RedisAI quick start tutorial

For this quick start tutorial, we recommend that you use:

- Either:
  - A Redis Enterprise Software database with the RedisAI module
  - Contact support if you want to try out RedisAI on Redis Cloud
  - redis-cli with connectivity to a Redis database

Getting started

You can connect to RedisAI using any Redis client. Better yet, some languages already have client implementations for RedisAI - the list can be found at the Clients page. RedisAI clients wrap the core API and simplify the interaction with the module.

We’ll begin by using the official redis-cli Redis client. If you have it locally installed feel free to use that, but it is also available from the container:

```
redis-cli
```

Using RedisAI tensors

A tensor is an n-dimensional array and is the standard representation for data in DL/ML workloads. RedisAI adds to Redis a Tensor data structure that implements the tensor type. Like any datum in Redis, RedisAI’s Tensors are identified by key names.

Creating new RedisAI tensors is done with the `AI.TENSORSET` command. For example, consider the tensor:

$$
\begin{bmatrix}
2 \\
3
\end{bmatrix}
$$

We can create the RedisAI Tensor with the key name ‘tA’ with the following command:

```
AI.TENSORSET tA FLOAT 2 VALUES 2 3
```

Copy the command to your cli and hit the <ENTER> on your keyboard to execute it. It should look as follows:

```
$ redis-cli
127.0.0.1:6379> AI.TENSORSET tA FLOAT 2 VALUES 2 3
OK
```

The reply 'OK' means that the operation was successful. We’ve called the `AI.TENSORSET` command to set the key named ‘tA’ with the tensor’s data, but the name could have been any string value. The FLOAT argument specifies the type of values that the tensor stores, and in this case a single-precision floating-point. After the type argument comes the tensor’s shape as a list of its dimensions, or just a single dimension of 2.

The VALUES argument tells RedisAI that the tensor’s data will be given as a sequence of numeric values and in this case the numbers 2 and 3. This is useful for development purposes and creating small tensors, however for practical purposes the `AI.TENSORSET` command also supports importing data in binary format.

The Redis key ‘tA’ now stores a RedisAI Tensor. We can verify that using standard Redis commands such as `EXISTS` and `TYPE`:
Using `AI.TENSORSET` with the same key name, as long as it already stores a RedisAI Tensor, will overwrite the existing data with the new. To delete a RedisAI tensor, use the Redis `DEL` command.

RedisAI Tensors are used as inputs and outputs in the execution of models and scripts. For reading the data from a RedisAI Tensor value there is the `AI.TENSORGET` command:

```
127.0.0.1:6379> AI.TENSORGET tA VALUES
1) INT8
2) 1) (integer) 2
3) 1) (integer) 2
   1) (integer) 3
```

---

## Loading models

A **Model** is a Deep Learning or Machine Learning frozen graph that was generated by some framework. The RedisAI Model data structure represents a DL/ML model that is stored in the database and can be run.

Models, like any other Redis and RedisAI data structures, are identified by keys. A Model's key is created using the `AI.MODELSET` command and requires the graph payload serialized as protobuf for input.

In our examples, we'll use one of the graphs that RedisAI uses in its tests, namely 'graph.pb', which can be downloaded from [here](https://github.com/RedisAI/RedisAI/raw/master/test/test_data/graph.pb). This graph was created using TensorFlow with [this script](https://github.com/RedisAI/RedisAI/blob/master/test/test_data/graph.pb).

Use a web browser or the command line to download 'graph.pb':

```
wget https://github.com/RedisAI/RedisAI/raw/master/test/test_data/graph.pb
```

You can view the computation graph using Netron, which supports all frameworks supported by RedisAI.

![Computation Graph](image)

This is a great way to inspect a graph and find out node names for inputs and outputs.

`redis-cli` doesn't provide a way to read files' contents, so to load the model with it we'll use the command line and output pipes:

```
$ cat graph.pb | redis-cli -x \
   AI.MODELSET mymodel TF CPU INPUTS a b OUTPUTS c
   OK
```
Like most commands, `AI.MODELSET`'s first argument is a key's name, which is 'mymodel' in the example. The next two arguments are the model's DL/ML backend and the device it will be executed on. 'graph.pb' in the example is a TensorFlow graph and is denoted by TF argument. The model will be executed on the CPU as instructed by the CPU argument.

TensorFlow models also require declaring the names of their inputs and outputs. The inputs for 'graph.pb' are called 'a' and 'b', whereas its single output is called 'c'. These names are provided as additional arguments after the 'INPUTS' and 'OUTPUTS' arguments, respectively.

### Running models

Once a RedisAI Model key has been set with `AI.MODELSET` it can be run with any Tensor keys from the database as its input. The model's output, after it was executed, is stored in RedisAI Tensors as well.

The model stored at 'mymodel' expects two input tensors so we'll use the previously-created 'tA' and create another input tensor:

\[
\begin{bmatrix}
3 \\
5
\end{bmatrix}
\]

with the following command:

```plaintext
AI.TENSORSET tB FLOAT 2 VALUES 3 5
```

The model can now be run with the `AI.MODELRUN` command as follows:

```plaintext
AI.MODELRUN mymodel INPUTS tA tB OUTPUTS tResult
```

For example:

```plaintext
127.0.0.1:6379> AI.TENSORSET tA FLOAT 2 VALUES 2 3
OK
127.0.0.1:6379> AI.TENSORSET tB FLOAT 2 VALUES 3 5
OK
127.0.0.1:6379> AI.MODELRUN mymodel INPUTS tA tB OUTPUTS tModel
OK
```

The first argument to `AI.MODELRUN` is the name of the key at which the RedisAI Model is stored. The names of RedisAI Tensor keys that follow the `INPUTS` argument are used as input for the model. Similarly, following the `OUTPUTS` argument are the key names of RedisAI Tensors that the model outputs.

The inputs for the example are the tensors stored under the 'tA' and 'tB' keys. Once the model's run had finished, a new RedisAI Tensor key called 'tResult' is created and stores the model's output.

For example:
The model we've imported from 'graph.pb' takes two input tensors as input and outputs a tensor that is the product of multiplying them. In the case of the example above it looks like this:

\[
tModel = tA \times tB = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \times \begin{bmatrix} 2 \\ 3 \times 5 \end{bmatrix} = \begin{bmatrix} 6 \\ 15 \end{bmatrix}
\]

**Model management**

The `AI.MODELGET` command can be used for retrieving information about a model and its serialized blob. The `AI.INFO` command shows runtime statistics about the model’s runs. Lastly, RedisAI Model keys can be deleted with the `AI.MODELDEL` command.

**Scripting**

RedisAI makes it possible to run TorchScript with the PyTorch backend. Scripts are useful for performing pre- and post-processing operations on tensors.

The RedisAI Script data structure is managed via a set of dedicated commands, similarly to the models. A RedisAI Script key is:

- Created with the `AI.SCRIPTSET` command
- Run with the `AI.SCRIPTRUN` command
- Deleted with the `AI.SCRIPTDEL` command

We can create a RedisAI Script that performs the same computation as the 'graph.pb' model. The script can look like this:

```python
def multiply(a, b):
    return a * b
```

Assuming that the script is stored in the 'myscript.py' file it can be uploaded via command line and the `AI.SCRIPTSET` command as follows:

```
cat myscript.py | redis-cli -x AI.SCRIPTSET myscript CPU
```

This will store the PyTorch Script from 'myscript.py' under the 'myscript' key and will associate it with the CPU device for execution. Once loaded, the script can be run with the following:

```
AI.SCRIPTRUN myscript multiply INPUTS tA tB OUTPUTS tScript
```

For example:
127.0.0.1:6379> AI.TENSORSET tA FLOAT 2 VALUES 2 3
OK
127.0.0.1:6379> AI.TENSORSET tB FLOAT 2 VALUES 3 5
OK
127.0.0.1:6379> AI.SCRIPTRUN myscript multiply INPUTS tA tB OUTPUTS tScript
OK
127.0.0.1:6379> AI.TENSORGET tScript VALUES
1) FLOAT
2) 1) (integer) 2
3) 1) "6"
   2) "15"

Updated: March 11, 2022