Task #1 - Topic Review

Garrido et al. make a unique contribution to Mobile computing. Similar to Weiser, the authors take a step back and look at the industry (Garrido focused on VR, while Weiser as a whole) to see what contributions have been made and what type of framework to have moving forward. Garrido finds a "fundamental imbalance" since ***no*** actual implementation of defense papers are used by industry (only 1 paper had an affiliation with Microsoft yet **not** used in production), whereas we are finding more and more attack patterns coupled with industry "trend towards more data harvesting". This is incredibly important framing since we are still at the early stages of VR with a surge of users growing more and more curious over the metaverse, and instead of laying a strong security foundation and allowing a safe transfer, we're running the risk of adversaries having access to more and more personal information. The specific attack that we will be running in Task 2-3 is an impersonation attack, where another user tries to fool the system into believing they are another user. This is significant as it relates to Device Protection, 1 of the 6 key VR defenses Garrido notes on his paper (pg.4-5), and it also relates to the tagging/tracking of users *across* VR sessions (the idea is, an attacker can leverage motion-based authentication features to track users – take advantage that to authenticate it means there is something unique about you… but that uniqueness can also track you!)

For the paper *Exploring the Unprecedented Privacy Risks of the Metaverse* the main motivation lies between recognizing that VR development has increasingly been gathering attention and resources, yet there is a lack of transparency and a security/privacy view over how regular user experience (even just a few minutes in a game) allows for personal and identifiable information *without you realizing*. The challenge for the paper is clear in that the authors's goal is trying to give a comprehensive security and privacy framework for VR environments. They tackle this problem by first describing general flow of *how* VR data is contributed, then they further break what type of adversaries may exist (and how much access to data they have), and end by looking at each adversary and seeing what attributes they could extract. There are similarities to the Garrido paper in this step, such as how there are 4 main types of adversaries (hardware, client, server, and user), but there are also unique interpretation done to get a better sense of the full extent that adversaries can potentially identify (for example, they go further at seeing how we can even detect vision acuity by putting certain interactive elements and measuring user response; or how to reconstruct user room size by encouraging movement within a VR application). In terms of how they evaluated the **need** for this security & privacy framework, they conducted a 30 user study where they *covertly* identified over 25 private data attributes through an open-source"escape room" game (their specific choice of this is important, they wanted to illustrate how game developers - which are the most popular VR developers - can maliciously get information from users easily) where users were only spending 10-20 minutes in VR. In terms of future work they mentioned that to create a defense against this, they encouraged other developers/researchers to keep security & privacy framework in mind because **all** 30 users reported not realizing **when** each attribute was collected (which could be exploited further as in their tests they limited what data to collect for ethical reasons - real adversaries would not limit themselves).

For the paper *A privacy-preserving approach to streaming eye-tracking data* the main motivation comes from recognizing that eye-tracking technology is being used in mixed reality devices. The authors recognize that there are many applications that can effectively use this for critical applications, **however** they also recognize that there are many risks for *uniquely*

identifying users even when they are using “natural viewing conditions” (again, from previous two papers there is an emphasis on this yet we see it echoed again). Thus, the authors are seeking some way to minimize rate of identification without sacrificing usability. A key challenge here is that part of the API for the “good” applications, require **raw** gaze data, yet this same raw data allows arbitrary apps to identify users; so how to modify so applications can still get data they require for legit application purposes yet protect from identification? The authors’ solution to this is creating a “gatekeeper” function such that it addresses the previous (unsafe) design assumption that “data is used only for the purpose for which it is collected”; so that instead of applications having access to the **raw** data, they can **request** certain attributes (such as number of fixations and the dwell time of a fixation). This type of solution brings about another challenge that the authors recognize: “gatekeeper” does not support streaming optimizations which low-battery devices rely on. In response to this, the authors created a procedure that in the event of streaming: they can add Gaussian noise to data, they can temporal downsample (which essentially is taking frequency of original stream and dividing it by a parameter), and they can do spatial downsample (which reduces resolution to a discrete set of horizontal and vertical gazes). Evaluation method was done by having multiple participants view images used for training and testing *identification classification*. Important to note: all users would be present in the validation and test set, since the valuation wanted to see how many users could still be identified. Results is that without “gatekeeper”, you can reach almost 85% identification on users; but with “gatekeeper” that is brought to about 30% while still supporting important features (such as AOI analysis, gaze prediction, and saliency map generation; and incurring less than 1.5 degrees of error for gaze position). Despite these results, the authors still posted that further defense could be implemented such as IF the platform itself can’t be trusted, that there should still exist some user-implementable defense.

For our last paper *Going Incognito in the Metaverse* the main motivation is how VR has been gaining more traction in the industry yet there are also many concerns about identification occurring. As such, the authors are motivated to create the first “incognito mode” for VR. The biggest challenge is that seemingly “anonymized” users, have been shown in multiple studies (and in practice) to be identified with a few minutes of data collection. Their solution to this involves creating an epsilon differential privacy framework (to protect sensitive data attributes, with an emphasis on the most commonly attacked by adversaries) which can be implemented as a plugin for Unity. Interestingly to note, linking back to the Garrido paper that had Hardware, Client, Server, and User adversaries, the MetaGuard plugin only focuses on Server & User adversaries. To create the differential privacy, the authors use Bounded Laplace Mechanism despite it being noisier due to the fact that Servers are potential adversaries. While talking about their solution, the authors are also very careful to mention that there is a difference between additive offsets and multiplicative offsets (and that the choice of each is vital in order to guarantee differential privacy). To explain further, there are some attributes such as Wingspan, where if they would just add “x” noise constantly, adversaries would still be able to classify people (afterall, they would know it’s just *wingspan + some x*. Thus, they introduce multiplicative offset into their solution which essentially makes it so the offset can change with regards to a range (imagine the offset varies so even if a user lays on the grand and we observe height *0 + offset*, that doesn’t mean we know what y is when considering height *y + offset* since this will be a **different** offset). Their evaluation method consisted in obtaining 30 anonymized data recordings from the MetaData study (which was the 1st reading we picked!) and running MetaGuard; with the results substantially reducing the accuracy at which attributes could be

predicted (for example, height within 7cm went from 100% to 44.47%). The authors also double checked MetaGuard by applying Machine learning that tries to identify Demographic Information from users; with results being again substantially reducing accuracy (for example, gender - a piece of information that is hard to prevent being leaked - went from 100% to 57.19%). The authors concluded by mentioning that while MetaGuard protects from Software Level, there should also be a Firmware level of protection.

Task #2 - User Identification Classifier

Methods:

Gesture Design:

To design our gesture, we first thought about what type of activities give **unique** data for each user and can't be **easily** duplicated by an adversary. We also took into consideration that the gesture must be something that can be replicated (more on this in a little bit). The gesture we ended up choosing was Jumping Jacks. This is because we realized that Jumping Jacks have many attributes to identify users (for example Wingspan, Height, and Speed of Jumping Jacks [extra note: we didn't tell users what speed to choose! This is a personal choice and in fact our three group members and adversaries in Task #3 seemed to have different methods: some rested a little bit after every jumping jack while some did continuously]), most everyone has done when they were in primary, and could be easily incorporated into a fitness app login for example. Our method was to have users do **10** jumping jacks **per** session, where each session starts with users having their hands to their side (natural standing position), and then ending when the last jumping jack is fully completed. Now, in regards to how "replicable" the gesture is, we decided from the onset off project to collect our validation data and test data **on different days.** We took this decision to make our gesture be more robust, and we also thought about how actual implementations could make it so every successful login attempt feeds that data to the model (this was actually mentioned in class a couple weeks ago: essentially the idea here is that by keeping up to date with the user's successful login attempts we would be able to account for aging users, either positive or negative changes in fitness, etc.)

Model Design

We implemented and tested five different non-deep learning classifiers: histogram-based gradient boosting, gradient boosting, random forest, k-nearest neighbors, and support vector. To pre-compute the relevant model weights from the training data, we created the file shallow_weights_accuracy_latency.py for our computations. The first function in this file, get_model_weights(num_files_per_person=25), reads the CSV files from the training data and appends them to a dataframe. The train_test_split() function is used to split 80% of the data to be a training subset and the remaining 20% to be a validation subset.The array y is used to account for the user identification labels of the lines appended to the dataframe. Each classifier is then trained on the training data and then evaluated on the validation data using the score() function simply as an accuracy check for the models. Each classifier is then retrained on the entire training data and the weights are saved using joblib.dump() into respective files.

Model Selection:

In deciding on the learning classifier to use, we researched different classifiers and took into consideration factors including accuracy and applicability to the dataset (helpful sources we used: https://www.activestate.com/blog/top-10-python-machine-learning-algorithms/ and

https://www.simplilearn.com/10-algorithms-machine-learning-engineers-need-to-know-article). We ended up choosing our classification models based on their relatively high accuracy and their ability to handle large datasets.

Model Explanation:

Histogram-based gradient boosting first creates histograms of the data. Each histogram represents a subset of the data, and the histogram bin size and number of bins can be adjusted to control the trade-off between bias and variance. The performance of the weak model is evaluated and a new model is built to correct the errors made by the previous model. The predictions made by the previous model are updated to include the predictions made by the new model. The updated predictions are used to train the next model. This model is repeated so the overall model improves with each iteration.

Gradient boosting builds a strong model by combining several weak models, typically decision trees. It works in a sequential manner, where each tree is trained to predict the residual errors made by the previous tree. The final prediction is obtained by summing up the predictions made by all the trees.

K-nearest neighbors stores all available cases and classifies a new case by taking a majority vote of its k neighbors. The case is then assigned to the class with which it has the most in common, measured by a distance function.

Random forest classification uses a collection of decision trees to classify an object based on its attributes. Each tree is classified and the tree "votes" for that class. The forest chooses the classification having the most votes. The classification starts off by using multiple trees with slightly different training data. The predictions from all of the trees are then averaged out, resulting in better performance than any single tree in the model.

Support vector classifiers find the best hyperplane that separates the data points of different classes with the maximum margin. Raw data is plotted as points in an n-dimensional space, where n equals the number of features. For non-linearly separable data, SVM uses kernel functions to map the data points into a higher-dimensional space, where they become linearly separable.

Results:

We compared the classifiers in terms of algorithm precision and latency required for training and testing. Latency is set to equal the sum of training latency and inference latency, and is calculated using the function latency() in the shallow_weights_accuracy_latency.py file and accuracy is inspected by comparing the given test files to prediction results. The results are shown in the table below:

Classifier	Latency (s)
HGB	6.330578

GB	287.546827
K-nearest Neighbors	2.781285
Random Forest	38.177482
SV	203.057954

To test accuracy, we observed the predictions made by the classifiers for the test data by running the script python3 Python/Lab3/predict_shallow.py --label_folder Data/Lab3/Test --output output_labels.txt from the root directory. The results from one trial are shown in the table below, with the misclassified labels highlighted:

	Classifier Predictions (25 Training Samples Per Person)				
Test File	**HGB**	**GB**	**KNN**	**RF**	**SV**
JO_01.csv	JO	JO	JO	JO	JO
JO_02.csv	MA	JO	JO	JO	JO
JO_03.csv	JO	JO	JO	JO	JO
JO_04.csv	JO	JO	JO	JO	JO
JO_05.csv	JO	MA	JO	MA	JO
MA_01.csv	MA	MA	MA	MA	MA
MA_02.csv	MA	JO	MA	MA	MA
MA_03.csv	MA	MA	MA	MA	MA
MA_04.csv	MA	MA	MA	MA	MI
MA_05.csv	MA	MA	MA	MA	MA
Mi_01.csv	MI	MI	MI	MI	MI
MI_02.csv	MI	MI	MI	MI	MI
MI_03.csv	MI	MI	MI	MI	MI
MI_04.csv	MI	MI	MI	MI	MI
MI_05.csv	MI	MI	MI	MI	MI

From these outputs and repeated trial outputs, we determined that K-nearest neighbors was the most accurate classifier.

We also tested the impact on prediction outcomes when the classifiers were trained on only 10 and 5 random samples per user by running get_model_weights(num_files_per_person=10) and get_model_weights(num_files_per_person=5), respectively. The results from one trial are in the tables below:

	Classifier Predictions (10 Training Samples Per Person)				
Test File	**HGB**	**GB**	**KN**	**RF**	**SV**
JO_01.csv	JO	JO	JO	JO	JO
JO_02.csv	JO	JO	JO	JO	JO
JO_03.csv	JO	JO	JO	JO	JO
JO_04.csv	JO	JO	JO	JO	JO
JO_05.csv	JO	MA	JO	MA	JO
MA_01.csv	MA	MA	MA	MA	MA
MA_02.csv	MA	MA	MA	MA	MA
MA_03.csv	MA	MA	MA	MA	MA
MA_04.csv	MA	MA	MA	MA	MI
MA_05.csv	MA	MA	MA	MA	MA
Mi_01.csv	MI	MI	MI	MI	MI
MI_02.csv	MI	MI	MI	MI	MI
MI_03.csv	MI	MI	MI	MI	MI
MI_04.csv	MI	MI	MI	MI	MI
MI_05.csv	MI	MI	MI	MI	MI

	Classifier Predictions (5 Training Samples Per Person)				
Test File	**HGB**	**GB**	**KNN**	**RF**	**SV**
JO_01.csv	JO	JO	JO	JO	JO
JO_02.csv	JO	JO	JO	JO	JO
JO_03.csv	JO	JO	JO	JO	JO
JO_04.csv	JO	JO	JO	JO	JO
JO_05.csv	JO	JO	JO	JO	JO
MA_01.csv	MA	JO	MA	MA	MA
MA_02.csv	MA	JO	MA	JO	MA
MA_03.csv	MA	JO	MA	JO	MA
MA_04.csv	MI	MI	MI	MI	MI
MA_05.csv	MA	MI	MA	MI	MA
Mi_01.csv	MI	MI	MI	MI	MI
MI_02.csv	MI	MI	MI	MI	MI
MI_03.csv	MI	MI	MI	MI	MI
MI_04.csv	MI	MI	MI	MI	MI
MI_05.csv	MI	MI	MI	MI	MI

From these results, it can be seen that when the number of training samples per person decreased down to 5, the classifiers unsurprisingly became visibly less accurate in predicting the correct label. The limitation of these outputs is that they reflect only a singular trial of random training files. To get a better idea of accuracy, a greater number of trials need to be run in order to determine average prediction results.

Discussion:

Some of the implications our results have on VR is just to show how much data we have access to. While yes, we recognize and agree with the 4 papers we read that Security & Privacy are of absolute vital importance as the field continues to expand and it can even be scary how easily we

can uniquely identify users, we also view how strong of an identification can be done with (a) limited data and (b) short time-frame (each session is far underneath 1 minute) as a positive thing **if** done with an appropriate framework and underneath certain applications. To elaborate, we look at the flipside to "typical" computing and the most common version of authentication: passwords. Passwords are extremely easy to use and that's what makes it the "default" option for many users, but it can also be guessed (most people don't do secure passwords) and frankly it's a bother to memorize for every account you hold (and if you decide to user password manager then you have a single-point failure opportunity). VR methods of authentication however, can rely on vast amounts of data to authenticate users by doing a normal every-day activity which is a great bonus to usability & security as false positives would be difficult to achieve.

A potential limitation to the results here is that we can't do weeks in between our tests due to the time limit of the Lab. It would be interesting to put in practice whether or not this type of identification (physical based activity) holds through the passage of time as users can obviously change and their behavior with them (aging users won't do jumping jacks as fast, becoming fitter may increase your speed and range of motion, etc.)

Task #3: Adversary Detection

Because our best identified model in Task 2 was kNN, we will be using kNN for adversary detection. Our goal in this task is to create an adversary detection model that accurately filters out adversaries without inconveniencing authenticated users.

Methods:

The adversary detection model is somewhat trivial but important to understand. Because our model is based on each frame of data collected in the recording, the function *predict_proba()* has around 700 rows, each of which make a prediction on which user we are seeing. Specifically, the function return can look something like this:

[[0, 0, 1],
[0, .02, .98],
… ((700 rows later))
[.01, .01, .98]]

Therefore, we decided to navigate this by averaging the probabilities and taking the highest probability. This is therefore equivalent to the confidence level of the user identification. Eyeballing the confidence levels showed us that the threshold decision would be difficult to make, however, we are confident in our decided threshold of 90%. Specifically, our analysis showed the following:

- Jose's confidence level was >98%
- Michelle's confidence level was 86-96%
- Matt's confidence level was 86-93%
- Adversary confidence levels were 40-70%
- However, one adversary achieved a confidence level of 85.16% for Matt

We chose the 90% threshold due to these data points among other reasons. Specifically, if we had chosen a threshold of 86%, then we would have achieved 100% identification accuracy for adversaries and authenticated users. However, this poses a significant issue: our adversary was extremely close to this threshold in this case, and therefore we would be concerned that the adversary would be able to authenticate with more trials. With the 90% threshold, we had 80% success with Michelle, 60% success with Matt, and 100% success with Jose (while maintaining 100% successful identification of adversaries). But, this threshold was more comfortable for us because we would prefer to have Matt and Michelle be slightly more inconvenienced to avoid authenticating an adversary.

Results:

	Jose	Matt	Michelle	Adversary
Accuracy	100%	60%	80%	100%
Latency	152ms			172ms

RAW OUTPUT FROM FUNCTION (for reference):

test data

—

Detected JO with probability 0.991218130311615
Detected JO with probability 0.9876923076923078
Detected JO with probability 0.9888111888111885
Detected JO with probability 0.9958041958041957
Detected JO with probability 0.991255289139633
Detected MA with probability 0.9392215568862277
Detected adversary
Detected MA with probability 0.9325190839694659
Detected adversary
Detected MA with probability 0.9296244784422808
Detected MI with probability 0.9676248108925872
Detected MI with probability 0.9241809672386899
Detected MI with probability 0.918012422360249
Detected MI with probability 0.9374021909233177
Detected adversary
+ avg latency = 0.15211136993333332

===

adversary data

—

Detected adversary

Detected adversary
Detected adversary
Detected adversary
Detected adversary
Detected adversary
Detected adversary
Detected adversary
+ avg latency = 0.1724322853750001

Discussion:

In terms of accuracy, we were hoping for high confidence levels (>95%) for each authenticated user, as this would allow us to set a high threshold that would assist in adversary detection. However, after seeing that much of the adversary confidence level data was lower than 70%, we realized that our threshold could afford to be a bit lower (90%) so we could hedge against adversaries while still identifying authenticated users. Nonetheless, we were surprised to see that the confidence level for one of the adversaries was very close to the confidence level of Matt's trials. Because Jose's and Michelles' trials did not seem to have this issue, in the future, we would want to have Matt create more training data to increase the confidence scores in his data. This can sometimes be the case in ML models, as the training data obtained may have some inconsistencies that make identification confidence levels lower.

The primary limitation in the model would be the inconsistencies in the training data which has resulted in lower-than-optimal confidence levels for detecting Michelle and Matt. Ideally, we would expect to see high probabilities (like in the case of Jose). However, we suspect that the variability in the data is due to the complex and strenuous nature of our task, and users may have slight variations in how they perform the task. Furthermore, the task is quite strenuous, so creating the training data could have resulted in users being more tired. Therefore, it might be the case that the main limitation is our choice of a complex, strenuous task, but it would be difficult to know this for sure without making the task less strenuous and complex.