset and produce a new dataset
    predictions = PredictOperator(
        dataset=unlabeled_data,
        model=dt.model,
        output_column=label_column[0],
    )

    # Return the outputs of the workflow from the individual operators
    return Output(
        model=dt.model,
        logloss=metrics.logloss,
        predictions=predictions,
    )
```

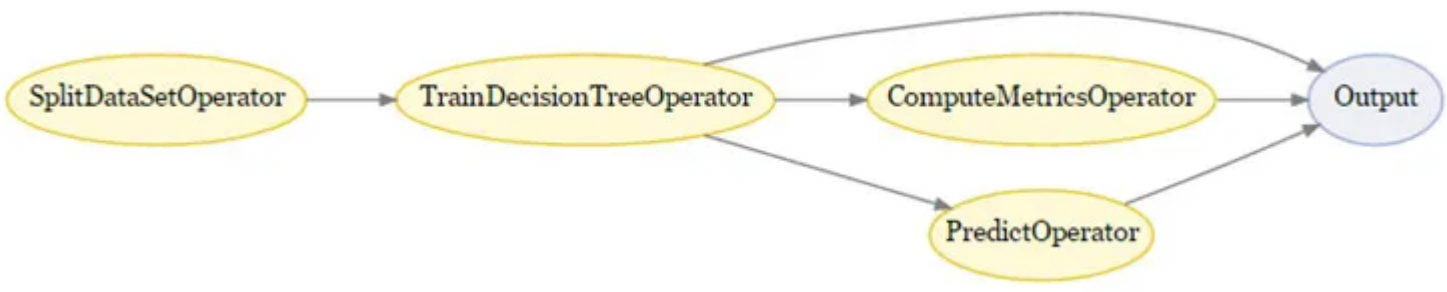
Let's examine this workflow closely to understand how FBLearner Flow works under the hood.

First, the `@workflow` decorator indicates to FBLearner Flow that the `iris` function is not a normal Python function but instead a workflow. The `input_schema` and `returns` arguments indicate the types of the inputs expected by the workflow and the outputs it produces. The execution framework will automatically verify these types at runtime and enforce that the workflow receives the data it expects. Looking at this example, the `labeled_data` input is marked as a data set input with four columns. If one of these columns were missing in the provided data set, then a `TypeError` exception would be raised, because the data set would be incompatible with this workflow.

The body of the workflow looks like a normal Python function with calls to several operators, which do the real machine learning work. Despite its normal appearances, FBLearner Flow employs a system of futures to provide parallelization within the workflow, allowing steps that do not share a data dependency to run simultaneously.

Rather than execute serially, workflows are run in two stages: 1) the DAG compilation stage and 2) the operator execution stage. In the DAG compilation stage, operators within the workflow are not actually executed and instead return futures. Futures are objects that represent delayed pieces of computation. So in the above example, the `dt` variable is actually a future representing decision tree training that has not yet occurred. FBLearner Flow maintains a record of all invocations of operators within the DAG compilation stage as well as a list of futures that must be resolved before it operates. For example, both `ComputeMetricsOperator` and `PredictOperator` take `dt.model` as an input, thus the system knows that `dt` must be computed before these operators can run, and so they must wait until the completion of `TrainDecisionTreeOperator`.

By the completion of the DAG compilation stage, FBLearner Flow will have built a DAG of operators in which edges represent data dependencies. This DAG can then be scheduled for execution where each operator can begin execution once its parents have completed successfully. In this example, there is no data dependency between the call to `ComputeMetricsOperator` and `PredictOperator`, so these two operators can run in parallel.



In the operator execution stage, each operator is run when its dependent operators complete. Each operator declares its CPU, GPU, and memory requirements, and FBLearner Flow will allocate a slice of a machine that matches the operator's requirements for the task. FBLearner Flow automatically handles deploying the relevant code to the machine and transporting inputs and outputs between operators.

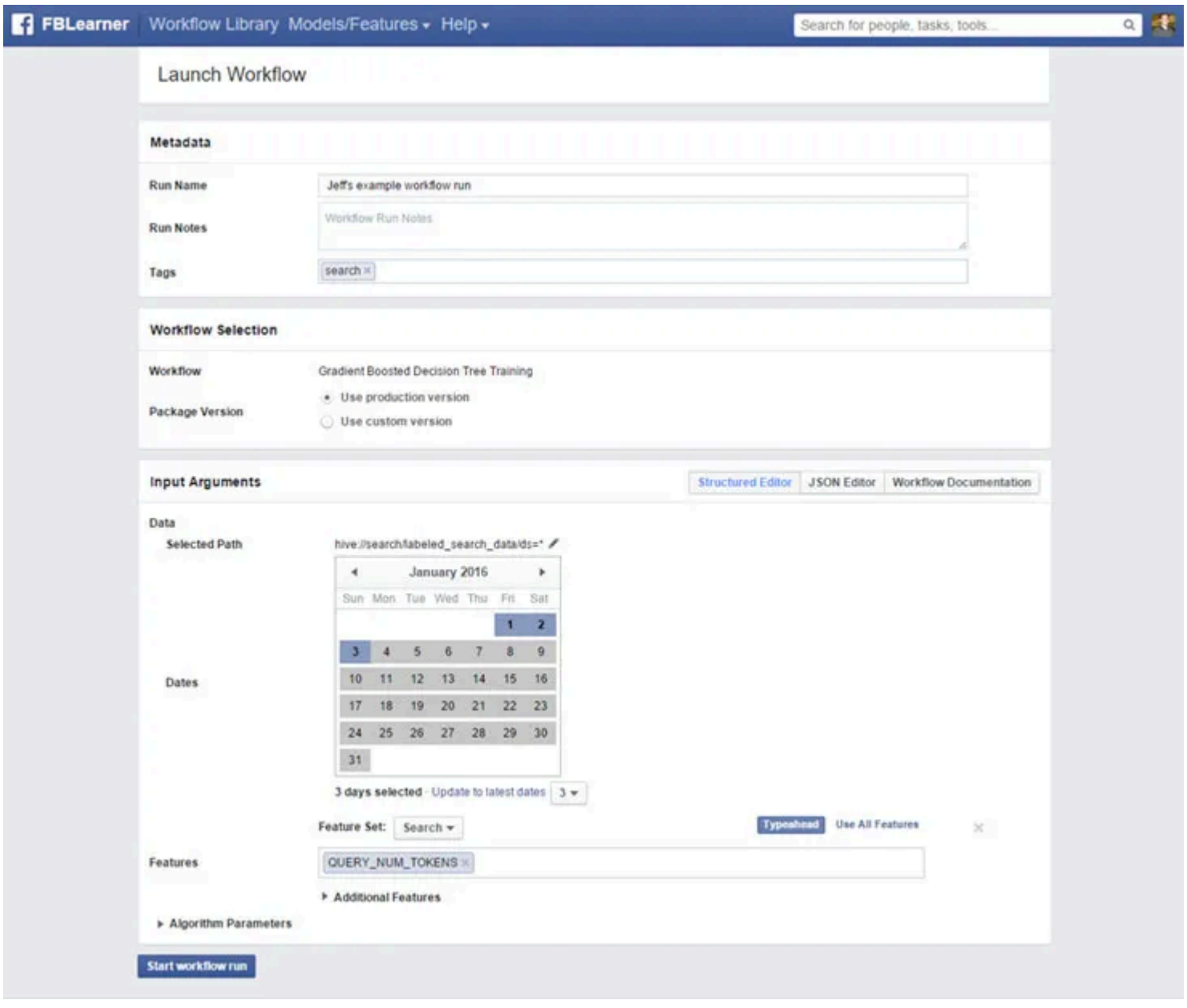
Experimentation management UI

Across Facebook, there are hundreds of different workflows performing myriad machine learning tasks. One challenge we faced was building a generic UI that works with the diverse set of workflows our engineers use. Using the custom type system, we built a UI that can interpret inputs and outputs in a workflow-agnostic manner without needing to understand the implementation details of each workflow. For further customization, the FBLearner Flow UI provides a plugin system that can be used for custom experiences for specific teams and integration with Facebook's systems.

The FBLearner Flow UI provides a few additional experiences: 1) launching workflows, 2) visualizing and comparing outputs, and 3) managing experiments.

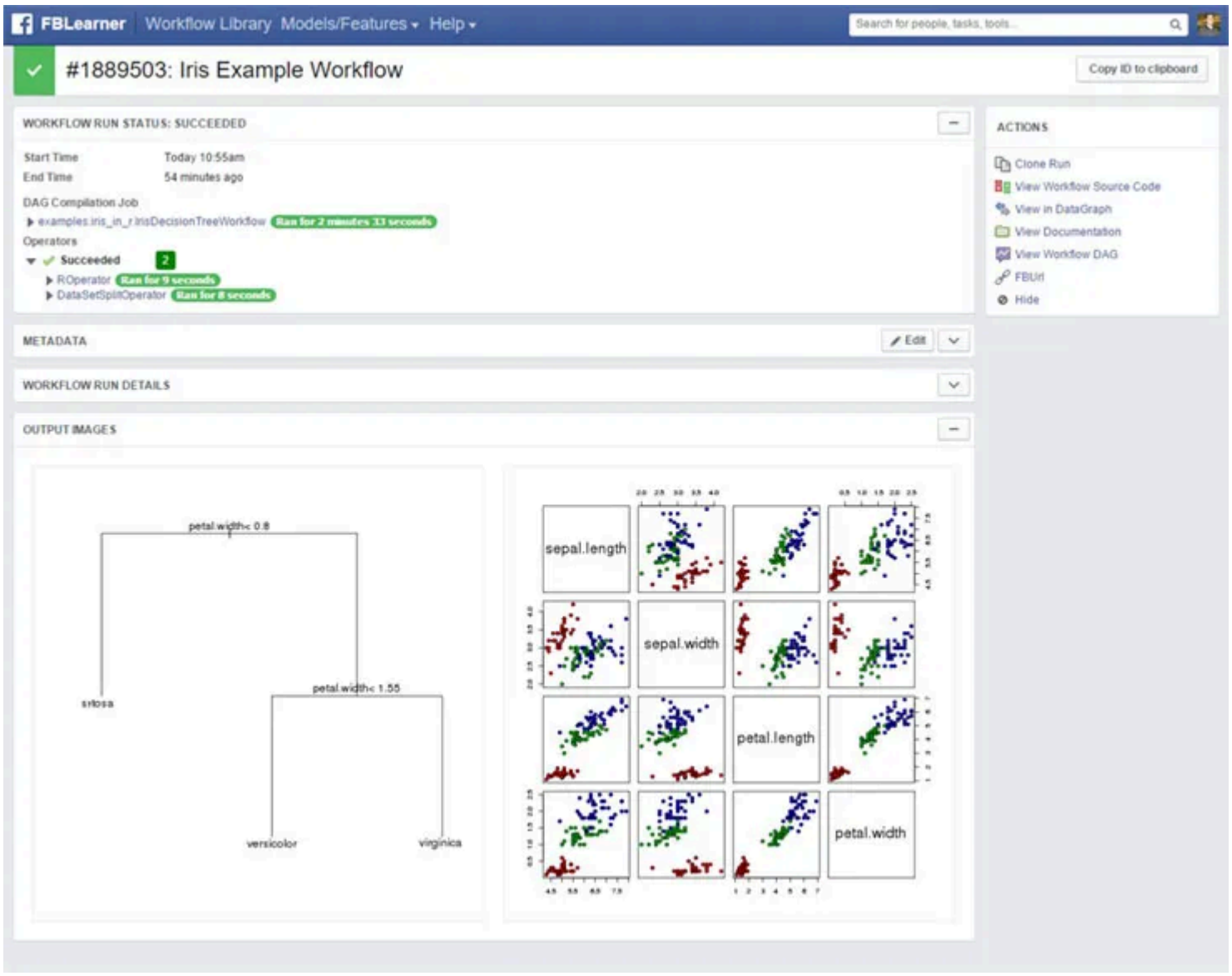
Launching workflows

We saw before that every workflow declares a typed input schema. When engineers launch a workflow, the UI reads this input schema and automatically generates a structured form with validation to specify inputs to the workflow. This allows machine learning engineers to get a rich UI for their workflows without writing a single line of frontend code. FBLearner Flow's custom type system has rich types for describing data sets, features, and many other common machine learning data types. This allows the UI to render complex input elements such as typeaheads for features and selectors for data sets.



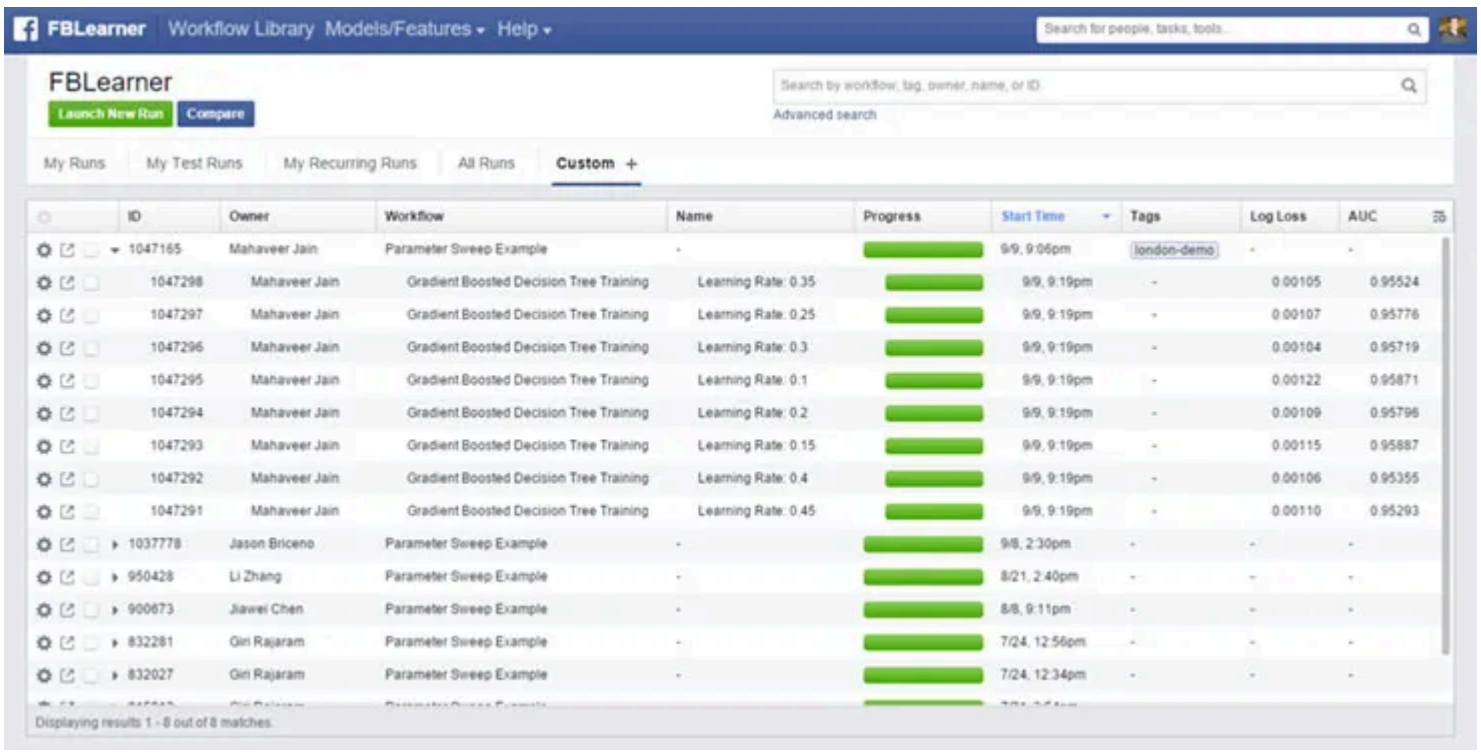
Visualizing and comparing outputs

Engineers can view each workflow run from the UI to see its outputs, to modify tags and other metadata, and to take actions like deploying the model to production. Inputs and outputs from workflow runs can be compared to allow engineers to evaluate the performance of experiments against baselines. We apply a similar technique for visualizing outputs to rendering input forms — the type system is used to provide type-specific rendering of each output. Through a plugin system, additional custom visualization and actions can be added to these views. For example, the News Feed team can add real-time system metrics for their models that have been deployed to production.



Managing experiments

Facebook engineers launch thousands of experiments every day, and the FBLearner Flow UI provides tools to manage all of these experiments. All workflow runs are indexed in Elasticsearch so they are easily searchable by numerous dimensions, and the system supports saved search queries to easily find experiments. When tuning models, engineers will often run complex parameter sweeps that have specialized rendering to easily see which configurations produced the best results.



Machine learning library

A core principle of the FBLearner Flow platform is that it is not tied to any specific algorithm. As a result, the platform supports numerous machine learning algorithms and innovative combinations of these algorithms. It's easily extensible too — any engineer can write a new workflow to make his or her favorite algorithm available to the entire company. Open source implementations of algorithms can easily be wrapped in a workflow and integrated with Facebook's infrastructure.

Facebook's Applied Machine Learning team maintains workflows that provide scalable implementations of commonly used algorithms, including:

- Neural networks
- Gradient boosted decision trees
- LambdaMART
- Stochastic gradient descent
- Logistic regression

Future plans

With FBLearner Flow, AI is becoming an integral part of our engineering fabric and providing Facebook engineers with the power of state-of-the-art AI through simple API calls. We're constantly working to improve FBLearner Flow to allow engineers to become increasingly productive and apply machine learning to a growing number of products. This enables gains in:

- **Efficiency**: This past April, more than 500,000 workflow runs were executed on a cluster containing thousands of machines. Some of these experiments require lots of computational resources and can take multiple days to complete. We are focused on improving the efficiency of executing these experiments to ensure that the platform can scale to the growing demand while simultaneously keeping execution latency to a minimum. We're exploring new solutions around data locality to co-locate computation with the source data, and we're improving our understanding of resource requirements to pack as many experiments onto each machine as possible.
- **Speed**: FBLearner Flow is a system that ingests trillions of data points every day, trains thousands of models — either offline or in real time — and then deploys them to the server fleet for live predictions. Engineers and teams, even with little expertise, can build and run experiments with ease and deploy AI-powered products to production faster than ever. We're working to minimize turnaround time for workflow completion to allow our products to learn on the latest data and to allow engineers to quickly iterate in their experiments.
- **Machine learning automation**: Many machine learning algorithms have numerous hyperparameters that can be optimized. At Facebook's scale, a 1 percent improvement in accuracy for many models can have a meaningful impact on people's experiences. So with Flow, we built support for large-scale parameter sweeps and other AutoML features that leverage idle cycles to further improve these models. We are investing further in this area to help scale Facebook's AI/ML experts across many products in Facebook.

In the coming months, we'll take a closer look at some of the specific systems and applications that leverage FBLearner Flow to enable engineers to more easily apply AI and ML to their products, as well as to deliver a more personalized experience for people using Facebook.

TAGS: INFRA NEWS FEED PYTHON

◂ Prev
Automatically push commits to GitHub with FBShipIt

Next ▸
Artificial intelligence, revealed

Read More in AI Research

View All ▸

AUG 23, 2024

How PyTorch powers AI training and inference

AUG 22, 2024

Inside the hardware and co-design of MTIA

AUG 21, 2024

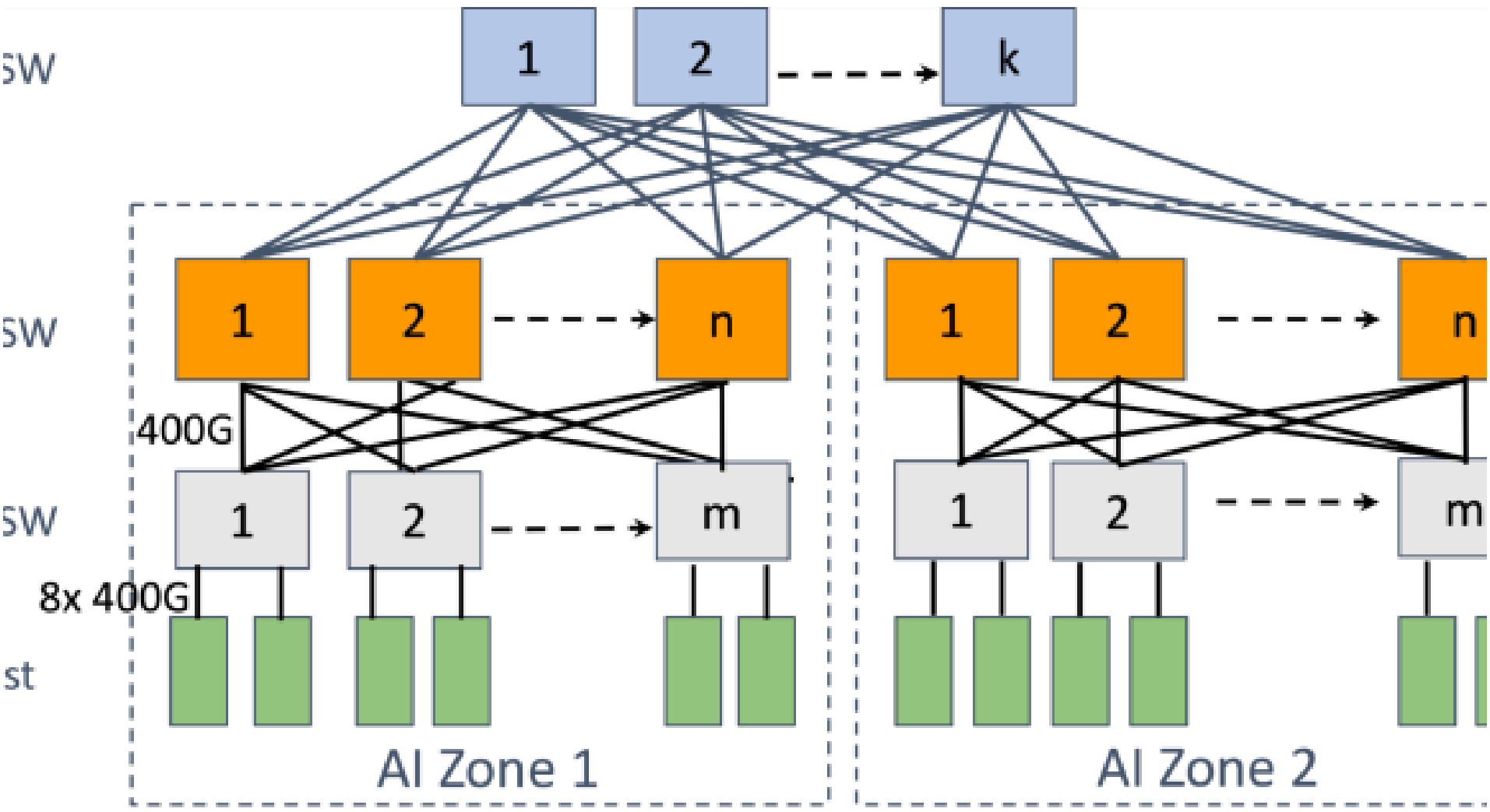
Bringing Llama 3 to life

AUG 20, 2024

Aparna Ramani discusses the future of AI infrastructure

AUG 14, 2024

How Meta animates AI-generated images at scale



AUG 5, 2024

RoCE networks for distributed AI training at scale

Related Posts

Related Positions

Research Scientist Manager - Input & Interaction

NEW YORK, US

Research Engineer - Reality Labs

MENLO PARK, US

Research Engineer - Reality Labs

BURLINGAME, US

Research Engineer - Reality Labs

NEW YORK, US

AI Research Scientist, Computer Vision - XR Tech

BURLINGAME, US

See All Jobs

Available Positions

Research Scientist Manager - Input & Interaction
NEW YORK, US
Research Engineer - Reality Labs
MENLO PARK, US
Research Engineer - Reality Labs
BURLINGAME, US
Research Engineer - Reality Labs
NEW YORK, US
AI Research Scientist, Computer Vision - XR Tech
BURLINGAME, US

See All Jobs

Technology at Meta

Engineering at Meta - X

Follow

AI at Meta

Read

Meta Quest Blog

Read

Meta for Developers

Read

Meta Bug Bounty

Learn more

RSS

Subscribe

Open Source

Meta believes in building community through open source technology. Explore our latest projects in Artificial Intelligence, Data Infrastructure, Development Tools, Front End, Languages, Platforms, Security, Virtual Reality, and more.

ANDROID

iOS

WEB

BACKEND

HARDWARE

Learn More

Meta

Engineering at Meta is a technical news resource for engineers interested in how we solve large-scale technical challenges at Meta.

Home

Company Info

Careers