

A Reusable Software Architecture for Deploying MATLAB Machine Learning Models in
Android and iOS Mobile Apps, and Interactive Websites

Benjamin Chen

Submitted under the supervision of Dr. Christopher Tignanelli, Dr. Chris Kauffman, and Dr. Chad Myers to the University Honors Program at the University of Minnesota-Twin Cities in partial fulfillment of the requirements for the degree of Bachelor Science, *summa cum laude* in Computer Science.

5/10/2021

Abstract

A novel and reusable software architecture was developed for productionizing a hospital trauma team activation model built in MATLAB. Previously, models were prototyped in MATLAB, then reimplemented, retrained, and retuned in Python and revalidated for deployment. This new architecture directly deploys the MATLAB model, expediting the release timeline and reducing cost. The roadblock preventing direct deployment of MATLAB models up to this point has been MATLAB's incompatibility with mobile ARM CPUs found in smartphones and tablets. This new architecture uses a server with an x86-64 CPU to conduct all model computations in a MATLAB instance running on the server, rather than attempting to transplant the MATLAB model to mobile devices. It leverages a RESTful API on the server to accept prediction requests containing validated and formatted patient data, and the Java Engine API to interact with the MATLAB Engine in which the model is run on the patient data. Three frontends, two apps and a website, accept patient data in a form containing number fields and selectable options, send prediction requests to the RESTful API upon form completion, and display the model output and corresponding activation recommendation to the user.

Introduction

In our increasingly data driven world, machine learning is the tool we use to discover insights, create models, and make decisions. The two most common pieces of software used for machine learning are MATLAB and Python. They both have an excellent selection of machine learning models and the necessary data manipulation tools; however, MATLAB is significantly more user friendly. Data import and manipulation can be done visually in MATLAB through intuitive graphical menus, while the same in Python requires knowledge of the filesystem, data structures, libraries such as Pandas, and the programming language itself [6]. As a result, MATLAB is a much more accessible platform than Python for machine learning, allowing for a much larger pool of talent to develop predictive models with it. The current lack of a software architecture for deploying MATLAB models to mobile apps and websites limits most of these models to an academic audience, with the few released to a wider audience being reimplemented in Python. A solution to this problem would unlock the work from this vast pool of talent and allow it to be widely deployed and made available to enhance decision making. Here, a software architecture that allows direct deployment of MATLAB models in mobile apps and websites to wide audiences is proposed and proven on a prehospital trauma activation model that has been deployed in a 10-center clinical trial with over a thousand predictions made.

Methods

MATLAB cannot run on devices with ARM CPUs, including mobile phones, tablets, and raspberry pi's. This is due to its reliance on the x86-64 instruction set supported by Intel and AMD CPUs. As a result, directly deploying models built with MATLAB in a self-contained mobile app is impossible. Thus, a novel mobile app and web architecture was created to deploy MATLAB models for provider and public use. This architecture consists of three parts: a MATLAB Engine instance running on a server, a RESTful API on the same server that interfaces with the MATLAB Engine, and web and native mobile frontends that accept user data and interface with the RESTful API.

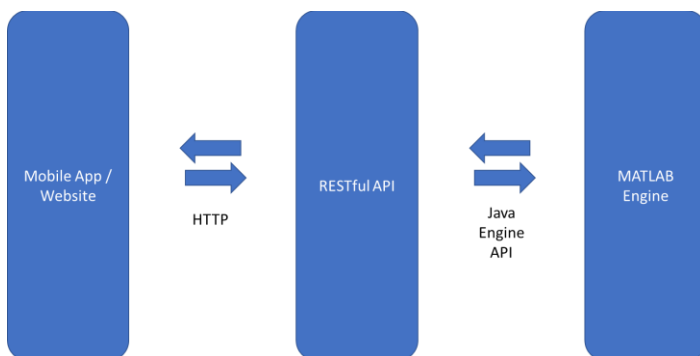


Figure 1: Data Flow Diagram

To deploy this architecture, a server with an Intel or AMD CPU must first be provisioned. In this project, the AWS EC2 cloud service was chosen [5]. A t2.small instance offers enough memory to run both the MATLAB Engine and RESTful API. Utilization at idle is just under 70%. Amazon Machine Images with MATLAB preinstalled on the Ubuntu Linux operating system are available. AMI-0ed83798ecd58abc4 was used for this project. 128GB of EBS standard storage was provisioned for this project, of which 40.5 GB was used after the server was fully configured. The automatically generated key pair was used to make the initial ssh connection to the server, and an additional user was created with password authentication enabled. The developer's IP Address was added to the allow list for RDP connections to enable graphical remote access to the server which is required for interacting with MATLAB activation prompts. Windows Remote Desktop Connection was used to connect to the server and activate MATLAB after opening it for the first time and to complete the license check after starting new instances of the API described below.

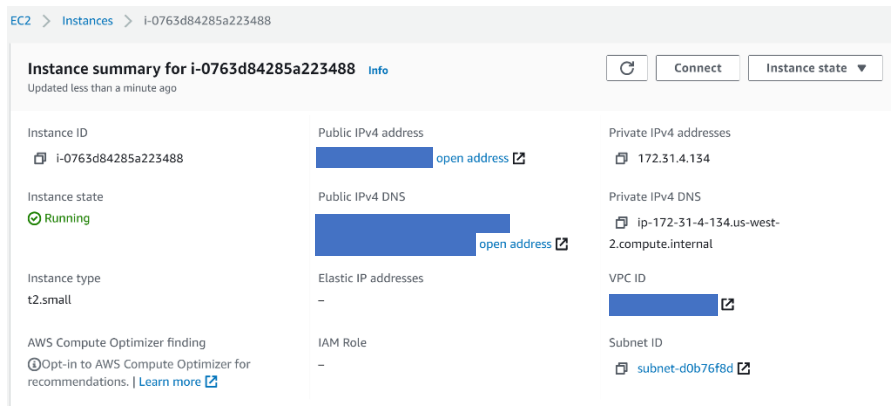


Figure 2: EC2 Instance Details Including Public IPv4 DNS URL

A RESTful API described in [1] was built using Java. The Spring framework documented in [3] was leveraged to allow the API to accept HTTP GET requests containing the patient data and respond to the requests with corresponding model predictions. The project was created using the spring initializer at start.spring.io with the options selected in figure 3. Notably, maven was chosen as the dependency manager and later used to add the MATLAB Java Engine API. A GetMapping, shown in figure 4, was then added to the RequestController.java file to create an API endpoint at /predict for HTTP GET requests. The endpoint accepts a parameter named values in the url string. It expects values to be a list of double precision floating point numbers corresponding to the encoded patient data. A sample GET request is shown being made with the Google Chrome browser in figure 4. The endpoint was temporarily set to always return a default value of 1.


```

    <dependency>
      <groupId>com.mathworks.engine</groupId>
      <artifactId>engine</artifactId>
      <version>1.0</version>
      <scope>system</scope>
      <systemPath>/usr/local/matlab/extern/engines/java/jar/engine.jar</systemPath>
<!--      <systemPath>C:/Program Files/MATLAB/R2019a/extern/engines/java/jar/engine.jar</systemPath-->
    </dependency>

```

Figure 6: Java Engine API Dependency in pom.xml

```
import com.mathworks.engine.MatlabEngine;
```

Figure 7: Java Engine API Import Statement

The MATLAB Java Engine API library imported in figure 7 was used to connect to a MATLAB session, import the machine learning model, send patient data to the session, compute a prediction on the data using the model, and retrieve the prediction. A MatlabEngine data member was added to the RequestController class defined in the RequestController.java file as shown in figure 8. It is initialized in the constructor with the static method MatlabEngine.connectMatlab(). This automatically starts a MATLAB session, connects to it, and returns a MatlabEngine object through which that session can be interacted with. If a session is already running, it connects to the existing session. Rarely, it may fail to start or connect to a MATLAB session, in which case it throws an exception that is caught in the RequestController constructor.

```

@RestController
public class RequestController {
    MatlabEngine ml;
    public RequestController() {
        try {
            ml = MatlabEngine.connectMatlab();
        } catch (InterruptedException | ExecutionException | IOException e) {
            System.out.println("connect/load exception");
            e.printStackTrace();
        }
    }
}

```

Figure 8: Starting and Connecting to a MATLAB Session

The MATLAB Classification Learner Machine Learning Model is saved in a mat file. This project expects it to be saved as a file named “model.mat” and placed in the same folder as the final compiled Java API JAR file. The RequestController constructor was expanded to load

this model into the connected MATLAB session using the instance method “eval” of the MatlabEngine connection. This is shown in figure 9. The File object of the Java standard library is used to determine the full path of the model.mat file, which is inserted as the argument of the load command sent to the MatlabEngine. This command loads the Classification Learner object defined in model.mat into the connected MatlabEngine session. In this case, the model saved in model.mat was named “cens.”

```
public RequestController() {
    try {
        ident = 1;
        ml = MatlabEngine.connectMatlab();
        String path = "model.mat";
        File f = new File(path);
        ml.eval(String.format("load('%s');", f.getCanonicalPath()));
    }
}
```

Figure 9: Loading the Machine Learning Model

Finally, the “/predict” API endpoint is updated as in figure 10 to use the imported model to make a prediction on the data passed in with the “values” parameter. First, the putVariable instance method is called on the MatlabEngine object to send the array of doubles stored in the java variable “values” to the connected MATLAB session, in which the array is saved as a new variable named “data.” Next, the eval method is used to get a prediction on “data” using the model imported earlier, and store that prediction in a new variable named “prediction.” Following that, the getVariable method is used to retrieve the data stored in the “prediction” variable just created. A new Java integer “result” is created to save the data just retrieved. Finally, “result” is returned in response to the HTTP request.

```
@GetMapping("/predict")
public int predict(@RequestParam double[] values) {
    ml.putVariable("data", values);
    ml.eval("prediction = cens.predict(data);");
    int result = ml.getVariable("prediction");
    return result;
}
```

Figure 10: Connecting the API to the Model

This concludes the implementation of the RESTful API that accepts patient data from frontend interfaces, interacts with MATLAB to evaluate the model on that data, and returns the output to the frontend. The code was moved to the server and built by running “maven package” in the root directory containing pom.xml. It is started using the command “java -jar myApi.jar.” Upon testing, one issue was discovered that caused MATLAB to prompt for the account


```
server:
  ssl:
    key-store: classpath:keystore.p12
    key-store-password: mypassword
    key-store-type: pkcs12
    key-alias: tomcat
    key-password: mypassword
  port: 8443
```

Figure 13: Spring Configuration for Enabling TLS

Discussion

A novel architecture has been proposed for the previously unsolved problem of deploying MATLAB models for use outside MATLAB. This allows the work of a vast pool of analytical talent without programming experience to be made available to all smartphone and internet users. Previously, architectures for deploying Python models as RESTful APIs have been proposed using frameworks such as Flask [7, 8] and FastAPI [9]. Unfortunately, these architectures are only compatible with models built in Python with libraries such as scikit-learn, PyTorch, or TensorFlow. This leaves out models built in MATLAB, as they must be reimplemented, retrained, revalidated, and retuned in Python by talent skilled in both programming and machine learning. The advantages of the novel architecture are demonstrated in the following use case example.

This architecture was implemented in the deployment of the Trauma Intervention Prediction MATLAB model. The logistic regression model was developed by a statistician to predict need for activating a trauma team at the destination hospital for patients picked up by EMS using prehospital metrics. It was designed to assist with what would otherwise be an arbitrary decision made using personal experience. By leveraging this architecture, over two months of development and countless hours of meeting, implementation, validation, and tuning that would have otherwise been spent duplicating the model in Python were saved. This will make these data driven trauma team activation decisions available for widespread use two months sooner and at lower cost. As a result of these benefits, the Android and iOS apps have been already released in their respective appstores and a 10-center clinical trial for proving their effectiveness in field use has begun. Screenshots are shown in Figure 14.

Trauma Intervention Prediction

Site

Level of Care

Patient Type

Transfer Method

Estimated Time to Arrival (min)

Please record most aberrant vitals below:

Systolic Blood Pressure

Pulse

GCS Score _____

Prehospital FAST Pericardial Window

End Tidal CO2 Reading Available

Base Deficit Reading Available

Base Deficit (Meq/L)

Head/Neck

Focal Neurologic Deficit

Abdomen

Emergent Intervention Predicted

Full Trauma Activation Recommended

Figure 14: Trauma Intervention Prediction App

This architecture is particularly valuable for health care research and applications, in collecting Patient Reported Outcome Measures (PROMs) for example, as it is HIPAA compliant. By hosting the RESTful API on AWS, this architecture takes advantage of AWS' adherence to the NIST 800-53 security standard, which satisfies the HIPAA Security Rule [5]. HIPAA requires a Business Association Agreement between the covered entity releasing the machine learning model and the cloud provider. This can be added for free to AWS accounts using the web portal in the Artifact page. The remaining responsibility rests with the developer to encrypt protected health information in transit and at rest [11]. This architecture does not include any storage of data, so only encryption in transit is needed. The TLS encryption added to the API and website satisfies this requirement, as the patient data sent as a URL parameter is encrypted in this standard [10].

The proposed architecture can be further expanded in the future to include data collection, real-time model updates, autoscaling, and usage logging. Data collection was implemented in the Trauma Intervention Prediction using REDCap, a commonly used data collection and storage service in medical research. A relational database hosted on AWS would be more appropriate for most projects. Real time model updates using collected data can be implemented with further MATLAB Engine API calls to send that data to the MATLAB Engine instance and build a new model. Autoscaling may become necessary for higher demand applications. Trauma Intervention Prediction is only experiencing rates of less than one request per minute during this clinical trial phase, so the t2.small easily handles the demand. Finally, usage logging may be useful for analytics and preventing malicious attacks such as denial of service.

Conclusion

A novel and reusable software architecture was developed for making machine learning models built in MATLAB available through RESTful APIs in mobile apps and websites. It has been proven in the implementation and release of Trauma Intervention Prediction. This expands the talent available for public use model development from only the small group of those with both analytical skills and programming experience to the entire pool of analytical talent. Now, instead of requiring the time of dedicated machine learning engineers of which there is currently a shortage, a statistician can build a model in MATLAB and a software engineer can deploy that model. Ideally, this will help increase the rate of adoption of machine learning and data driven decision making.

Works Cited

- [1] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". *Architectural Styles and the Design of Network-based Software Architectures (Ph.D.)*. University of California, Irvine.
- [2] MathWorks. *Java Engine API*. https://www.mathworks.com/help/matlab/matlab_external/java-api-summary.html.
- [3] VMware. *Spring*. <https://spring.io/>.
- [4] Internet Security Research Group. *Let's Encrypt*. <https://letsencrypt.org/>.
- [5] Amazon. *Amazon EC2*. AWS. <https://aws.amazon.com/ec2/>.
- [6] Colliau, Taylor; Rogers, Grace; Hughes, Z.; and Ozgur, Ceyhun, *Business Faculty Publications*. "MatLab vs. Python vs. R" (2017).
- [7] Dario Radecic. Medium. *Towards data science*. "Deploy Your Machine Learning Model as a REST API" (Nov 13, 2019).
- [8] Bamigbade Opeyemi. Medium. *Towards data science*. "Deployment of Machine learning Models Demystified" (Nov 25, 2019).
- [9] Tyler Folkman. Medium. *Towards data science*. "How To Deploy Machine Learning Models" (Mar 23, 2021).
- [10] Internet Engineering Task Force. "The Transport Layer Security (TLS) Protocol Version 1.2".
- [11] 45 C.F.R. § 164