MLMODELSCOPE WEBSITE DEVELOPMENT WITH REACT

BY

XINRUI ZHU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Professor Wen-Mei Hwu

ABSTRACT

With the rapid growth of the MLModelScope project, there is an urgent need for a user interface to show users the available resources, functionalities, and facilitate their experiments. The project was designed to develop a website with React for MLModelScope which entailed providing an interactive user interface to easily demonstrate its functionalities. Also, other users can use our platform to run some experiments without setting up the whole system.

In this thesis, we will first give an introduction to MLModelScope and React and also the goal of this project. Then a discussion follows to describe the whole design and development process and the problems we faced in each stage. Next, we dive into the details of the technologies we used and how we dealt with some technical challenges. Finally, we presented our opinions about this project, thoughts for developing in React and plans for further development.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Prof. Wen-Mei Hwu of the Electrical and Computer Engineering Department at the University of Illinois at Urbana Champaign. He consistently allowed this thesis to be my own work, but steered me in the right direction whenever he thought I needed guidance.

I would also like to thank the MLModelScope research group members who were involved in this research project: Jinjun Xiong, Abdul Dakkak and Cheng Li. Without their advice and help, the project could not have been successful.

Finally, I must express my very profound gratitude to my parents and to my boyfriend for providing me with financial support and continuous encouragement throughout my years in the master's degree program and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 MLModelScope

MLModelScope [1] is a platform which can facilitate the experimentation and evaluation of machine learning models. MLModelScope lowers the cost and effort for performing model evaluation and profiling, making it easier for others to reproduce, evaluate, and analyze accuracy, efficiency or performance claims of models and systems. It allows users to understanding the model performance (within real-world AI workflows) and its interaction with all levels of the hardware/software stack. The profiling levels of MLModelScope are shown in Figure 1.1.

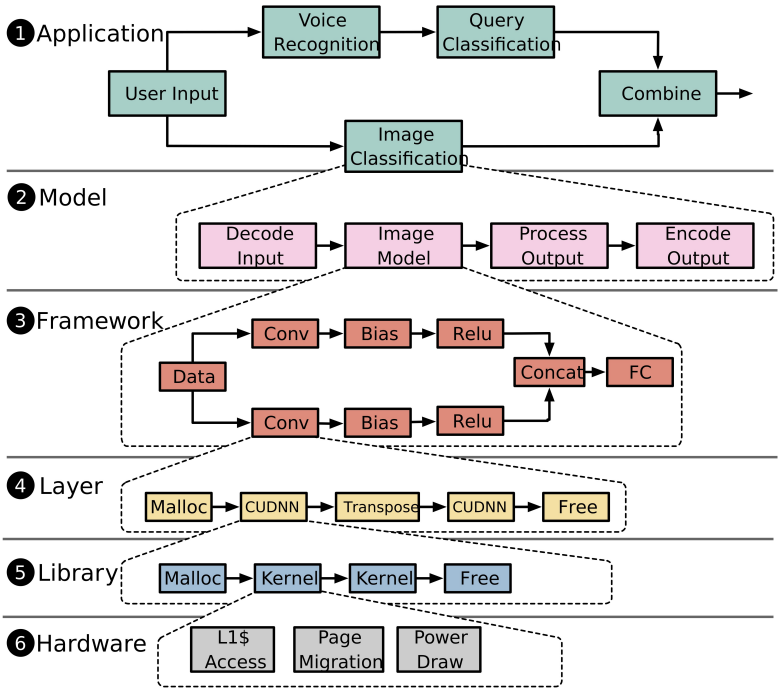

Figure 1.1: MLModelScope Profiling Levels

Figure 1.2: Website Overview in the Landing Page (see www.mlmodelscope.org for more details)

We provide both a command line interface (CLI) to give users more control of the tools and a website user interface (Web UI) to allow users to easily evaluate and profile models without familiarity with the underlying frameworks or profiling tools. Compared to CLI, Web UI is a more convenient and straightforward way to use the tool. So, Web UI is one of the most important components of this project which is an easily accessible start point that makes our tool available to most users. Figure 1.2 demonstrates part of the website landing page which gives an overview of the project.

1.2 React

React [2] is a declarative, efficient, and flexible JavaScript library for building user interfaces. It was developed by Facebook in 2011 and given open-source status in 2013 under the BSD3 license. It is currently one of the most popular front-end development libraries. Figure 1.3 shows that the Google search trend of React has surpassed Angular which was viewed as the number one front-end web development framework in recent years and keeps increasing. Since its first release, React's Github repository has generated 132,000 stars from developers and has amassed a community of almost 1300 active contributors, regularly pushing updates to enrich the library.

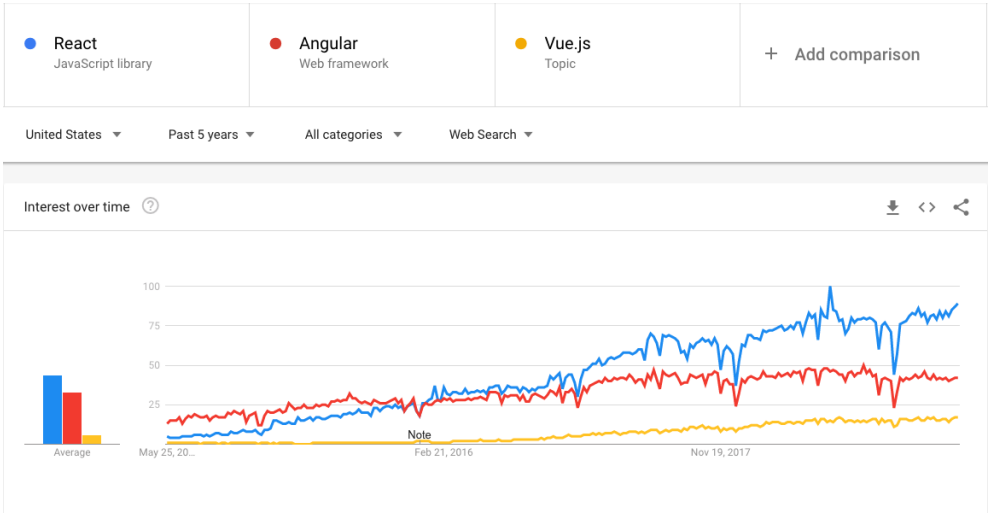

Figure 1.3: Google Search Trends of Three Popular UI Development Tools in the Past 5 Years in the United States

Next is a more detailed discussion about why we choose to use React.

1.2.1 Reusable Components

React enables the creation of module-like pieces of code called "components". These code snippets reflect a particular part of the user-interface, which can be repeated across several web pages. This is what we meant by reusability, which is an efficient and valuable time-saving strategy for development. The declarative nature of React also makes designing Web UI seamless by relieving product developers from detailed description of how to perform functions

so they could focus on more important functions and business logic.

1.2.2 Fast Learning Curve

React is actually not a framework; unlike Angular or Vue.js, it is a library that is consistently used in association with other JavaScript libraries. Hence, there is a shorter learning curve involved in understanding React compared to other comprehensive libraries.

1.2.3 Virtual DOM

Virtual DOM [3] (short for document object model [4]) is the core reason why React enables the creation of fast, scalable web apps. Through the memory reconciliation algorithm, React constructs a representation of the page in a virtual memory, where it performs the necessary updates before rendering the final web page into the browser.

1.3 Objective

The objective of this project is to build a website for MLModelScope. The goal is to clearly describe MLModelScope and show what features we are supporting, allow users to run machine learning experiments and explore the results and performances across frameworks, models, and systems and also provide clear and interactive visualizations of those experiment results.

CHAPTER 2

DESIGN AND DEVELOPMENT WORKFLOW

This chapter discusses our workflow while designing and developing the MLModelScope website which contains three parts: wireframe design, visual design and development. While our main focus was the development of the site, we cooporated with the IBM design team during the design process to nail down the high-level design from a developer's point of view. This chapter starts with an understanding about the design, which is followed by a focus on the actual development. It further elaborates on the main challenges in each step and how we handled them.

2.1 Wireframe Design

In this step, we focused on the functionalities, user needs and the user journey. The design was done in close collaboration with the IBM design team and we were meeting with them weekly and discussed these details.

During the design review, we provided background information on machine learning so that the design would be intuitive to the users. For example, how would a user want to set up the experiment, what data might interest a user, what to visualize and how to provide a clearer result.

Also, we considered some technical details and assessed the feasibility of the design. Some functionality might not be available at that time but we have planned to add it, and our goal is to make the design more compatible and easily extended. In some other situations, the functionality might need a lot of effort. We will judge the necessity of an alternative by determining if we have a better replacement or if we can eliminate it.

2.2 Visual Design

In this part, we addressed issues related to the appearance of the website. Examples include what colors, what font and size for different text, the layout and also how to display different kinds of data.

In Figure 2.1, we put the same page's wireframe design and the visual design side by side to clearly show the difference that the wireframe focus is more on the content of each page and the visual design focus is on making the page more attractive.

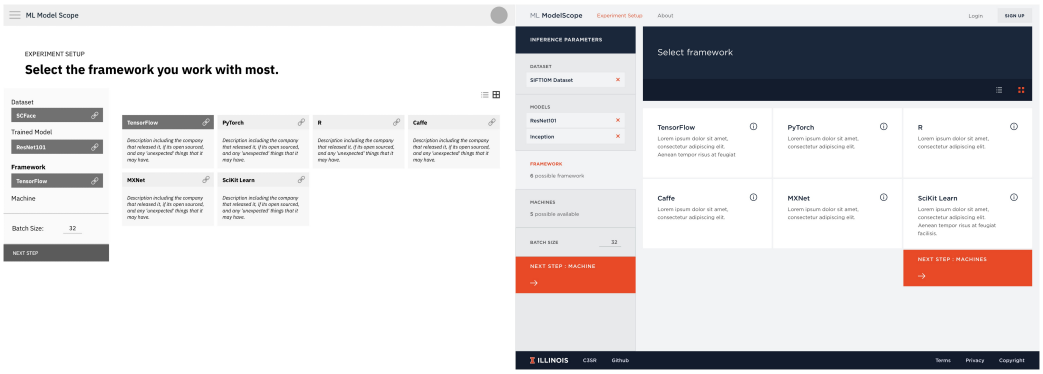

Figure 2.1: Wireframe Design and Visual Design

2.3 Development

The technical details in the development progress will be covered in Chapter 3. The focus of this section is to discuss what else is needed to address the development besides simply converting the visual design images into code and adding interactive logic.

First, we want to use existing UI design libraries to save time on styling those UI components. Therefore, we assessed a few such libraries and finally decided to use Ant Design which includes all the components we want to use and has a style similar to our visual design.

Further, as we extended the functionality, updating was needed for some of the initial design. So, it is essential to write extensible and easily maintainable code.

With the increasing use of different mobile devices like smart phones and

tablets to surf the web, it is important to guarantee our website is adaptive to different devices. Another important problem we encountered was making our website mobile compatible. We resolved that issue by using the responsive web design which means the layout changes based on the size and capabilities of the device. For example, on a phone users would see content shown in a single column view; a tablet might show the same content in two columns. The detailed layout changes will be discussed in Chapter 3.

CHAPTER 3

TECHNICAL DETAILS

3.1 Layout

3.1.1 Page Layout

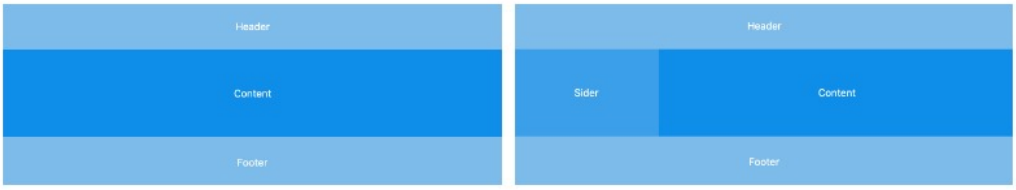

Figure 3.1: Web Layouts

The layout of each page is the first issue to clarify during development. We are mainly using two kinds of layout in the website which are shown in Figure 3.1. The layout on the right was used only in the experiment pages where the sider shows the experiment setups. All the other pages used the layout on the left.

3.1.2 Mobile Friendly Design

Mobile device compatibility is a very important part of this project. Our website is passing the mobile friendly test [5] of Google. Figure 3.2 shows the results.

Currently, there are two popular design patterns for building a mobile friendly website: responsive web design [6] and adaptive web design [7].

With the responsive web design, the layout of the page grows or shrinks based on the resolution of the screen. Adaptive web design is more like

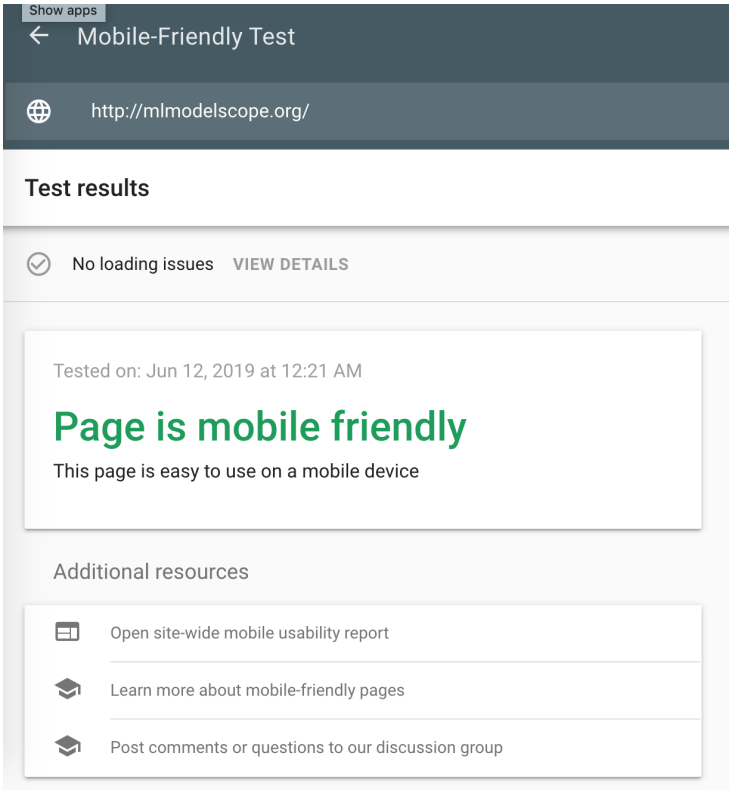

Figure 3.2: Google Mobile Friendly Test Result

having two completely different websites, one designed to fit on the user's phone and the other designed to fit on a desktop. So, with an adaptive web design, the website can be more optimized for mobile devices, while the look on desktop and mobile devices will be more consistent with a responsive web design.

We finally selected the responsive web design since our website is not very complicated and the Ant Design [8] library can be easily used to implement a responsive web design with its grid system. The grid system was defined based on rows and columns. Each row is divided into 24 aliquots and each column has an argument value of 1-24 to represent its range spans. If the total span of columns in a row is over 24, then the overflowing columns as a whole will start a new line arrangement. This is useful as we develop the cards in the experiment page which is shown in Figure 3.3.

The general idea of transforming the page layout is to change the horizontal layout to a vertical one. Fortunately, the column also allows specification of the span based on window size. For example, Figure 3.3 shows the com-

parison of an experiment page shown on a desktop (left) and on a mobile device (right). We are displaying three cards (span = 8) in a normal window and displaying only one card (span = 24) in a small window, which improves the website appearance in mobile devices.

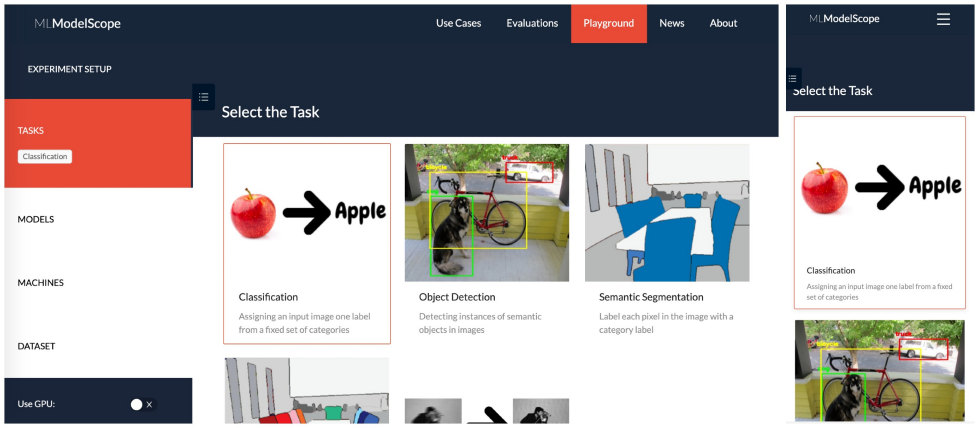

Figure 3.3: Different PC and Phone Layouts

3.2 UI Components

The whole journey of building a website with React is like playing Lego where small pieces (components) are used and put together to build the whole website. Building extensible and reusable components is the key to improve the reusability, maintainability and scalability of our code. Before presenting how to design and implement those components, we discuss JSX which is a special syntax that is recommended by React to build components.

3.2.1 JSX

The syntax of JSX is very similar to regular HTML except that some of the tags are React components instead of HTML tags. This syntax is not required to create React applications and it can be easily converted to regular JavaScript code as shown in the Figure 3.4 example. But, we chose to use it since it helps to keep the code clean and gives a visual aid about what the website looks like.

```
function hello() {
  return
    <div style={{color: "red"}}>
        <Button>Hello world!</Button>
        <Button>Hello world!</Button>
    </div>;
}
```

```
function hello() {
  return;
  React.createElement(
    "div",
    {
      style: {
        color: "red"
      }
    },
    React.createElement(Button, null, "Hello world!"),
    React.createElement(Button, null, "Hello world!")
  );
}
```

Figure 3.4: JSX (left) to JavaScript (right)

3.2.2 Component Style UI Development

Previously, most UI development was done by separating logic with markup. Each page usually has an HTML template to describe how the website looks and a JavaScript file to define how to handle different events.

But with component style, the markup and handling functions are put together in components. Components are in fact special JavaScript functions. They take arbitrary inputs (props) and return React elements which describe how things will appear on the screen.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}


class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Figure 3.5: Function Component vs. Class Component

There are two ways to define a component: as a function or a class. As shown in Figure 3.5, the function Welcome and the class Welcome are actually equivalent. Function components are simply functions that take props as input and return React elements. However, they cannot handle some complicated functionalities since unlike class components, they do not support

state management. We will talk about the details of state management in Section 3.5.

In our project, we prefer to define components using a class component even when the component is really simple for consistency. In the class, a render function which will return the React elements is essential. And handler functions like the "handleClick" function in Figure 3.6 can be added to handle the events which is very similar to handling events on DOM elements. Note that the reason we are manually binding "this" to the "handleClick" function is because in JavaScript class methods are not bound by default.

```
class Welcome extends React.Component {
  constructor(props) {
    super(props);

    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    window.alert("Hello")
  }

  render() {
    return (
      <button onClick={this.handleClick}>Welcome</button>
    );
  }
}
```

Figure 3.6: Class Component with Event Handling

3.2.3 Components Development Pattern

In this section, we talk about how to implement UI components. The development process and basic patterns are covered.

Build Components Top-Down

We tend to start from a big picture and work toward small components during development. In our case, we first draw boxes for the header, content and footer which are shared by all the pages and then fill in contents to those boxes using the same strategy.

An obvious benefit of this method is that it follows the data flow direction in React components. The data flow in React is unidirectional which is depicted in the Figure 3.6 example. That means components can only take data from their parent component. For example, the props object in Welcome comes from its parent component and Welcome component passes the "handleClick" function to its child component button. Therefore, building from the parent component to the child components can help developers to clearly know what data is accessible in the child components.

Extract Components

When following the top-down developing style, we usually encountered the problem of growing components since we kept adding things to existing components. There is no strict rules about how to split components but a good reference is the single responsibility principle [9], that is, a component should ideally only do one thing. Further, similar components might be used in many different places and we definitely want to generalize those components to avoid duplicate code.

We follow three general guidelines for minimizing component growth and maximizing component reuse in our development. First, an obvious approach is to take advantage of common opportunities, such as the header, sider and footer. Those areas are normally independent and can be easily extracted.

Secondly, reusable components such as buttons and cards should also be extracted and generalized. This not only avoids duplicate code, but also contributes to the consistency of the whole website. In this case, components normally should be customizable based on props.

Finally, conditional rendering is very useful for customizable parts mentioned before, but will also result in low readability. So, when the conditional rendering parts expand, we might want to extract them to separate components and pass all the necessary data into those components.

Static to Dynamic

As we mentioned before, web pages normally include two parts: the markup and the interactive handling functions. It is much easier to start with a static version using some mock data and then add interactivity. Because

updating interactive logic can sometimes be complicated, we always verify the static page with other members in our team before moving forward to avoid significant changes.

3.2.4 Components Organizing

Recall that we discussed when to split the components in the Section 3.2.3. After extracting the component, we need to determine where to put this component.

In order to make the source code directory structure clear and easy to maintain, we organized the components based on the lowest common ancestor rule. Each directory contains an index file which describes the component and all the child components (which can be a single file or a directory depending on whether it has a child) it is using. If a child component is used by other components, it will be lifted up to the lowest common ancestor directory of all the components that are using it. Figure 3.7 shows part of the directory structure of our project.

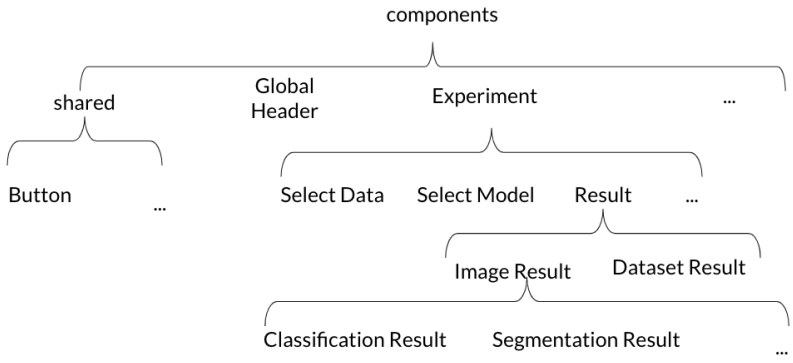

Figure 3.7: Components Directory Structure

A problem with this method is that when a component gets lifted, all the import paths might need to be updated. Considering the scope that a component will be used in advance and put the component to the correct place directly can reduce the problem. Besides, most editors today have the functionality to automatically update import paths.

3.3 Visualization

Visualization is a very important and challenging part of the website. We are using different tools for visualizing the dataset inference results and single image inference result. (We currently only support images as input and plan to add more modalities later.)

3.3.1 Dataset Inference

For dataset inference, we are using BizCharts [10] to create tables and charts for users to compare the performance of different models and frameworks.

The data is collected from inference experiments and are being stored in JSON files. When the user clicks predict, the web will fetch corresponding data, processing them to specific format and generate the charts. Right now, we only display accuracy, latency and layer duration information which are shown in Figure 3.8, but we plan to add more charts in the future. Layer duration for different models and frameworks are shown in different charts because models have different implementation on different frameworks and therefore has totally different neural net layers. Also, we only showed one example of a layer duration chart in Figure 3.8 since there are too many to show in one figure.

3.3.2 Image Inference

The goal for visualizing the image inference result is to give users an intuitive understanding of the result. The methods are different based on the task. Most of them include manipulating images which are done by react-konva [11]: a JavaScript library for drawing complex canvas graphics using React.

Classification

Image classification is a process which takes in an image and outputs an array of probabilities of classifying the input image as each class. We are pulling out the top 10 possible classes and display the class name and probabilities in a table. The response data structure is shown in Figure 3.9 on the left side and on the right side is the sample result table.

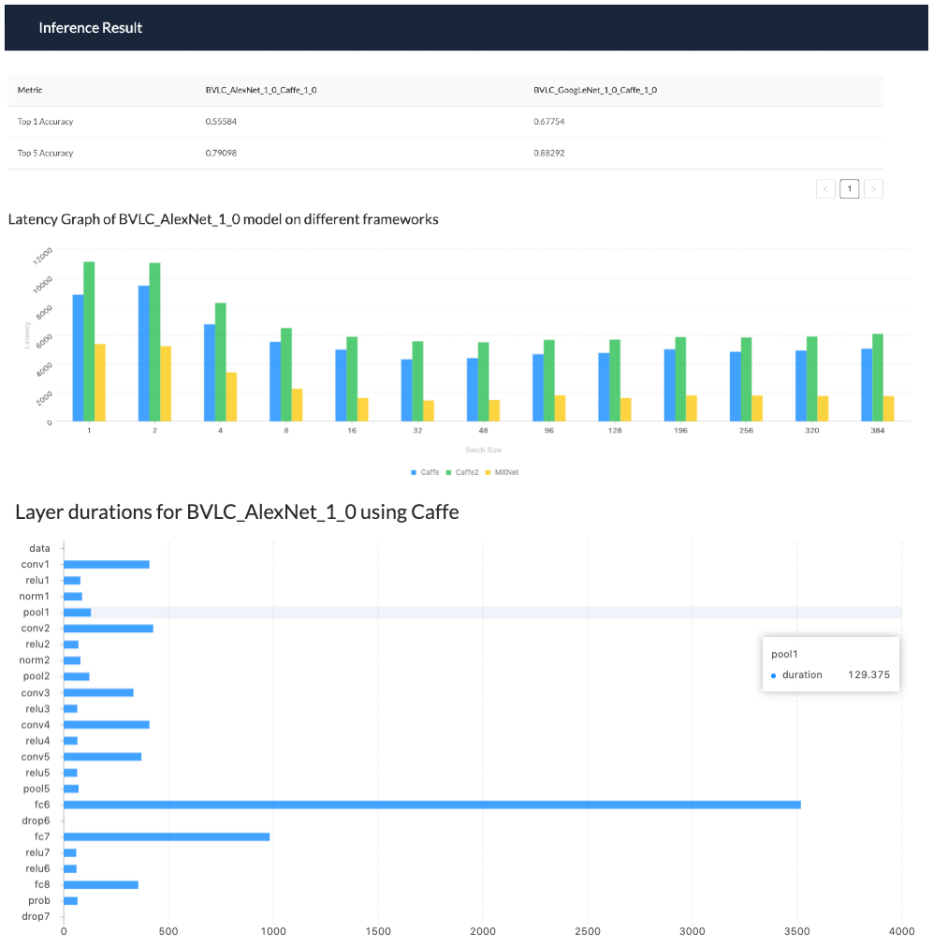

Figure 3.8: Dataset Inference Result

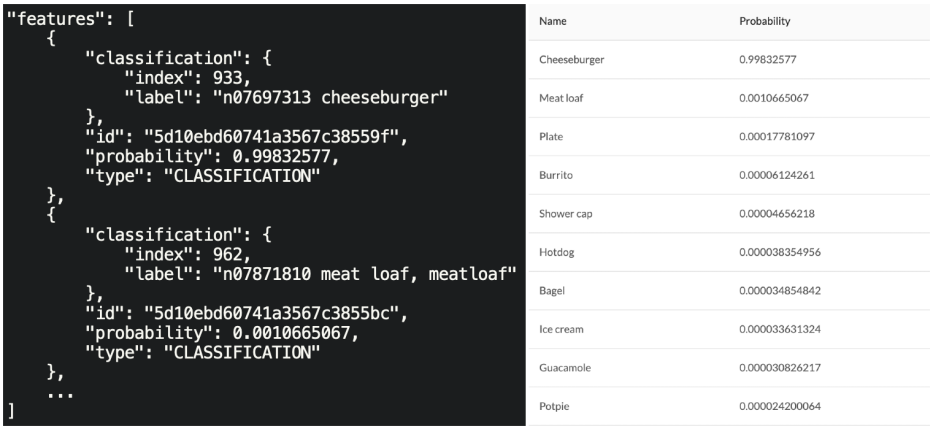

Figure 3.9: Image Classification Sample Result

Object Detection

The object detection task takes an input image and outputs a list of bounding boxes which are rectangular areas around detected objects, the class of each object and the possibility that the object belongs to that specific class as shown in Figure 3.10.

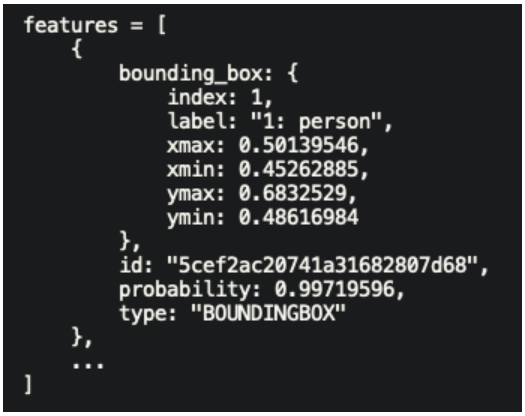

Figure 3.10: Object Detection Response Data Sample

Each bounding box is represented by two points: the upper-left point (xmin, ymin) and the lower-right point (xmax, ymax). We need to convert these four numbers into the coordinates of upper-left point (x, y), the width and height of the bounding box according to the following equations:

$$x = xmin * imageWidth$$
$$y = ymin * imageHeight$$
$$width = (xmax - xmin) * imageWidth$$
$$height = (ymax - ymin) * imageHeight$$

Then, we can pass those values into React-Konva Rect component to generate the bounding box and layer it over the original image. When the user moves the mouse over a bounding box, its label and probability will show up. But sometimes, the label is not clear on the image, so we introduced a table which can clearly display the label and probability. The corresponding row will be highlighted when the user puts the mouse over a bounding box and vice versa. We also added a slider to adjust the probability filter so that the

user can filter out the data that is not of interest. An example of the result is shown in Figure 3.11.

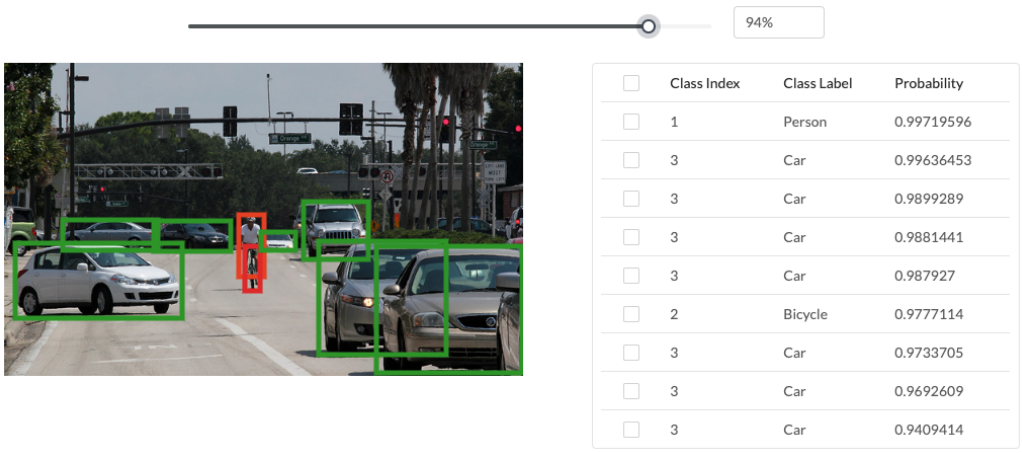

Figure 3.11: Object Detection Result

Semantic Segmentation

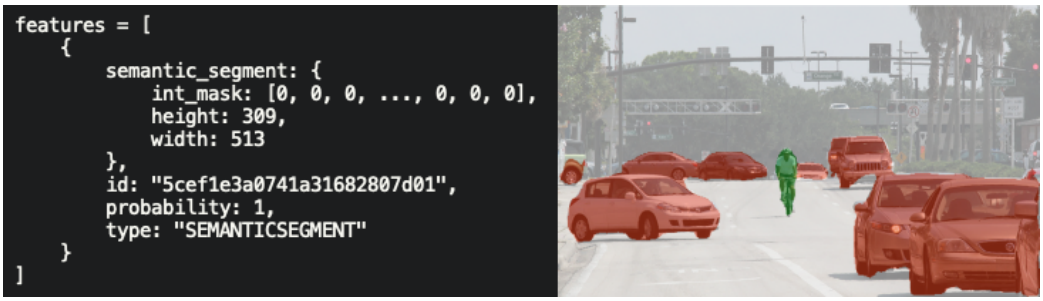

Figure 3.12: Semantic Segmentation Sample Response and Result Image

Semantic segmentation task takes an input image to generate an int mask which includes the label of each pixel in the image where 0 indicates that pixel does not belong to any class.

In the front end, we need to generate a mask image from the int mask first. This can be easily done by replacing each integer by its corresponding RGB color (for example, 0 will be changed to [255, 255, 255]). At the same time, we need to convert the 1-D int_mask into a 2-D image with the specified width and height as shown in the left part of Figure 3.12. The result, in this

example, will be a 513*309 2-D array where each element is an RGB color. We can easily change the array into an image with some JavaScript libraries.

Most of the time, the mask image will not have the same size as the original image. Therefore, we will need to resize the mask image to be the same size as the input image and finally overlap the mask image above the original image to create the resulting image shown in Figure 3.12.

Instance Segmentation

Instance segmentation is very similar to object detection. The only difference is that each bounding box also comes with a float mask which represents the possibility of each pixel in the bounding box to be classified as its class. A sample response is shown in Figure 3.13.

```
features = [
    {
        instance_segment: {
            float_mask: [0.00020763278, 0.00009891391, ..., 0.031030864, 0.0031805933]
            index: 1,
            height: 15,
            width: 15,
            label: "1: person",
            mask_type: "float",
            xmax: 0.50139546,
            xmin: 0.45262885,
            ymax: 0.6832529,
            ymin: 0.48616984
        },
        id: "5cf041940741a31682807de0",
        probability: 0.99860686,
        type: "INSTANCESEGMENT"
    },
    ...
]
```

Figure 3.13: Instance Segmentation Response Sample

We used the same method to draw the bounding boxes as we used for object detection. Then, we need to generate a mask image from the float mask. This step is very similar to what we did for semantic segmentation int masks. However, this time we assign each instance a random color. (Different instances with the same label will have a different color.) When converting the floats to RGB colors, we are using 0.5 as the threshold to filter out the object and the background. If the number is larger or equal to 0.5, we use the assigned color, otherwise we use white. Finally, we just need to resize

the mask image to be same as the bounding box and overlap it above the bounding box. The sample output is shown in Figure 3.14.

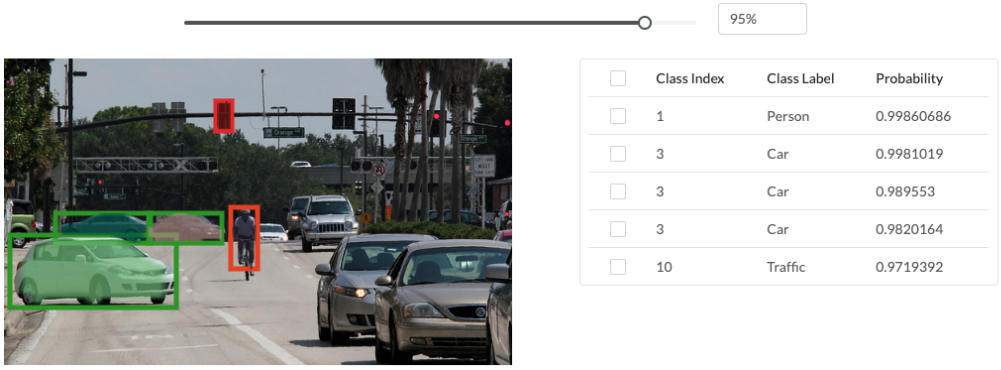

Figure 3.14: Instance Segmentation Result

Image Enhancement

The goal of the image enhancement process is to improve the quality and the information content of the original image. Thus, the output is also an image which is in JPEG format and is encoded with Base64 (jpeg_data in Figure 3.15) and we then need to display the new image so that the user can compare it with the original image.

```
features = [
    {
        id: 5cf09e1a0741a31682807e44,
        raw_image: {
            channels: 3,
            char_list: null,
            data_type: float32,
            float_list: null,
            height: 1396,
            width: 1720,
            jpeg_data: "..."
        },
        type: RAW_IMAGE
    }
]
```

Figure 3.15: Image Enhancement Response Sample

3.4 REST API

We discussed how we display the data in the previous section, and now we discuss how we obtain those data. We are using REST API [12] to interact with the server side and get the data we need. Three major sets of requests used in the website are discussed in the following sections.

3.4.1 Manifest

During the experiment setup steps, we use manifest requests to get the information about all available frameworks, models and machines. Those are GET requests and the result can be filtered out by specifying some arguments but we are requesting the whole list and do the filtering in the UI part to avoid duplicate requests.

3.4.2 Predict

This is the core functionality of our website. The whole predict process is a set of three requests: open, url/images/dataset (depends on the input) and close. If the user selects multiple frameworks and models, each combination of framework and model will need a set of predict calls. For example, if a user selects two frameworks and two models, four sets of requests are needed.

The open request will tell the server what framework and model to use and the server will open a predictor and respond with the predictor ID. Then, the UI part will pass the predictor ID and experiment data to the server to run the experiment and get the experiment result in the format we mentioned before in the visualization part. Finally, a close request will be sent which tells the server that it is time to close the predictor.

3.4.3 Authentication

The authentication system contains four requests: login, logout, signup and userinfo. Currently, username and password are transferred through basic authentication [13] which is a simple authentication scheme built into the HTTP protocol. The login and signup requests are send with the Authorization header that contains a base64-encoded string username:password.

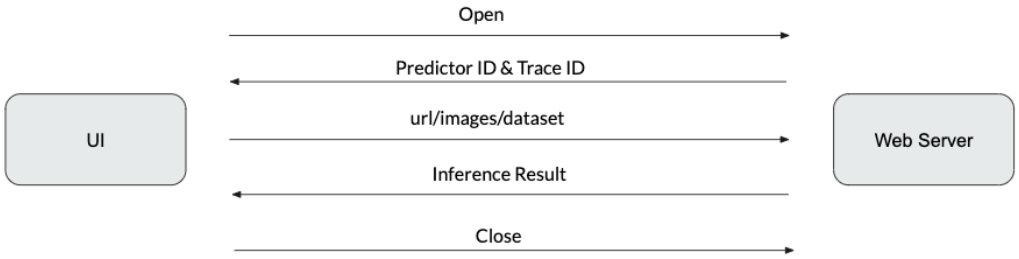

Figure 3.16: Predict Requests Flow

3.5 State Management

State management refers to the management of the state of one or more user interface control mechanisms such as text fields, OK buttons, radio buttons, etc. in a graphical user interface. In this user interface programming technique, the state of one UI control depends on the state of other UI controls. For example, a state managed UI control such as a button will be in the enabled state when input fields have valid input values and the button will be in the disabled state when the input fields are empty or have invalid values.

Redux is one of the most popular state management libraries for React. But we choose to use Context API which was introduced with React 16.3 because it is built into React and therefore needs no extra third-party dependencies. It is also relatively more straightforward to use.

In our website, we are mainly maintaining two contexts: experiment context and user context. Context normally contains both the state data and the update functions. Direct data manipulation is not allowed for consistency and security. The life cycle and functionality of these two contexts will be explained in the following sections.

3.5.1 User Context

The user context stores information related to the user. It will be created once the user opens our website and will be destroyed when the user closes it. This context was added recently when we decided to support authentication.

Right now it only contains the username and we will keep adding user-related data to it in the future.

3.5.2 Experiment Context

The experiment context stores information related to experiment setup and results. It will be created when a user goes to the experiment page and will be destroyed when the user goes to other pages. The data it stores are listed in Table 3.1.

Table 3.1: Experiment Context Content

Property	**Description**	**Type**	**Default**
modelManifests	all models manifest	object	null
frameworkManifests	all frameworks manifest	object	null
machineManifests	all machines manifest	object	null
currentPage	current experiment page	string	task
batchSize	batch size	int	1
traceLevel	trace level to show	string	framework
useGPU	whether to use a GPU	boolean	false
isPredicting	if an experiment is running	boolean	false
task	selected task	object	null
imageUrls	input image urls	object	[ ]
dataset	selected dataset	object	null
models	selected models	object	[ ]
frameworks	selected frameworks	object	[ ]
machines	selected machines	object	[ ]
result	inference result	object	null

3.6 Routing

In a web application, routing is also a very important part. It is the process of using URLs to drive the user interface (UI). From the users' point of view, there are three main functions of URLs:

1. Users can bookmark a specific page and easily come back later.
2. Users can share the link to others and they should be looking at the same page.

3. Users can use web browser's back/forward functions.

Table 3.2: MLModelScope Routing List

Route	Page Description
/	Landing Page
/playground	Experiment Page
/news	News Page
/about	About Us Page
/login	User Login Page
/signup	User Signup Page
/logout	User Logout Page
/my	User Information Page

Table 3.2 listed all the routes we are having right now. Generally, it is straightforward and the only one we want to discuss is /playground. Actually, there are multiple sub-pages for experiment setup steps and the result. But currently, we are using the same URL since it does not make sense for a user to visit the result page directly. In the future, we have plans to store experiment results and allow users to share the results page with others and will use pattern matching to display the result of different experiments.

3.7 Error Handling

3.7.1 Error Boundary

React rendering is a tree style progress, the render function in a component will call the render functions of its child components. JavaScript errors inside components used to corrupt React's internal state and cause it to emit cryptic errors on next renders and the errors we see are always caused by a previous error which makes the error information useless and hard to debug the application.

However, an error in part of the component should not cause the corruption of the whole website. In order to solve such problems, we used error boundary which is a new concept deployed with React 16. Error boundaries are React components that catch JavaScript errors anywhere in their child component

tree, log those errors, and display a fallback UI instead of the component tree that crashed.

Further, the usage of error boundary is straightforward. Defining a new lifecycle function ("componentDidCatch") in component will make it an error boundary where render function will render the fallback UI. Then it can be used as regular components as shown in Figure 3.17. If an error occurs in Component1, Component1 and Component2 will be replaced by the fallback UI in ErrorBoundary1. However, if an error occurs in Component2, Component1 will still be displayed and Component2 will be replaced by the fallback UI in ErrorBoundary2. Figure 3.18 is an example where one of the experiments failed, but the result page will not crush.

```
<ErrorBoundary1>
    <Component1 />
    <ErrorBoundary2>
        <Component2 />
    </ErrorBoundary2>
</ErrorBoundary1>
```

Figure 3.17: Error Boundary Usage

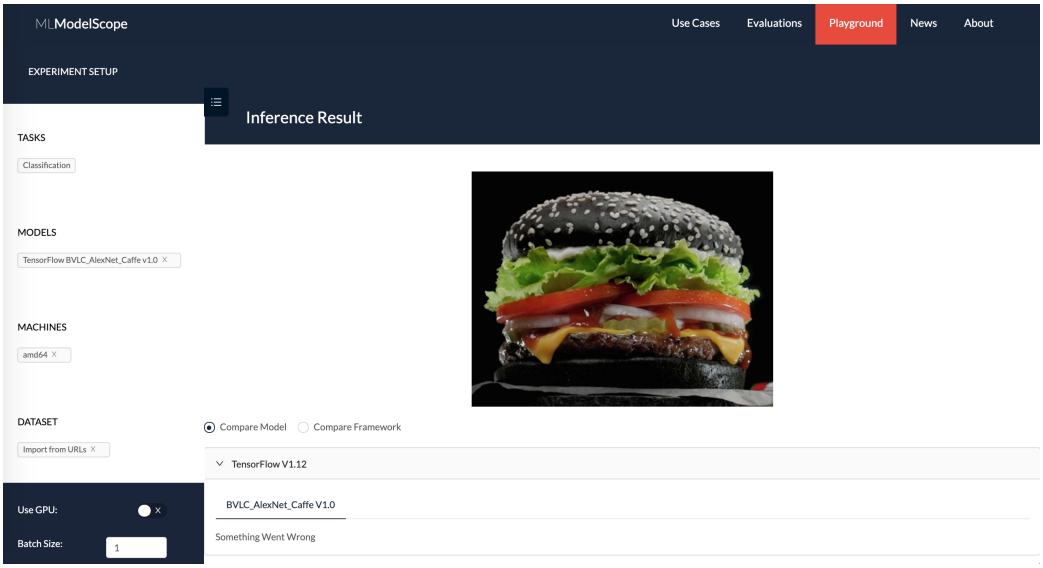

Figure 3.18: Error Boundary Example on MLModelScope

Another problem we need to solve is what to display in fallback UI and determine when we should add an error boundary. In our case, we choose

to display the trace information for debug purpose. The granularity of error boundaries really depends on the pages and contents. Simply speaking, we wrap strongly related components in the same Error Boundary. For example, when we have both images and tables in the result page, we will wrap them in the same error boundary since if one of them crushes the other one does not make any sense. However, if the user-run experiment on different models and one of the models goes wrong, we will still display the results for other models since they are independent.

3.7.2 Handling Null Properties

In JavaScript, we usually need to deal with deeply nested structures. Especially in our project, the responses are deeply nested JSON format. Sometimes, part of the properties will be empty which is a normal behavior and we do not want the application to crush because of that. Normally, we will need do a serial of check of null (an example was given in the upper part of Figure 3.19) to avoid crushing but this method will sacrifice the readability of code. So, we choose to use idx: Library for accessing arbitrarily nested, possibly nullable properties on a JavaScript object. If an intermediate property is either null or undefined when accessing a nested structure, the null or undefined is instead returned rather than throwing an error. The two code snippets in Figure 3.19 are equivalent and demonstrate how idx simplifies and cleans up the code.

```
var features;
if (
    d !== null &&
    d.response !== null &&
    d.response[imageIndex] !== null
) {
    features = d.response[imageIndex].features;
} else {
    features = null;
}
```

```
var features = idx(d, _ => _.response[imgIndex].features);
```

Figure 3.19: Example for Code Simplification Using Idx

3.8 Performance Optimization

Performance problems usually will not be noticed at the very beginning of development. But as the content on our website increasing and the functionality getting more complicated, performance issues will show up and become more serious. In this section, we will first introduce a great tool to evaluate the website performance and then discuss some techniques we used to optimize the performance of our website.

3.8.1 Chrome DevTools Performance Recording

The performance tab in Chrome DevTools [14] is a very helpful tool to assist in identifying what tasks are taking too much time. We just need to click start recording, perform the actions we want to profile (no more than 20 seconds) and then click stop. It will record the time each action took and display a tree style breakdown of the time consumed by different small tasks in the action as shown in Figure 3.20. Please note that the timing numbers are just relative and components will normally render faster in production.

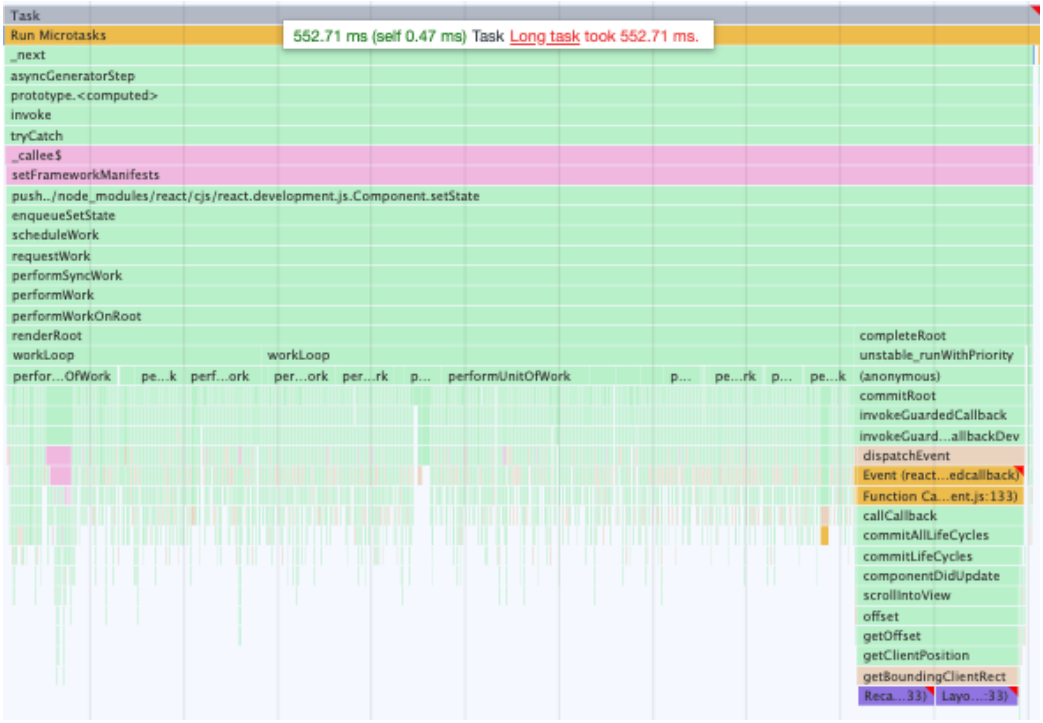

Figure 3.20: Performance Evaluation without Using Window Rendering

3.8.2 Window Rendering

This example in Figure 3.20 was taken on the model select page. It shows that the model cards rendering process takes more than 500ms because we have hundreds of models to render at the same time. We only enabled TensorFlow when we did this experiment and the condition would be much worse if we enable all frameworks. Also, we can feel obvious delay when selecting models since it will re-render those cards.

There is a technique designed to solve such rendering problems of a long list known as windowing. Since we only have limited space on the screen and therefore only part of the content can fit into the window, this technique only renders a small subset of all rows at any given time, and can dramatically reduce the time it takes to re-render the components as well as the number of DOM nodes created.

We used the react-virtualized [15] library to implement this. Basically, we create a box with the library and define a function to render the contents inside the box. This library will help to only render the components that are visble. The time dropped more than 50% as shown in Figure 3.21.

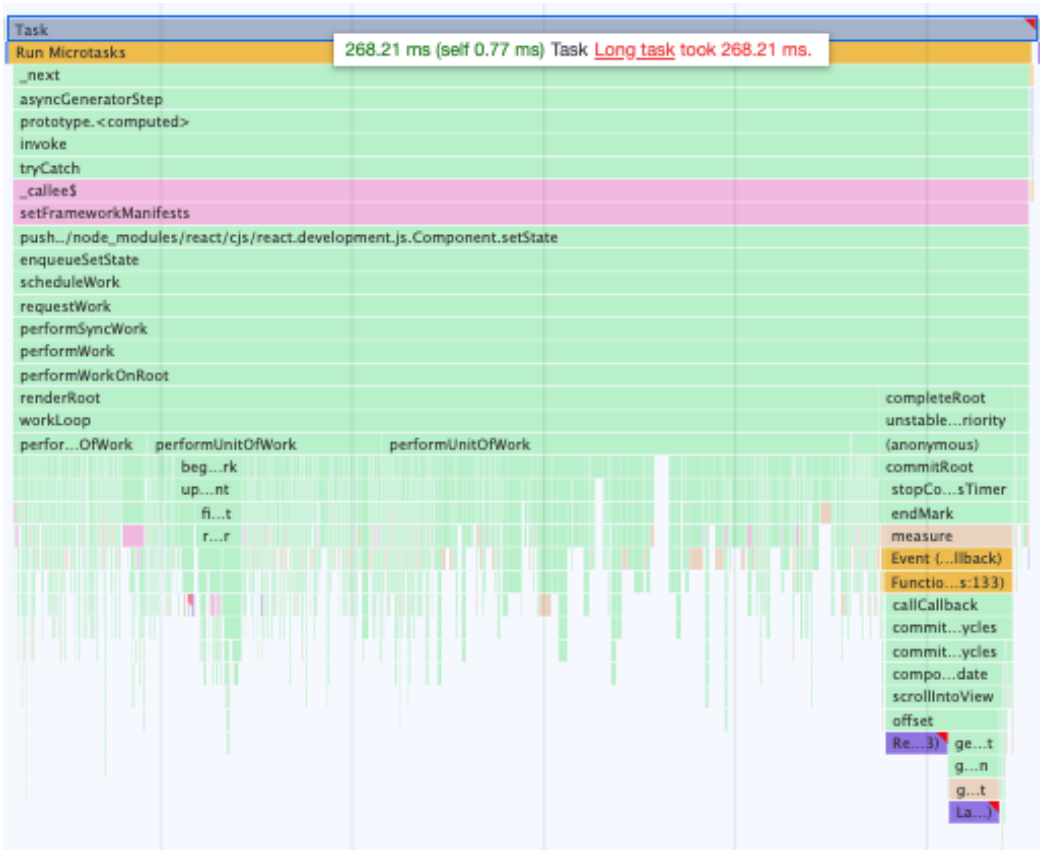

Figure 3.21: Performance Evaluation Using Window Rendering

CHAPTER 4

CONCLUSIONS

4.1 Accomplishment

In conclusion, the MLModelScope website is very successful. At this moment, it allows users to compare the pre-run experiment performance on public datasets as well as live experiment results on input images of different frameworks and models. We support hundreds of neural net models for five common computer vision experiments: classification, object detection, semantic segmentation, instance segmentation and image enhancement on almost all popular frameworks like Caffe, Caffe2, TensorFlow, Pytorch, etc. We have also implemented an authentication system which is not very useful right now, but provides the potential to support more user-specific functionality in the future.

4.2 Lesson Learned

The experience of developing a website in React is very different from traditional development frameworks because of its reusable, composable and stateful components. The biggest challenge is how to create those components. There are hundreds of ways to break down an application, but we want to make it readable and extensible. Small components will make the code easy to maintain and well-designed interfaces will simplify the composition of components.

As PC hardware improvements continue, the browser becomes much more powerful and can support more fancy functionalities. But the performance optimization is still really important. Sometimes, only a small change will smooth the surfing experience. To guarantee a great performance, a stan-

dardized coding pattern is critical. We are not doing well at this point initially and usually have to overwrite previous code which wastes a lot of time. But right now, we do have a well-documented development guide to help future developers write efficient and extensible code.

Starting from the easiest part is also a helpful technique for new developers. We were totally new to React when we stated to work on this project. At the very beginning, we just composed some components, fill in some text and create a static webpage, there was no need to consider state management, error handling, HTTP requests at that point. As the project grew, we found some specific techniques are needed and we just learned and applied them to the project. Trying to come up with the whole architecture is hard for new developers and might scare them. Working on easy things first and continuously expanding the knowledge base will help start a project.

4.3 Future Improvement

There are several potential improvements we can make in the future. First, we plan to support more data types such as text and videos in the future. We prepared some interface for those new data types and also generalized the components to be able to handle different data types in the future. But different tasks will need different visualization methods which will be a big challenge in the future.

In addition, we plan to support user-specific functionalities like uploading their own model and datasets, re-visiting previous experiment results and sharing the results with others. To achieve this, we will need to expand our database to store related data and implement corresponding API to allow the UI to query such information. On the UI part, we need to design and develop such pages for users to easily view and update those data types.

Besides, the current error handling on our website is generally naive. We plan to handle more special cases in the future and make our website more robust to provide users with a better experience.

REFERENCES

[1] A. Dakkak, C. Li, A. Srivastava, J. Xiong, and W.-M. Hwu, "MLModelScope: Evaluate and Measure ML Models within AI Pipelines," 2018.

[2] "The React website," 2019. [Online]. Available: https://reactjs.org/

[3] "What is Virtual DOM," 2019. [Online]. Available: https://reactjs.org/docs/faq-internals.html

[4] "The Document Object Model (DOM) website," 2005. [Online]. Available: https://www.w3.org/DOM/

[5] "The Google Mobile Friendly Test website," 2019. [Online]. Available: https://search.google.com/test/mobile-friendly

[6] A. E. S. Tafreshi, K. Marbach, and M. C. Norrie, "Proximity-based adaptation of web content on public displays," *International Conference on Web Engineering (ICWE): Web Engineering Lecture Notes in Computer Science*, vol. 10360, pp. 282–301, June 2017.

[7] A. Gustafson, *Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement*. New Riders, 2015.

[8] "The Ant Design website," 2019. [Online]. Available: https://ant.design/

[9] R. C. Martin, J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, *Agile Software Development: Principles, Patterns, and Practices*. The University of California: Pearson Education, 2003.

[10] "The BizCharts website," 2019. [Online]. Available: https://bizcharts.net/index

[11] "The React-Konva website," 2019. [Online]. Available: https://github.com/konvajs/react-konva

[12] "The REST API website," 2019. [Online]. Available: https://restfulapi.net/

[13] "The Basic Access Authentication Wiki website," 2019. [Online]. Available: https://en.wikipedia.org/wiki/Basic_access_authentication

[14] "Evaluate Performance with Google Chrome DevTools," 2019. [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/

[15] "The React Virtualized Library website," 2019. [Online]. Available: https://bvaughn.github.io/react-virtualized