GizaModel

Welcome to the step-by-step guide for usingGizaModel within the Actions SDK, a key component of AI Actions.GizaModel is designed to streamline the integration and management of machine learning (ML) models in blockchain workflows using ZKML. This guide will walk you through the process of model building, transpilation and deployment ofGizaModel in your projects.

Running an unverifiable inference involves executing the ONNX model locally, whereas a verifiable inference entails executing the transpiled ZK version on the Giza platform to generate a ZK proof of the inference.

Executing Unverifiable Inference with GizaModel and ONNX

To execute an unverifiable inference, initiate aGizaModel with the path to your ONNX model file. Use thepredict function, setting theverifiable parameter toFalse . Below is an example in Python:

```

Copy fromgiza_actions.modelimportGizaModel fromgiza_actions.taskimporttask fromgiza_actions.actionimportAction,action

@task(name='Prediction with ONNX') defprediction(image): model=GizaModel(model_path="./mnist_model.onnx") result=model.predict( input_feed={model.session.get_inputs()[0].name: image}, verifiable=False ) returnresult

@action(name='Execution: Prediction with ONNX', log_prints=True) defexecution(): image=preprocess_image("./imgs/zero.png")# Assuming preprocess_image is defined earlier predicted_digit=prediction(image) print(f"Predicted Digit:{predicted_digit}") returnpredicted_digit

execution()

```

Executing Verifiable Inference with GizaModel and ONNX

To enable verifiable inference, the model must first be transpiled toOrion Cairo , allowing execution on CairoVM to generate trace and memory files for inference proof.

Transpiling from ONNX to Orion Cairo

To transpile your ONNX model to Orion Cairo, execute the following command in your terminal:

```

Copy gizatranspilePATH_TO_YOUR_ONNX_MODEL--output-pathNAME_OF_TRANSPILED_MODEL

```

This converts your model to Cairo, providing amodel-id andversion-id for deployment on the Giza platform. The deployment process on our platform sets up services that handle prediction requests via a specific endpoint, utilizing Cairo for provable inferences.

When we transpile a model we have two possibilities: a fully compatible model and a partially compatible one.

A model is fully compatible when all the operators that the model uses are supported by the Transpiler and Orion, if this happens the model is compiled after transpilation and we save the .sierra file on behalf of the user to use later for deployment. This will be shown in the output of the transpile command:

```

Copy [giza][2024-02-07 16:32:13.511] Transpilation is fully compatible. Version compiled and Sierra is saved at Giza ✅

```

If a model is partially supported, we will create a warning in the output stating that not all the operators are supported right now. If it is partially supported the Cairo code can still be modified for later compilation and deployment.

```

Copy [WARN][2024-02-07 16:42:31.209] Transpilation is partially supported. Some operators are not yet supported in the Transpiler/Orion [WARN][2024-02-07 16:42:31.211] Please check the compatibility list in Orion: https://cli.gizatech.xyz/frameworks/cairo/transpile#supported-operators

```

You can check all operators supported by the transpilerhere . We are actively working to support more operators.

If a model is partially supported

As previously mentioned, if a model is only partially supported during transpilation, you can finetune your Cairo project to ensure successful compilation. Once your Cairo project compiles without errors, you can proceed to build it and obtain the Sierra file necessary for deployment. The Sierra file serves as an intermediate representation bridging the high-level Cairo code and compilation targets. To deploy your Cairo model, it is essential to compile your transpiled model using Scarb to generate this file:

```
```

Copy cdyour_transpiled_model/inference scarbbuild

```

Once your transpiled project compiled, Sierra file will be located in thetarget/dev/inference.sierra directory.

Deploying the Transpiled Model

For deployment, use thedeploy command as shown below, which prepares your machine learning service to accept predictions at the/cairo_run endpoint:

```

Copy

> giza deployments deploy --model-id 1 --version-id 1 inference.sierra ▰▰▰▰▰▱▱Creatingdeployment! [giza][2024-02-0712:31:02.498] Deployment is successful ✅ [giza][2024-02-0712:31:02.501] Deployment created with id ->1 ✅ [giza][2024-02-0712:31:02.502] Deployment created with endpoint URL: https://deployment-gizabrain-38-1-53427f44-dagsgas-ew.a.run.app

```

If a model is fully compatible the sierra file is not needed and can be deployed without using it in the command:

```

Copy

> giza deployments deploy --model-id 1 --version-id 1 ▰▰▰▰▰▱▱ Creating deployment! [giza][2024-02-07 12:31:02.498] Deployment is successful ✅ [giza][2024-02-07 12:31:02.501] Deployment created with id -> 1 ✅ [giza][2024-02-07 12:31:02.502] Deployment created with endpoint URL: https://deployment-gizabrain-38-1-53427f44-dagsgas-ew.a.run.app

```

For a partially compatible model the sierra file must be provided, if not an error will be shown.

Running a Verifiable Prediction

With your Cairo model deployed on the Giza platform, you can execute verifiable predictions. This involves running the Sierra program on CairoVM to generate trace and memory files for proof, returning the output value and initiating a proving job for a Stark proof:

```

Copy fromgiza_actions.modelimportGizaModel fromgiza_actions.taskimporttask fromgiza_actions.actionimportAction,action

@task(name='Prediction with Cairo') defprediction(image,model_id,version_id):

Initialize a GizaModel with model and version id.

model=GizaModel(id=model_id, version=version_id)

Call the predict function.

Set verifiable to True, and define the expecting output datatype.

(result,request_id)=model.predict(input_feed={"image": image}, verifiable=True,) returnresult,request_id

@action(name='Execution: Prediction with Cairo', log_prints=True) defexecution(): image=preprocess_image("./imgs/zero.png") (result,request_id)=prediction(image,1,1) returnresult,request_id

execution()

```

A prediction in verifiable mode, return result and request id. The request id can be used to download the ZK proof.

Downloading the ZK Proof of an Inference

After running your model, CairoVM generates the necessary files to prove your program. You can download the ZK proof generated for your inference as follows:

```

Copy importrequests fromgizaimportAPI_HOST

Update the following variables with your specific details

```
MODEL_ID=1 VERSION_ID=1 DEPLOYMENT_ID=3 REQUEST_ID="YOUR_REQUEST_ID" API_KEY='YOUR_API_KEY'

url=f'{API_HOST}/api/v1/models/{MODEL_ID}/versions/{VERSION_ID}/deployments/{DEPLOYMENT_ID}/proofs/{REQUEST_ID}:download'
headers={"X-API-KEY":API_KEY} d_url=requests.get(url, headers=headers).json()["download_url"] proof=requests.get(d_url)

withopen("zk.proof","wb")asf: f.write(proof.content)
```

Previous Deployments Next Build a Verifiable Neural Network with Giza Actions

Last updated1 day ago