Homework 7 - A Giza tutorial

Install the Giza -cli using the instructions in Homework 5

In addition use pip to install the following packages if you don't already have them.

`pip install giza-actions`

`pip install numpy`

`pip install scikit-learn`

`pip install skl2onnx`

Create an account on giza

```
giza users create # Create a user
giza users login # Login to your account
giza users create-api-key # Create an API key.
```

We will now follow the tutorial [here]() to create and train a simple model.

For the code below , you can start a python shell to run the code, or if you prefer run it from a file.

Create and train a simple model with Scikit-Learn

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Generate some dummy data
X = np.random.rand(100, 1) * 10  # 100 samples, 1 feature
y = 2 * X + 1 + np.random.randn(100, 1) * 2  # y = 2x + 1 + noise

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

Now that our model is trained we need to convert this to ONNX format so that it can be used with Giza.

```python
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

# Define the initial types for the ONNX model
initial_type = [('float_input', FloatTensorType([None, X_train.shape[1]]))]

# Convert the scikit-learn model to ONNX
onnx_model = convert_sklearn(model, initial_types=initial_type)

# Save the ONNX model to a file
with open("linear_regression.onnx", "wb") as f:
    f.write(onnx_model.SerializeToString())
```

Once this completes you should see the onnx file : `linear_regression.onnx` in your filesystem

We can now use the giza-cli to convert this to Cairo code, which is then provable.

`giza transpile linear_regression.onnx --output-path verifiable_lr`

This creates a verifiable_lr directory containing a Cairo project, you can find the Cairo code in `verifiable_lr\inference\src\lib.cairo`
The main function will look like this

```
fn main(node_float_input: Tensor<FP16x16>) -
> Tensor<FP16x16> {
let node_variable =
    ml::LinearRegressorTrait::predict(
        ml::LinearRegressor {
            coefficients:
get_linearregressor_coefficients(),
            intercepts:
Option::Some(get_linearregressor_intercepts(
)),
            target: 1,
            post_transform:
ml::POST_TRANSFORM::NONE
        }
        , node_float_input)
    ;

        node_variable
    }
```

The output from the transpile command will give some details that you need to keep

```
[giza][2024-03-19 10:43:11.586] Model
Created with id -> 447! ✅
[giza][2024-03-19 10:43:12.093] Version
Created with id -> 1! ✅
```

You need the Model ID and the Version.

We now deploy an endpoint, using giza-cli

`giza endpoints deploy --model-id 447 --version-id 1`

You should use the model and version ids from the previous step.

The output will give us the endpoint id

`[giza][2024-03-19 10:51:48.557] Endpoint created with id -> 109 ✅`

You should make a note of this as we need it later.

We are now ready to use our actions workspace. To find the URL run
`giza workspaces get`
This will give an output similar to

```
[giza][2024-03-19 11:09:38.610] ✅ Workspace
URL: https://actions-server-raphael-doukhan-
```

```
dblzzhtf5q-ew.a.run.app ✅
{
  "url": "https://actions-server-raphael-doukhan-dblzzhtf5q-ew.a.run.app",
  "status": "COMPLETED"
}
```

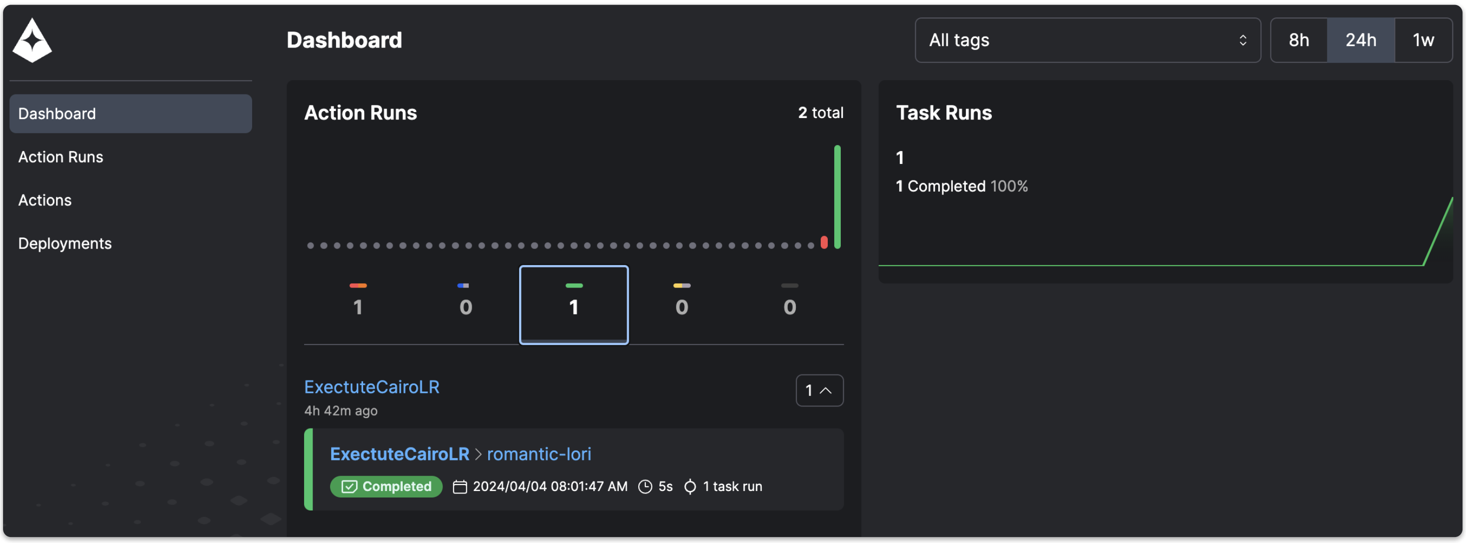
We can now write a script to automate our workflow using the `@task` decorator and `@action` decorator.
You need to substitute the MODEL_ID and VERSION_ID variables with the ones that apply to your model.

```python
from giza_actions.model import GizaModel
from giza_actions.action import action
from giza_actions.task import task
import numpy as np

MODEL_ID = 447  # Update with your model ID
VERSION_ID = 1  # Update with your version ID


@task(name="PredictLRModel")
```

```
def prediction(input, model_id, version_id):
    model = GizaModel(id=model_id,
version=version_id)

    (result, proof_id) = model.predict(
        input_feed={'input': input},
verifiable=True
    )

    return result, proof_id


@action(name="ExectuteCairoLR",
log_prints=True)
def execution():
    # The input data type should match the
model's expected input
    input =
np.array([[5.5]]).astype(np.float32)

    (result, proof_id) = prediction(input,
MODEL_ID, VERSION_ID)

    print(
        f"Predicted value for input
{input.flatten()[0]} is {result[0].flatten()
[0]}")
```

```
    return result, proof_id


execution()
```

You should also see the results in your dashboard



You will get an output finishing with a proof id

```
View at https://actions-server-raphael-doukhan-dblzzhtf5q-ew.a.run.app/flow-runs/flow-run/637bd0e0-d7e8-4d89-8c07-a266e6c280ce
...
11:34:08.194 | INFO    | Task run 'PredictLRModel-0' - Finished in state Completed()
11:34:08.197 | INFO    | Action run 'proud-perch' - Predicted value for input 5.5 is 12.20851135253906 2
11:34:08.313 | INFO    | Action run 'proud-
```

```
perch' - Finished in state Completed()
(array([[12.20851135]]),
'"3a15bca06d1f4788b36c1c54fa71ba07"')
```

Here the proof id is `3a15bca06d1f4788b36c1c54fa71ba07`

Before we can download the proof, we need to check that it is ready with

```
giza endpoints get-proof --endpoint-id 109 --proof-id 3a15bca06d1f4788b36c1c54fa71ba07
```

When the proof is ready you can download it

```
giza endpoints download-proof --endpoint-id 109 --proof-id 3a15bca06d1f4788b36c1c54fa71ba07 --output-path zklr.proof

>>>>
[giza][2024-03-19 11:55:49.713] Getting proof from endpoint 109 ✅
[giza][2024-03-19 11:55:50.493] Proof downloaded to zklr.proof ✅
```

Congratulations, you've created a simple model, turned it into provable code and produced a proof of the inference step.