# IoT Smart Mirror

Jeffrey Murray Jr.
*M.S. Cybersecurity Engineering*
*University of Washington - Bothell, WA*

***Abstract*—IoT devices are growing rapidly with consumer products such as TVs, cameras, locks, lights, and many more. Google, Amazon, Apple, and Microsoft have released some form of organizing and controlling devices on the Local Area Network via a mobile application and/or voice assistant. Devices that have an embedded virtual assistant have limited security capabilities for authorizing active users and executing commands. This work contributes an open-source, Google Assistant SDK with Facial Recognition. The Smart Mirror is cost-competitive and provides similar functionality to current products on the market. Using Google Cloud Platform, the device can respond to the user in less than a second and handle local execution commands while relying on existing infrastructure to control other smart home devices.**

## I. Introduction

Smart Home Devices have become increasingly popular in the last decade alongside the need to organize, connect, and communicate with them. In the last 5 years, large companies have released numerous products to address this need for users, however, a problem remains. Personal Voice Assistants embedded within smart home devices are triggered via wake words, and the concern is that unauthorized users can access sensitive data and/or control network-connected devices [12].

In recent years, Amazon and Google have proposed solutions via voice recognition, face recognition, two-factor authentication, and secret phrases or pins [2]. The application of voice and facial recognition allows households with more than one member to access personal information that does not conflict with other users. The Smart Mirror approaches this problem with a pre-trained facial recognition model running on a Jetson Nano. It publishes login and logout events to the Google Cloud Platform and a Raspberry Pi 4 acknowledges the message. Once acknowledged this application will provide personalized content and access to the embedded Google Assistant and subsequent user data. The device integrates into the Home Graph API provided by the Google Cloud Platform which the user can access on their Google Home App.

This work proposes an open-source implementation of an embedded Google Assistant activated by a local wake word engine, Porcupine with custom user profiles authorized by facial recognition. The application can also be controlled by the Google Home App or other Google Assistant instances to control the Smart Mirror. Section II provides a feature-based comparison of similar devices on the market. Section III discusses privacy concerns with existing smart home devices. Section IV proposes the design and implementation of three modules, cloud software, and facial recognition. Section V evaluates the final product on the responsiveness to authentication events. Section VI summarizes the key points and takeaways of this implementation.

## II. Related Work

There has been widespread adoption and integration of Virtual Personal Assistants (VPAs) in the last decade with Amazon and Google taking the lead in this growing market [6]. In this section, the Smart Mirror features and functionalities will be compared to similar devices available in the market. The devices shown in Figure 1 demonstrate the market share with an estimated price, year of product release, facial recognition option, the type of voice assistant integrated, and the key features.

### A. Google Assistant

Google debuted their digital assistant at the Google I/O Developer conference in 2016, and rapidly grew each year to include more user-focused features such as Voice Match and Face Match to deliver more personalized content. Along with this, Google Home has provided a developer-friendly cloud service to incorporate almost any device, going as far as embedding the assistant service into the device's source code. The integration options have paved the way for the rapid growth of compatible devices with Google Assistant and Google Smart Home. The embedded assistant requires a Speech-to-Text and Google Assistant API to be enabled in the cloud, and out of the box, it does not include any way to trigger the assistant via voice commands.

However, Google has released products with their assistant such as Pixel phones, Chromecast, and Nest Smart Home. Only recently, Google has released Face Match to personalize user content displayed on compatible products. It works alongside Voice Match to drive the functionality of identifying the user speaking to the device. However, Google notes that this is not a security feature, rather automating content view based on user's routine patterns. It cannot differentiate between a photo of a user and the user standing in front of the device.

The Google Nest Hub Max includes a webcam and microphone for users to video chat with friends and family, and optionally can enable Face Match and Voice Match if the device is in a shared household where personalized content is desired. Not to mention the previous generation, the second generation of the Nest Hub smart display does not include a webcam, but it can make voice calls and utilize Voice Match to personalize content. Both devices can handle local intents such as controlling the volume, media playback, and display content. For both devices, Face Match and Voice Match are

| Device | Est. Price* | Release | Facial Recognition | Voice Assistant | Function | User-Interface |
|---|---|---|---|---|---|---|
| *DIY Smart Mirror* | $220 | 2021 | DLIB, OpenCV | Google Assistant, Amazon Alexa | Volume, Screen, Smart Home | Open-source, Configurable |
| *10" Lenovo Smart Display* | $250 | 2018 | None | Google Assistant | Video Chat, Volume, App Selector, Smart Home | Touchscreen, Embedded, Dynamic View |
| *Google Nest Hub Max* | $229 | 2019 | Face Match | Google Assistant | | |
| *Nest Hub (2nd gen)* | $100 | 2021 | None | Google Assistant | | |
| *Echo Show 8 (2nd gen)* | $130 | 2019 | Visual ID | Amazon Alexa | | |
| *Echo Show 10 (3rd gen)* | $250 | 2021 | Visual ID | Amazon Alexa | | |
| *Echo Show 15* | $250 | 2021 | Visual ID | Amazon Alexa | | |

** Prices may vary based on location and seasonal sales*

Fig. 1. Feature Comparison of similar products on the market.

trained locally on the device and sent to the cloud, opposed to the Smart Mirror which does local training and storage for both the Wake Word and Facial Recognition.

### B. Amazon Alexa

Amazon released the first integration of Alexa in 2014 with the Amazon Echo smart speaker to provide results for web searches, ordering products on Amazon Marketplace, and limited capability for IoT Smart Hub to control other devices via the user's voice. The Echo Show was the first Smart Home Display to debut in 2017 with their newest iteration in 2022, the Echo Show 15. There have been many iterations of the Echo Show, but more recently the inclusion of Visual ID to personalize the user experience.

The Visual ID function will also be released for the Echo Show 8 and Echo Show 10. Opposed to Google, images of user faces will be stored and trained on locally. Like Google and the Smart Mirror, facial recognition cannot discern a user from a photo of them, so it cannot be justified as a secure method of authorizing users. Instead, Amazon is using this technology to personalize content such as calendar, upcoming events, and to do lists. All of Amazon's 2021 smart home displays in Figure 1 are compatible with upcoming rollout of Visual ID to locally identify a user based on their face to personal content while in a household with more than one member.

In 2019, Amazon unveiled their Amazon Voice SDK [1] to enable developers and 3rd party companies access to embed the assistant to various applications. AVS is an all-inclusive embedded assistant, for it includes a wake word engine and seamless authentication for users and developers. However, the integrated wake word utilizes a shared data stream with the Alexa Communication Library which forwards the processed input to cloud services. The concerning aspect of this data flow is that the wake word engine requires connection to the internet [8]. Comparatively, the wake word engine on the Smart Mirror does not require an internet connection, and thus does not pose a privacy concern.

## III. Problem Statement

With Smart Home Devices gaining much popularity in recent years, a shared flaw of the current market is in the Wake Word Engine, a constantly listening data stream awaits a trigger word to handle a user command. As this wake word is always listening poses itself as a privacy concern [8], [12] and security issue for unauthorized users to execute commands on the device [3].

The proposed Smart Mirror utilizes open-source software to address these concerns with competing devices on the market. Embedded smart home devices currently listen for the incoming request, and the user has no control over where the data is going. In this regard, the Smart Mirror is equipped with a local wake word engine that does not transmit any data. This approach ensures that there are no saved recording or analytics being captured while inactive.

The system relies on Facial Recognition from another device to only allow authorized users access to the embedded assistant and to sensitive user data. Other devices will activate when any user says the wake word and only after tries to personalize the displayed content and/or verbal response. The Smart Mirror performs the opposite operation where the only pre-authorized users have access to the embedded assistant where they can access their sensitive data and control other devices. For performance, the overall design focuses on resource consumption by utilizing two devices, Raspberry Pi 4, and Jetson Nano 2 GB, to share the required computing resources to run this system. To draw an evaluation for the proposed system, the Smart Mirror will be analyzed on its responsiveness to user commands and authentication latency.

## IV. Design and Implementation

The overall architecture and data flow of this system has been illustrated in Figure 2. The Smart Mirror software contains three sub modules and require the prefix 'MMM' designated for extensions of the Magic Mirror repository [10]. This open-source, modular smart mirror platform provides a bare bones user interface running as a NodeJS electron application. Each sub module is dependent of the others, for each module communicates with a single component in

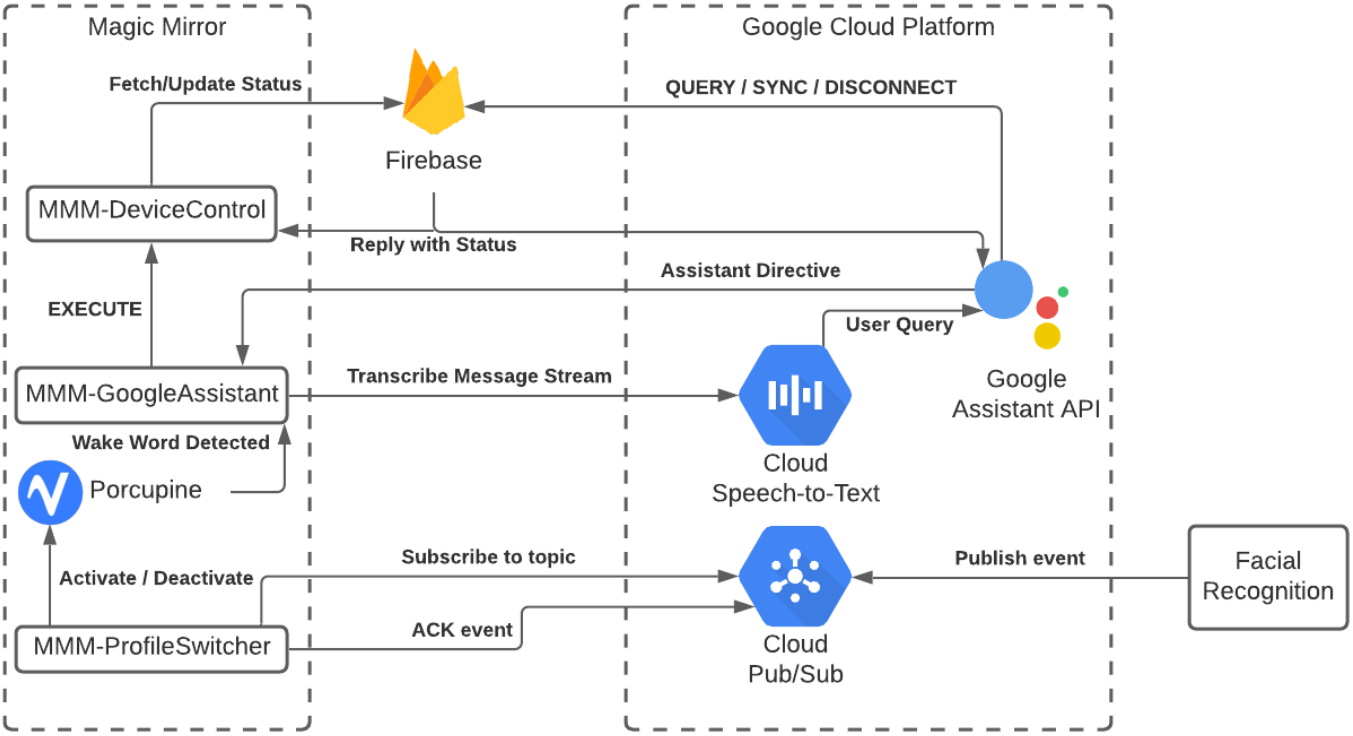


Fig. 2. Smart Mirror Data Flow

the cloud. For the back-end service provider, Google Cloud Platform handles user requests to generate directives with the Google Assistant API, act as a broker for incoming login and logout events with Google Pub/Sub API and maintain the Smart Mirror state and actions with Firebase.

### A. *Software*

*1) Google Assistant:* This module was forked from MMM-Porcupine [9] by Alex Sikand that contained depreciated version of Porcupine and limited functionality. MMM-GoogleAssistant extended the previous work by embedding the Google Assistant SDK [4] and importing the most recent Porcupine deployment. Porcupine [6] is a pre-trained, local wake word engine that does not require any network connectivity. It has 14 pretrained wake words including but not limited to: “Jarvis”, “Alexa”, “Hey Google”, and “OK Google”. The open-source software allows for multiple wake words at the same time and provides a fluid handoff to trigger the Google Assistant SDK. As it is a constantly listening data stream, it decodes the bit stream in 1 second intervals searching for the desired wake work. Once detected, it closes the stream and activates the Google Assistant SDK.

The Google Assistant SDK requires the user to authenticate themselves on the platform to access the Speech-to-Text, Google Assistant, and Home Graph APIs. All services have access to sensitive user data and profile settings. The Speech-to-Text API translates audio data to text and is known as a transcription, and once the user stops talking triggers an “end-of-utterance” command on the Google Assistant SDK. The intent is processed in the cloud via the Google Assistant API and generates a directive within the cloud platform. Intents are actionable items that the user would like to execute, and the back-end system handles directives for maintaining the device’s state, accessing other devices via Home Graph API, and requesting other services such as games or queries with the Google Assistant API. For directives that are seeking to modify or seek resources such as other smart devices or the current device, an intent has a design pattern of “devices.action.XYZ” where XYZ is a command.

The Home Graph Intent design pattern includes the following commands: SYNC, QUERY, DISCONNECT, EXECUTE. The sync intent requests the list of devices associated with the given user and their capabilities, and it is usually triggered during account linking via OAuth 2 token generation. The sync can also request to update the attributes and traits of the device, so that when the Firebase Function is updated will trigger changes in the Realtime Database. The query intent requests the current state and status of the device, and it acts as a read only action. The disconnect intent informs the back-end service that the user has unlinked, or unauthorized, access to the user’s data.

For an execute intent the Smart Mirror handles it locally with the Assistant SDK and updates the Firebase Realtime Database. From other Google Assistants connected to the same user account accesses the web hook hosted by the Firebase Function. A single execute intent can only target one device at a time. Such that, when the user says, “Turn off the lights”, N number of requests are sent for each light in the

room. For the implementation of execute, the Assistant SDK identifies “action.devices.EXECUTE” in the intent parameters and handles the request locally by passing the command to MMM-Device Control.

*2) Device Control:* This module is responsible for requesting the current status of the Smart Mirror from the Firebase Realtime Database. These monitors, known as daemons, ping the server every three seconds, and if the status has changed, runs a command on the Raspberry Pi. This feature is required for the user to control the device from other Google Assistant instances. The traits of the Smart Mirror are Volume and OnOff. Each request is handled asynchronously by each daemon: Volume, Display, and Mute. Volume and Mute must be handled separately, as they are two different intents for the Google Assistant API and access different states in the Realtime Database.

The Volume daemon stores a local integer variable with a value between 1 and 100, it compares the current state and state in the database, if they do not match, updates the local state to match and executes a command. On the raspberry pi, to change the volume via the command line, ”alsa mix” can modify the volume level. A typical request will contain *isPercentage* within the body of the request. If true, the command uses the raw value in the *volumeLevel* key, otherwise the value is a scale within 1 to 10. The corresponding percentage is then stored locally and updated in the database.

The Mute daemon checks the database value *isMuted* and it is read only. When the user intends to mute the Smart Mirror, the daemon executes the ”alsa mixer” to mute. No local value is stored, for the Realtime Database stores two values under the Volume trait: *isMuted* and Mute. Mute acts as a command and *isMuted* reflects the changes of the command. When the user intends to unmute the Smart Mirror, the Google Assistant API updates the database *isMuted* value, and the daemon executes the “alsa mixer” to unmute. Note that all mute and unmute intents are not handled by the Assistant SDK, rather passed to the Assistant API in the cloud platform to make changes to the Realtime Database.

The Display daemon checks the database value under the *OnOff* trait. The Smart Mirror screensaver can be controlled with ”xscreensaver-command” and sets the screen to black to appear turned off. The screensaver has been configured to automatically turn on after thirty minutes of inactivity, and it can be turned off when a user walks in front of the mirror and a login event is published or the Realtime Database value changes.

*3) Profile Switcher:* MMM-Profile Switcher [5] was forked from Brian Janssen, and acts as a user session manager for login and logout events. The schema of the profile switcher attaches to each DOM object in the global configuration. For example, the Google Assistant module can only be activated when known users are recognized. The default page only displays the current time and date, but when a user is recognized, the profile switcher enables all DOM elements associated with the user.

The Profile Switcher must be pre-configured with names of authorized users. For example, when a face is recognized on the Jetson Nano, it sends a publish event with “User Active”, the message is acknowledged by the Profile Switcher. The module then broadcasts the change and activates one or more DOM elements for the main user-interface based on the config.json file. Each module and DOM element must be labeled with a name or group, otherwise it will be enabled by default. The transition from one user to another is almost instantaneous, for every module and corresponding position is static and this module just changes its visibility status.

*4) Firebase Function:* The function only interacts with external parties such as the Google Home and other Google Assistant instances on the user account. When deployed, it will send a SYNC request to the Home Graph API to ensure that both have an up-to-date copy of the Smart Mirror traits and current state. This external dependency can be deployed from any device, and it serves as a webhook for Google Assistant and Home Graph APIs to interact with. It hosts */login*, */fakeauth*, and */faketoken* endpoints when a user connects their Google account to the Smart Mirror. The */reportstate*, */requestsync*, and */smarthome* endpoints handle incoming requests and reply with the Realtime Database values for the corresponding device id. The schema of the Realtime Database resembles the following parameters:

- Trait: OnOff
  - State: on – True or False
- Trait: Volume
  - Command: Mute – True or False
  - State: isMuted – True or False
  - Command: relativeSteps – 0 to 10
  - State: volumeLevel – 0 to 100

Each trait has numerous attributes that are preconfigured with a Firebase function that handles SYNC requests at the endpoint, requestsync, to reply with the device’s traits, attributes, and state. The report state endpoint handles incoming QUERY and DISCONNECT requests.

*5) Facial Recognition:* This software runs as a standalone application that is pretrained with DLIB with an accuracy of 99.38% from the benchmark Labeled Faces in the Wild [7], [11]. It stores local encodings with OpenCV from labeled images and loads them upon initialization. This implementation accesses the webcam’s data stream and looks for faces on every other frame to lower the workload. When a known face encoding is detected, it transmits a login or logout event based on user state. The user state is stored locally as a Python object. It also has a configurable user session variable, set to ten seconds, where once a user enters, it will not send a logout request for at least ten seconds. For communicating with Google Cloud Platform Pub/Sub, the software only publishes events via a service key stored as an environment variable. The user session variable is a minimum logout threshold that ensures that each logout event is at least ten seconds after a login event. The login is handled instantly if no other users are logged in. The ten second threshold is included in the

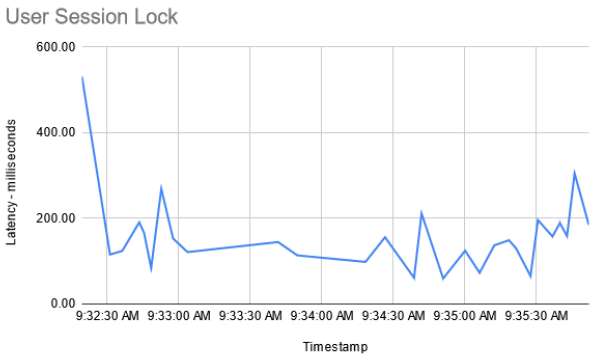


Fig. 3. Publish / Subscribe with User Session Lock of 10 seconds

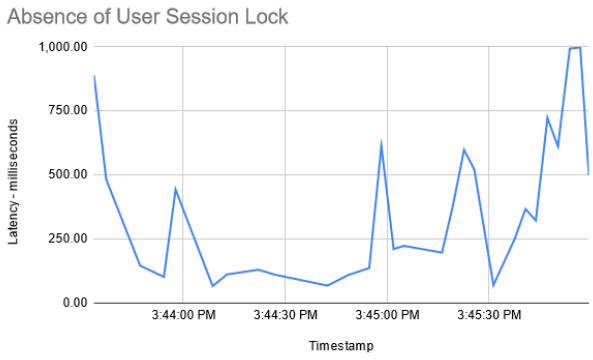


Fig. 4. Publish / Subscribe with no User Session Lock

evaluation showcasing why there must be some threshold to not overflow the Smart Mirror.

### *B. Hardware and Resource Consumption*

There are two main hardware components to create a facial recognition smart mirror: Raspberry Pi 4 and Jetson Nano 2 GB. The Raspberry Pi runs the Magic Mirror software and sub modules, and it has an external microphone and portable monitor with a built-in speaker. This allows the user to interact with Magic Mirror and control the device with his or her voice. The Jetson Nano runs the facial recognition software and when initialized loads or saves local face encodings, and has a USB webcam plugged in.

In this design, resource consumption was a crucial consideration of why the design includes two devices opposed to one. The Magic Mirror software idles at about 60% CPU consumption with a 4 core Raspberry Pi 4. The Facial Recognition in Python uses around 50% of the Jetson Nano's 4 core CPU and caches the facial encodings consuming about 1 GB of RAM.

Both devices together allow plenty of overhead for processing system requests and event driven requests. Furthermore, each device is equipped with a fan to cool down the CPU to avoid overheating. An extension to this work would be monitoring the temperature for both devices and alert the user when approaching a dangerous threshold, and power down the devices.

## V. Evaluation

The contribution of this work focuses on the responsiveness and length of user sessions. A quantitative metric for the systems robustness is by measuring the time of login and logout events. To set this experiment up, the Jetson Nano outputs each published event with a current timestamp, and on the Raspberry Pi, outputs each received event to a text file. A python parser would concat both files and compare matching events. The difference between each event is measured in milliseconds and graphed over the duration of the experiment. The Raspberry Pi 4 has a wireless network adapter, and the Jetson Nano must be connected via Ethernet.

The environment consisted of the Jetson Nano connected to the Raspberry Pi's Ethernet adapter and forwarded a sub

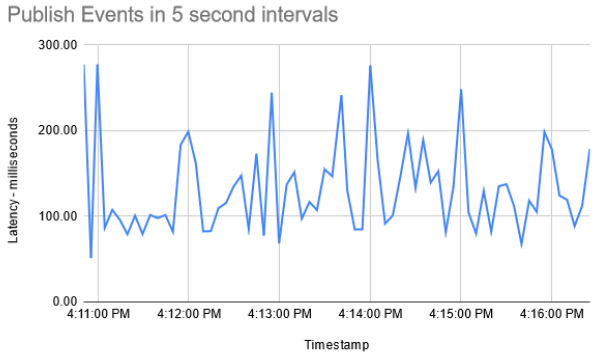


Fig. 5. Publish Events in 5 second intervals

network IP address via Network Address Translation. This 'daisy chain' allows for only a single wireless connection from the Raspberry Pi and both devices can communicate with the Google Cloud Platform.

The first experiment measured the 'sunny day' case of user sessions with a minimum session threshold of 10 seconds. This experiment yielded an average latency of 146.7 milliseconds and standard deviation of 92.9 milliseconds. The fastest response was 59.5 milliseconds and slowest response time was 531.3 milliseconds. Shown in Figure 3, the experiment lasted 3 minutes with 28 events recorded. The average user session was 11 seconds and average inactivity was 6 seconds.

From these results, the user session lock allows for more consistent publishing of events. The Profile Switcher is configured to poll for new messages to acknowledge every second, so by halting a flood of notifications at once, it allows the system to quickly respond to incoming messages.

The second experiment measured the rapid response time with the removal of the minimum session duration. With the removal of the session lock, there was a larger stream of unpredictable events. This resulted in an average latency of 370.7 milliseconds and less consistent records with a standard deviation of 285.9 milliseconds. The fastest response time was 66.3 milliseconds and maximum latency of 997.7 milliseconds. Shown in Figure 4, the experiment lasted 2 minutes and 30

seconds with 28 events recorded.

By removing the user lock, the subscriber is constantly processing new messages that adjust the user-interface, which slows down the response time of the request. With a maximum response of almost 1 second, the publisher sent 5 requests within 10 seconds, and due to this, was unable to promptly acknowledge all of them. The subscriber is only able to process one message at a time, so that there is not a conflict of resources. If the subscriber was able to acknowledge two login requests at the same time, it could potentially crash the system or show incorrect user data.

The third experiment measured a consistent stream of login and logout events every 5 seconds. This tests the responsiveness of the application with consistent publish events. On average events were processed within 132.1 milliseconds with a standard deviation of 53.7 milliseconds. The fastest response time was 51.1 milliseconds and maximum latency of 277.6 milliseconds. Shown in Figure 5, the experiment lasted 5 minutes and 30 seconds with 68 events recorded.

To stress the system latency, an event was published every 5 seconds and swapped user status. This method proved to be sufficiently handled by the subscriber, for it had time to acknowledge each request when it arrived.

## VI. Conclusion

The purpose of this work was to create a system utilizing open-source tools to deliver a customizable user interface and add an additional security layer before accessing sensitive data. The Smart Mirror uses Facial Recognition to label faces standing right in front of the device and allows for users to customize their individual content to interact with voice control. This product is privacy preserving for it conducts facial recognition locally, and uses a pre-trained, local wake word engine to activate the assistant. With Google Cloud Platform, the mirror can authenticate the user in less than a second and display their customized interface. The hardware used for this project poses a competitive price point to similar products and the software is compatible with many other devices to lower the overall cost. This project was successful because it is able to handle voice commands, form intents to control the screen and volume, and provide almost immediate responses played back over the speaker.

## VII. Future Work

There are many active developers around the world contributing to the open-source framework, Magic Mirror. Developers are working on new modules that integrate into the framework, and extend the current functionality of this project. Many modifications and improvements can be made to the Smart Mirror implementation by creating a more end-to-end system that would be easier to maintain and serve as an acceptable product to bring to market. To extend functionality, the Firebase Function would need to include the App Selector trait with Media Playback. This would require integration with 3rd Party services and generate authorization tokens to access user data and content on such platforms. Additionally, the embedded assistant can be swapped with other virtual assistants such as Amazon Alexa, Apple Siri, and Microsoft Cortana. This would provide a larger market of users and functionality to further customization of the device. The current implementation depends on the Google Home app to control the device, and an extension of this product would be a smart phone application to control the device directly including onboarding new users, customizing the user interface, and controlling the device.

## References


[1] Amazon. Alexa Voice Service (AVS) Device SDK, November 2021.

[2] Jide S. Edu, Jose M. Such, and Guillermo Suarez-Tangil. Smart Home Personal Assistants: A Security and Privacy Review. *ACM Computing Surveys*, 53(6):1–36, February 2021.

[3] Pardis Emami-Naeini, Henry Dixon, Yuvraj Agarwal, and Lorrie Faith Cranor. Exploring How Privacy and Security Factor into IoT Device Purchase Behavior. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, Glasgow Scotland Uk, May 2019. ACM.

[4] Google. Google Assistant SDK, June 2019.

[5] Brian Janssen. MMM-ProfileSwitcher, January 2017.

[6] Alireza Kenarsari and Ian Lavery. Porcupine, March 2018.

[7] Davis E King. Dlib-ml: A Machine Learning Toolkit. Journal of Machine Learning Research:4, July 2019.

[8] Lea Schönherr, Maximilian Golla, Thorsten Eisenhofer, Jan Wiele, Dorothea Kolossa, and Thorsten Holz. Unacceptable, where is my privacy? Exploring Accidental Triggers of Smart Speakers. *arXiv:2008.00508 [cs]*, August 2020. arXiv: 2008.00508.

[9] Alex Sikand. MMM-Porcupine, April 2020.

[10] Michael Teeuw. MichMich/MagicMirror, February 2014.

[11] Dujuan Zhang, Jie Li, and Zhenfang Shan. Implementation of Dlib Deep Learning Face Recognition Technology. In *2020 International Conference on Robots Intelligent System (ICRIS)*, pages 88–91, November 2020.

[12] Serena Zheng, Noah Apthorpe, Marshini Chetty, and Nick Feamster. User Perceptions of Smart Home IoT Privacy. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):200:1–200:20, November 2018.