Using `grpc` for Cross-language Communication

If you need more scalability and better performance, you can create a gRPC service in Python and use gRPC in Node.js to communicate between the two.

Steps:

1. Define a `.proto` file for your gRPC service.
2. Implement the gRPC service in Python.
3. Use the Node.js gRPC client to communicate with the Python gRPC service.

Define a Proto File (`addition.proto`):

```
syntax = "proto3";

service AdditionService {
  rpc Add (AddRequest) returns (AddResponse);
}

message AddRequest {
  int32 a = 1;
  int32 b = 2;
}

message AddResponse {
  int32 result = 1;
}
```

Python gRPC Server (`addition_server.py`):

```
import grpc
from concurrent import futures
import addition_pb2
import addition_pb2_grpc

class AdditionService(addition_pb2_grpc.AdditionServiceServicer):
    def Add(self, request, context):
        result = request.a + request.b
        return addition_pb2.AddResponse(result=result)

def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    addition_pb2_grpc.add_AdditionServiceServicer_to_server(AdditionService(), server)
    server.add_insecure_port('[::]:50051')
    server.start()
    server.wait_for_termination()

if __name__ == '__main__':
    serve()
```

Node.js gRPC Client (`client.js`):

```
const grpc = require('@grpc/grpc-js');
const protoLoader = require('@grpc/proto-loader');

const PROTO_PATH = './addition.proto';
const packageDefinition = protoLoader.loadSync(PROTO_PATH);
const additionProto = grpc.loadPackageDefinition(packageDefinition).AdditionService;

const client = new additionProto('localhost:50051', grpc.credentials.createInsecure());

client.Add({ a: 5, b: 7 }, (error, response) => {
    if (!error) {
        console.log(`Result from gRPC: ${response.result}`);
    } else {
        console.error(error);
    }
});
```