Proto File that specifies service, structure of data and functions

```
syntax = "proto3";

package example;

Defines Service
service ResNet {
  rpc ClassifyImage (Message) returns (ClassifyResult) {}
}
Defines the Message structure being sent from Loadgen
message Message{
  repeated int32 image_matrix = 1;
  int32 sample_idx = 2;
}
Defines the Message Response being returned by SUT
message ClassifyResult {
  int32 output_label = 1;
}
```

Function on Loadgen side that sends the data to SUT and receives response

```
// Prepares query, sends it across the network to SUT and returns output
label

int client_predict(int samples_idx)

  {
    // Prepare request

    ClientContext context;
    Status status;
    Message request;

    torch::Tensor query_image = qsl.get_features(samples_idx)
```

```
    // Set value of Sample_ID (Assume integer)

    request.set_sample_idx(samples_idx);

    // Set value of Query Image (Image Tensor in this case)

    int* query_image_array =
query_image.to(torch::kInt32).data_ptr<int>();

    google::protobuf::RepeatedField<google::protobuf::int32>
field{query_image_array,query_image_array + 1*3*224*224};

    request.mutable_image_matrix()->Swap(&field);

    // Serialize the query and send it to SUT

    status = _stub -> ClassifyImage( & context, request, & response);

    // Return output_label from received response

    return response.output_label();
  }
```

After we define the service and message structure in .proto file like shown above, we can then generate the client and server interfaces using protobuf compiler *protoc.* The generated files will contain protobuf code to populate, serialize, and retrieve our request and response message types. For instance, in the above code, the Message type object request has a function called set_sample_idx() which is readily generated by the protobuf compiler.

The information in the .proto file is not sufficient to identify the size and shape of data. Specifically, the *repeated* field in the file just assumes the data will be a list/array of arbitrary size. However, the shape of the data can be easily inferred from the source code when the data is fed to the Message object.

In this particular case, we can clearly see from the code that a single image is being read from QSL with sample_id, then we retrieve the address of the first element and then load it into the google protobuf repeated field with the help of the address and the image shape.

Given that this particular code and all generated protobuf code files will be open and available, anyone can easily infer batching, padding or caching on the loadgen node.