

A RELIABLE FIRE SAFETY SYSTEM FOR THE AVERAGE HOME OWNER USING MACHINE LEARNING AND A MOBILE APPLICATION

Yuxuan Li ¹, Garret Washburn ²

¹ United World College South East Asia, 1207 Dover Road,
Singapore 139654

² California State Polytechnic University, Pomona, CA 91768

ABSTRACT

House fires in Singapore are quite common due to the widespread lack of consumer fire prevention and safety systems installed in residential areas [3]. This paper proposes a solution, the ScorchVision system, that uses a machine learning model and a Raspberry Pi with camera to detect fires in camera view and alert users through the accompanying mobile application. The whole system consists of a Raspberry Pi hardware configuration with an on-board custom trained machine learning model using PyTorch, a mobile application written in Dart using the Flutter framework, and a back-end server created with the Flask framework that communicates directly with a Firebase database [2]. Throughout development there were a few major challenges that required troubleshooting, all of which had to do with working with the individual technologies as there are a lot of moving parts within the system. To ensure the system works as expected, two different experiments were performed to find the accuracy and reliability of the machine learning models classifications as well as the average updating time from the hardware to the database [4]. Both experiment results, included in this paper, were quite positive in displaying the reliability of the ScorchVision system. Overall, the ScorchVision system is a promising new way for homeowners to take charge of keeping their home safe from fires in a fashion that is much cheaper than other posed solutions on the market. Additionally, the system is entirely open source and free to download and use, enabling privacy with complete ownership over the system and no middlemen.

KEYWORDS

Fire Detection, Machine Learning, Artificial Intelligence, Home Safety, Mobile Application

1. INTRODUCTION

In Singapore, house fires are a very common occurrence and wreak a lot of destruction throughout the country. It has been recorded that the average numbers of injuries and deaths due to fires in Singapore has remained quite stagnant from 2003 to 2014, with a death toll of 8 people and injury count of 111 in the year 2014 [1]. With these numbers showing no significant evidence of any change, the problem of unchecked fires causing damage remains. Most Singaporeans, when asked about the problem of fires in their country, would confirm that fires are a major problem. The issue of fire safety in Singapore ultimately derives from the lack of fire prevention and reaction systems, creating a demand for systems that can prevent the start of fires in residential areas. Having a system in place for citizens in Singapore to monitor their property and

alert them of gas leaks for fires starting would remedy this issue and likely bring down the yearly total of fire deaths and injuries.

Additionally some other methodologies, aimed at solving the same or a similar problem identified, are considered and compared to the method proposed in this paper. The first method, a convolutional neural network model trained in a similar fashion to the model proposed in this paper, is an effective solution at identifying house fires [5]. However, this system is entirely reliant upon internet connection as well as only having an accuracy of 92%. The next method, a multi-sensor classification system for identifying fires that included temperature, light obstruction, CO₂, etc., sensors. This system was also quite effective, however, it lacked in reproducibility since the sensors needed to be calibrated to the room and drifted over time. The third and final alternative method explored within this paper is a system for escaping wildfires in real time that integrated satellite fire data and GPS tracking [14]. This system did not provide any way of identifying fires immediately in view of the user and only showed fires that had been reported. ScorchVision seeks to resolve all of these issues by providing a self contained fire recognition and notification system that can be entirely run and hosted by a single individual in their home or wherever they set up their camera.

The solution proposed in this paper is the ScorchVision system. The Scorchvision system consists of a Raspberry Pi hardware camera with an on board machine learning model capable of detecting fires and a Mobile Application that is notified by the camera on fires or gas being detected. This solution solves the problem of house fires through a reliable, affordable and efficient approach. When a fire has happened, the user and fire services immediately. This solves the main issue where this prevents fires from growing, becoming a greater problem and causing injuries and property damage. Moreover, it is a lot more reliable compared to other solutions. Smoke detectors for example, often detect the fire too late before it has already become too big of a problem. Or on the contrary, it often creates false alarms, to the point that some people might even turn them off. The solution proposed is able to detect far more reliably and faster (idk add comparison statistic), with real time analysis. The proposed solution is also much more affordable and sustainable compared to other solutions which provide the same effectiveness. Creating a robust fire prevention system often takes a large amount of resources and time, which can be inconvenient and potentially undermine the importance of fire safety. ScorchVision provides the same protection for a fraction of the resource and time, meaning its impact across communities would be greater and faster. Overall, the solution proposed solves the problem in multiple dimensions.

Within this paper, two experiments are performed to test the efficacy of the ScorchVision system. The first experiment is designed to find the accuracy of the machine learning model on an unseen testing set of fire and non-fire images. The concluding results of the experiment are indicative of an efficiently trained model, as it boasts an accuracy score of around 98%. The second experiment, designed to test the efficiency of the system, measures the time it takes for the hardware to update the backend so the user can be notified of the status. The experiment demonstrated that the update time is quite quick with an average update time of around 5.7 seconds. Overall, the experiments demonstrated the efficiency and reliability of the ScorchVision system, and were a very positive highlight of the project.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Optimizing AI Training for Fire Detection Accuracy

The effectiveness of this element relies significantly on the AI model's precision, necessitating thorough training and top-tier data. Issues might stem from scarce or uneven datasets, prolonged training periods, and hardware limitations. Subpar data could result in flawed fire detection, whereas inadequate processing capacity might introduce lag. To tackle these concerns, I could acquire sizable, validated datasets and employ data augmentation to bolster model resilience. Training productivity could be boosted via GPU acceleration and batch refinement. Lastly, I could assess various architectures and fine-tune hyperparameters to strike a balance between accuracy, velocity, and computational efficacy for peak model output.

2.2. Raspberry Pi Sensor Integration Issues

Combining several sensors on the Raspberry Pi might trigger significant compatibility and efficiency problems. The Bullseye Legacy 64-bit OS has restricted support for specific Python libraries and camera drivers, potentially resulting in setup failures or program faults. Moreover, linking both the camera and gas sensor through GPIO pins might yield erratic data or inadequate power supply. To resolve this, I could utilize library versions certified for the OS, establish a virtual setting to handle dependencies, and refine power allocation using external components or a steady source to ensure consistent sensor operation.

2.3. Mobile App UI and Backend Integration

Developing the mobile app requires balancing functionality with a clean, intuitive design. The main challenge could be creating a user interface that is both visually appealing and easy to navigate while maintaining reliable communication with the backend server. Compatibility across different devices and screen sizes might also present layout issues. To solve these, I could follow modern UI/UX guidelines, perform usability testing, and use a responsive framework such as Flutter. Implementing real-time error handling and ensuring smooth API integration would help provide a seamless experience for users receiving fire alerts and system status updates.

2.4. Backend Performance and Security Optimization

Building the backend server requires establishing swift, dependable interactions between the AI system, hardware components, and the mobile app. Possible challenges involve handling multiple simultaneous requests, securing data transfers, and avoiding server strain during high demand. Unstable networks or poorly optimized APIs might slow alerts and weaken system reliability. To counter this, I could introduce asynchronous processing, distribute workloads, and leverage cloud scalability to manage surges. Adopting secure communication methods such as HTTPS and authentication tokens would boost protection. Frequent testing and monitoring solutions would help maintain server performance and ensure constant readiness for real-time fire detection notifications.

3. SOLUTION

The main structure of the system includes the camera, the backend server and the mobile app. The camera is the hardware portion of the system, it includes a camera, which has a built in fire detection MLM, and a gas sensor [6]. Its purpose is to detect fire. The next component is the backend server. This is where the data is processed. The backend handles requests from the hardware and the database to ensure that users receive the correct information as well as storing

user information. The last component, the mobile app, connects everything together as well as provides the user with evidence, notifications and acknowledgements.

The system begins at the mobile app for the user. The user registers and creates an account. Next, they connect their account to the camera with the camera id. By pairing the camera and account. Information from the camera will be sent to the backend server and back to the user through the mobile app. Once the camera is turned on, it will periodically capture an image and determine if there is fire or not, this is sent to the mobile app and will show them the images. It will also notify users if there is a fire.

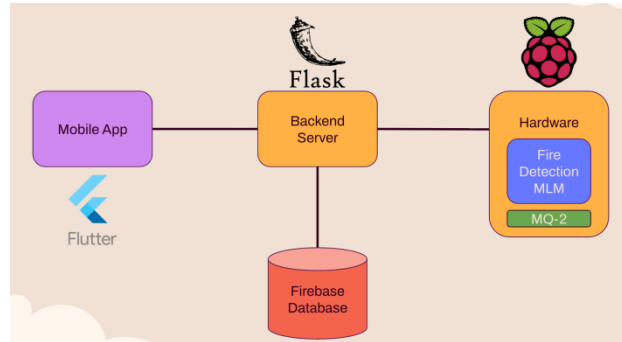


Figure 1. Overview of the solution

The mobile app is the user interface. It lets users register and pair the camera through the camera ID. It sends commands to the backend and receives results. It uses authentication push notifications and REST API calls [7]. It displays captured images and alerts. It guides setup and monitors status remotely.

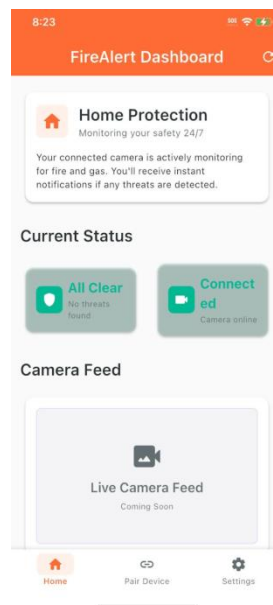


Figure 2. Screenshot of the dashboard

```

29 Future<void> checkFire() async {
30   var response = await http.post(
31     Uri.parse("${sapiUrl}/check_fire"),
32     headers: {"Content-Type": "application/json"},
33     body: jsonEncode({
34       "user_id": user_id,
35       "camera_id": camera_id,
36       "key": key,
37     })),
38   );
39
40   if (response.statusCode == 200) {
41     var body = jsonDecode(response.body);
42     var isFire = body['isFire'];
43     var isGas = body['isGas'];
44     var imageUrl = body['image_path'];
45     print("Response Body: $body");
46
47     if (isFire == null || isGas == null) {
48       PopupAlert(
49         "Fire or Gas Status Missing.",
50         "The response came in missing important data. Please try again or reconnect camera if this happens again.",
51         context,
52       );
53     }
54     return;
55
56     if (isFire == "True") {
57       _isFire = true;
58       PopupAlert(
59         "Fire Detected",
60         "A fire has been detected on your camera. It is best for you to check to ensure your house's safety.",
61         context,
62       );
63     }
64
65     if (isGas == "True") {
66       PopupAlert(
67         "Gas Detected",
68         "Gas readings have been detected. It is best for you to check to ensure your house's safety.",
69         context,
70       );
71     }
72
73     // explicitly set the state for the image to appear
74     setState(() {
75       _imageUrl = imageUrl;
76       print("Setting imageUrl: $imageUrl");
77     });
78   } else {
79     PopupAlert(
80       "Error Response: ${response.statusCode}",
81       "Please ensure your camera was properly paired. Please re-pair if this happens again.",
82       context,
83     );
84   }
85
86   setState(() {
87     _isLoading = false;
88   });
89 }

```

Figure 3. Screenshot of code 1

The checkFire() method seen in the image above is the function that runs in the background that automatically checks to see if a fire has been detected from the database. The function does so by creating a HTTP request to the backend server at the /check_fire route, which is configured to return the fire and gas status. Immediately after receiving the response, the function then moves into parsing it, checking first the status code. If the status code is anything other than 200, then an error popup alert is displayed. Otherwise, the contents of the response are parsed and statuses such as 'isFire' and 'isGas' are extracted. From there, if a positive detection for fire or gas has been identified, the user is then notified with a popup alert [8]. The function purposefully handles every case, creating proper popup alerts to notify the user of any potential miscommunications in the system to keep them completely in the loop.

The camera is a key component of ScorchVision as it detects the fire. The camera uses a custom trained machine learning model to quickly and accurately detect fires. It also has a gas sensor to detect gas leaks. The camera will constantly feed data into the backend server and notifies it when a fire or gas leak has been detected.

```

65 # update firebase by requesting to /update_fire in server to update the fire status
66 def update_fire(isFire):
67     global key
68     try:
69         # create data to send
70         data = {
71             "camera_id": camera_id,
72             "camera_key": key if key != "" else None,
73             "isGas": GPIO.input(DO_PIN), # read input and return True or False
74             "isFire": isFire
75         }
76         # print(f"Data to post: {data}")
77
78         # perform an http post request
79         files = {'image': open('capture/cap.png', 'rb')}
80         response = requests.post(
81             "https://rain-fire-detection.onrender.com/update_fire",
82             headers={"Content-Type": "application/json"},
83             json=data,
84             files=files
85         )
86         # print(f"Response Data: {response.json()}")
87
88         # check to ensure the request was successful
89         if response.status_code == 200:
90             # print("Data Successfully posted!")
91             body = response.json() # save response json as a dictionary
92
93             # parse key and set if present
94             new_key = body.get("key", None)
95             if new_key:
96                 # update the KEY environment variable and in this instance
97                 update_env("KEY", new_key)
98                 key = new_key
99                 print(f"Saved new key: {new_key}")
100
101             # check if should shutdown, then do so
102             shut_off = body.get("shut_off", False)
103             if shut_off:
104                 shut_down()
105
106             return True
107         else:
108             print(f"Issue posting data: {response.reason} {response.status_code}")
109             print(response.json())
110         return False
111     except Exception as error:
112         print(error)
113         return False

```

```

115 # main process where the image is taken and isFire is determined
116 def main():
117     try:
118         # take picture
119         frame = picam.capture_array()
120         frame = cv2.cvtColor(frame, cv2.ROTATE_180) # rotate camera
121         cv2.imwrite("capture/cap.jpg", frame)
122
123         # check to ensure the frame was saved
124         if not os.path.isfile("capture/cap.jpg"):
125             print("Issue getting frame. Frame was never saved.")
126             return False
127
128         # NOT GENERATING HEATMAP BECAUSE IT TENDS TO BE MORE ACCURATE!!!
129         # generate_heatmap("capture/cap.jpg", "capture/")
130
131         # determine isFire
132         result = predict(model, "capture/cap.jpg")
133         print(f"result: {result}")
134         isFire = True if result == "fire" else False
135
136         # update database if fire is detected
137         is_set = update_fire(isFire)
138
139         # remove picture file
140         os.remove("capture/cap.jpg") # remove file so next check will be accurate
141
142         return True
143     except Exception as error:
144         print(f"Error running main: {error}")
145         return False
146
147 if __name__ == "__main__":
148     while True:
149         if not main():
150             break
151         time.sleep(1)

```

Figure 4. Screenshot of code 2

The code in the screenshots seen above are significant portions of the software for the RaspberryPi Camera hardware. The first image, the `update_fire()` method, is responsible for receiving the 'isFire' status and sending the HTTP POST request to the backend server so it may update the value within the database. Additionally, it also grabs the MQ-2 sensor's value to see if gas has been detected, and updates that as well [9]. The image the camera has taken is also uploaded so the user can see the most recent image as well. The `main()` method is the main system loop that runs constantly to keep everything updated. It starts by taking a quick image with the camera, saving it, and then processing it with the machine learning model. After it has been predicted and found, the value is then passed to the `update_fire()` method. This loop runs with a timer offset of 1 second, this is a placeholder value and can be adjusted for better or more stable performance.

The server processes image data from the camera device using trained AI models to detect flames. The server handles sending out notifications to specific users and handles user information.

```

122 # route for the hardware to update whether a fire has been detected or not
123 @server.route("/update_fire", methods=["POST"])
124 def update_fire():
125     try:
126         # grab the camera_id and camera_key
127         camera_id = request.json.get("camera_id")
128         camera_key = request.json.get("camera_key")
129         fire_status = request.json.get("isFire")
130         gas_status = request.json.get("isGas")
131
132         # TODO: fully implement managing image file
133         image = request.files.get("image", None)
134
135         # print(f"Camera_id: {camera_id}\nCamera_key: {camera_key}\nFire_status: {fire_status}")
136
137         if not camera_id or fire_status is None or gas_status is None: # have to be specific
138             return jsonify({"error": "Missing info for updating."}), 404
139
140         doc = get_document("fire_cameras", camera_id)
141
142         # handle if there is no camera_key
143         if camera_key is None:
144             # check to see if the key is empty in the doc
145             if doc["camera_key"] == "":
146                 # generate a unique key
147                 new_key = str(uuid.uuid4())
148
149                 # set the new camera_key in the database and update isFire and isGas
150                 update_document("fire_cameras", camera_id, "camera_key", new_key)
151                 update_document("fire_cameras", camera_id, "isFire", fire_status)
152                 update_document("fire_cameras", camera_id, "isGas", gas_status)
153
154                 # place image in storage if sent over
155                 if image:
156                     image_path = f"fire_camera/{camera_id}/image.png"
157                     place_file_into_storage(image_path, file)
158                     update_document("fire_cameras", camera_id, "image_path", image_path)
159
160                 # return the status
161                 return jsonify({"status": "success", "key": new_key, "shut_off": doc.get("shut_off", False)})
162             else:
163                 return jsonify({"error": "Missing key."}), 404
164         else:
165             # handle if the keys match
166             if camera_key == doc["camera_key"]:
167                 update_document("fire_cameras", camera_id, "isFire", fire_status)
168                 update_document("fire_cameras", camera_id, "isGas", gas_status)
169                 return jsonify({"status": "success", "shut_off": doc.get("shut_off", False)})
170             else:
171                 return jsonify({"error": "Incorrect key."}), 404
172
173     except Exception as error:
174         return jsonify({"error": f"Error updating: {error}"}), 404
175

```

Figure 5. Screenshot of code 3

The /update_fire route in the image is responsible for the receiving of data from the user, the verification of the user's credentials, and the proper updating of the database with the appropriate values. After parsing the values from the request and receiving the image, the 'camera_key' is checked to ensure it is present and correct. If the key is not present, then it is assumed that the user is trying to pair their camera, so the key is checked from the backend to see if it has been set previously. If it has, obviously, the user is informed that that is an invalid operation. After verifying or updating the key, the values are then placed into the database.

The Machine Learning Model is used to detect flames. It is trained on a large collection of images with and without fire. On average over a collection of tests, the model brags a solid 98% accuracy rate, able to identify even tiny fires so long as it is present within the image and visible.

```

13 # function to train the model
14 def train_model(model, dataset_path="fire_dataset", epochs=10, batch_size=32, lr=0.001):
15     # epochs: the amount of times the ML model will be trained on images
16     # batch_size: the amount of data the ML model will be trained on during each epoch
17     # lr or learning rate: rate at which the ML model will make tweaks to become more accurate
18
19     # configure gpu if available
20     device = "cuda" if torch.cuda.is_available() else "cpu"
21     model.to(device) # move the model onto the gpu or cpu
22
23     # process for cleansing and preparing our dataset
24     transform = transforms.Compose([
25         transforms.Resize((224, 224)),
26         transforms.ToTensor(),
27         transforms.Normalize(mean=[.5], std=[.5])
28     ])
29
30     # load the data and transform it
31     dataset = datasets.ImageFolder(root=dataset_path, transform=transform)
32     # configure train and testing size
33     train_size = int(.8 * len(dataset))
34     test_size = len(dataset) - train_size
35     print(f"Length of Dataset: {len(dataset)}\nLength of Train Size: {train_size}\nLength of Test Size: {test_size}")
36     train_dataset, test_dataset = random_split(dataset, [train_size, test_size]) # split the dataset into the train and test
37
38     train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
39     test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=True)
40
41     # configure our loss and optimizer
42     criterion = nn.CrossEntropyLoss()
43     optimizer = optim.Adam(model.parameters(), lr=lr)
44
45     # training loop
46     for epoch in range(epochs):
47         print(f"### Epoch: {epoch} ###")
48         model.train() # configure and prep model for training
49         loss = 0.0
50         for images, labels in train_loader:
51             images, labels = images.to(device), labels.to(device) # move the images and labels onto gpu if necessary
52
53             optimizer.zero_grad()
54             output_labels = model(images) # output labels for the prediction
55             image_loss = criterion(output_labels, labels) # compare the output labels to labels and also determine the loss
56             image_loss.backward()
57             print(f"Image Loss: {image_loss}")
58             optimizer.step() # self reflective step to actually learn
59             loss += image_loss.item()
60         print(f"Epoch Loss: {loss}")
61
62         # loop for testing immediately after each training loop
63         model.eval() # configure and prep the model for evaluation
64         correct = 0
65         total = 0
66         with torch.no_grad():
67             for images, labels in test_loader:
68                 images, labels = images.to(device), labels.to(device)
69                 output_labels = model(images)
70                 _, predicted = torch.max(output_labels, 1)
71                 total += 1
72                 correct += (predicted == labels).sum().item()
73
74         accuracy = 100 * (correct / total)
75         print(f"Model Accuracy: {accuracy}\n")

```

Figure 6. Screenshot of code 4

The `train_model()` function is responsible for the training of whatever model is passed to it, given the number of epochs and `batch_size` as parameters. The model passed to this function is a Affine Linear model from PyTorch's neural network modules collection. This model seemed to perform the best in our testing. The method starts by identifying if there is a 'cuda' or GPU option available on the OS, as training takes a long time on the CPU, but isn't unbearable [10]. For the actual training, the data transformer is initialized firstly and the dataset is loaded from the `dataset_path`, as all of the training data were images received from Kaggle. The data is then split for training and testing and the `DataLoaders` are initialized. Finally, the epochs are iterated through, and with each iteration the model is trained on the training data and the loss is calculated. Each iteration also validates the model, displaying the new accuracy scores so the trainer can keep track of how the model is performing.

4. EXPERIMENT

4.1. Experiment 1

AI accuracy, this is important for this system because this will increase the chance that fire is correctly identified and that the chance for false alarms will be decreased.

To test the AI's accuracy, we can use a set of 999 images of fire and non-fire to test the accuracy. Then, the AI model will be tested using these images to identify whether these are fire or non-fire. After the AI has sorted the images into the 2 categories, the correct answer will be revealed. To present the data, use a confusion matrix where we have the Actual and the predicted.

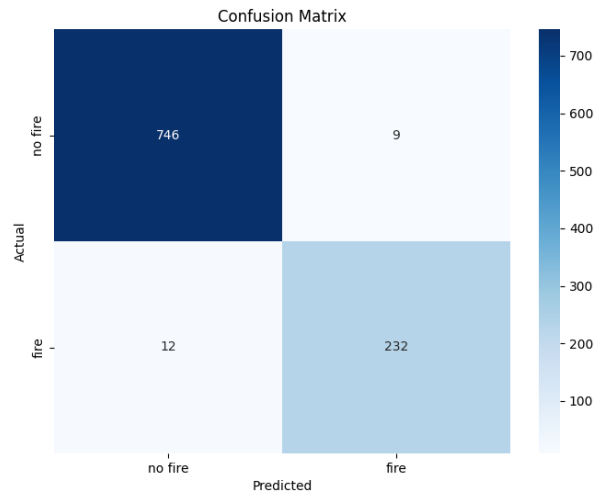


Figure 7. Figure of experiment 1

Overall, the AI model is very successful at identifying fires. This is evident from our results. Out of the 999 images tested, 978 images were correctly categorised. This gives us an estimated accuracy around 97.9%. Additionally, the AI model is more accurate at predicting non-fire images with an accuracy of 98.4% compared to predicting fire at 96.2%. According to current literature, a good accuracy for a fire detection model is from 95-99%. Our model falls in the upper range, meaning it is satisfactory. I think our model was successful because we trained it across a large sample size. Along with that, the samples were split into reasonable batch sizes and trained across multiple epochs. I believe the greatest effect on the results are the sample quality of the images used to test.

4.2. Experiment 2

Another issue that we foresaw with the ScorchVision system was the total update time of the system. If the update time, the total time it takes for the user to get an update from their hardware and see it within the app, is too long, that could seriously impact not only the public perception of the app but also potentially put houses in danger if not enough time is given to react.

To find a good metric of the update time, finding the upload time from the hardware to the backend server and database is a great metric as it is indicative of the raw time for an update. The time it takes for the mobile device to grab the status from the backend server is not a very good metric because not only is the user moving around constantly with their phone, but they are also just grabbing statuses from the database and not performing any major processing. Because of this, we will measure the total time it takes for the hardware to update to the backend and find the average update time.

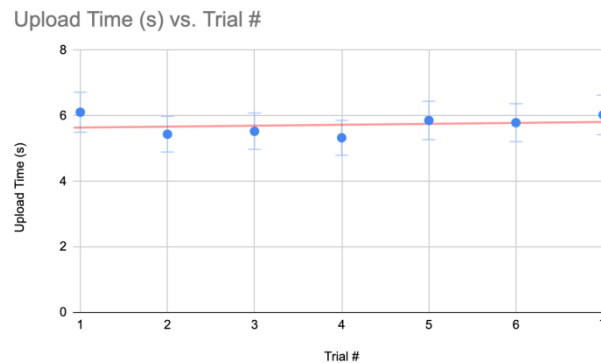


Figure 8. Figure of experiment 2

After performing the experiment it is clear that the updating time is nowhere near too long, as the update time is consistently under 10 seconds with an average time of 5.7 seconds. Considering all that the hardware is doing, including taking a picture, processing it with the machine learning model, and uploading the results and image to the backend, this is actually quite fast and efficient. What surprised us the most with this experiment was the average upload time, as we imagined it was going to be much higher. However, we believe it is as quick as it is simply due to the fact that the process was built with efficiency in mind, specifically in how it reuses the same objects throughout the program and avoids copies as much as possible.

5. RELATED WORK

In 2020, research conducted by Zhang et al. demonstrated a residential fire detection approach employing convolutional neural networks (CNNs) to recognize flames and smoke from live video streams [11]. Their framework analyzed visual input via a trained artificial intelligence algorithm, categorizing fire imagery and notifying occupants through IoT-linked alerts. Performance reached roughly 92% precision, showing dependability in regulated indoor conditions. Yet, efficacy declined under inadequate illumination or when smoke partially concealed flames. Further, consistent internet access was mandatory, and fires beyond the camera's scope remained undetected. My initiative, ScorchVision, refines this by refining the AI model for quicker edge computation, incorporating offline detection features, and merging thermal with optical inputs to lower incorrect dismissals and bolster initial fire identification.

A 2024 study by Vorwerk et al. developed an IoT-based multi-sensor fire detection system that uses transfer learning to classify early fire signals [12]. The researchers combined temperature, gas, and smoke sensors linked to a neural network model trained to distinguish real fires from false triggers. Their system achieved over 95 % accuracy and demonstrated strong potential for early fire detection. However, it required extensive calibration and struggled with long-term reliability due to sensor drift and environmental noise. It also relied on dense sensor networks, increasing cost and complexity. My project, ScorchVision, enhances this concept by integrating camera-based AI for visual verification and edge processing, reducing both hardware cost and false positives while improving scalability and deployment simplicity.

A 2023 paper by Kamilaris explored how mobile applications can help people detect and escape wildfires in real-time [13]. The app integrated satellite fire data, GPS tracking, and alert notifications to guide users safely during emergencies. It was effective for large-scale outdoor fires, improving situational awareness and communication. However, its accuracy depended heavily on network connectivity and satellite data frequency, making it unsuitable for fast-

developing indoor or household fires. It also lacked built-in image recognition or automatic detection. My project builds upon this by providing real-time camera-based detection directly within a mobile app, allowing users to receive instant, locally generated alerts even without internet access — combining visual AI detection with a user-friendly safety interface.

6. CONCLUSIONS

ScorchVision has multiple limitations currently as it is still in its early developmental stages. The first, and most prominent, issue with the current hardware is the inconsistency of the fire recognition machine learning model as it occasionally has a false positive on what could be sunlight, a very bright screen, or even smoke from safe cooking. Solving this issue would look like finding more data and carefully crafting the training dataset to account for all of these scenarios [15]. Additionally, the Raspberry Pi camera doesn't have a high picture quality, and can therefore miss things in low light or that are far away. Getting a better camera, or an attachment onto the Raspberry Pi camera could help this. Another potential issue that could keep a user from hopping into ScorchVision are the privacy concerns. The system does include having a camera right in the middle of the users home, which can scare some concerned about privacy. Because of this, we have made the ScorchVision system source code completely available on Github, however, that is not a solution for those not as technically inclined. An alternative to this could be designing the system to be easy to deploy and host locally for a not so tech-savvy user. Finally, the last issue of ScorchVision is the lack of direct intervention of the hardware. It would be awesome if the hardware was able to deploy some sort of fire suppression system, however, if it were to do so, the accuracy of the model would need to be much higher first, as close as we could get to 100% as possible.

To conclude, house fires and kitchen fires have been an ongoing issue for the last century and ScorchVision effectively tackles this. ScorchVision is a solution that integrates both hardware and software as a system, using a custom trained machine learning model and accompanying hardware to send notification updates to the users mobile phone.

REFERENCES

- [1] Rahim, M. S. N. A. "The current trends and challenging situations of fire incident statistics." *Malaysian Journal of Forensic Sciences* 6.1 (2015): 63-78.
- [2] Mufid, Mohammad Robihul, et al. "Design an mvc model using python for flask framework development." 2019 International Electronics Symposium (IES). IEEE, 2019.
- [3] Runefors, Marcus, Nils Johansson, and Patrick van Hees. "The effectiveness of specific fire prevention measures for different population groups." *Fire Safety Journal* 91 (2017): 1044-1050.
- [4] Sullivan, Emily. "Understanding from machine learning models." *The British Journal for the Philosophy of Science* (2022).
- [5] Ketkar, Nikhil, and Jojo Moolayil. "Introduction to pytorch." *Deep learning with python: learn best practices of deep learning models with PyTorch*. Berkeley, CA: Apress, 2021. 27-91.
- [6] Relan, Kunal. "Beginning with flask." *Building REST APIs with flask: create python web services with MySQL*. Berkeley, CA: Apress, 2019. 1-26.
- [7] Chougale, Pankaj, et al. "Firebase-overview and usage." *International Research Journal of Modernization in Engineering Technology and Science* 3.12 (2021): 1178-1183.
- [8] Ghael, HIRAK DIPAK, L. SOLANKI, and GAURAV SAHU. "A review paper on raspberry pi and its applications." *International Journal of Advances in Engineering and Management (IJAEM)* 2.12 (2020): 4.
- [9] Aung, Soe Thandar, et al. "A study of learning environment for initiating Flutter app development using Docker." *Information* 15.4 (2024): 191.
- [10] Hernandez-Matas, Carlos, et al. "FIRE: fundus image registration dataset." *Artificial Intelligence in Vision and Ophthalmology* 1.4 (2017): 16-28.

- [11] Huang, Lida, et al. "Fire detection in video surveillances using convolutional neural networks and wavelet transform." *Engineering Applications of Artificial Intelligence* 110 (2022): 104737.
- [12] Vorwerk, Pascal, et al. "Classification in early fire detection using multi-sensor nodes—a transfer learning approach." *Sensors* 24.5 (2024): 1428.
- [13] Kamilaris, A., et al. "Examining the potential of mobile applications to assist people to escape wildfires in real-time." *Fire Safety Journal* 136 (2023): 103747.
- [14] Michael, Katina, Andrew McNamee, and Michael G. Michael. "The emerging ethics of humancentric GPS tracking and monitoring." 2006 International Conference on Mobile Business. IEEE, 2006.
- [15] Takano, Nao, and Gita Alaghband. "Srgan: Training dataset matters." arXiv preprint arXiv:1903.09922 (2019).