Microsoft

Designing and implementing a data science solution on Azure Exam-prep

aka.ms/DP100-Course

Hello!

Thank you for joining me today

Instructor: Morten Nordli

Microsoft Certified Trainer

AI consultant

AI Certified instructor, startup entrepreneur, ...

www.linkedin.com/in/datavettmorten

Let’s get to know each other

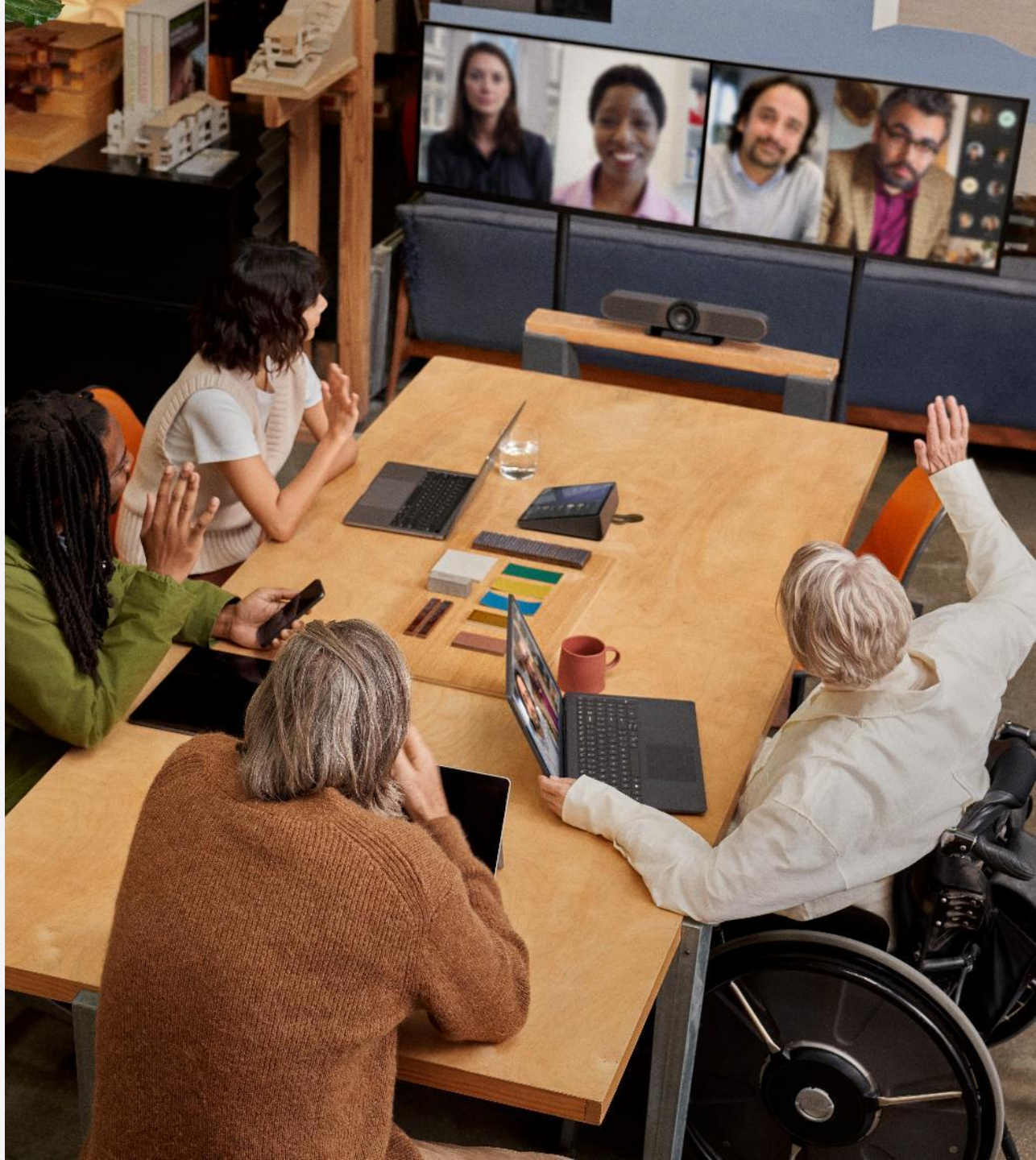
Your name

Company affiliation

Title/function

Your experience

Your expectations for the course



Course objectives

After completing these days, you should be able to:

- Design and create a suitable **working environment** for data science workloads.
- **Explore** data and **train** machine learning models on Azure Machine Learning.
- Run **jobs** and **pipelines** to prepare your models for production.
- **Deploy** and monitor scalable machine learning solutions.
- **Optimize** language models for AI applications.

Audience profile

Candidates for this exam:

- Leverage **Azure Machine Learning** to manage their assets and resources for data science workloads.
- Interact with the Azure Machine Learning workspace primarily with the **Python SDK** (v2).
- Track and manage machine learning models with **MLflow**.
- Optimize language models with **Azure AI Foundry**.

Why use Azure ML

- On-demand compute that you can customize based on the workload
- Data ingestion engine which I found to be extensive in terms of the sources it accepts.
- Workflow orchestration for machine learning is incredibly simple with azure.
- Machine Learning model management – if you like evaluating multiple models before selecting the final one, azure machine learning has dedicated capabilities to manage this.
- Metrics & logs of all the model training activities and services we utilize are readily available on the platform.
- Model deployment – With azure ML, you can deploy your model in real-time
- Governance, security, management and security in Azure

[What is Azure Machine Learning? - Azure Machine Learning | Microsoft Learn]

Course schedule – Module 1

- Design a machine learning solution
- Explore Azure Machine Learning workspace resources and assets
- Use automated machine learning to explore optimal models

Labs:

- Explore the Azure Machine Learning workspace
- Explore developer tools for workspace interaction
- Find the best classification model with Automated Machine Learning

Course schedule – Module 2

- Use notebooks for custom model training
- Run model training scripts
- Automate hyperparameter tuning

Labs:

- Track model training in notebooks with MLflow
- Run a training script as a command job
- Use MLflow to track training jobs
- Perform hyperparameter tuning with a sweep job

Course schedule – Module 3

- Implement training pipelines
- Manage models
- Deploy a model

Labs:

- Run pipelines
- Create and explore the Responsible AI dashboard
- Deploy a model to a batch endpoint
- Deploy a model to a managed online endpoint

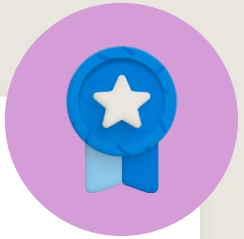
Course schedule – Module 4

- Explore and deploy models from the model catalog
- Optimize model performance through prompt engineering
- Optimize through Retrieval Augmented Generation (RAG)
- Optimize through fine-tuning

Get ready for your Microsoft Certification exam

Exam DP-100: Designing and Implementing a Data Science Solution on Azure

Study area	Percentage
Design and prepare a machine learning solution	20-25%
Explore data, and run experiments	20-25%
Train and deploy models	25–30%
Optimize language models for AI applications	25–30%



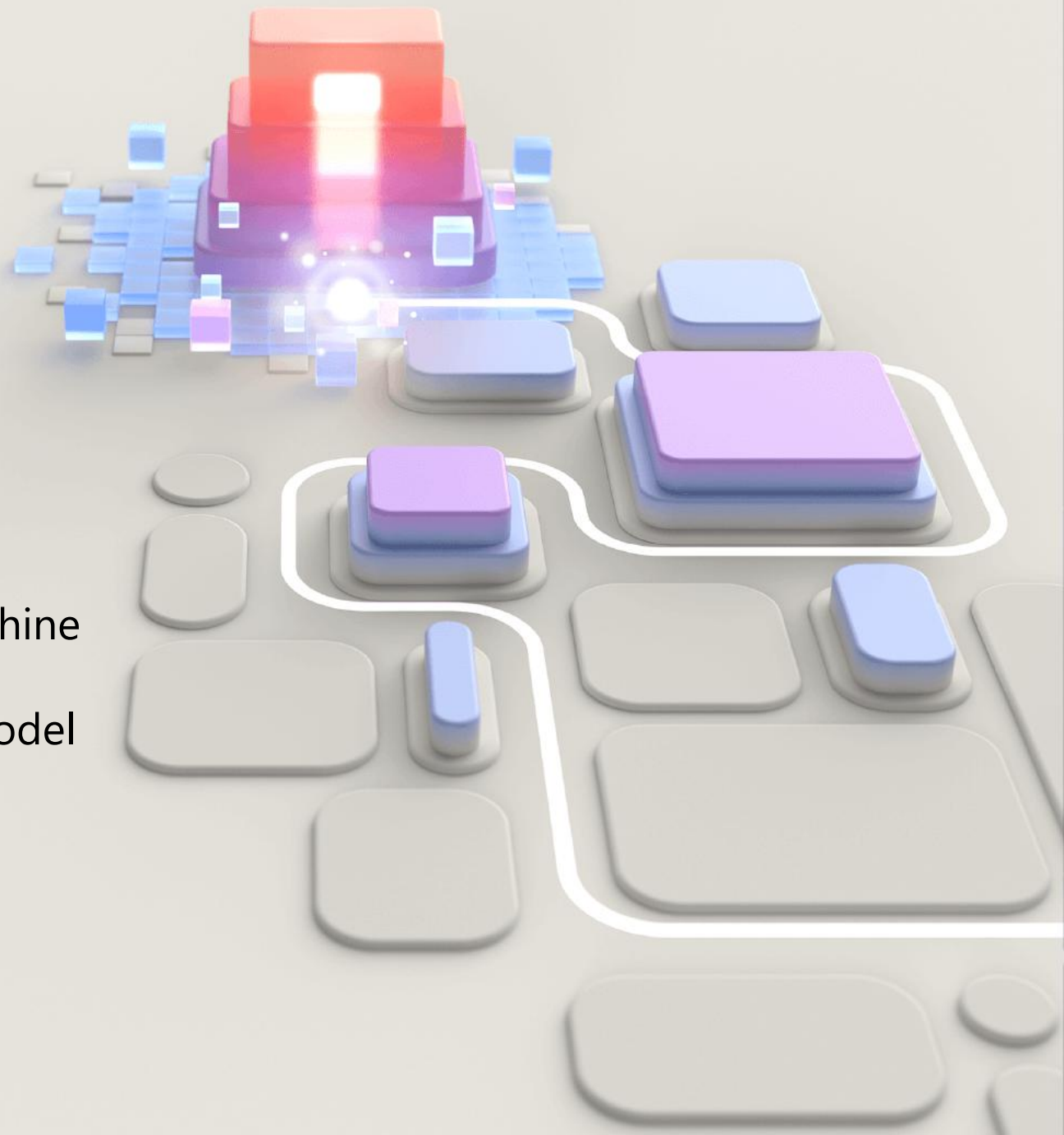
Exam preparation resources:

- Watch exam prep videos
- Review the exam study guide
- Demo the exam experience with the exam sandbox
- Take a practice assessment

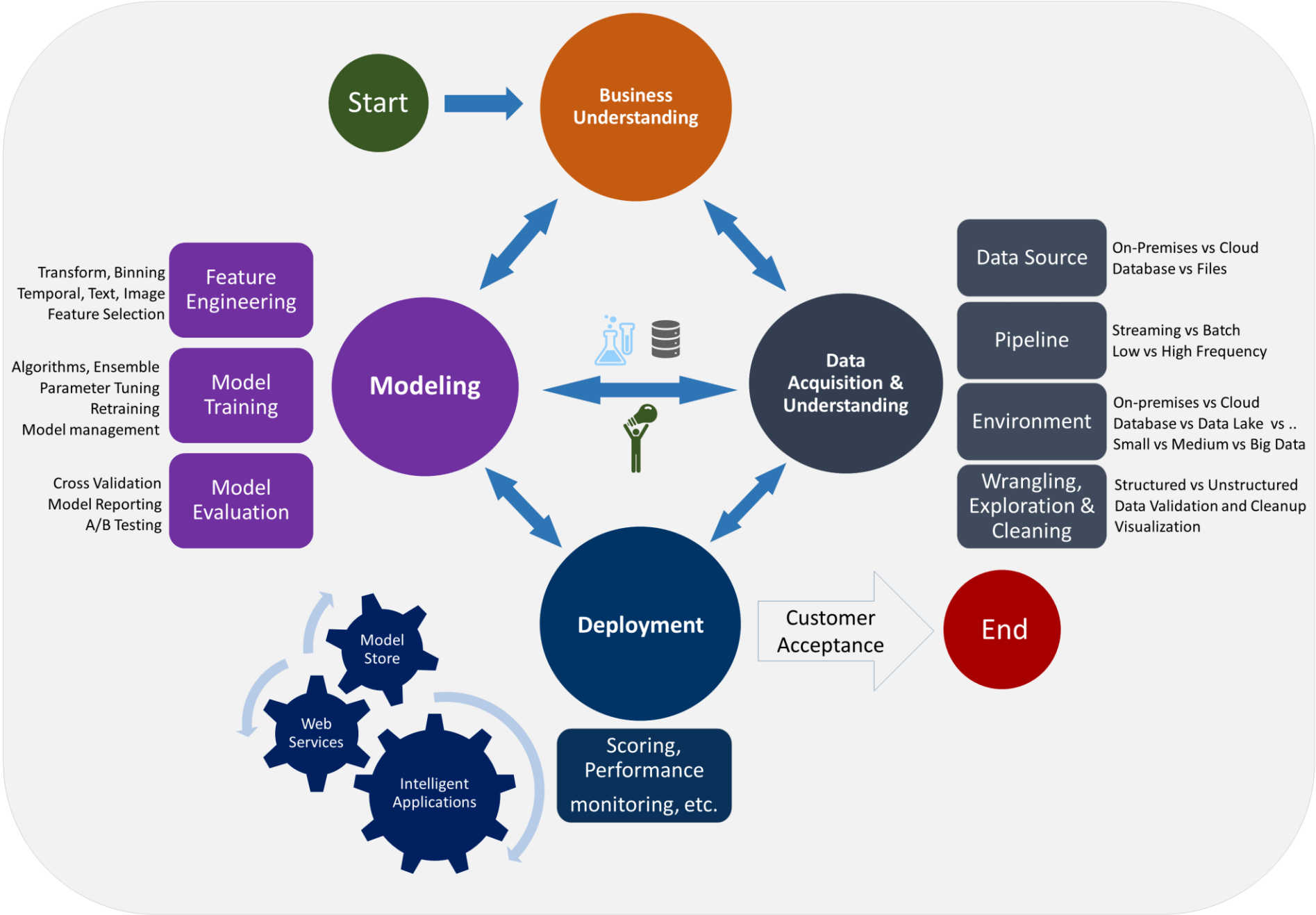
Microsoft

Design a machine learning training solution

- Identify the structure and format for datasets
- Determine the compute specifications for machine learning workload
- Select the development approach to train a model



Data Science Lifecycle



AI Services in Azure

- Azure Machine Learning: Manage assets and resources for data science workloads, interact with the workspace using the Python SDK, and track and manage models with MLflow.
- Azure AI Foundry: Optimize language models for AI applications.
- Azure AI Services: Provides pre-built AI models that can be easily integrated into applications .
- Agent services and frameworks (Copilot Studio, Semantic Kernel, Agent Service ….

Create the workspace

You can create a workspace in any of the following ways:

- Use the user interface in the Azure portal.
- Create an Azure Resource Manager (ARM) template.
- Use the Azure Command Line Interface (CLI) with the Azure Machine Learning CLI extension.
- Use the Azure Machine Learning Python SDK.

Python

```python
from azure.ai.ml.entities import Workspace
workspace_name = "mlw-example"
ws_basic = Workspace(
    name=workspace_name,
    location="eastus",
)
ml_client.workspaces.begin_create(ws_basic)
```

Give access to the Azure Machine Learning workspace

Access is granted in Azure using role-based access control (RBAC)

There are three general built-in roles that you can use across resources and resource groups to assign permissions to other users:

- **Owner:** Gets full access to all resources and can grant access to others using access control.
- **Contributor:** Gets full access to all resources but can't grant access to others.
- **Reader:** Can only view the resource but isn't allowed to make any changes.

Additionally, Azure Machine Learning has specific built-in roles you can use:

- AzureML Data Scientist
- AzureML Compute Operator

To fully customize permissions, create a custom role.

Work with resources - DEMO

1. **The workspace** – The top-level resource for Azure Machine Learning. The workspace keeps an overview of all logs, metrics, outputs, models, and snapshots of your code.

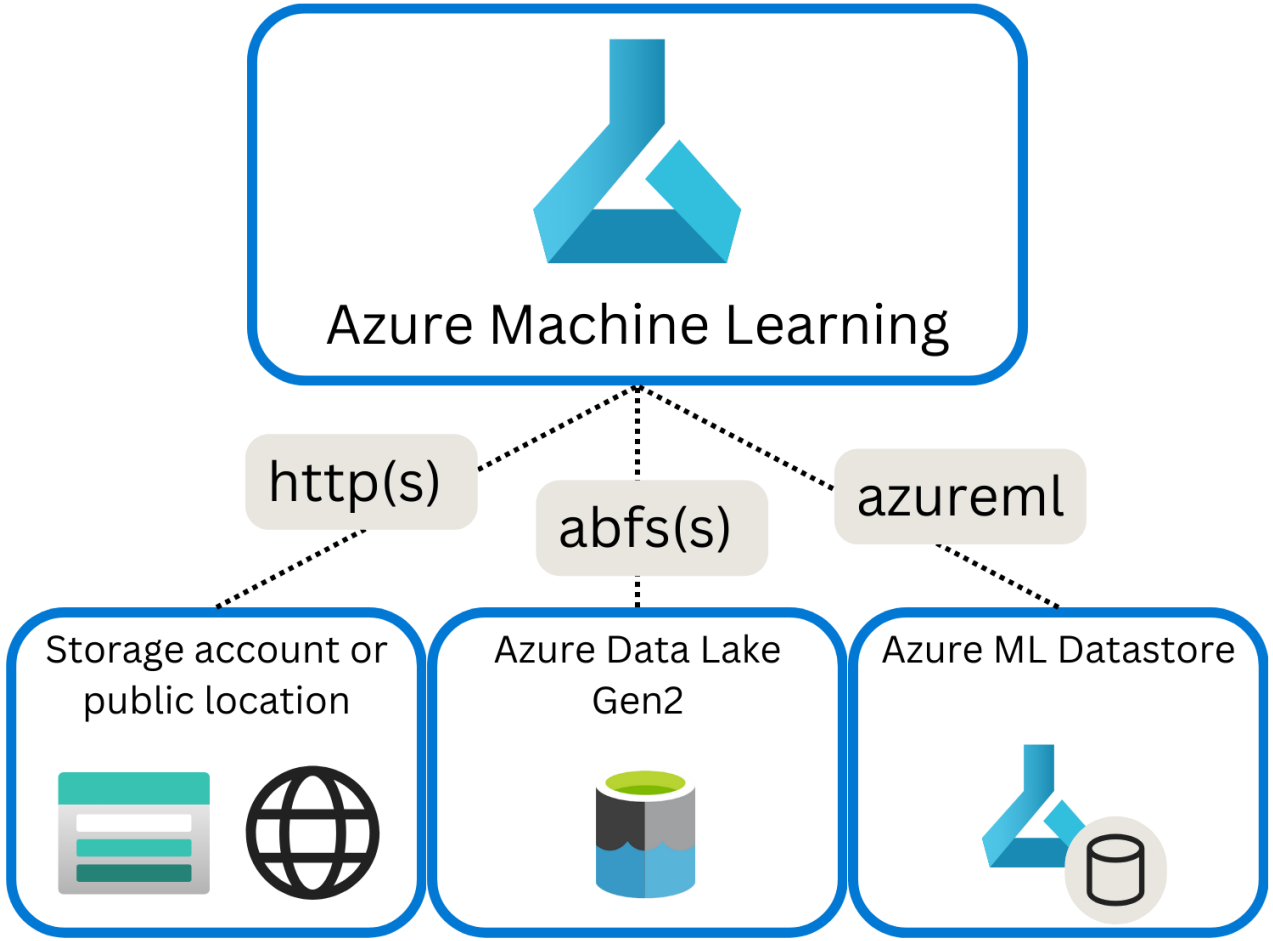
2. **Compute resources** – There are five types of compute in the Azure Machine Learning workspace: compute instances, compute clusters, Kubernetes clusters, attached computes, and serverless compute.

3. **Datastores** – All data is stored in datastores, which are references to Azure data services. Four datastores will exist by default.

Understand URIs - DEMO

A URI references the **location of your data.**

For Azure Machine Learning to connect to your data directly, you need to prefix the URI with the appropriate protocol.

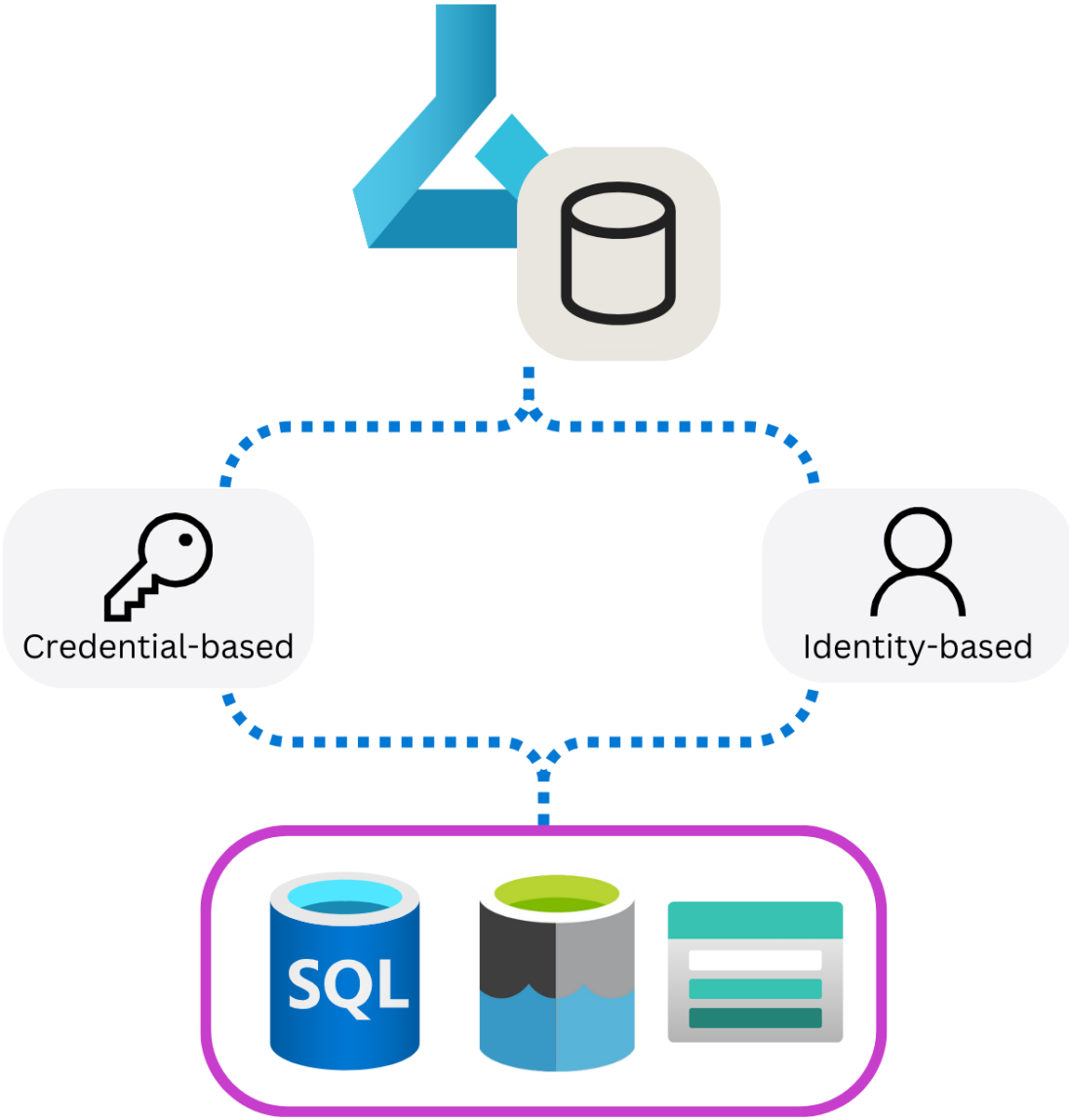


Understand datastores

Datastores are **abstractions** for cloud data sources, storing the **connection information**.

The benefits of datastores:

- Provide easy-to-use URIs to your data storage.
- Facilitates data discovery within Azure Machine Learning.
- Securely stores connection information, without exposing secrets and keys to data scientists.



Create and manage assets - DEMO

1. **Data** – You can use data assets to easily access data every time, without having to provide authentication every time you want to access it.

2. **Environments** – Specify software packages, environment variables, and software settings to run scripts. An environment is stored as an image in the Azure Container Registry created with the workspace when it's used for the first time.

3. **Models** – Save trained models in the workspace. A common way to store such models is to package the model as a Python pickle file (.pkl extension).

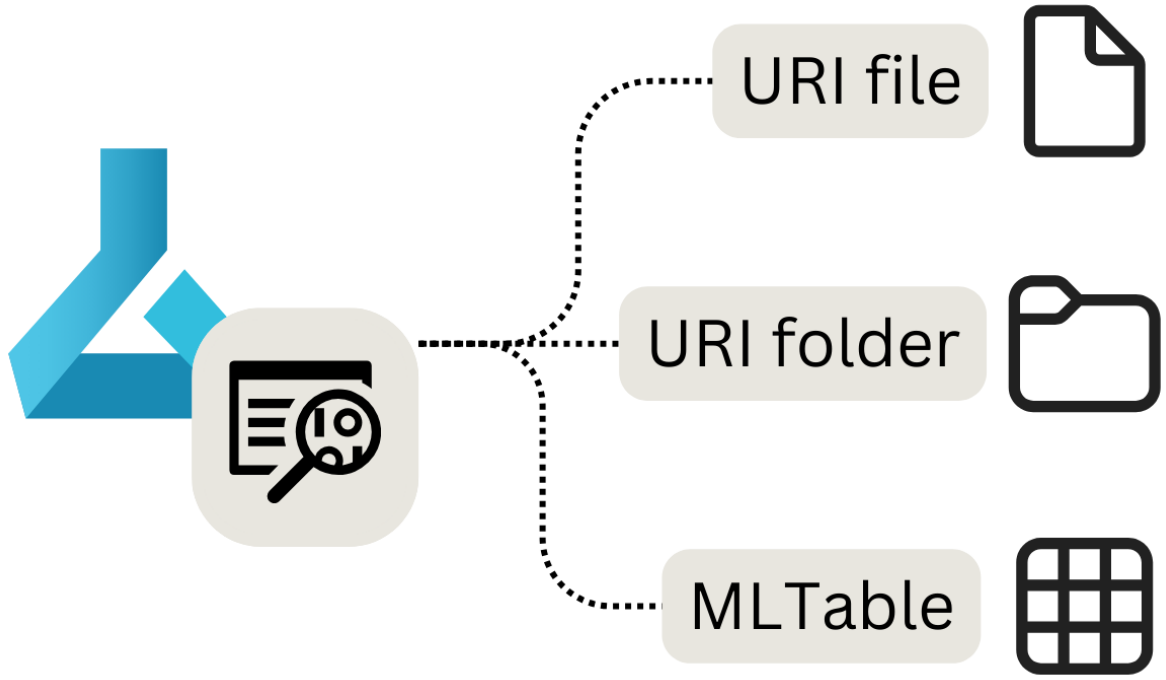
4. **Components** – Make it easier to share code with component in a workspace.

Understand data assets

Data assets are **references** to where the data is stored, how to get access, and any other relevant metadata.

The benefits of data assets:

- Share and reuse data with other members.
- Seamlessly access data during model training (on any supported compute type) without worrying about connection strings or data paths.
- Version the metadata of the data asset.



Understand data assets

The benefits of using data assets are:

- You can share and reuse data with other members of the team such that they don't need to remember file locations.
- You can seamlessly access data during model training (on any supported compute type) without worrying about connection strings or data paths.
- You can version the metadata of the data asset.

Three main types of data assets you can use:

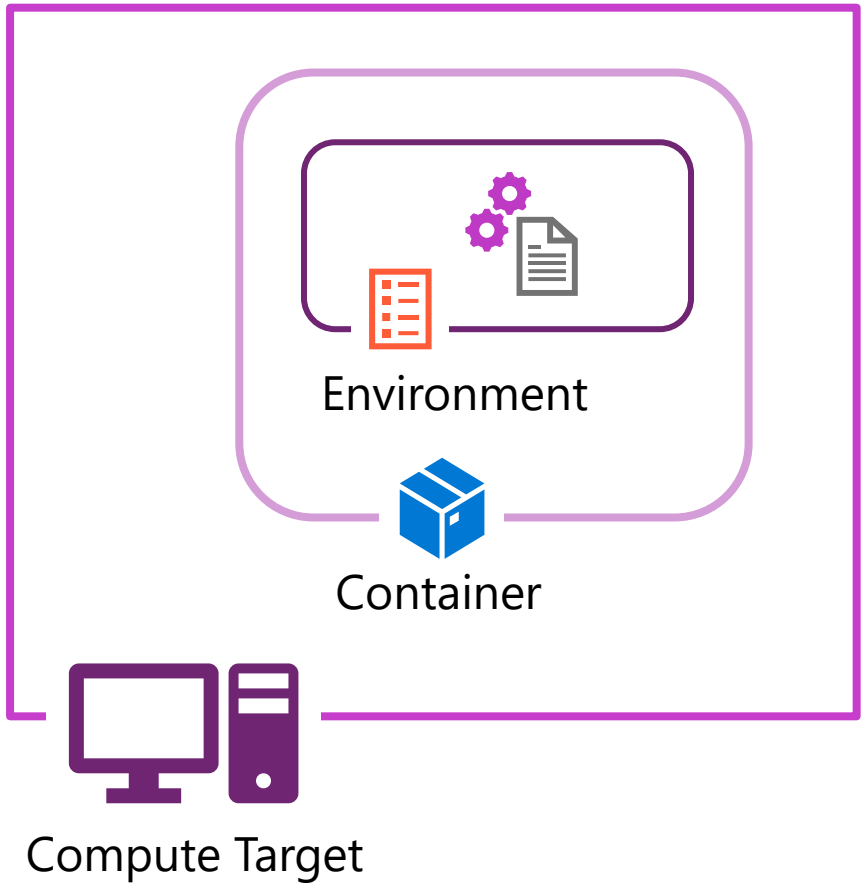
- **URI file:** Points to a specific file.
- **URI folder:** Points to a folder.
- **MLTable:** Points to a folder or file, and includes a schema to read as tabular data.

Understand environments

Environments define the context (runtime and packages) needed to run code on a compute target.

An environment is used to create the **Docker container** that your code runs in on the specified compute target.

Use a predefined **curated** environment or create your own **custom** environment.



Create custom environments

Creating a custom environment within the Azure Machine Learning workspace from a:

- **Docker image**: choose an existing image from a (public) repository
- **Docker build context**: reference a path that includes a Dockerfile and requirements.txt
- **Conda specification**: reference an image and add a conda YAML file that includes additional dependencies

Dockerfile

```
#FROM mcr.microsoft.com/azureml/openmpi4.1.0-ubuntu20.04
FROM python:3.8

# python installs
COPY requirements.txt .
RUN pip install -r requirements.txt && rm requirements.txt

# set command
CMD ["bash"]
```

Conda YAML file

```
name: pydata-example
channels:
  - conda-forge
dependencies:
  - python=3.8
  - pip=21.2.4
  - pip:
    - numpy==1.22
    - scipy==1.7.1
    - pandas==1.3.0
    - scikit-learn==0.24.2
```

Install the Azure Machine Learning Extension

1. Install on Windows

```
az extension add -n ml -y
```

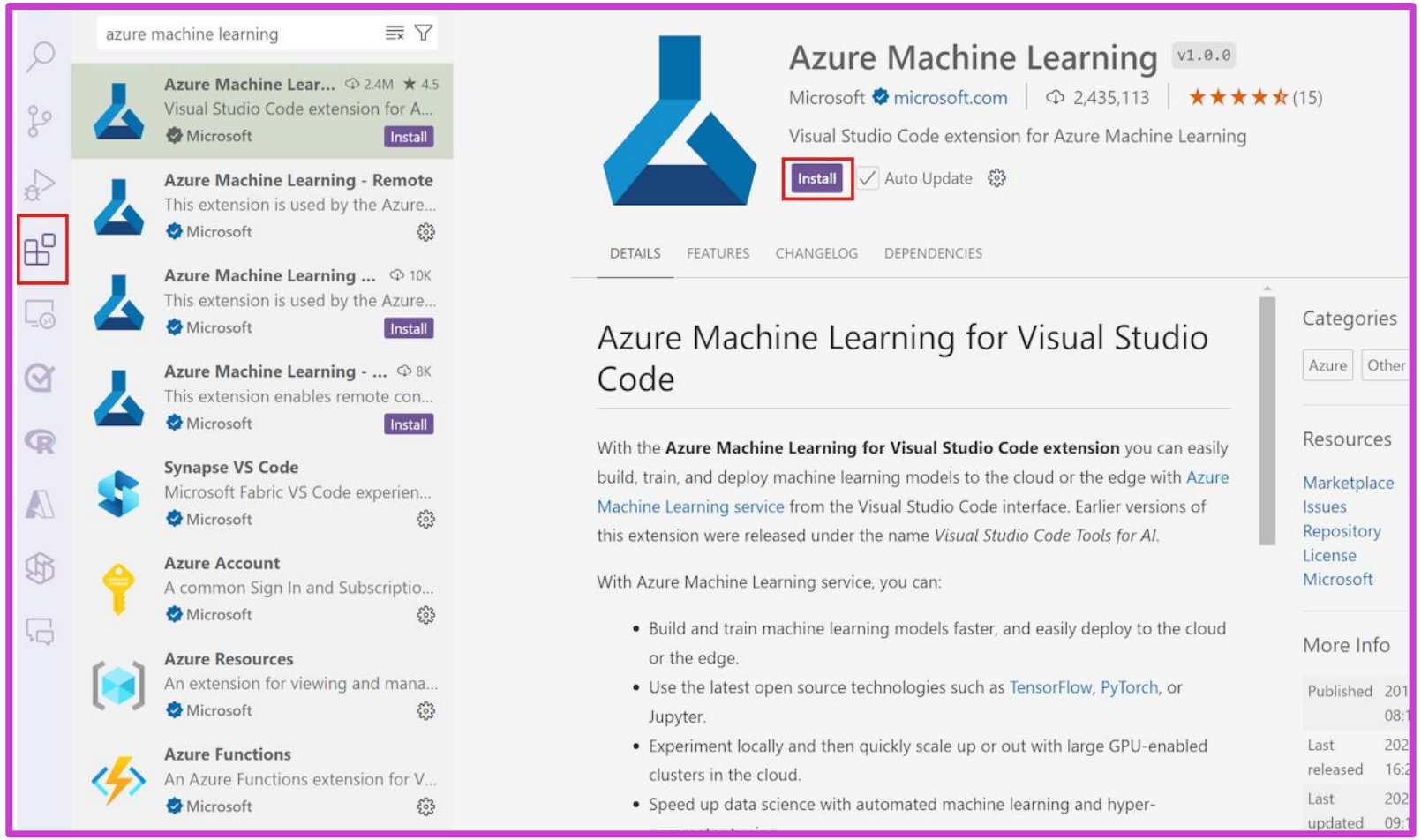
2. Work with the Azure CLI

```
az ml compute create --file compute.yml --resource-group my-resource-group
--workspace-name my-workspace
```

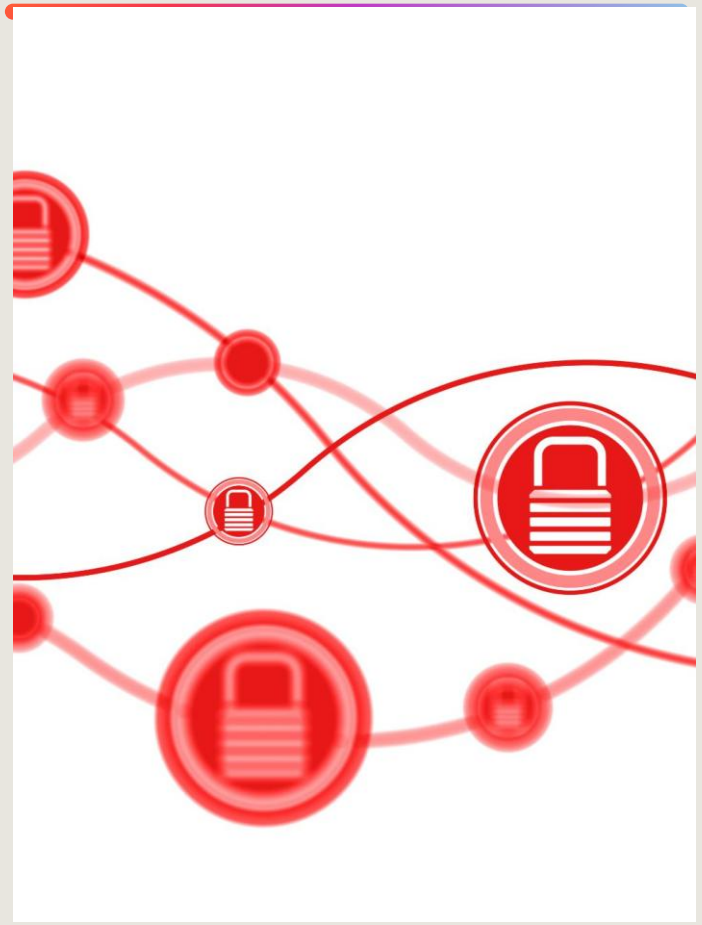
Explore the Visual Studio Code extension

When working in VS Code, you can use the Azure Machine Learning extension for:

- Managing Azure Machine Learning resources and assets
- Developing locally using remote compute instances
- Schema-based language support, autocompletion and diagnostics for specification file authoring



Azure ML Registries



Centralized Storage

Azure ML registries provide centralized storage for machine learning assets to facilitate collaboration and reuse across teams.

Cross-Region Access

They enable seamless cross-region and cross-workspace access to machine learning resources, boosting efficiency.

Improved Version Control

Azure ML registries improve version control and discoverability of machine learning resources, ensuring proper tracking and management.

Role-Based Security

Role-based access control in Azure ML registries ensures that machine learning assets are secure and only accessible by authorized users.

GitHub Actions with Azure ML

Fork and clone a repo

git clone https://github.com/YOUR-USERNAME/azureml-examples

Add GitHub Secrets

Start by adding secrets like `AZURE_CREDENTIALS`, `AZURE_SUBSCRIPTION_ID`, and `AZURE_ML_WORKSPACE` to GitHub for secure access.

Create Workflow File

Create a workflow file in `.github/workflows/` to define steps like checkout code, set up Python, and authenticate with Azure.

Update Pipeline

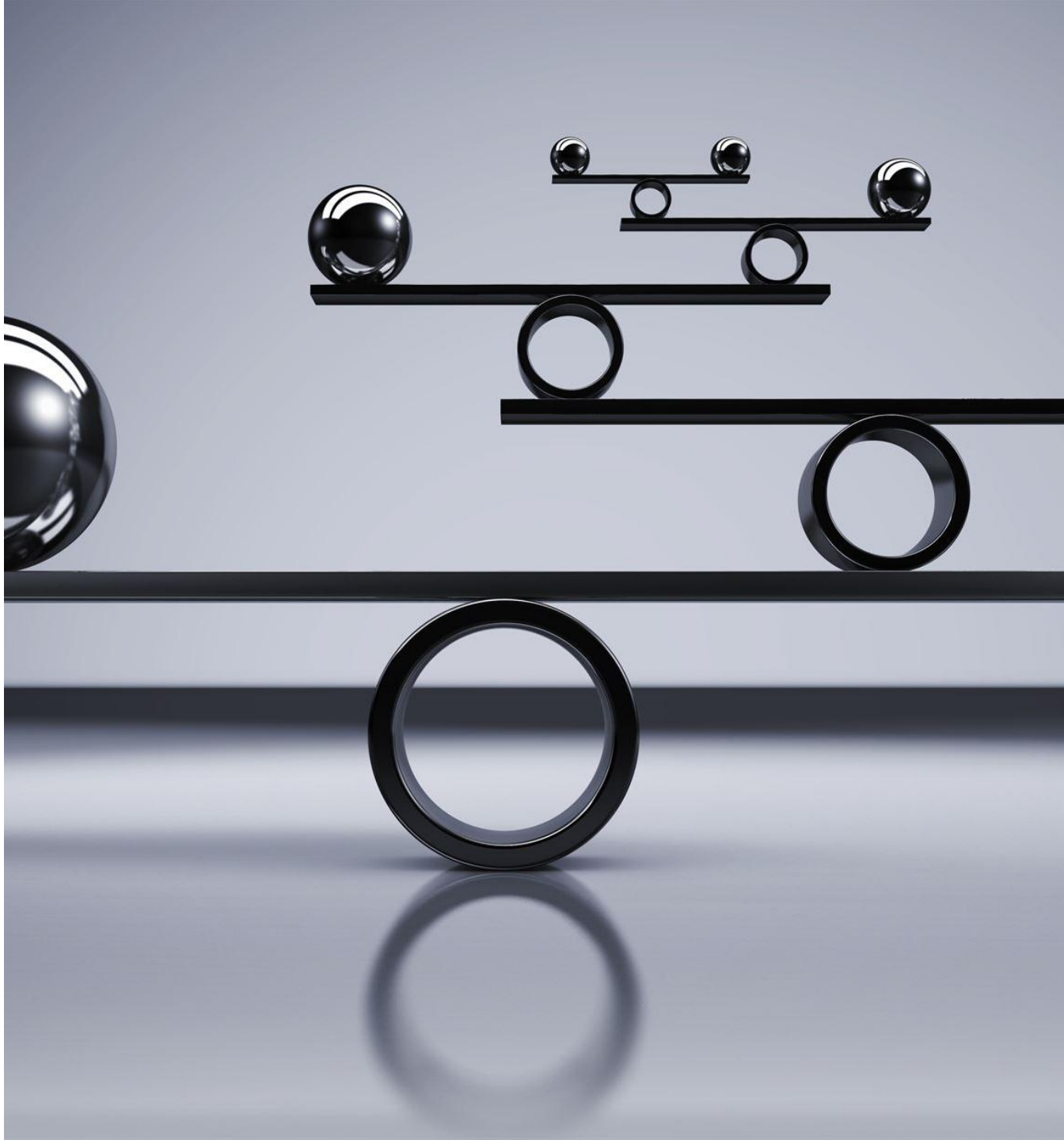
Customize `pipeline.yml` with your compute cluster and environment settings, and include the training script for Azure ML.

Monitor Job Status

Push changes to GitHub to trigger the workflow and monitor job status in Azure ML Studio for real-time updates.

Probability and Thresholds

- Class Prediction and Probability
  - Probability values range from 0 (impossible) to 1 (certain)
  - Total probability for all classes is always 1
- Example of Probability Calculation
  - If diabetic probability is 0.3, non-diabetic probability is 0.7
- Threshold Value for Classification
  - Common threshold value is 0.5
  - If diabetic probability exceeds threshold, patient is classified as diabetic



Supervised Learning Technique

Definition of Classification	• Example of supervised machine learning • Relies on known feature and label values
Feature Values	• Diagnostic measurements for patients
Label Values	• Classification of non-diabetic or diabetic
Classification Algorithm	• Fits a subset of data into a function • Calculates probability for each class label
Model Evaluation	• Uses remaining data to evaluate the model

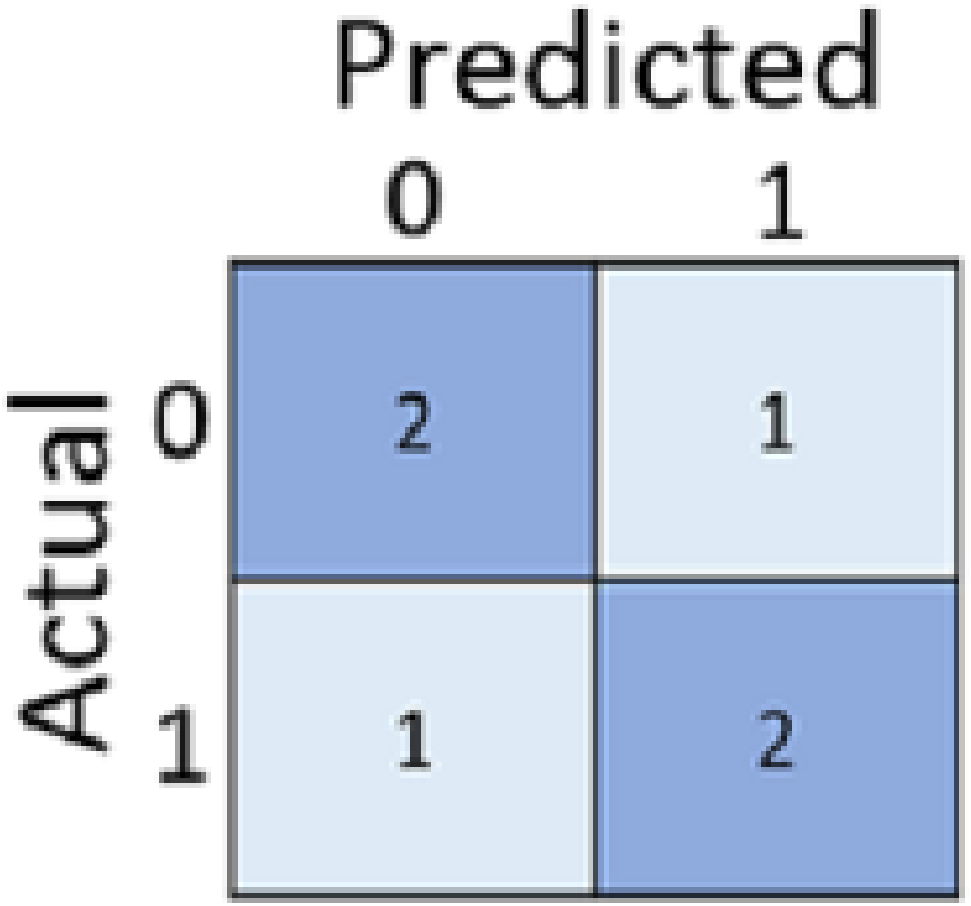
X	Y	Ŷ
83	0	0
119	1	1
104	1	0
105	0	1
86	0	0
109	1	1

Evaluation Table

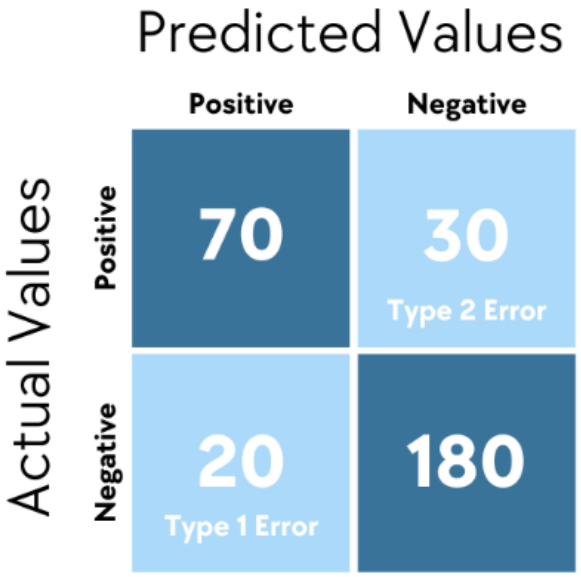
- Key Variables in Diabetes Prediction
  - x: Blood glucose level
  - y: Actual diabetic status
  - ŷ: Model's prediction of diabetic status

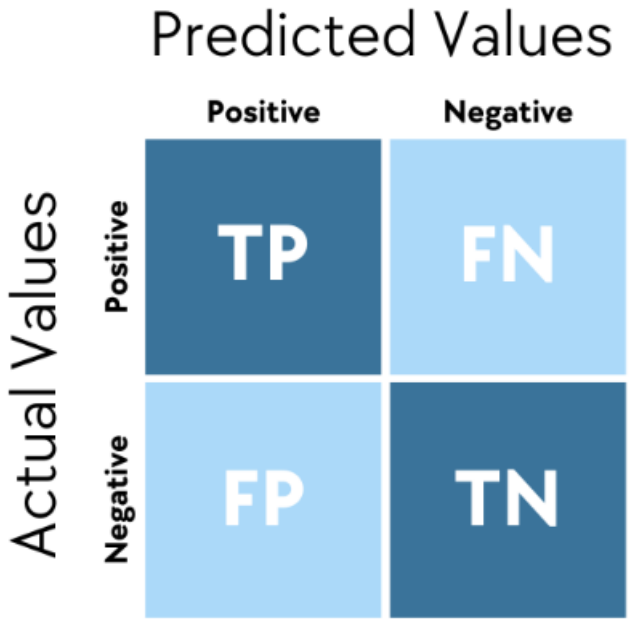
Confusion Matrix

- Limitations of Simple Accuracy Calculation
  - Can be misleading
  - Too simplistic for real-world error understanding
- Need for Detailed Information
  - Tabulate results for better insights
  - Use a confusion matrix



Confusion matrix





Confusion matrix

Precision and Recall

Now, let's dive into precision and recall:

1. **Precision:** This measures the accuracy of the positive predictions. It's the ratio of true positives to the total predicted positives. In simpler terms, it tells us how many of the predicted spam emails were actually spam.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

2. **Recall:** This measures the model's ability to detect positive instances (spam emails in this case). It's the ratio of true positives to the total actual positives. In other words, it tells us how many of the actual spam emails were correctly identified by the model.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

Imagine we have a model that predicts whether an email is spam (positive) or not spam (negative):

	Predicted Spam (Positive)	**Predicted Not Spam (Negative)**
Actual Spam	True Positive (TP)	False Negative (FN)
Actual Not Spam	False Positive (FP)	True Negative (TN)

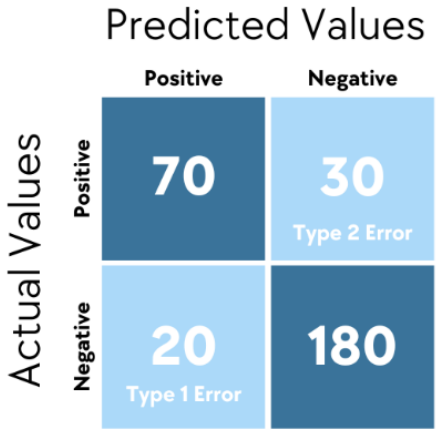
True Positive (TP): The model correctly predicts spam as spam.

False Negative (FN): The model incorrectly predicts spam as not spam.

False Positive (FP): The model incorrectly predicts not spam as spam.

True Negative (TN): The model correctly predicts not spam as not spam.

Precison and Recall



$$\text{Precision} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)} + \text{False Positive (FN)}}$$

Therefore, for our spam classification model, the Precision would be 70/90 ≈ 0.78 or 78%.

This means that the classifier is 78% accurate in identifying an email as spam.

Precision is especially important in scenarios where the cost of a False Positive is high or when incorrectly predicting something as positive can have significant consequences. For example, in medical diagnosis, predicting a disease when it's not actually present (a False Positive) can lead to unnecessary stress, additional testing, or even harmful treatments for the patient. Similarly, in fraud detection, falsely labeling a legitimate transaction as fraudulent can result in inconvenience to customers, damage reputations, and cause financial losses.

$$\text{Recall} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)} + \text{False Negative (FN)}}$$

Therefore, for our spam classification model, the Recall would be 70/100 ≈ 0.70 or 70%. This means that the classifier correctly identifies 70% of all actual spam emails.

Recall is very important in scenarios where the cost of a False Negative is very high or when wrongly predicting a positive case as negative has serious implications. For example, failing to diagnose a patient with cancer (False Negative) can have very serious implications in the healthcare industry.

SDK - Demo

LAB 02 – Run training script.ipynb

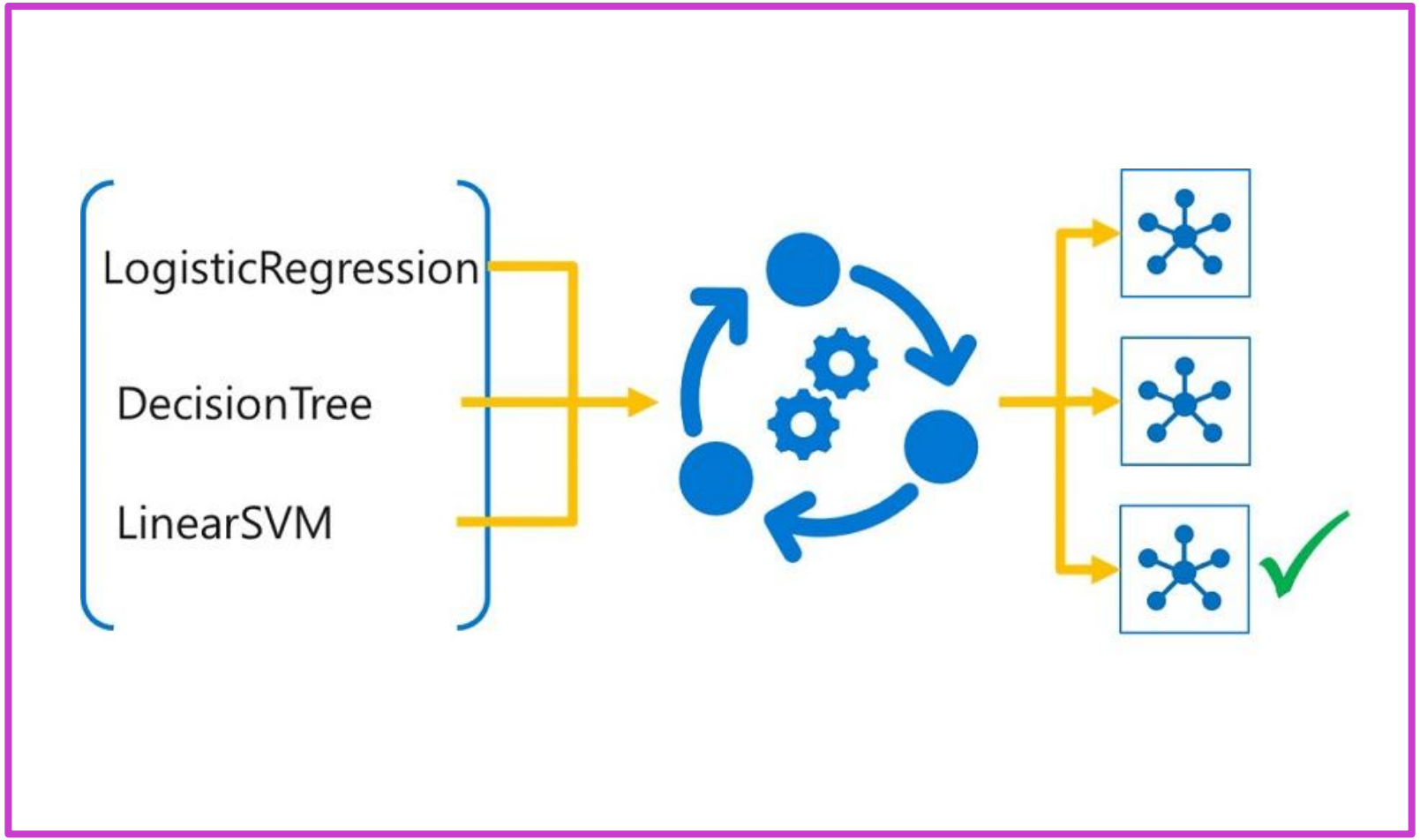
Microsoft

Explore data and run experiments DEMO

Explore Automated Machine Learning - DEMO

Instead of manually having to test and evaluate various configurations to train a machine learning model, you can automate it with automated machine learning or AutoML.

- Train multiple models in parallel, varying preprocessing and algorithm selection.
- Find the "best" model based on a specific performance metric.

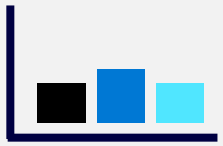


Choose a task

The training data, featurization options, algorithms, and performance metrics will depend on the task you choose.

Classification

Predict a categorical value.



Regression

Predict a numerical value.

Time-series forecasting

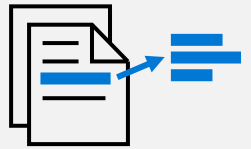
Predict future numerical values based on time-series data.

Computer vision

Classify images or detect objects in images.

Natural language processing (NLP)

Text classification or named entity recognition.



© Copyright Microsoft Corporation. All rights reserved.

Preprocess data and configure featurization

Before you can run an AutoML experiment, you need to prepare your data.

- Classification requires tabular data.
- Create a **data asset** in Azure Machine Learning.
- Create a **MLTable** data asset: Store your data in a folder together with a MLTable file.

Python

```
from azure.ai.ml.constants import AssetTypes
from azure.ai.ml import Input


my_training_data_input = Input(type=AssetTypes.MLTABLE, path="azureml:input-data-automl:1")
```

Understand scaling and normalization

AutoML applies scaling and normalization to numeric data automatically, helping prevent any large-scale features from dominating training.

During an AutoML experiment, multiple scaling or normalization techniques will be applied.

Configure optional featurization:

- Missing value imputation to eliminate nulls in the training dataset
- Categorical encoding to convert categorical features to numeric indicators
- Dropping high-cardinality features, such as record IDs
- Feature engineering (for example, deriving individual date parts from DateTime features)

Restrict algorithm selection

- By default, AutoML will randomly select from the full range of algorithms for the specified task.
- You can choose to block individual algorithms from being selected; which can be useful if you know that your data isn't suited to a particular type of algorithm.
- You also may want to block certain algorithms if you have to comply with a policy that restricts the type of machine learning algorithms you can use in your organization.

Configure an AutoML experiment (2/2)

To configure an AutoML experiment for classification, use the **automl.classification** function:

Python

```
from azure.ai.ml import automl


classification_job = automl.classification(
                compute="aml-cluster",
                experiment_name="auto-ml-class-dev",
                training_data=my_training_data_input,
                target_column_name="Diabetic",
                primary_metric="accuracy",
                n_cross_validations=5,
                enable_model_explainability=True)
```

Set the limits

There are several options to set limits to an AutoML experiment:

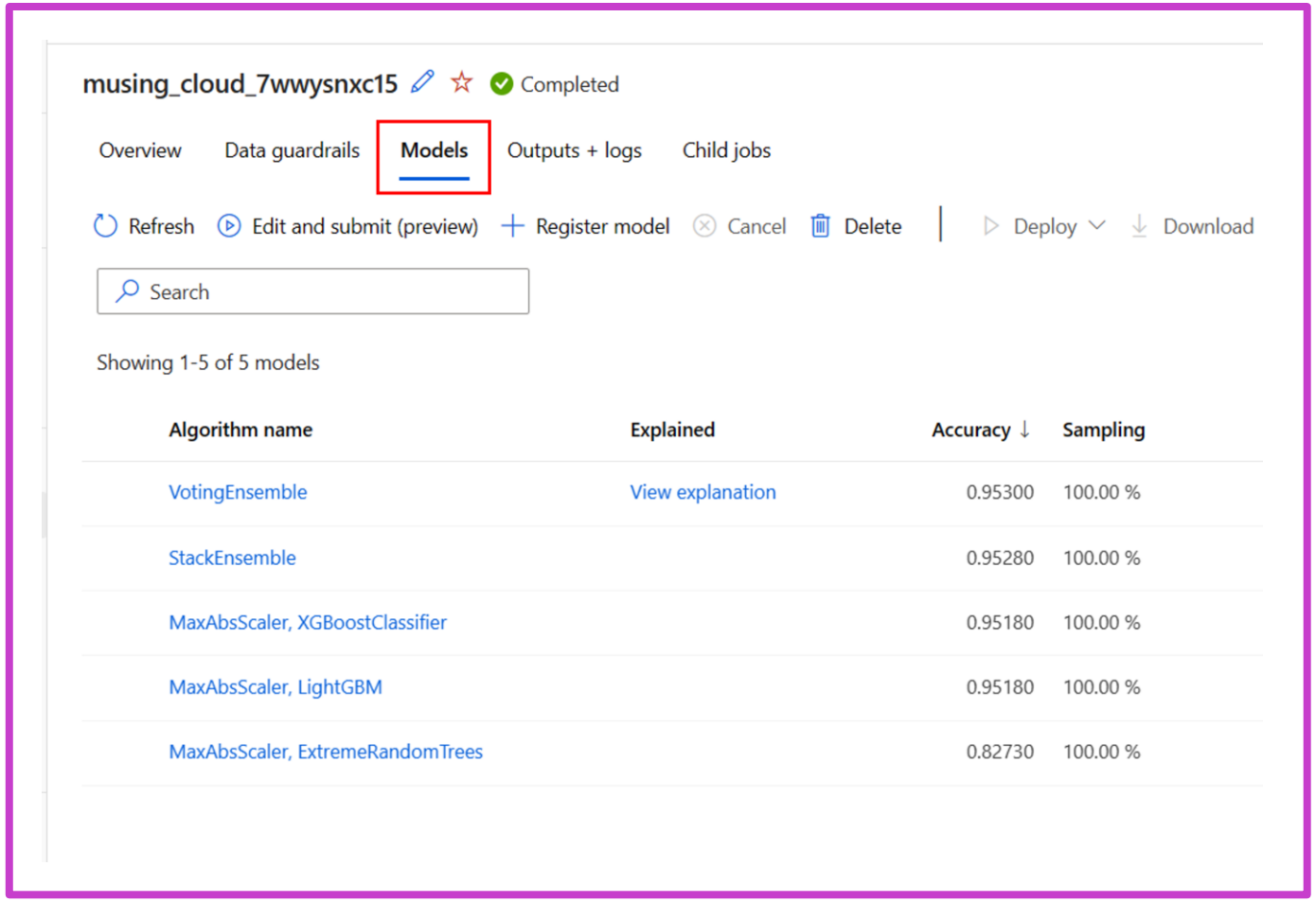
- Timeout_minutes: **Number of minutes after which the complete AutoML experiment is terminated.**
- **Trial_timeout_minutes:** Maximum number of minutes one trial can take.
- **Max_trials:** Maximum number of trials, or models that will be trained.
- **Enable_early_termination:** Whether to end the experiment if the score isn't improving in the short term.

Python

```
classification_job.set_limits(
        timeout_minutes=60,
        trial_timeout_minutes=20,
        max_trials=5,
        enable_early_termination=True)
```

Evaluate and compare models

- In the Azure Machine Learning studio, you can select an AutoML experiment to explore its details.
- On the **Overview** page of the AutoML experiment run, you can review the input data asset and the summary of the best model. To explore all models that have been trained, you can select the **Models** tab:



Explore preprocessing steps

When you've enabled featurization for your AutoML experiment, data guardrails will automatically be applied too.

The three data guardrails that are supported for classification models are:

- Class balancing detection
- Missing feature values imputation
- High cardinality feature detection

Each of these data guardrails will show one of three possible states:

- **Passed:** No problems were detected and no action is required
- **Done:** Changes were applied to your data. You should review the changes AutoML has made to your data
- **Alerted:** An issue was detected but couldn't be fixed. You should review the data to fix the issue

Track model training in notebooks with MLflow

Introduction to Mlflow - DEMO

MLflow is an open-source platform, designed to manage the complete machine learning lifecycle.

MLflow is an open-source library for tracking and managing your machine learning experiments. **MLflow Tracking** is a component of MLflow that logs everything about the model you're training, such as **parameters**, **metrics**, and **artifacts**.

Whether training in notebooks or scripts, MLflow allows you to easily compare different models you trained to find the best performing one, which you can then deploy.

Track metrics with MLflow with notebooks

To track parameters, metrics, and artifacts with MLflow:

1. Create an **experiment**.
2. Train a model and track metrics.
3. Review experiment runs with logged metrics to compare models.

There are two options to track machine learning experiments with MLflow:

- Enable autologging using mlflow.autolog()
- Use logging functions to track custom metrics using mlflow.log_*

Train and track models in notebooks

Create an MLflow experiment:

- You can create a MLflow experiment, which allows you to group runs.
- To create an experiment, run the command on the right in your notebook.

Log results with Mlflow:

- To track the model, you can enable automatic logging and use custom logging.

Python

```
import mlflow

mlflow.set_experiment(experiment_name="heart-condition-classifier")
```

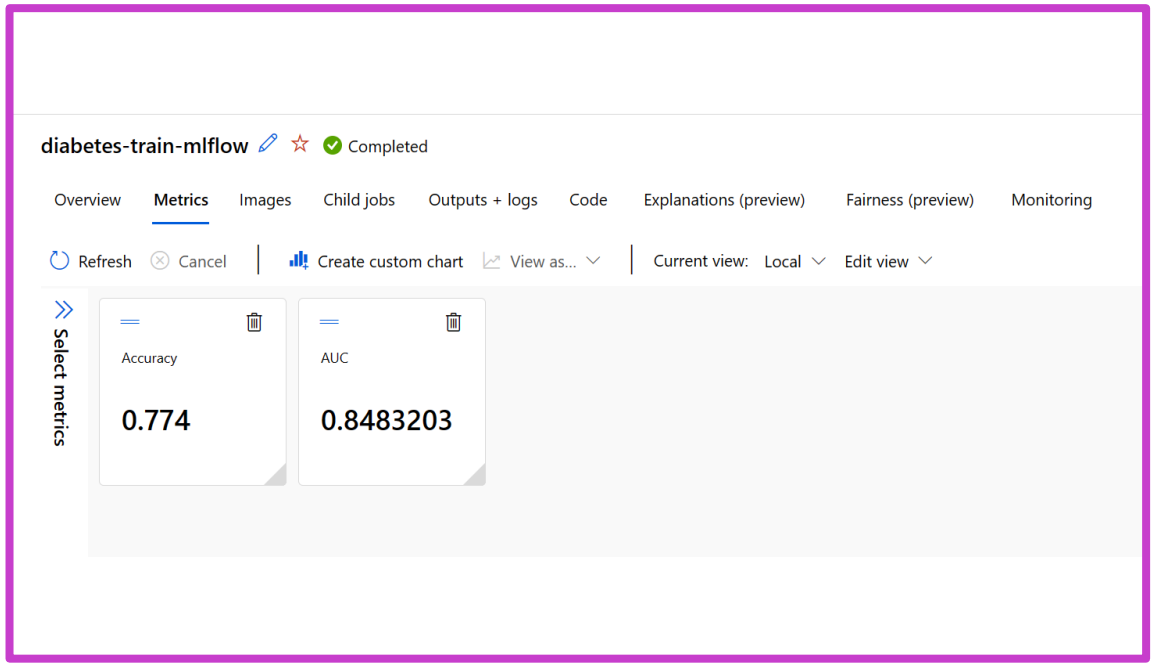
Log custom metrics with MLflow

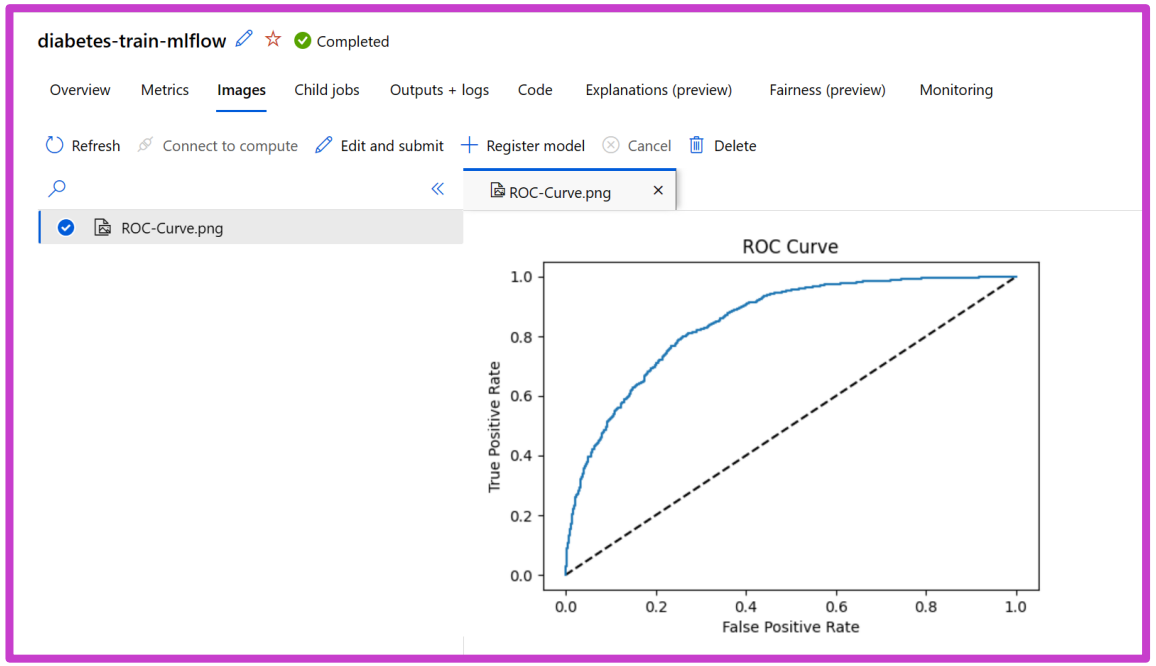
Depending on the type of value you want to log, use the relevant MLflow method to store the parameter, metric, or artifact with the experiment run:

- **mlflow.log_param():** Log single key-value parameter. Use this function for an input parameter you want to log.
- **mlflow.log_metric():** Log single key-value metric. Value must be a number. Use this function for any output you want to store with the run.
- **mlflow.log_artifact():** Log a file. Use this function for any plot you want to log, save as image file first.

View the metrics in the Azure Machine Learning studio

- Logged *parameters* and *metrics* will show in **Overview** and **Metrics** tabs.
- Plots that are logged as *artifacts* are shown under **Images**.
- Find other *artifacts* like model files under **Outputs + logs**.

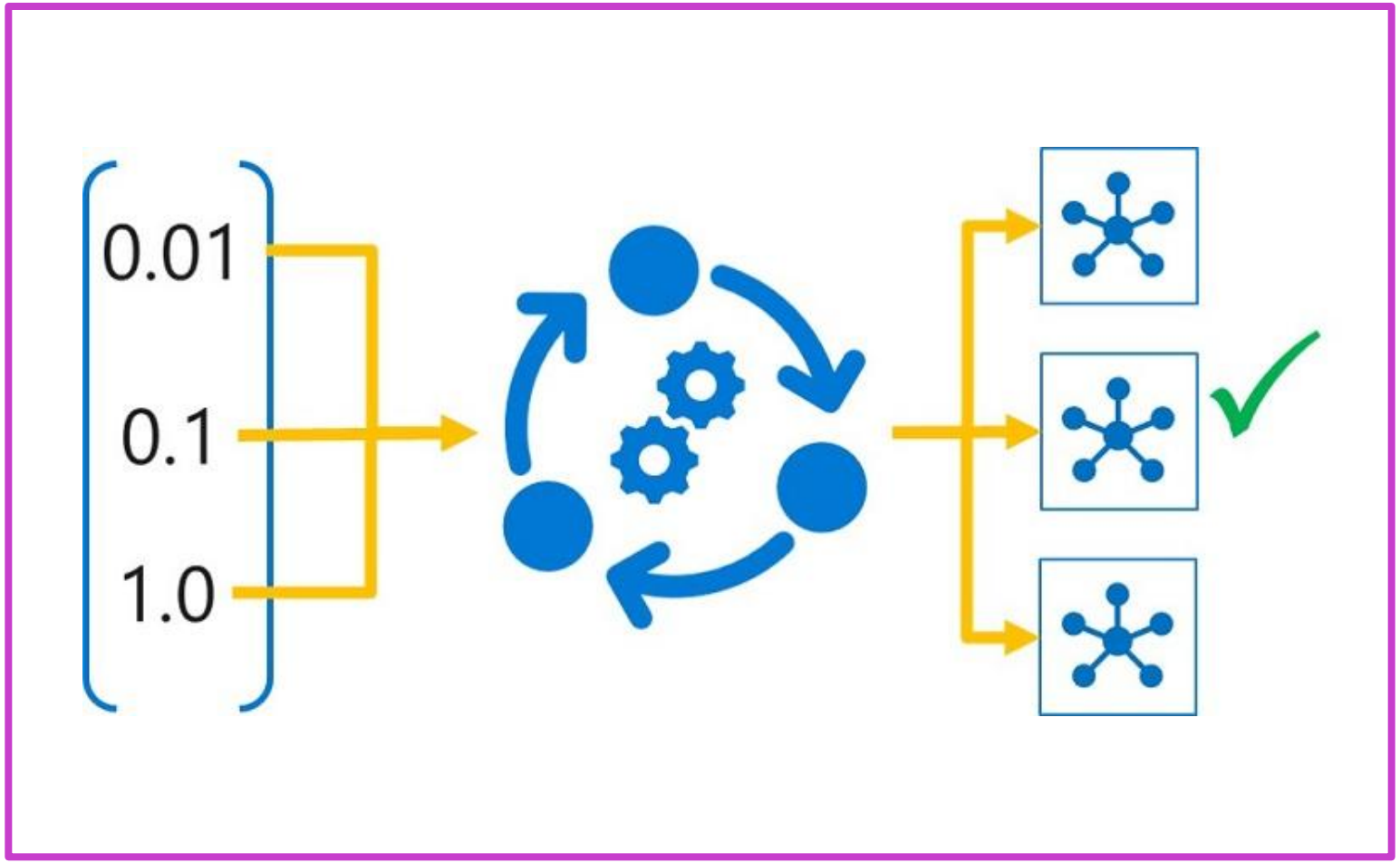




Perform hyperparameter tuning with Azure Machine Learning

Understand Hyperparameter Tuning

Hyperparameter tuning is accomplished by training the multiple models, using the same algorithm and training data but different hyperparameter values.



Use a sweep job for hyperparameter tuning

In Azure Machine Learning, you can tune hyperparameters by running a sweep job.

1. Create a training script for hyperparameter tuning
2. Configure and run a sweep job
3. Monitor and review sweep (child) jobs

Define a search space

The set of hyperparameter values tried during hyperparameter tuning is known as the **search space**. The definition of the range of possible values that can be chosen depends on the type of hyperparameter.

1. **Discrete hyperparameters:** Some hyperparameters require discrete values – In other words, you must select the value from a particular *finite* set of possibilities.

2. **Continuous hyperparameters:** Some hyperparameters are *continuous* – In other words you can use any value along a scale, resulting in an *infinite* number of possibilities

Defining a search space: To define a search space for hyperparameter tuning, create a dictionary with the appropriate parameter expression for each named hyperparameter

Configure a sampling method

The specific values used in a hyperparameter tuning run, or **sweep job**, depend on the type of **sampling** used.

There are three main sampling methods available in Azure Machine Learning:

1. **Grid sampling**: Tries every possible combination

2. **Random sampling**: Randomly chooses values from the search space
   - **Sobol**: Adds a seed to random sampling to make the results reproducible

3. **Bayesian sampling**: Chooses new values based on previous results

Configure early termination

- When you configure a sweep job in Azure Machine Learning, you can set a maximum number of trials.
- A more sophisticated approach may be to stop a sweep job when newer models don't produce significantly better results.
- To stop a sweep job based on the performance of the models, you can use an **early termination policy.**

When to use an early termination policy

Whether you want to use an early termination policy may depend on the search space and sampling method you're working with.

An early termination policy can be especially beneficial when working with continuous hyperparameters in your search space.

Configure an early termination policy - DEMO

There are two main parameters when you choose to use an early termination policy:

`evaluation_interval:`

Specifies at which interval you want the policy to be evaluated

`delay_evaluation:`

Specifies when to start evaluating the policy

New models may continue to perform only slightly better than previous models. To determine the extent to which a model should perform better than previous trials, there are three options for early termination:

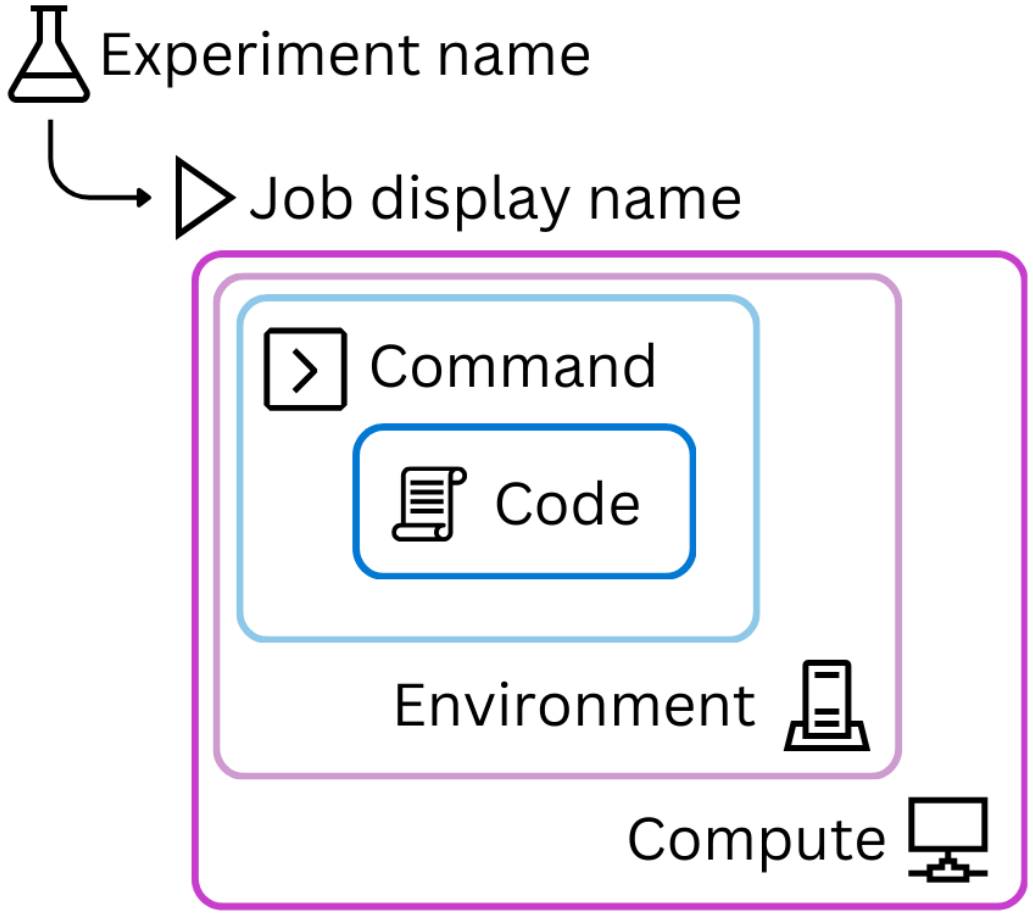
- **Bandit policy**
- **Median stopping policy**
- **Truncation selection policy**

Run a training script as a command job in Azure Machine Learning

Configure a command job - DEMO

When you have a **script** that you want to execute, you can run it as a **command job**.

Configure the command job by specifying the necessary job parameters and submit the job to the Azure Machine Learning workspace to run the script.



Use parameters in a command job

To run one script with different inputs, use **parameters**:

1. Import the **argparse** library in the script.
2. Use the ArgumentParser() method to define arguments for parameters.
3. Specify the parameters in the script, include name, type, and default value.
4. When running the script, specify the value for the defined parameters you want the script to use for this specific run.

```
python train.py --training_data diabetes.csv --reg_rate 0.01“
```

Track model training with MLflow in jobs

Track metrics with MLflow with scripts

To track parameters, metrics, and artifacts with MLflow:

1. Create a command job (automatically tracked as an **experiment**).
2. Train a model and track metrics.
3. Review experiment runs with logged metrics to compare models.

There are two options to track machine learning experiments with MLflow:

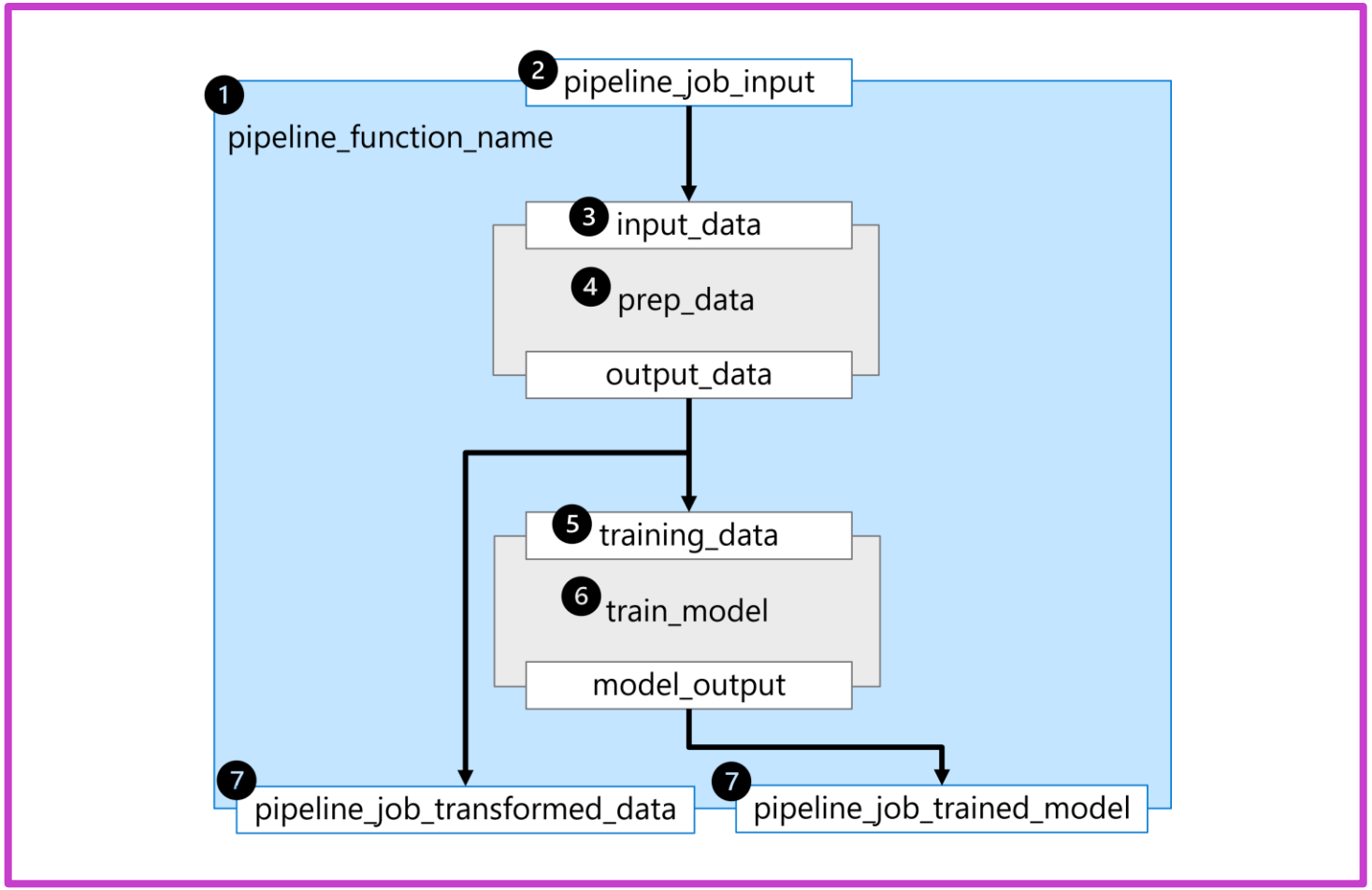
- Enable autologging using mlflow.autolog()
- Use logging functions to track custom metrics using mlflow.log_*

Run pipelines in Azure Machine Learning

Create a pipeline

In Azure Machine Learning, a **pipeline** is a workflow of machine learning tasks in which each task is defined as a **component**.

An Azure Machine Learning pipeline is defined in a YAML file. The YAML file includes the pipeline job name, inputs, outputs, and settings.



Run a pipeline job - DEMO

When you've built a component-based pipeline in Azure Machine Learning, you can run the workflow as a **pipeline job.**

Configure a pipeline job

A pipeline is defined in a YAML file, which you can also create using the `@pipeline()` function

Run a pipeline job

When you've configured the pipeline, you're ready to run the workflow as a pipeline job

Schedule a pipeline job

To schedule a pipeline job, you'll use the `JobSchedule` class to associate a schedule to a pipeline job

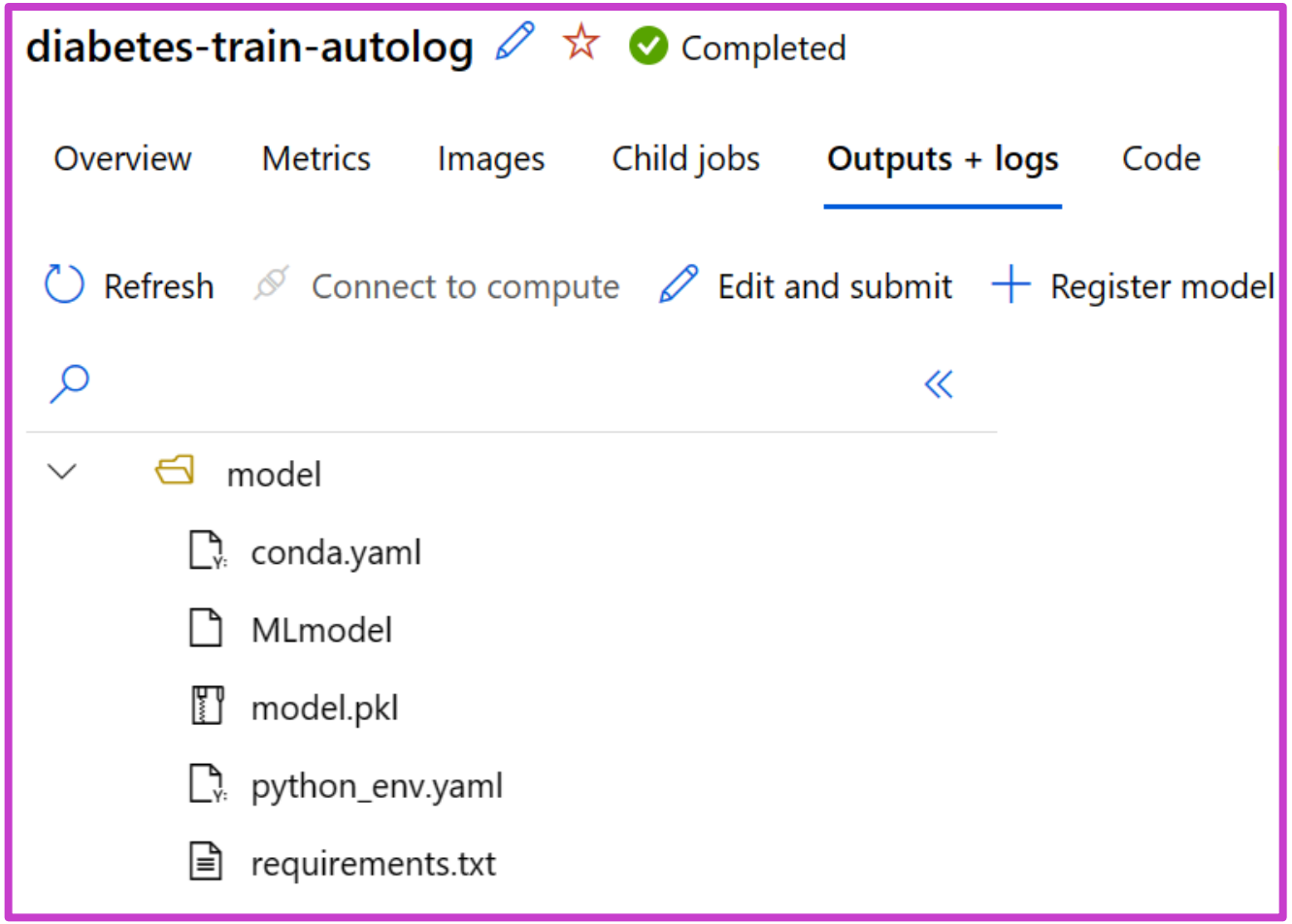
Register an MLflow model in Azure Machine Learning

Log a model with MLflow

MLflow allows you to log a model as an **artifact**, or as a **model.**

- When you log a model as an artifact, the model is treated as a file.
- When you log a model as a model, you're adding information to the registered model that enables you to use the model directly in pipelines or deployments.

When you log as a model, an MLmodel file is created in the output directory. The MLmodel file contains the model's metadata, which allows for model traceability.



Understand the MLmodel file format

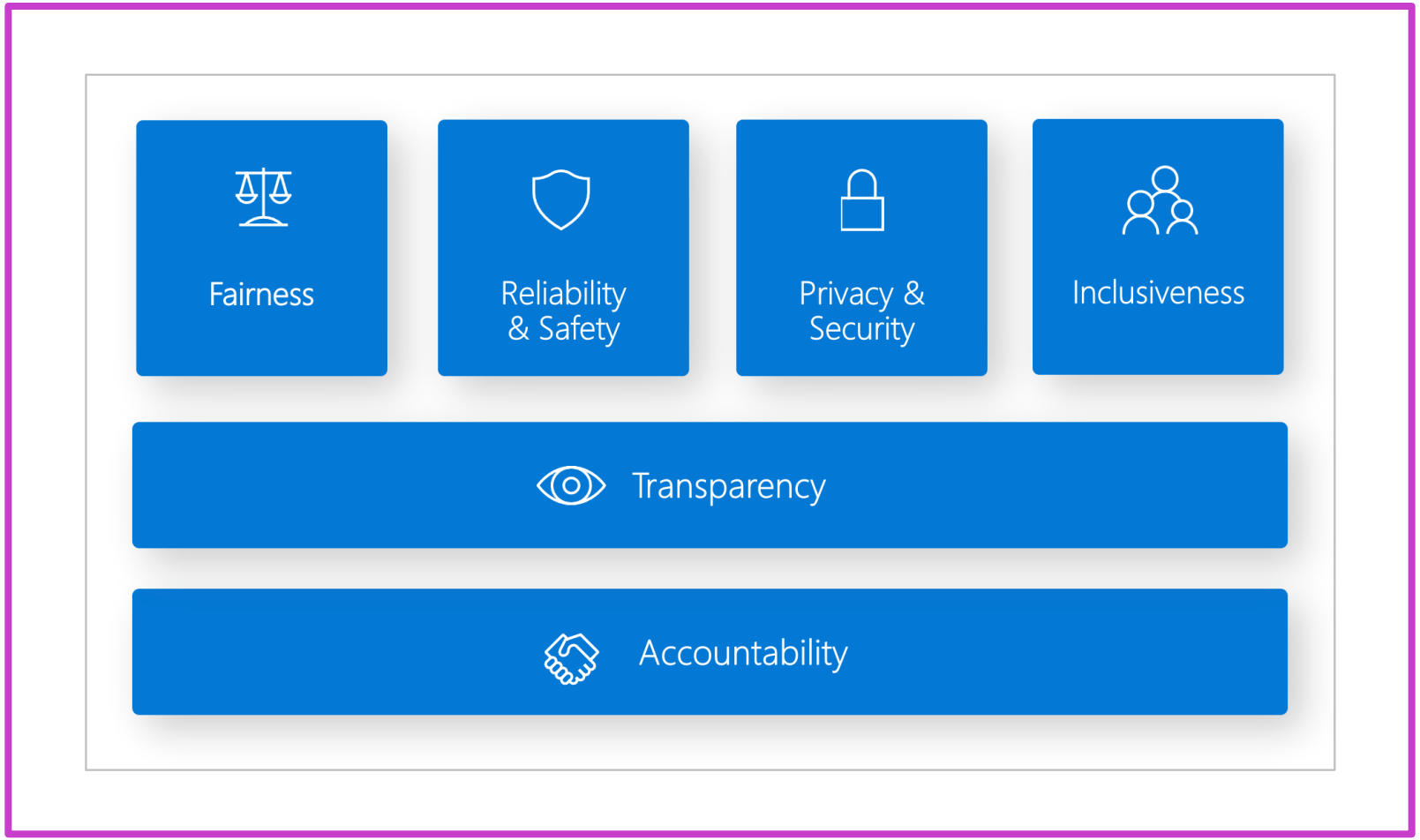
The MLmodel file may include:

- **artifact_path:** During the training job, the model is logged to this path.
- **flavor:** The machine learning library with which the model was created.
- **model_uuid:** The unique identifier of the registered model.
- **run_id:** The unique identifier of job run during which the model was created.
- **signature:** Specifies the schema of the model's inputs and outputs:
  - **inputs:** Valid input to the model. For example, a subset of the training dataset.
  - **outputs:** Valid model output. For example, model predictions for the input dataset.

Create and explore the Responsible AI dashboard

Understand Responsible Artificial Intelligence (AI)

Responsible Artificial Intelligence (Responsible AI) is an approach to developing, assessing, and deploying AI systems in a safe, trustworthy, and ethical way.



Reliability and safety: understand your datasets

Use data analysis to identify issues or overrepresentation and underrepresentation and to see how the data is clustered in the dataset.

Use data analysis when you need to:

- Explore your dataset statistics by selecting different filters to slice your data into different dimensions (also known as cohorts).
- Understand the distribution of your dataset across different cohorts and feature groups.
- Determine whether your findings related to fairness, error analysis, and causality (derived from other dashboard components) are a result of your dataset's distribution.
- Decide in which areas to collect more data to mitigate errors that come from representation issues, label noise, feature noise, label bias, and similar factors.

Reliability and safety: assess errors in your model

Identify erroneous cohorts of data. Explore whether your model underperforms for specific demographic groups.

- **Error tree:** Partitions the data into interpretable subgroups.
- **Error heatmap:** Slices the data based on one- or two-dimensional grids of input features.

Fairness: mitigate disparity

The model overview and fairness assessment evaluates the performance of your model and evaluates your model's group fairness issues.

- **Disparity in model performance:** Do different cohorts of the data perform differently when comparing selected performance metrics?
- **Disparity in selection rate:** Are there cohorts of the data that more often get a favorable prediction?

Create a Responsible AI dashboard

To create a Responsible AI (RAI) dashboard, you need to create a **pipeline** by using the built-in components. The pipeline should:

1. Start with the RAI Insights dashboard constructor.
2. Include one of the **RAI tool components**.
3. End with Gather RAI Insights dashboard to collect all insights into one dashboard.
4. *Optionally* you can also add the Gather RAI Insights score card at the end of your pipeline

Microsoft

Deploy and consume models with Azure Machine Learning

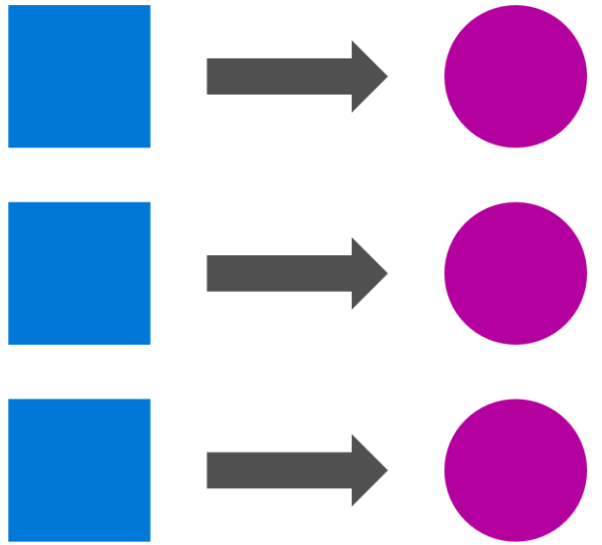
Deploy a model to a managed online endpoint

Explore managed online endpoints

An **endpoint** is an HTTPS endpoint to which you can send data, and which will return a response (almost) immediately.

Online endpoints are used to generate real-time predictions for individual data points.

With managed online endpoints, Azure Machine Learning manages all the underlying infrastructure.



Deploy a model to a managed online endpoint

When you deploy a model to a managed online endpoint, you have two options:

Deploy a MLflow model:

1. Register a MLflow model with a MLModel file.
2. Create deployment.
3. Deploy model to endpoint.

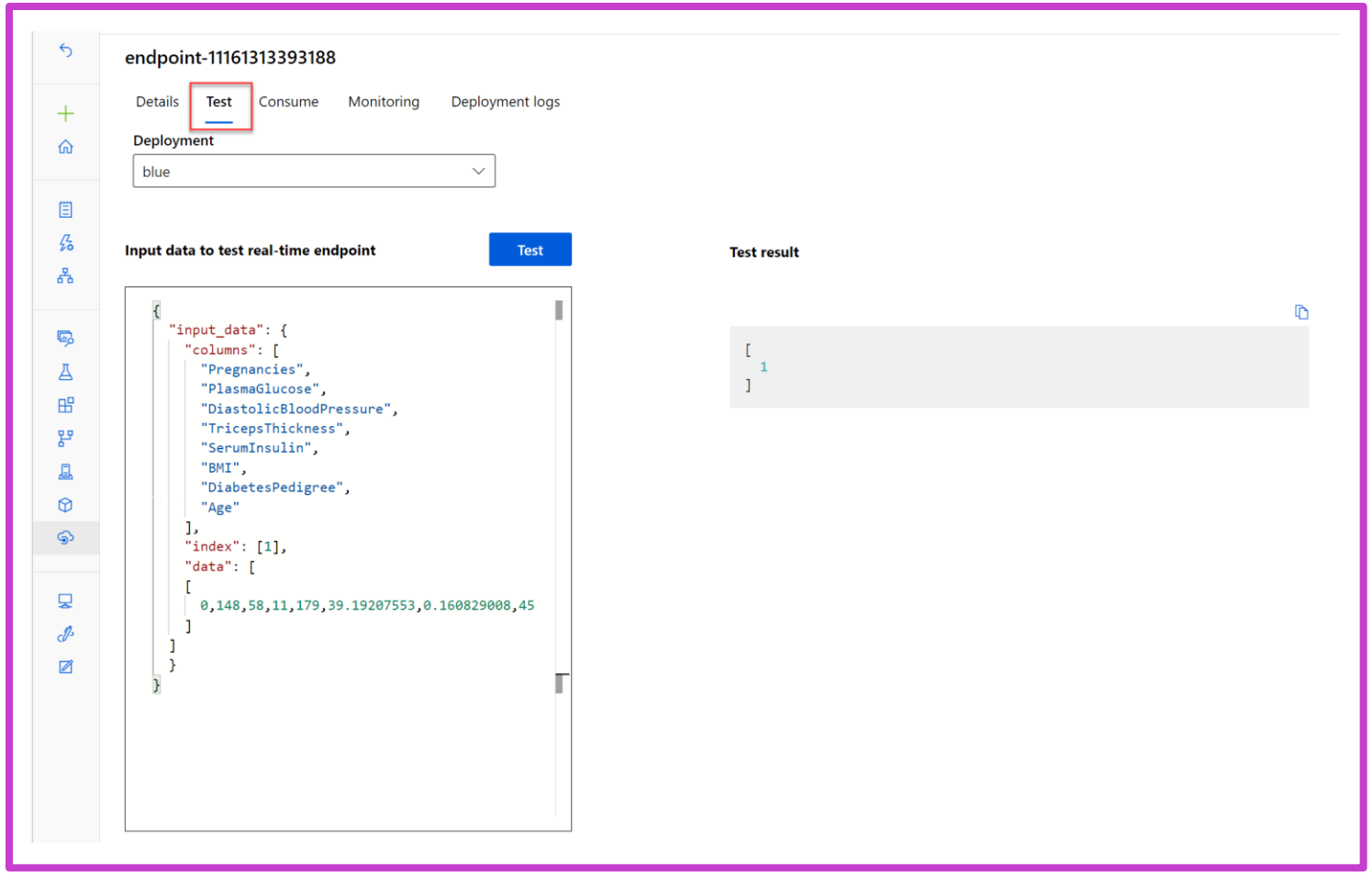
Deploy a (custom) model:

1. Register a model with the necessary model files (subject to model type).
2. Create scoring script.
3. Define execution environment.
4. Create deployment.
5. Deploy model to endpoint.

Test managed online endpoints - DEMO

Use the Azure Machine Learning studio to:

- List all endpoints.
- View an endpoint's details and deployment logs.
- Test the endpoint.



Deploy a model to a batch endpoint

Understand batch endpoints

1. To get batch predictions, you can deploy a model to an endpoint
2. An **endpoint** is an HTTPS endpoint that you can call to trigger a batch scoring job
3. The advantage of such an endpoint is that you can trigger the batch scoring job from another service, such as Azure Synapse Analytics or Azure Databricks
4. Whenever the endpoint is invoked, a batch scoring job is submitted to the Azure Machine Learning workspace

Create a batch endpoint

To deploy a model, you first have to create the batch endpoint.

To create a batch endpoint, you'll use the `BatchEndpoint` class. Batch endpoint names need to be unique within an Azure region.

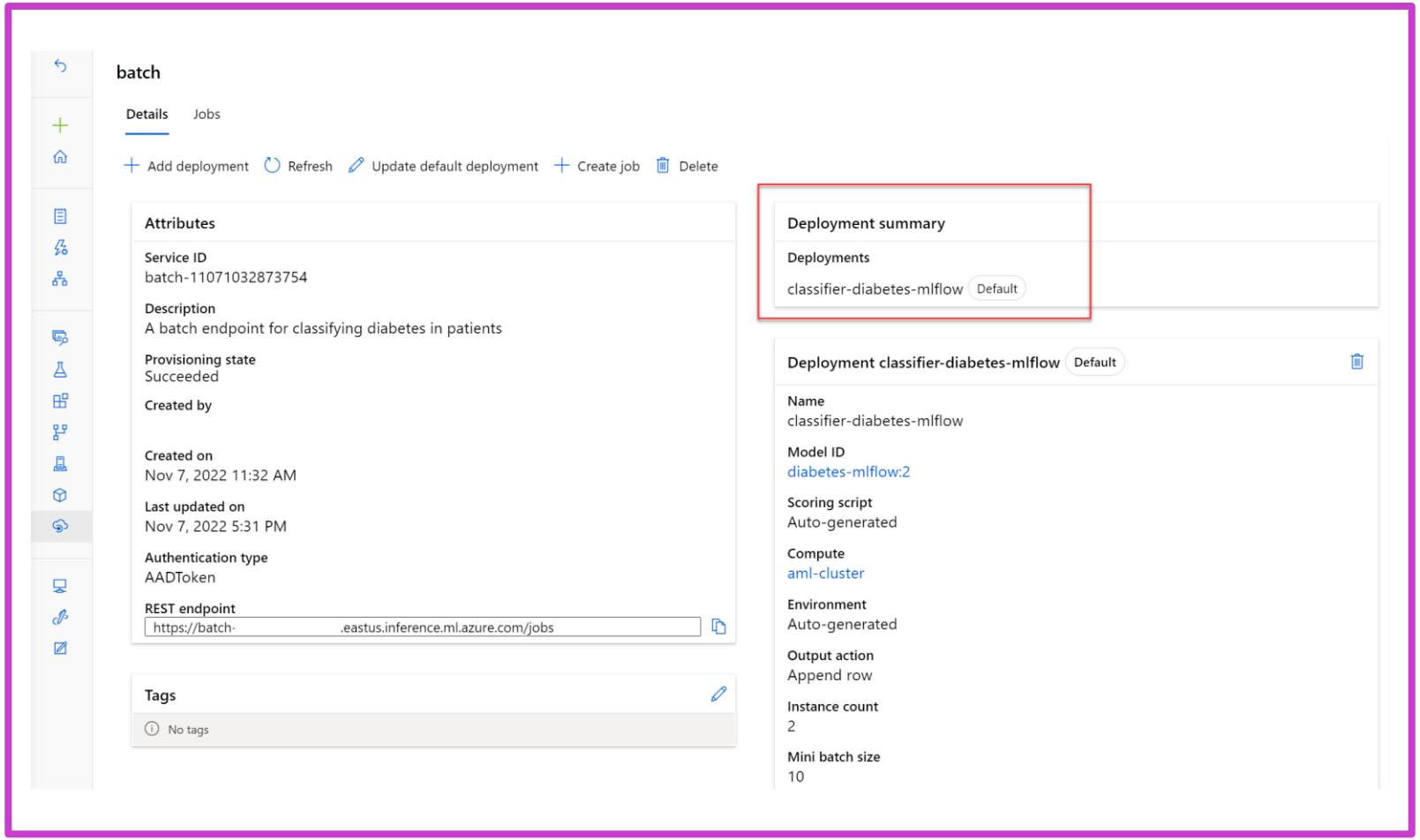
Python

```
# create a batch endpoint
endpoint = BatchEndpoint(
    name="endpoint-example",
    description="A batch endpoint",
)
 ml_client.batch_endpoints.begin_create_or_update(endpoint)
```

Deploy a model to a batch endpoint

You can deploy multiple models to a batch endpoint.

Whenever you call the batch endpoint, which triggers a batch scoring job, the **default deployment** will be used unless specified otherwise.



Run batch endpoints

Use compute clusters for batch deployments

- To process the new data in parallel batches, you need to provision a compute cluster with more than one maximum instances.
- To create a compute cluster, you can use the `AMLCompute` class.

Python

```
from azure.ai.ml.entities import AmlCompute

cpu_cluster = AmlCompute(
    name="aml-cluster",
    type="amlcompute",
    size="STANDARD_DS11_V2",
    min_instances=0,
    max_instances=4,
    idle_time_before_scale_down=120,
    tier="Dedicated",
)

cpu_cluster =
ml_client.compute.begin_create_or_update(cpu_clu
ster)
```

Deploy a custom model to a batch endpoint

If you want to deploy a model to a batch endpoint without using the MLflow model format, you need to create the scoring script and environment.

1. **Create the scoring script:** The scoring script is a file that reads the new data, loads the model, and performs the scoring.

2. **Create an environment:** Your deployment requires an execution environment in which to run the scoring script. Any dependency your code requires should be included in the environment.

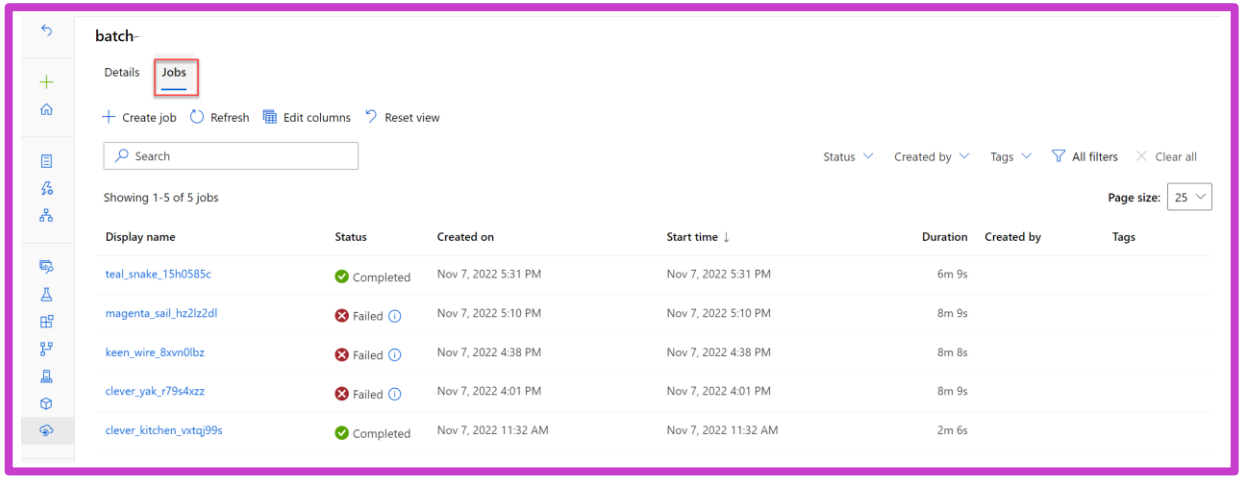
3. **Configure and create the deployment:** Finally, you can configure and create the deployment with the BatchDeployment class.

Invoke a batch endpoints

- To prepare data for batch predictions, you can register a folder as a data asset in the Azure Machine Learning workspace.
- You can then use the registered data asset as input when invoking the batch endpoint with the Python SDK.

Python

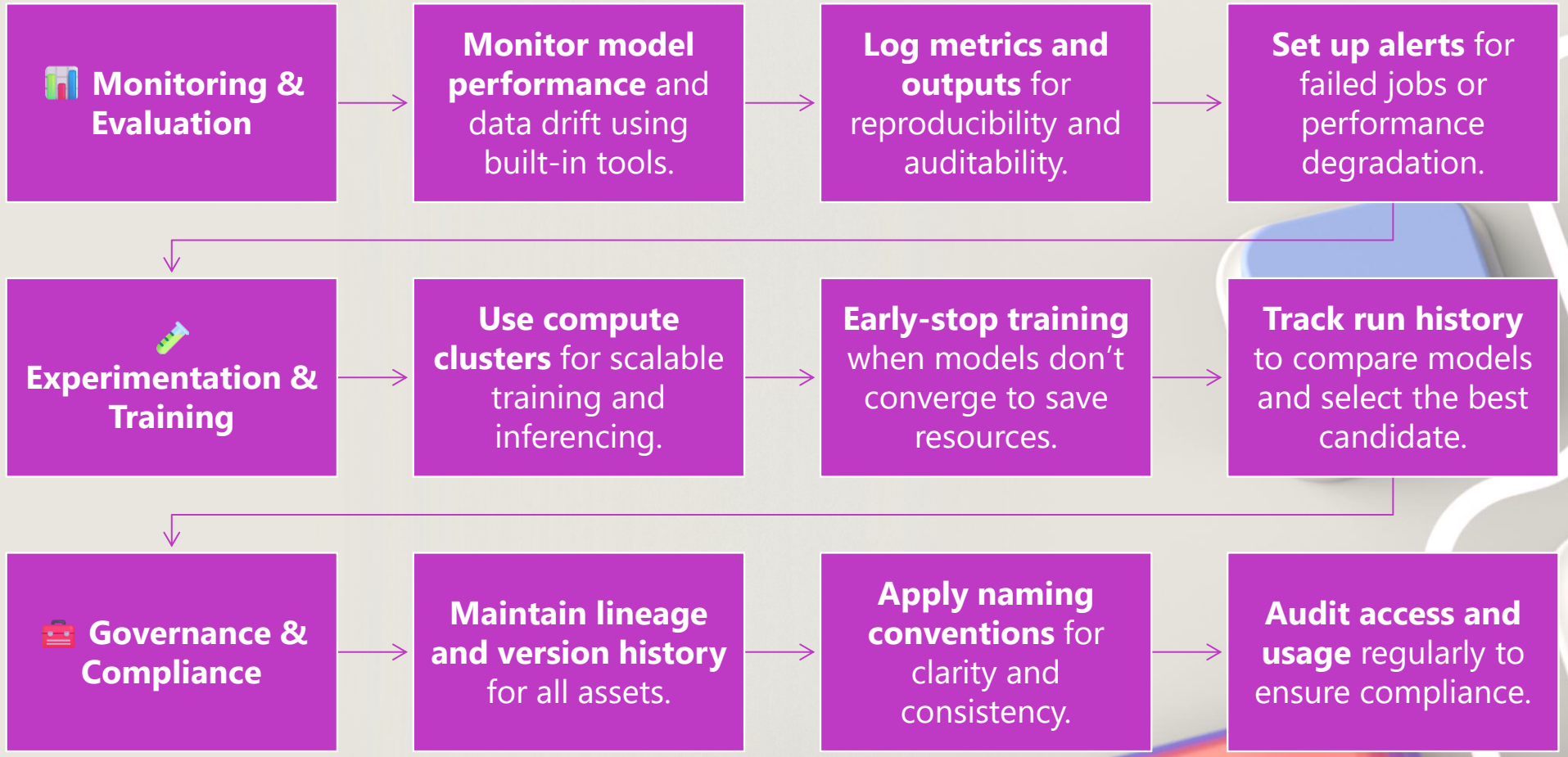
```
from azure.ai.ml import Input
from azure.ai.ml.constants import AssetTypes

input = Input(type=AssetTypes.URI_FOLDER,
path="azureml:new-data:1")

job = ml_client.batch_endpoints.invoke(
    endpoint_name=endpoint.name,
    input=input)
```



Best practice

Project Organization	- **Use separate workspaces** for dev, test, and production environments to isolate workloads.	- **Tag resources** (datasets, models, experiments) for easy tracking and governance.	- **Structure your codebase** with reusable templates and modular components.	### 🔐 Security & Access
- **Enable role-based access control (RBAC)** to manage permissions across teams.	- **Use managed identities** for secure access to Azure resources.	- **Store secrets** (like keys and credentials) in Azure Key Vault, not in code.	### 📦 Asset Management	- **Register datasets, models, and environments** to enable versioning and reuse.
- **Use ML registries** for cross-workspace sharing and collaboration.	- **Track experiments** with MLflow or Azure ML's built-in tracking tools.	### 🚀 Automation & CI/CD	- **Integrate GitHub Actions or Azure DevOps** to automate training and deployment.	- **Parameterize pipelines** to support flexible experimentation.
		- **Use pipeline templates** for consistent job orchestration.		

Best practice



Common mistakes in small projects

Setup & Configuration

- **Skipping workspace configuration**: Not setting up compute targets, datastores, or environments properly can lead to runtime errors.
- **Hardcoding secrets**: Storing credentials in scripts instead of using Azure Key Vault is risky and hard to maintain.

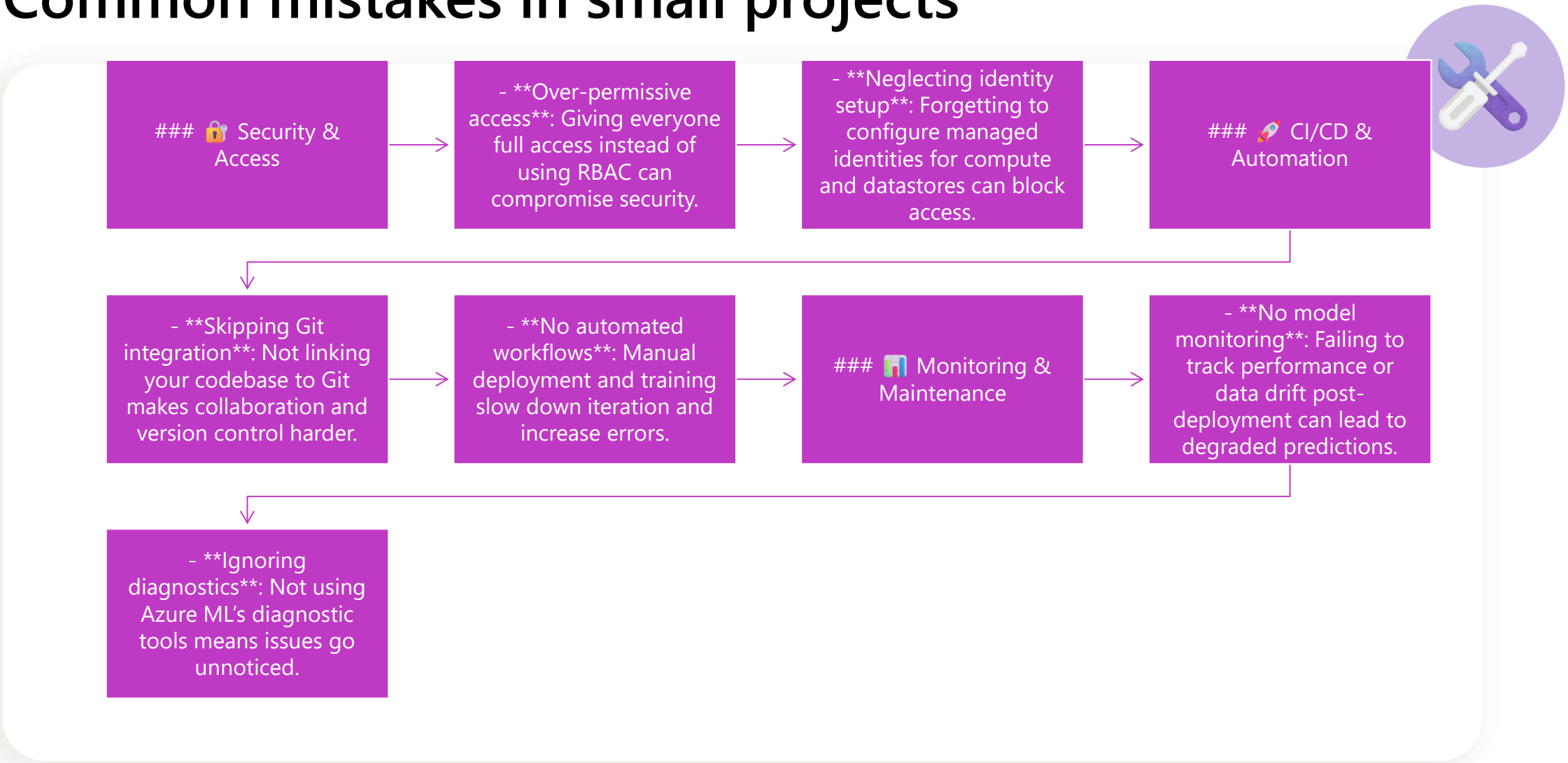
Asset Management

- **Not registering datasets or models**: This makes versioning and reuse difficult.
- **Ignoring ML registries**: Missing out on cross-workspace sharing and collaboration.

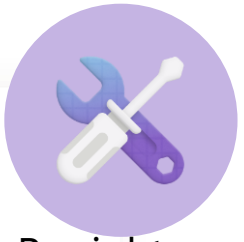
Experimentation

- **Running everything manually**: Not using pipelines or automation leads to inconsistent results.
- **No experiment tracking**: Without logging metrics and parameters, it’s hard to compare runs or reproduce results.

Common mistakes in small projects



When not to use



Simpler Projects

- **Lightweight models**: If you're building basic models with small datasets, local tools like scikit-learn or R might be faster and cheaper.

- **No need for cloud scale**: Azure ML shines with distributed training and deployment—if you don’t need that, it might be overkill.

Budget Constraints

- **Cost-sensitive environments**: Azure ML can get pricey with compute clusters, storage, and long training jobs. For small teams or personal projects, local or open-source alternatives may be more economical.

Preference for Full Code Control

- **Advanced customization**: Some users find Azure ML’s abstraction limits flexibility. If you need full control over algorithms, hyperparameters, or infrastructure, coding directly in Python or R might be better.

When not to use

Experimental or Research Work
- **Rapid prototyping**: Researchers often prefer lightweight environments like Jupyter notebooks or Colab for quick iteration.
- **Unsupported techniques**: Azure ML may lack built-in support for niche algorithms or cutting-edge research methods.

Team Skillset Mismatch
- **Non-developer data scientists**: Some teams find Azure ML's interface and DevOps integration challenging if they're not familiar with cloud tools.

Regulatory or Data Privacy Concerns
- **Sensitive data**: If your data can't leave a secure on-prem environment due to compliance, cloud-based ML might not be allowed.

Microsoft

Optimize language models for generative AI applications

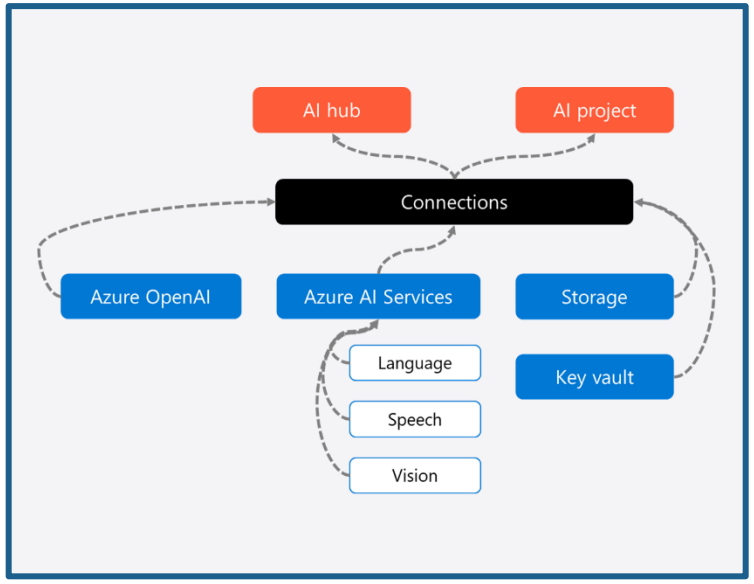
Agenda

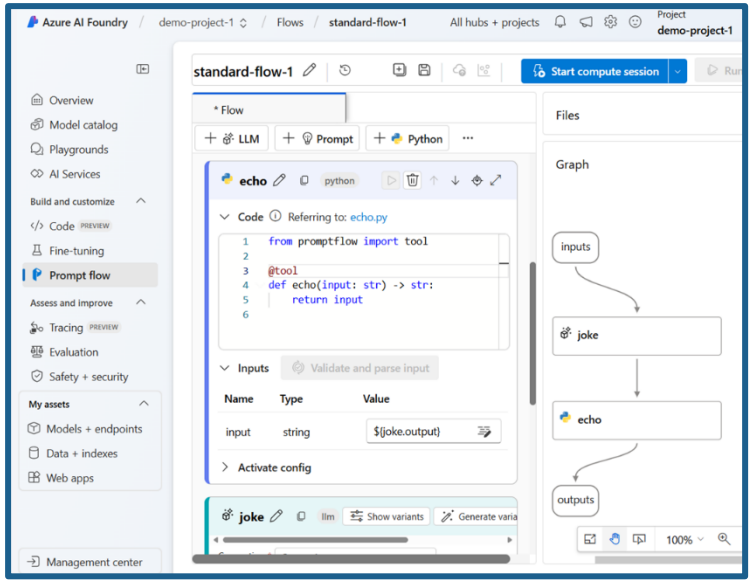
- Explore and deploy models from the model catalog
- Optimize model performance through prompt engineering
- Optimize through Retrieval Augmented Generation (RAG)
- Optimize through fine-tuning

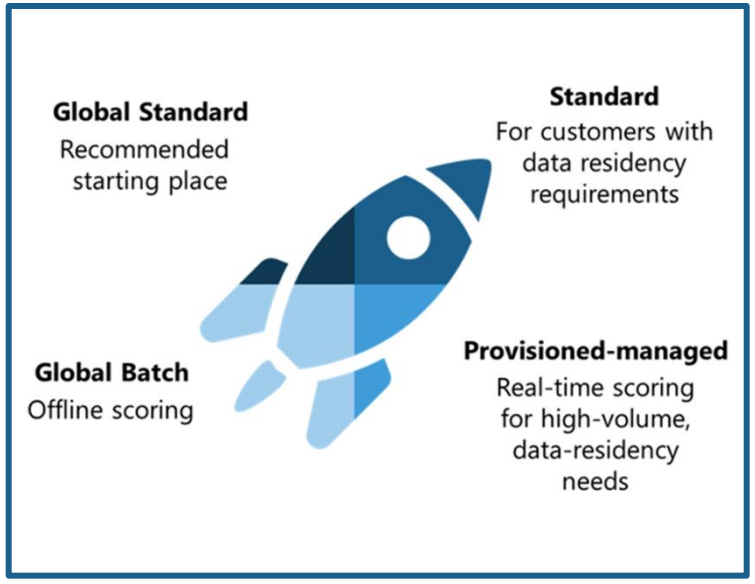
Introduction to Azure AI Foundry

- Select and deploy a language model from the model catalog
- Compare language models using benchmarks
- Test a deployed language model in the playground
- Select an optimization approach

Introducing Azure AI Foundry







Set up Azure AI Foundry

- Manage with **hubs**
- Build apps in a **project**
- Configure **connections**

Build and customize

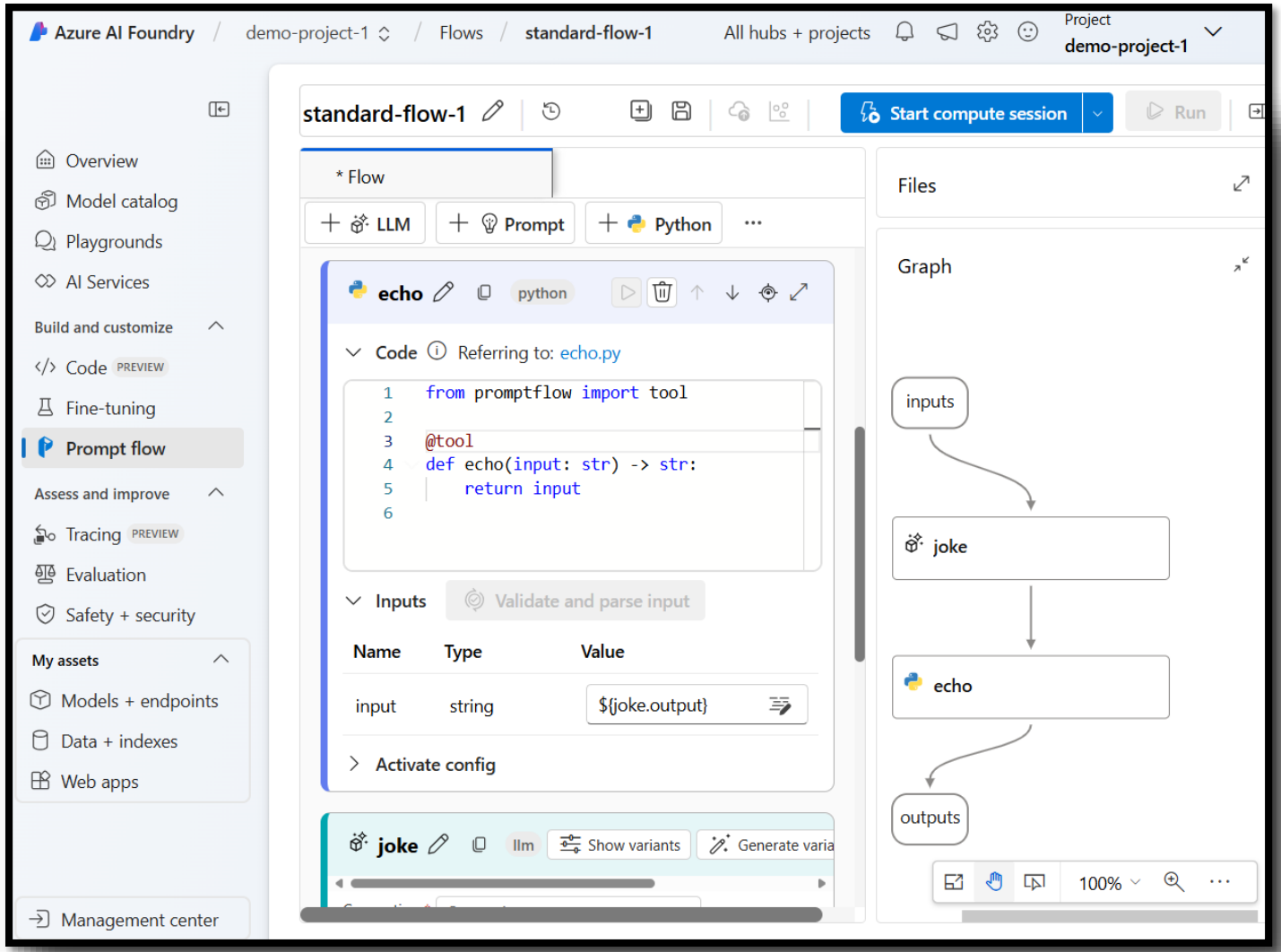
- Select a **model**
- Add your **data**
- Orchestrate **workflows**

Deploy a model

- Select the **deployment type**
- Customize the **tokens per minute rate limit**
- Consider **regional** availability

Build and customize

- ☑ Select a **model**
- ☑ Add your **data**
- ☑ Orchestrate **workflows**

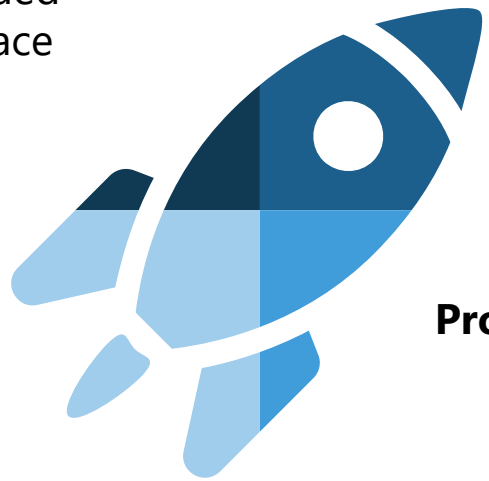


Deploy a model

- Select the **deployment type**
- Customize the **tokens per minute rate limit**
- Consider **regional** availability

Global Standard
Recommended starting place

Standard
For customers with data residency requirements

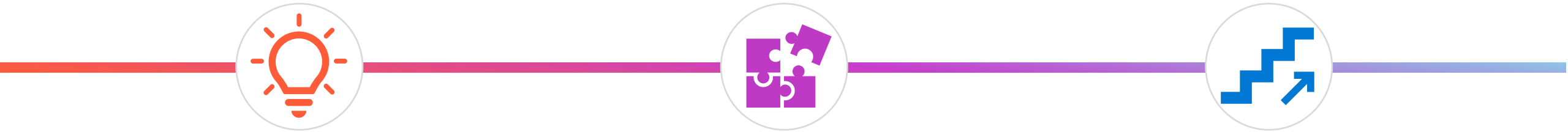


Global Batch
Offline scoring

Provisioned-managed
Real-time scoring for high-volume, data-residency needs

Explore and deploy models from the model catalog

Select a model using a structured approach



Can AI solve my use case?

- Define use cases
- Select a model
- Establish feasibility
- Build prototype

How do I select the "right" model for my use case?

- Select a model
- Prompt engineering
- Retrieval Augmented Generation
- Fine-tuning

Can I scale for real-world workloads?

- Content filtering
- Security and data privacy
- Capacity and cost trade-offs
- Monitoring

Build a prototype to solve your use case

LLMs vs SLMs

Large Language Models:

- GPT-4
- Mistral Large
- Llama3 70b
- Llama 405B
- Command R, R+

Small Language Models:

- Phi3
- Mistral OSS models
- Llama3 8b

Modalities, tasks and tools

Multi-modal:

- GPT-4o, Phi3-vision

Image generation:

- Dalle3, Stability AI

Embedding models:

- Ada, Cohere

Function calling & JSON support

Regional and domain specific

- Core42 JAIS Arabic language LLM
- Mistral Large is focused on European languages
- Nixtla TimeGEN-1 - Timeseries forecasting

Open and propriety

Premium models first on Azure:

- OpenAI, Mistral Large, Cohere Command R+

100s of Open models from HuggingFace

Open models from Meta, Databricks, Snowflake, Nvidia

Assess the model performance

Evaluate your model performance at different phases, using a variety of evaluation approaches:

Model benchmarks

- Compare publicly available metrics across models and dataset

Manual evaluations

- Rate your model responses

Traditional machine learning metrics

- Measure ratio of number of shared words between generated and ground truth answers

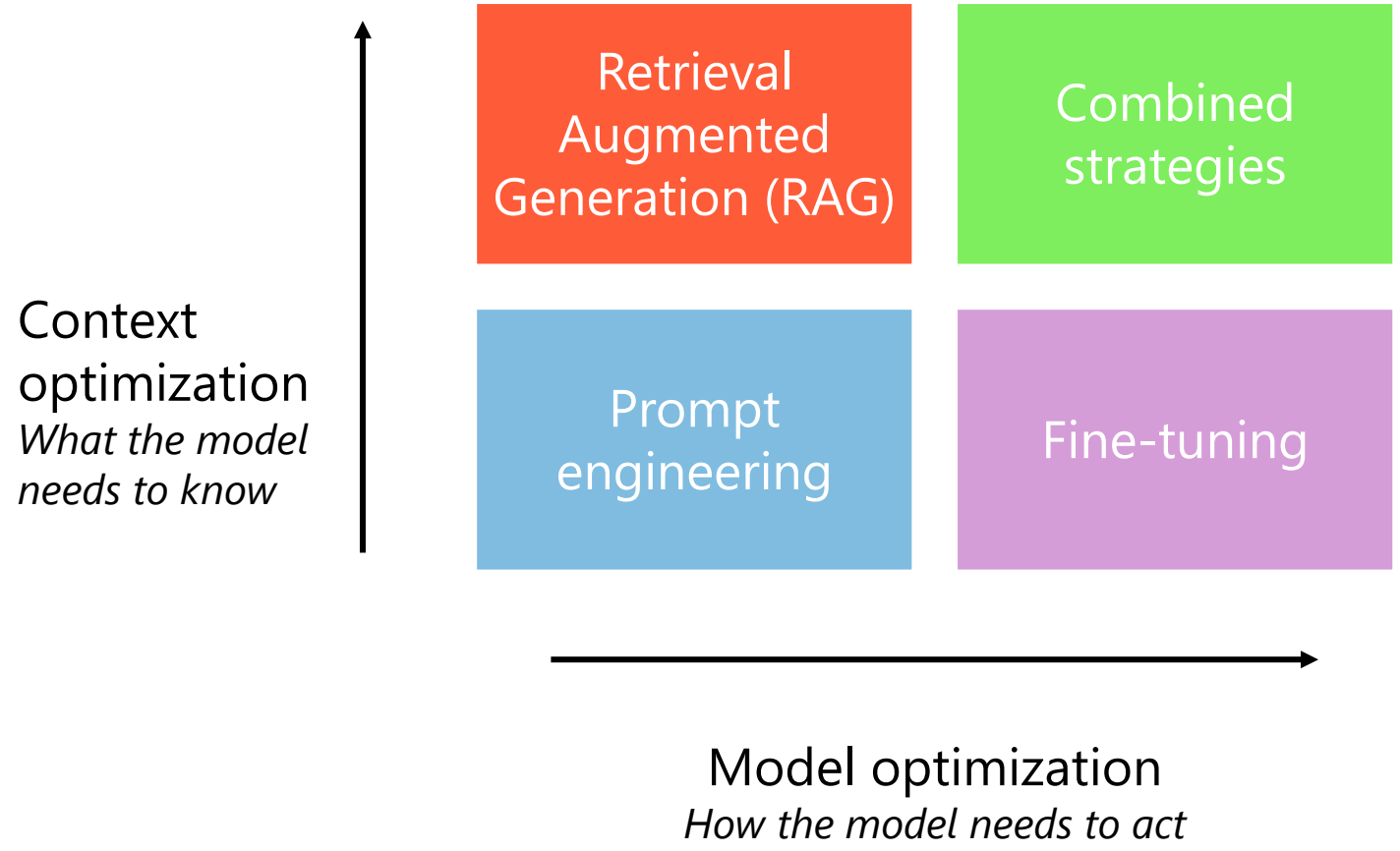
AI-assisted metrics

- Risk and safety metrics
- Generation quality metrics

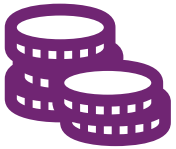
Optimize for precision after selecting a model

Optimization strategy will depend on your requirements:

- **Optimize for context** when the model lacks contextual knowledge, and you want to **maximize response accuracy**.
- **Optimize the model** when you want to improve the response format, style, or speech by **maximizing consistency of behavior**.



Optimize performance to prepare for scale



By **cost**: Limits and tradeoffs

By **metrics**: Quality and safety

By **trust**: Curated and compliant

By **size**: SLM (edge) and LLM (cloud)

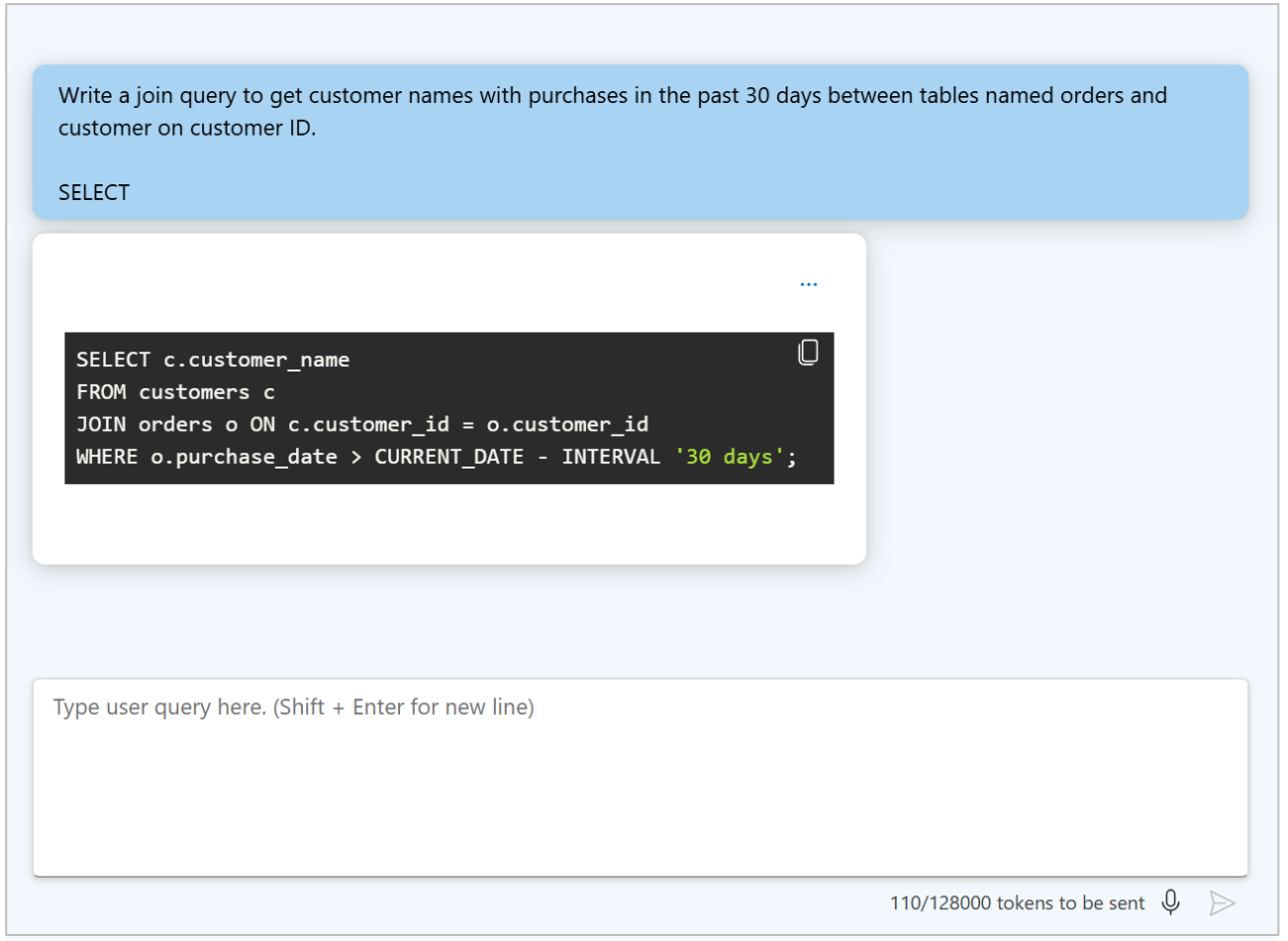
Optimize model performance through prompt engineering

- Test prompts with manual evaluation
- Define and track prompt variants
- Create prompt templates
- Define chaining logic with the prompt flow SDK
- Use tracing to evaluate your flow

Apply prompt engineering

To improve the model's output as a user, you can apply **prompt engineering**:

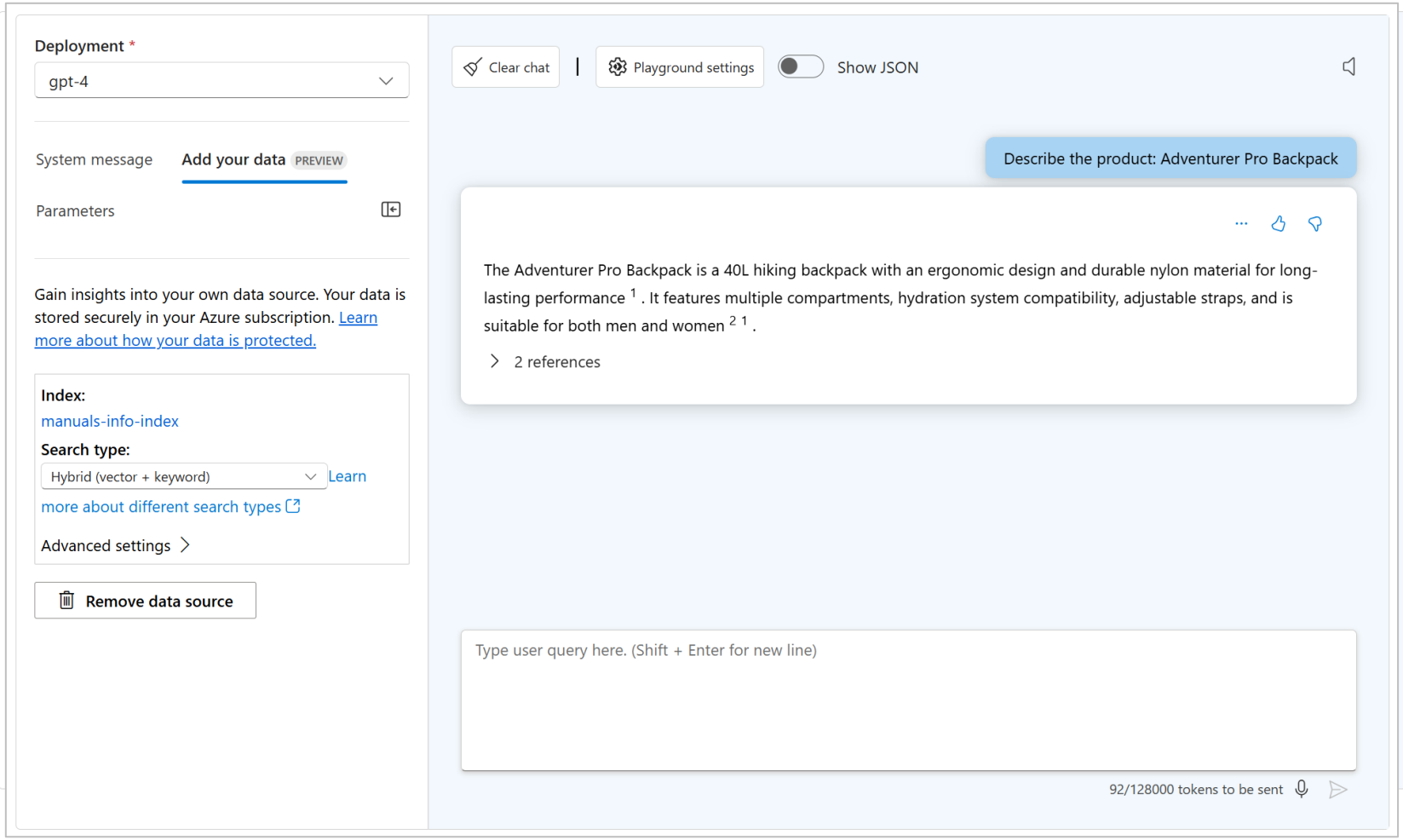
- Provide clear instructions
- Format your instructions
- Use cues



Update the system message

To improve the model's output as a developer, you can update the **system message**:

- Use one/few shot(s)
- Use chain-of-thought
- Add context

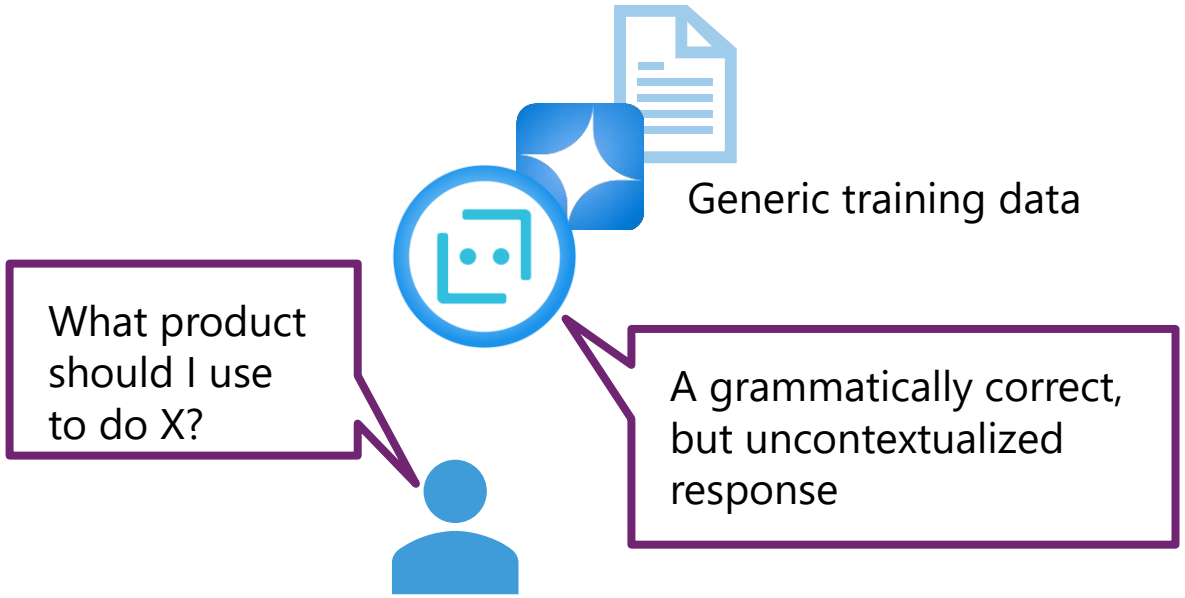


Grounding a copilot with your own data

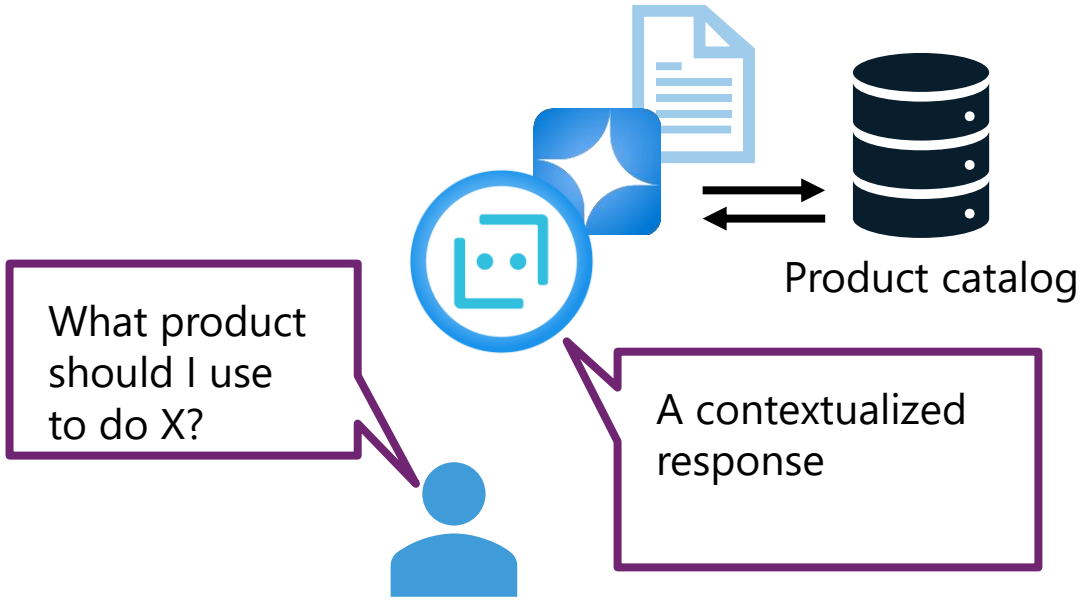
Language models create coherent answers to questions, but what are those answers based on?

Grounding provides specific context to the model to provide accurate, relevant responses.

Ungrounded



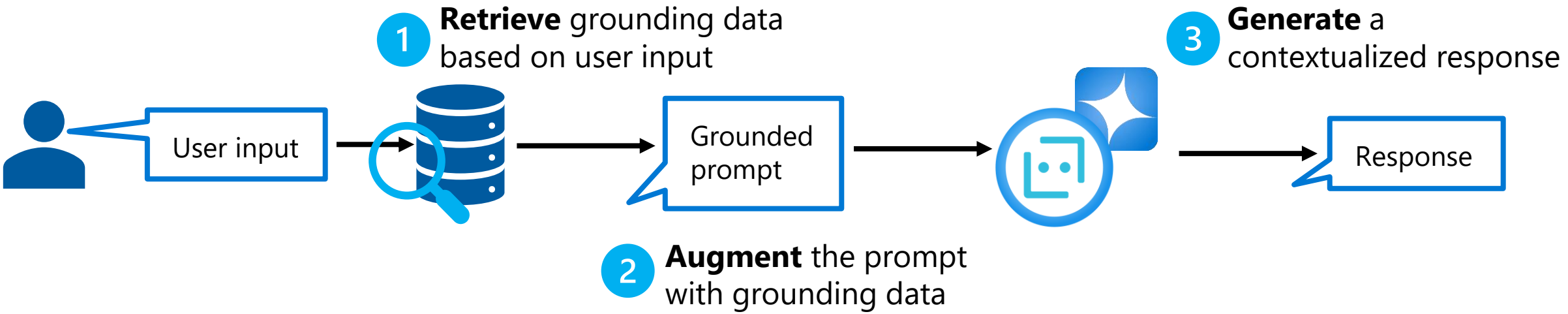
Grounded



Retrieval Augmented Generation (RAG)

RAG gathers relevant data to include with a prompt for the language model to use for grounding context.

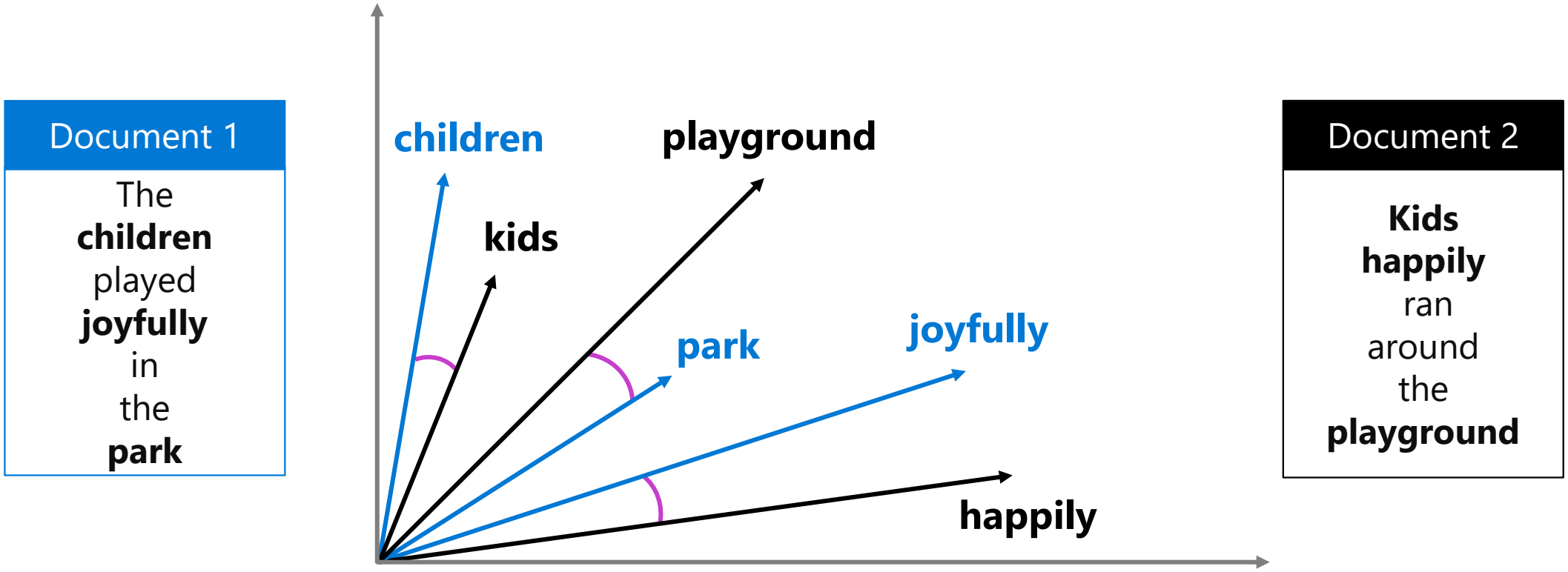
A **RAG pattern** is an architectural design for RAG by including retrieved relevant data in a prompt:



Retrieval Augmented Generation (RAG)

Data is stored as embeddings in a **vector-based search index**.

Embeddings allow better data retrieval for *semantic similarities* in the search results.

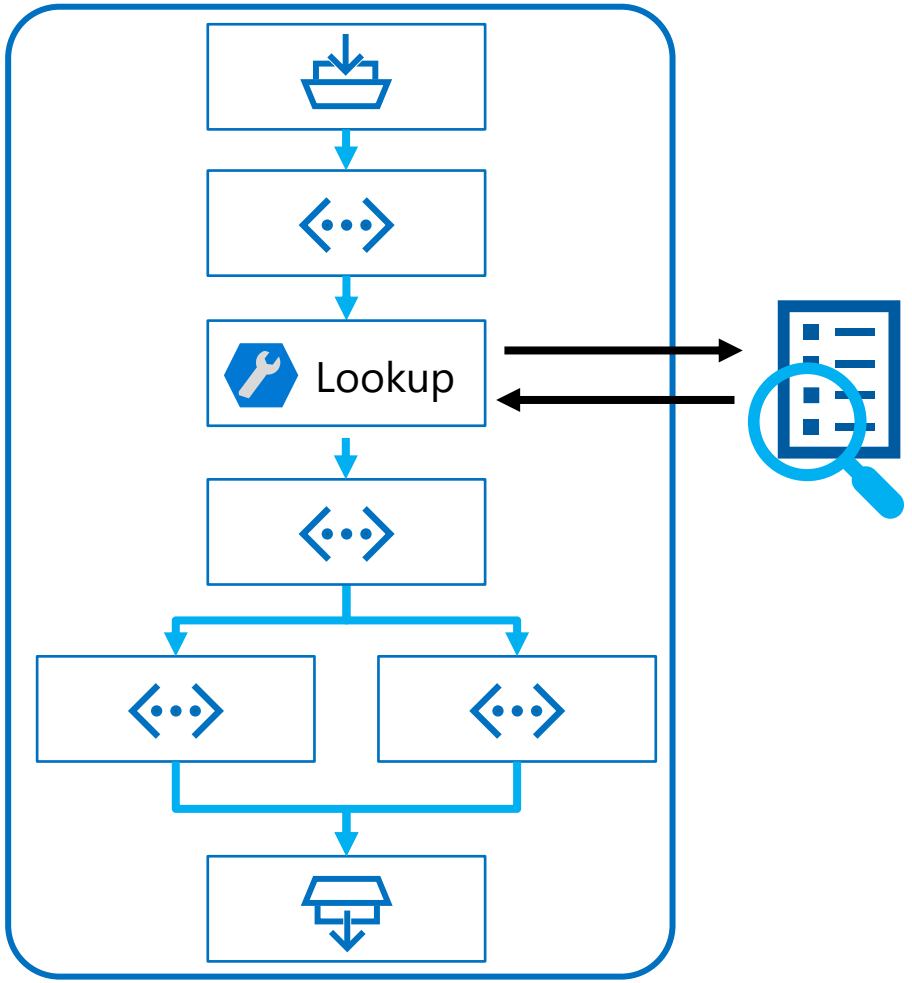


Add your data in a prompt flow

To ground your language model's responses, add your own data source to retrieve relevant context:

1. Add your **data** to an Azure AI project.
2. Index your data with **Azure AI Search**.
3. Query your indexed data in a prompt flow with the **Index Lookup** tool.
4. Reference the retrieved **context** in a prompt.
5. Send the prompt with context to an **LLM**.

The generated response will be grounded by the retrieved context.



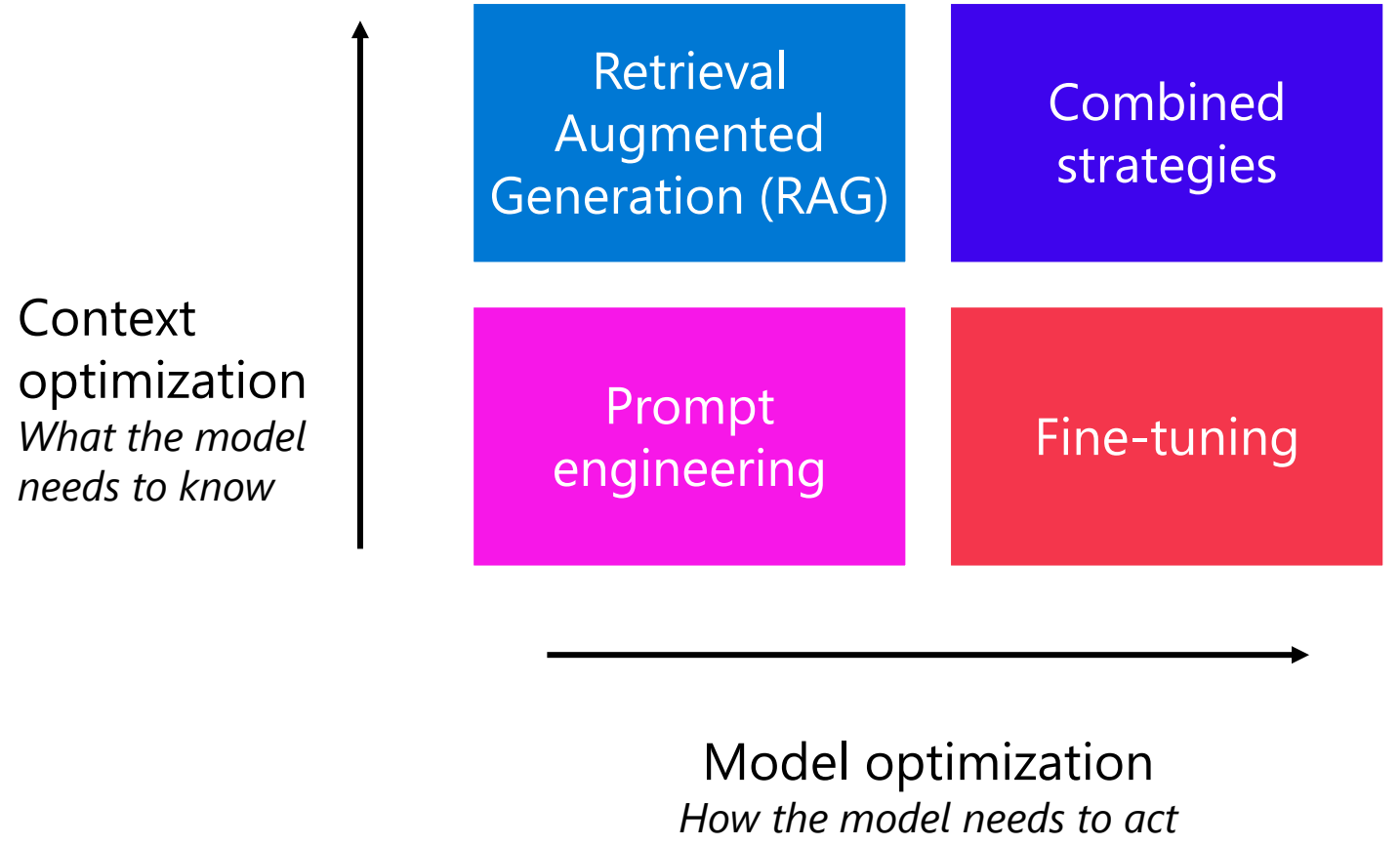
Optimize through fine-tuning

- Prepare data for fine-tuning
- Select an appropriate base model
- Run a fine-tuning job
- Evaluate your fine-tuned model

Understand when to fine-tune a model

Fine-tuning will improve the model when:

- You have a clear specific use case.
- You need the model's output to be in a specific customized style.
- You want to have a more consistent performance by providing more few shots than can fit in your prompt.



Evaluate the performance of your models in the Azure AI Foundry

Understand model benchmarks

Datasets are publicly available to calculate individual benchmarks and compare across models. Some commonly used benchmarks are:

Accuracy:
Compares model generated text with correct answer according to the dataset. Result is one if generated text matches the answer exactly, and zero otherwise.

Coherence:
Measures whether the model output flows smoothly, reads naturally, and resembles human-like language

Fluency:
Assesses how well the generated text adheres to grammatical rules, syntactic structures, and appropriate usage of vocabulary, resulting in linguistically correct and natural-sounding responses.

GPT Similarity:
Quantifies the semantic similarity between a ground truth sentence (or document) and the prediction sentence generated by an AI model