# TraffickCam: GWCS Senior Capstone Final Report

Oliver Broadrick, Graham Schock, Marshall Thompson, and Jack Umina

# Section 1: The Team

**This section includes our team member names and a description of who did what in the project.**

- Marshall Thompson
    - Responsible for developing the search and related features
        i. Analysis of the R2D2 key points of the Hotels50k dataset
        ii. Developing ideas on how to reduce dataset size
            - Implement machine learning models
            - Writing analytical methods
        iii. Devise searching and scoring method to yield best results
- Oliver Broadrick
    - Developed, implemented, tested, and compared multiple approaches to the search algorithm (with Marshall)
        i. Use FAISS for large scale k-nearest neighbors search
        ii. Implement machine learning models to classify points useful or not useful ('good' or 'bad') for hotel recognition
        iii. Develop analytical classification of 'good' and 'bad' points for hotel recognition
        iv. Implement image match scoring metric
- Graham Schock
    - Front end
        i. Uploading images
        ii. Editing images
        iii. Displaying results
- Jack Umina
    - Responsible for architecting and developing our backend framework. This included:
        i. Designing a database schema
        ii. Implementing the database
        iii. Designing our API
        iv. Implementing the API in our backend Flask server

# Section 2: Non-Technical Overview

**A brief non-technical overview of our project.**

In the United States, it is estimated that there are roughly 15,000 to 50,000 women and children forced into sexual slavery every year[1]. However, this number may very well be an underestimate as most sex trafficking cases go unreported. In most cases, sex traffickers will post pictures of their victims in hotel rooms to be used as online advertisements. TraffickCam is an application that seeks to help law enforcement agencies pinpoint the exact hotel in which a picture was taken in an effort to rescue the victim.

Currently, TraffickCam's search algorithm works by first compressing pictures of victims to a resolution of 256 by 256 pixels. This allows the compressed picture to be vectorized and fed into a neural network. Unfortunately the tradeoff here is that many important details of an image will be lost due to the down-scaled quality. Take for example a unique painting on the wall of a hotel room—with a resolution of just 256 by 256, the wall art will not be a useful feature in the search process. Our work on TraffickCam aspires to increase the accuracy of the search by retaining unique features of the photos. Using Repeatable and Reliable Detector and Descriptor (R2D2)[2] key points, the search algorithm can be expanded by searching on specific and unique items in a picture. The updated TraffickCam would continue to be utilized by intvestigators who are trying the find victims of sex trafficking and apprehend those responsible.

# Section 3: Project Website

**A website containing our project information and presentations can be found at: grahamschock.github.io/tcam**

# Section 4: Project Environment

**A description of your project "environment" - a list of libraries, packages and APIs that you used for the project. This is not inclusive of your own code.**

- Reliable and Repeatable Detector and Descriptor (R2D2)
  - https://github.com/naver/r2d2
- Facebook AI Similarity Search (FAISS)
  - https://github.com/facebookresearch/faiss

[1] https://deliverfund.org/blog/facts-about-human-trafficking-in-the-us/
[2] https://arxiv.org/pdf/1906.06195.pdf

- CUDA Toolkit
- Numpy
- PIL
- Flask
- Requests
- OS
- Shutil
- Skimage
- Pickle
- MariaDB
- Pandas

# Section 5: Technical Overview

R2D2 generates thousands of key points per image. If it were to be used on the full scale Hotels-50k dataset of 8 million images, that would result in 8 billion key points. Using all 8 billion key points to conduct our search would be infeasible both in terms of time and memory. Our approach to tackle this issue was two fold. First, we needed to reduce the size of the dataset. To do this we needed to discriminate between points that would be more useful for matching an image ("good" points) and points that would be less useful for matching an image ("bad" points). We call a point "good" if, when a k-nearest neighbors(kNN) search is conducted in a subsampled dataset, a point from the same hotel appears in the first 100 nearest neighbors. For the same reasons that just a search is infeasible, labeling 8 billion points by 8 billion other points is also infeasible, hence a subsampled labeling dataset. This is done for every point. Second, The "good" points are kept, and used to build a FAISS index. We chose the FAISS Inverted File Index(IVF), which, at build time, clusters data based on their cosine distances. At search time, an arbitrary number of probes are sent to the most likely clusters to find the nearest neighbor to the query point. Since the number of probes is very low, 10 in our case, and the number of clusters is very high, in the thousands, we significantly reduce our search space and significantly reduce our time to search. Of course, there is an accuracy tradeoff as the IVF index is an approximate kNN algorithm, but this is preferable to having searches that would take on the order of hours to days given the urgency of the underlying task of finding victims of sex trafficking and stopping the people responsible.

To support the search algorithm, we needed to develop both a backend server that would handle our custom API, as well as a frontend web application. Our backend server is an instance of a Flask server that has a number of different endpoints to support our application. These include functions to upload new query images, functions to generate key points on images, a function to call our custom built search algorithm, as well as a few functions to return images for the visualization parts of our application. To demonstrate our search pipeline, we built a frontend. Our frontend uses React to allow the user to upload an image, edit the image, display key points and display results and why they are a good match.

# Section 6: If I had to do this again!

- Marshall Thompson

At our scale of data, experiments take days to weeks to conduct. So when you fail an experiment because of a bug in your code, or due to an oversight you are wasting weeks. As these errors pile up, you waste months. Additionally, I could have been more organized, and attacked this problem with more precision which would have helped with the previous problem and would have maybe yielded better results.

- Oliver Broadrick

It would be difficult to understate the value of visualizations while working with images. Checkpoints in code to see key points on images, correspondences, various distributions, etc, were always helpful in developing understanding of the problem and making forward progress. There is also significant value in tracking progress meticulously. Weeks after an experiment that at the time seemed unimportant, those results may suddenly become useful or important, but they are only available if they were tracked from the beginning without regard for perceived importance.

- Graham Schock

When we first started this project, I thought it would be a good idea to use React for our frontend. React was used to build the original TraffickCam and is a popular highly documented framework, so it seemed like a good fit. However, using react on our "simple"

frontend caused it to become a lot more complex then it needed to be. We had to deal with complicated layers of abstraction in the "Model View Control" model. In the future it would probably have been better to write the frontend in HTML and only use javascript where it was needed.

- Jack Umina

Our project utilized many different libraries and packages to achieve our solution. Because of this, some of the install process and environment setup became quite complicated. This issue was only compounded with each new library added to our solution. Getting Faiss, CUDA Toolkit, and a few other libraries to work well with one another required a couple different virtual environments. It's likely that this issue could have been avoided with a cleaner, more streamlined solution.

# Section 7: Advice for Follow On Projects

As discussed above, we needed a way to discriminate between "good" and "bad" points in our dataset. Our analytical approach to this problem works, but it requires much more time, memory, and storage than a machine learning method. We attempted several machine learning techniques to this problem, however none of the models we built generalized well, and became inaccurate at larger scale datasets. We consider this still an open question, and one that is well worth research, as this would not only allow quicker database building, but would also allow us to remove potentially "bad" points from query images at search time improving accuracy.

A piece of advice to any future researchers: Double, and triple check the experiments that you run, or consider running them at smaller scales before moving on to larger scales. With the scale of data that this project deals with, a mistake can end up costing you days or weeks of time. Multiple mistakes can lead to months!

# Section 8: How to Run This

**This section provides instructions for getting our project running.**

*Required equipment:*

Our application is quite computationally intensive. Here are some of the minimum system requirements:

- At least 64GB of RAM
- A CUDA compatible Nvidia GPU
- Minimum of 30GB of available disk space for the database and necessary index files

*To run frontend:*

1. Clone repo (https://github.com/jtumina/tcam-web)
2. $ cd tcam-web/frontend
3. Install dependencies (React)
4. npm start
5. Go to port 3000

*To run backend:*

6. Clone repo (https://github.com/jtumina/tcam-web)
7. cd into /tcam-web/backend
8. Install all required packages
    - `pip install -r requirements.txt`
9. Add execute privileges to run.sh
    - `chmod +x run.sh`
10. Execute run.sh
    - `./run.sh`

# Section 9: What Works, What Doesn't

**What works, what doesn't, what to be aware of (pitfalls, issues).**

We have built a web application and the supporting backend and algorithmic features to find a hotel match to a query image. However, we have only been able to do this at the relatively small scale of 100 hotels, 2600 images and 18 million points. Because of time constraints, we have not been able to scale up to the 0.1k Hotels50k dataset(~5000 hotels), let alone the full 50k.

# Appendix

**All previous homework assignments.**

Graham Schock
Oliver Broadrick
Marshall Thompson
Jack Umina

Ms. Nicolas
CSCI 4243W: Capstone Senior Design
October 22, 2021

# TraffickCam: HCI Goals

TraffickCam is an application that helps combat sex trafficking by providing an accurate and effecient way of matching a picture of a hotel room to its location in an effort to help law enforcement find victims. Our interface will seek to provide a convenient way for users to visualize R2D2 points, as well as provide a way to search based on particular R2D2 points. We will maximize efficiency by creating a simple UI that allows users to quickly and effectively upload new images and view the search results. We will provide and encourage options to anonymize sensitive user data and add UI elements to prevent users from accidentally deleting their work to provide a safe user interface. The interface will be easily learnable so that current users continue to have a familiar experience and new users can quickly learn our application. By increasing the learnability, memorability, efficiency, and safety of our project we are necessarily increasing the utility our users will receive from our project.

The George Washington University

# TraffickCam:
# HCI Goals

Graham Schock
Oliver Broadrick
Marshall Thompson
Jack Umina

Ms. Nicolas
CSCI 4243W: Capstone Senior Design
October 22, 2021

## 1. Introduction

In the United States, it is estimated that there are roughly 15,000 to 50,000 women and children forced into sexual slavery every year[1]. However, this number may very well be an underestimate as most sex trafficking cases go unreported. In most cases, sex traffickers will post pictures of their victims in hotel rooms to be used as online advertisements. TraffickCam is an application that seeks to help law enforcement agencies pinpoint the exact hotel a picture was taken in an effort to rescue the victim.

Currently, TraffickCam's search algorithm works by first compressing pictures of victims to a resolution of 256 by 256 pixels. This allows the compressed picture to be vectorized and fed into a neural network. Unfortunately the tradeoff here is that many important details of an image will be lost due to the down-scaled quality. Take for example a unique painting on the wall of a hotel room—with a resolution of just 256 by 256, the wall art will not be a useful feature in the search process.

Our work on TraffickCam aspires to increase the accuracy of the search by retaining unique features of the photos. Using Repeatable and Reliable Detector and Descriptor (R2D2)[2] points, the search algorithm can be expanded by searching on specific and unique items in a picture.

## 2. The Interface

With the inclusion of R2D2 points in TraffickCam, it will be important for users to understand exactly what makes a match accurate. Our idea for the interface is to allow users to visualize matching R2D2 points from the query image to the search results. This gives users a way of visually verifying that the search results are accurate.

Stemming from this visualization, we also plan to develop a feature in which users can click on a specific R2D2 point in the query image and restart the search where the search parameter is solely looking for images that contain the selected R2D2 point. This feature can also be expanded by creating a bounding box around multiple R2D2 points to be used as the search parameter.

## 3. Usability Goals

### *3.1. Efficiency - Jack Umina*

Efficiency is an extremely important goal of the interface for TraffickCam. Because this is an application that is currently, and will continue to be, used by law enforcement on real sex trafficking cases, we must ensure that it is efficient to use. The current UI for TraffickCam is quite simple and efficient: users simply upload a search query image, specify any particular parameters such as hotel names or locations, and then the search results are displayed in a sorted order from best to worst match. When we implement features to allow for a more refined search using R2D2 points, we will ensure that this efficiency is not lost. The UI will be simple to allow users to look at specific R2D2 points when searching for matches.

### *3.2. Safety - Graham Schock*

As UI designers we want to prevent our users from getting into undesirable or dangerous situations. In order to accomplish this we will make sure to incorporate safety as a major design element in our user interface. Our users deal with sensitive data. Users will often upload personal details about minors and victims of sex trafficking to our website. In order to protect our users

---

[1] https://deliverfund.org/blog/facts-about-human-trafficking-in-the-us/
[2] https://arxiv.org/pdf/1906.06195.pdf

and their data, it is vital we give them the option to anonymize their data. As soon as a user uploads an image, the first screen they will see will give them the option to mask/delete sensitive parts of the image. We can also create a safer UI by making sure the user knows they are exiting the program or deleting an image. Users can spend a lot of time masking or editing their image when they first upload. Therefore, it is important that when they move out of the image editing page or exit the website they know their work will not be saved. We can accomplish this by adding a popup when a user exits to let them know and save them time from having to do all of the edits again. By incorporating some of these features in our user interface we can create a safer and better user experience for our users.

### *3.3. Learnability - Oliver Broadrick*

Learnability is an important principle of user interface design and is essential to our project being useful to the users. We need users to relatively quickly and easily understand how to use our tool and how to use all of its features. After a simple login, our main user interface will be largely similar to the current version of Traffickcam so that current users have their usual experience maintained. That said, we are implementing new features (namely, the use of keypoint matching in the search) that lends itself well to some new front end visualizations and interfaces. The main way to help new users learn how to use our new features is to make them use simple, familiar ways to perform tasks. For instance, it is common that hovering over something in a UI will make new information appear about the item. In the case of our keypoints, hovering will make information about the closeness of the match and value of the point become displayed on the screen. Since this is a common technique, the user will find it easy and natural to remember how to use this tool. If we do end up implementing the ability of users to select a small section of the image and search on just those keypoints contained in it, we will make the search very learnable by using a standard frame creation method accompanied by simple instructions. Since the users of this tool do have the time and willingness to invest some effort into being able to use the tool well, we have the advantage of being able to implement even some complicated features with terse but precise descriptions and reasonably expecting that they will learn them.

### *3.4. Utility - Marshall Thompson*

Utility is dependent on the design choices that we make regarding each of the other usability goals. As with any project, we want to maximize the utility that our end users will receive. In the context of the Traffickcam Project, this not only means that we want to provide the highest quality image matches, but that we provide a high quality, easy to use interface with which the investigators can narrow down or edit the search. Generating high quality image matches can be next to useless if we do not display the results to the users in a meaningful, and intuitive way. It is similarly true that our matches would be of less value to investigators if we cannot visualize what we believe to be the similarities between images. Any feature, or design choice that can make an investigators workflow more efficient, intuitive, or accurate will increase the usability of our project and will also increase the utility that they gain from our product. Similarly, if the new features we add are both easy to learn and memorable, then we can increase the efficiency with which investigators can operate and also increase their utility. Furthermore, the security of our system also adds to the utility that the investigators and victims receive. This is an obvious increase in utility to our users as we do not want their information leaked. By increasing the learnability, memorability, efficiency, and security of our project we are necessarily increasing the utility our users will receive from our project.

The George Washington University

# TraffickCam:
# Architecture and Design

Graham Schock
Oliver Broadrick
Marshall Thompson
Jack Umina

Mr. Toombs
CSCI 4243W: Capstone Senior Design
December 10, 2021

**Overview** **3**
**Logical/Run-time Software Components** **4**
**Architecture choices** **6**
**Physical Layer** **7**
CPU: 7
RAM: 7
GPU: 7
Local Disk: 7
**Data Storage** **7**
**Backend** **8**
API 8
Faiss 9
Reliable and Repeatable Detector and Descriptor (R2D2) 9
**Frontend** **10**
Software Design Patterns 10
**Input and Output Mechanisms** **10**
**Team Responsibilities and Estimated Timelines** **10**
Frontend Timelines 11
Backend Timelines 11

# Overview

In the United States, it is estimated that there are roughly 15,000 to 50,000 women and children forced into sexual slavery every year[1]. However, this number may very well be an underestimate as most sex trafficking cases go unreported. In most cases, sex traffickers will post pictures of their victims in hotel rooms to be used as online advertisements. TraffickCam is an application that seeks to help law enforcement agencies pinpoint the exact hotel a picture was taken in an effort to rescue the victim.

Currently, TraffickCam's search algorithm works by first compressing pictures of victims to a resolution of 256 by 256 pixels. This allows the compressed picture to be vectorized and fed into a neural network. Unfortunately the tradeoff here is that many important details of an image will be lost due to the down-scaled quality. Take for example a unique painting on the wall of a hotel room—with a resolution of just 256 by 256, the wall art will not be a useful feature in the search process.

Our work on TraffickCam aspires to increase the accuracy of the search by retaining unique features of the photos. Using Repeatable and Reliable Detector and Descriptor (R2D2)[2] points, the search algorithm can be expanded by searching on specific and unique items in a picture.

The updated TraffickCam would continue to be utilized by intvestigators who are trying the find victims of sex trafficking and apprehend those responsible.

[1] https://deliverfund.org/blog/facts-about-human-trafficking-in-the-us/
[2] https://arxiv.org/pdf/1906.06195.pdf

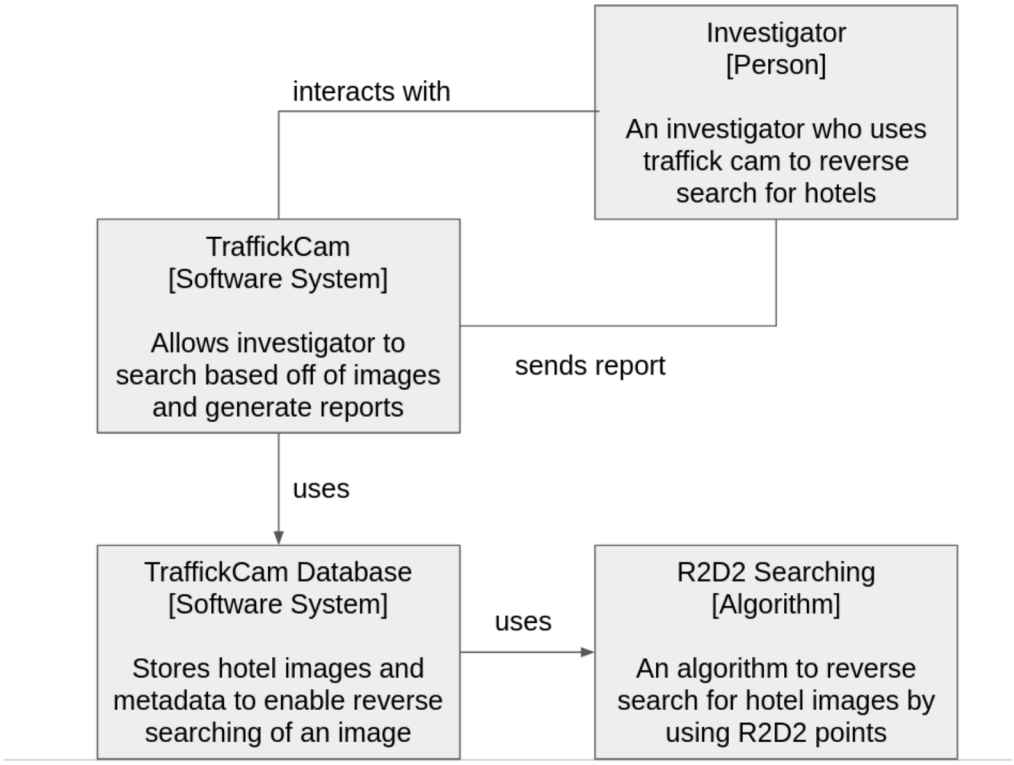



Figure 1: A high level overview of how a user will interact with a system

## Logical/Run-time Software Components

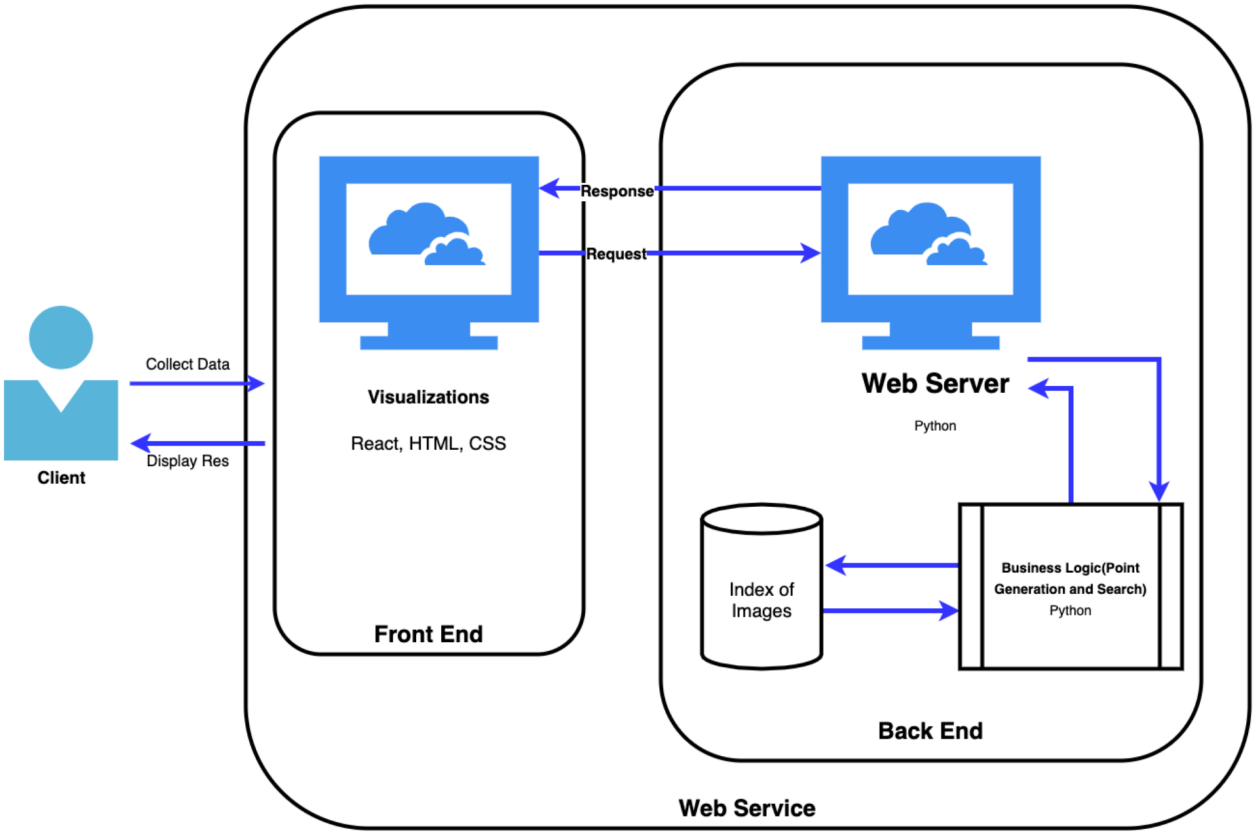



Figure 3: Container diagram of our architecture

Above is an architecture diagram of our entire system. Our clients will interact with a front end and will enter an image to evaluate and that image will be sent over a custom API to our backend server. This server has three functions. To store the indexed images of the Hotels50k dataset, evaluate images given by the clients and produce R2D2 points, and lastly search the indexed points for the best matches to the query image's points.

The front end is built using React, HTML, and CSS. In order to be functional, the clients must be able to upload an image, and be able to see and interact with the visualized points once the backend algorithms have completed their analysis.

The back end is built in Python. In order to be functional, the backend must be able to produce results from a query image, accurately, reliably, and efficiently and then send the results back to the front end for them to be visualized.

## Architecture choices

The TraffickCam web application has already been created. Our team is trying to add functionality. Because of this, we are locked into certain architectural choices. This includes using React, HTML, CSS to build the front end, a monolithic architecture, and existing APIs for accessing Hotels50k images (not the indexes we create for searches).

When designing our new back end algorithms for image recognition we had more freedom as we were not building on top of old algorithms or approaches, but developing novel ones.

The R2D2 model, like most machine learning algorithms, was written in Python, utilizing Pytorch to create the model, and Numpy to serialize the results of image evaluation. Since the output of the R2D2 evaluation can only be interpreted through Python and Numpy, it was the most logical choice of language to develop the algorithms to perform analysis on those results. This reduces compute time and wasted storage space by translating the data into a format readable to another language. This comes into play again when designing the backend API (written in Python Flask) to receive query images and send the results back to the front end. Having everything in one language on the back end reduces complexity, and minimizes the chance of error when switching between them. Additionally, python has many supported libraries for data analysis and machine learning. This means we do not have to spend time writing a lot of the boilerplate code, and we can spend time on more important issues.

The Hotels50k Dataset contains 9 million images. The R2D2 algorithm creates upwards of 1000 features per image. That means at full scale we would have to store over 9 billion features. That is too much for any database or indexing system. A main focus of this project is to figure out a way to reduce the size of the data we would have to store by eliminating redundancies, and eliminating features that bear little significance to finding a good match. We are hoping to reduce the data by two or more orders of magnitude. However, this would still be too large to store in a conventional database, and when we are searching for a match it would be impossible to store that many vectors in memory. So, instead of using a more conventional database system some form of index is required. Writing our own indexing algorithm is ancillary to our main goal, and it has already been done many times. Thus we plan to use Facebook's FAISS library for Python. This library will generate an index of our dataset that can be stored in memory making our search problem feasible.

Searching over that many index entries is still a daunting task. Computing distance on high dimensional vectors is expensive, and to do that over millions of images would be near impossible. Our current solution is to utilize GPUs and write our code in Cuda, the language of NVidia GPUs. While this is much faster than on a CPU, it is still far too slow to produce results for an investigator working on a case. As the project progresses, we plan to utilize FAISS again. FAISS has several built-in methods of an approximate nearest neighbor search, which will be

much faster than an exact nearest neighbor search. This alone makes it a better choice than a brute force nearest neighbors search. Since FAISS uses approximate nearest neighbor searches there will be a loss to accuracy, but parameters can be experimented with and to create the optimal results.

# Physical Layer

Our server is a workstation running in the fourth floor labs in SEH. It has the following specs:

### CPU:

Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz
10 cores

### RAM:

128 GB

### GPU:

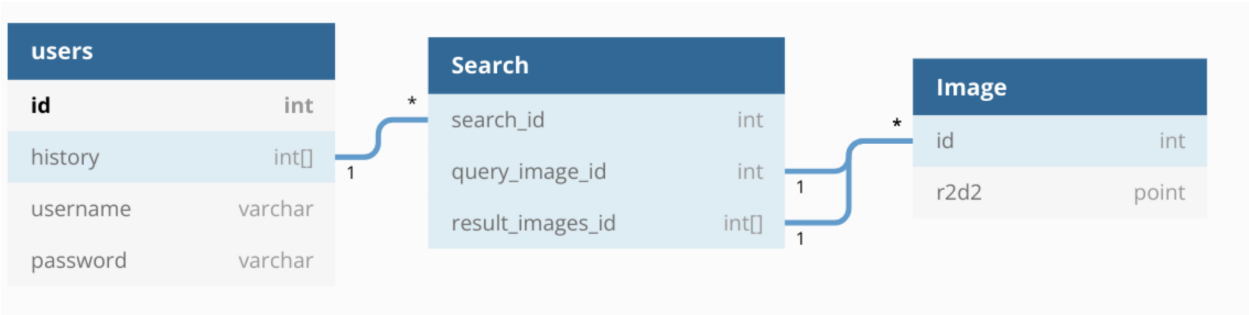
2 x Nvidia Titan RTX

### Local Disk:

2 TB SSD

# Data Storage

In our web application we will be using a MySQL database to store our data. In our project the main thing we will be storing is images as well as metadata about those images that help the user search. Here is our schema.




As we can see from this the user will have a list of searches which will contain a query image and a list of result images. Each image will also have metadata (like R2D2) that will help us to be able to search for an image and generate a result array.

While at a high level our backend will be interacting with a SQL schema. It is important to note what the underlying file system of the OS will be. We are considering two main file systems that our OS will run.

XFS
This filesystem is focused on having lots of threads read or write to files. XFS achieves this by using allocation groups which allow multiple threads to read and write from the filesystem at one time.

EXT4
EXT4 is what is used in Ubuntu/Debian by default. It optimizes for being secure and reliable. However, it may not scale as well as XFS.

In our project we will need to be interacting and searching a large amount of images on the filesystem. Due to the reasons and explanations that XFS will be more scalable and parallel we will be using XFS as our underlying filesystem for our datastore.

# Backend

## API

We are working on building a Flask server on the backend that will act as our API server. Currently, we have two REST methods implemented: uploadImage and getPoints.

uploadImage is a POST request that takes in a data url as the payload. The data url contains the query image data for the image we are attempting to match. This image data is posted to the

backend server, given a unique image id, and then stored in the filesystem for future processing. The method returns the newly generated, unique image id.

getPoints is a GET request that takes three arguments: image id, k values, and grid size. The image id is used to select which photo the frontend is requesting R2D2 points on; k values is an integer that specifies the number of R2D2 points to be returned; grid size is used to specify the size of the grid that the R2D2 coordinate points will be normalized to. The method returns a json list where each element contains the x and y coordinates of the R2D2 point, as well as the scale of that point.

Future development plans will include another GET method to return the list of hotel matches to the query image. This will be arguably the most important method as it will provide the main functionality of our project. This method will only need the image id as an argument, and will return a list of images and hotel names that match the query image.

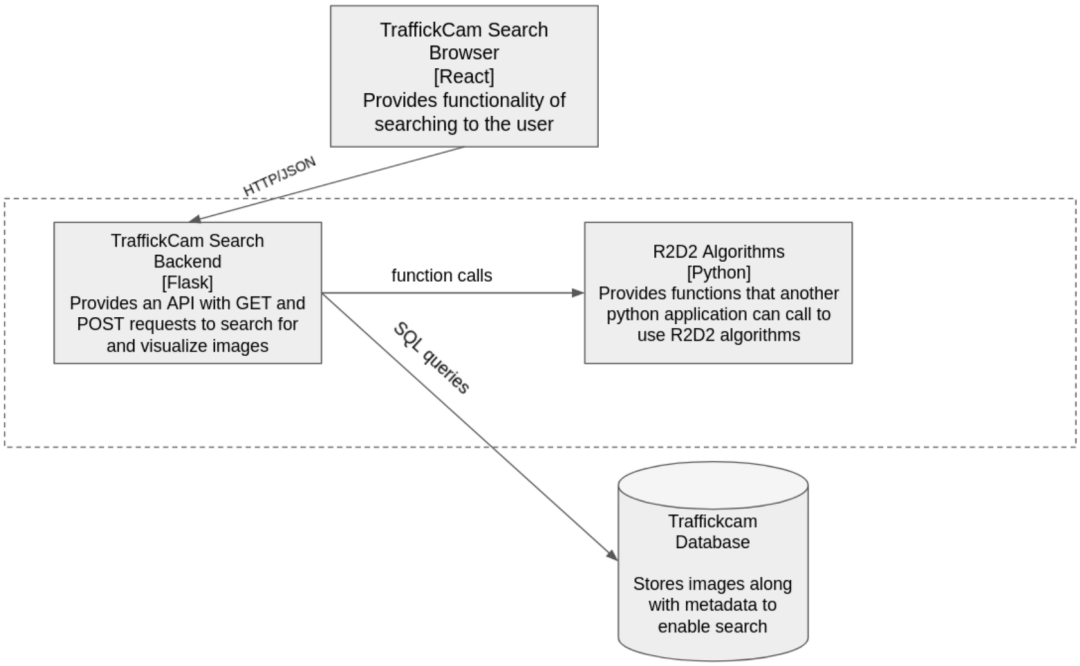



Figure 2: Diagram of frontend interacting with backend API

## Reliable and Repeatable Detector and Descriptor (R2D2)

Key point detection and description is a classic problem in computer vision. Reliable and repeatable detector and descriptor is a state-of-the-art tool for this job. The key insight of the R2D2 work is that some key points are more useful in the domain of matching than others, since some points, although unique in terms of colors and shapes are not discriminative within the context of their image. Thus, by training a network which jointly detects and describes the most

reliable and repeatable points, we will have the best opportunity to find correct matches. We will use R2D2 prebuilt and pretrained neural networks to generate key points for images in the Hotels-50k database. We will store these points in our own database and use them for the searching of matches to query images. On query images, we will compute R2D2 points as well which will be used for the searching process at query time.

## FAISS

Computing the R2D2 features on the entirety of the 1 million images in the Hotels-50k dataset would result in roughly 10 billion features. A naive, brute force solution to the R2D2 based Traffickcam problem is not viable for two reasons. First, reading and writing to disk is an expensive operation- roughly 30 times slower than reading from main memory(RAM). Therefore, a better solution would be to store a representation of our features in RAM as an index (Note, the word “representation” was used as 10 billion 128-dimensional floating point vectors would not fit in memory). On the scale of 10 billion features, this is imperative, as the time costs of reading from disk would accrue quickly. Second, a brute force k-nearest neighbors, O(kn) search on the features is far too slow on the scale of 10 billion features. There exists approximate nearest neighbors searches that reduce this time.

The Fast Approximate Similarity Search Library (FAISS), developed by Facebook, solves both of these problems. First FAISS will build an index of your database that can be loaded into RAM. There are several options available for indexing type and further experimentation is required to determine which would be the best fit for the Traffickcam problem. Second, FAISS provides several options for approximate searches that also will need to be experimented with in the Spring. While we have not chosen a specific indexing scheme or a searching algorithm, there are certain constraints that can be decided upon, or considered now. First is the memory cost. The server we will be using for this project has 128Gb of RAM. Therefore we are constrained by the amount of RAM that the server uses to host the backend, and the total amount of RAM. With more memory, an index that more closely resembles the full dataset can be built and provide better accuracy. While we cannot decide at this point what searching algorithm will be required, it is certain that an approximate search is necessary. Furthermore, we know that we will be using the cosine distance to represent the distance between two features. The more dimensions there are in your data, the more the Euclidean distance metric becomes less descriptive of the relationship between vectors as all vectors are almost equally far apart in high dimensional space. Instead, the cosine of the angle between two vectors can be used as that relationship remains constant in high dimensional space.

## Binary Classification of Points for Hotel Recognition

Another way to make the search problem more achievable is to reduce the size of the overall search space by removing points that aren’t useful for the problem of hotel recognition. We will develop a binary classification algorithm which identifies *good* points and *bad* points where good points are those useful for hotel recognition and bad points are those that are not useful in hotel recognition. Consider, for example, the corner of a white bed and a grey wall. This is the kind of point that is reasonably detected and described as an R2D2 point but one

which may not be useful in hotel recognition since it may be common in many different hotels. Thus, we will consider the ratio of the distance of the nearest point among those in the same hotel to the distance to the nearest point among those in different hotels. The distribution over these ratios will inform our selection of the threshold at which the cutoff between *good* and *bad* points is made for labeling training data. Then, a neural network will be trained to predict this class for arbitrary new points, and we will classify the rest of the points in our dataset.

# Frontend

Due to the fact that our tool is designed for and used by people working at desks on investigations that take place over multiple days, it is appropriate to only build our tool for use in the browser. This is justified by the current version of Traffickcam, designed by and for experts in the field, which mirrors this design decision.

## Software Design Patterns

In our frontend we will be using react. More specifically our image visualization will be a component that will be integrated into the entire traffickcam ecosystem. The general traffickcam component will have instances of our image point visualization component and will be interacting by sending props to our component.

## Input and Output Mechanisms

*Input:*

The user will upload an image or several images using our front end web page. The user will also (using already implemented functionality) input search restraint including location restraints, hotel chain restraints, image area restraints in the form of maskings of sensitive pixels.

*Output:*

The user will see, on the front end web page, visualizations of the search results. They will be able to interact with these visualizations in some capacity (get more or less information about particular key points for example). The user will also have the option to save pdf files summarizing these results; this functionality already exists for the current search, but the new key point matches will provide additional evidence for strong matches. It is useful for investigators to be able to download pdf files showing matches because these files can be used as evidence in their cases.

# Team Responsibilities and Estimated Timelines

The table below shows approximate estimates for the percentage of work each team member will contribute to each task. Notice that overall, the division of work consists of Graham and Jack working on the system architecture, while Marshall and Oliver work on the algorithm for the new key point search.

| | Frontend | System Administration | Backend | Algorithms |
|---|---|---|---|---|
| Graham | **100%** | **50%** | 0% | 0% |
| Jack | 0% | **50%** | **100%** | 0% |
| Marshall | 0% | 0% | 0% | **50%** |
| Oliver | 0% | 0% | 0% | **50%** |

## Frontend Timelines

- Integrate visualization tool with our fork of TraffickCam repo (February 1)
- Build search framework (March 1)
- Build key point comparison visualizations (April 1)

## Backend Timelines

- Binary classification algorithm (January 15)
- Finalize search algorithm (February 15)
- Integrate search algorithm with backend (March 1)
- Build full database (March 15)
- Build search API (April 1)

# TraffickCam

**Jack Umina, Graham Schock, Marshall Thompson, and Oliver Broadrick**
CSCI 4243W - Team 8
George Washington University

## Introduction

In the United States, it is estimated that there are roughly 15,000 to 50,000 women and children forced into sexual slavery every year [4]. Identifying the location an image was taken in through reverse image searching is an important problem for investigating human trafficking cases. In Figure 1, there are a few example images of real law enforcement investigations. Our goal is to improve TraffickCam to more accurately identify and locate images like these.

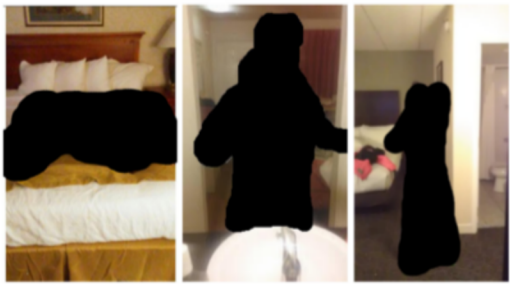


Figure 1: Examples of sex trafficking investigation images.

Currently, TraffickCam uses a dataset of over 1 million hotel room images known as "Hotels-50k". TraffickCam then uses a pretrained Resnet-50 model and a downscaled version of Hotels-50k as input to find similarities between the query image and images in the dataset. However, in order to protect sex trafficking victims, any identifiable information relating to the victim must be masked. In many cases, 75% or more of the image will be obscured leading to a more challenging search. Figure 2 presents this issue: as the level of masking increases, the chance of the query image matching to one of the images in the top 100 results decreases. Clearly, instead of searching on the entire image as a single vector, we need a better solution.


Figure 2: A bar chart correlating masking level to image search accuracy.

## Related Work

**Robust Wide Baseline Stereo from Maximally Stable Extremal Regions.** Rather than searching for the entire image, images can be broken down into multiple, separate elements. Maximally stable extremal regions (MSERs) attempt to do this by using elements called extremal regions. These extremal regions are important for the properties they possess. The set is closed under the continuous transformation of image coordinates and monotonic transformation of image intensities. In other words, extremal regions can be easily manipulated in a way such that it will match an extremal region in another image of a slightly different view[2].

**Object Recognition from Local Scale-Invariant Features**

Essentially, we transform an image into local feature vectors that can not change from doing various linear algebra transformations, because these features are invariant to transformations they will remain the same when an image is distorted or scaled. We can identify these key features by looking at the maxima or minima difference relative to the rest of the image. Once we have identified these key features we can use the nearest neighbour algorithm to identify similar objects and features between images [1].

**Repeatable and Reliable Detector and Descriptor**

Detecting and using key points for matching problems is a standard technique in computer vision. The Repeatable and Reliable Detector and Descriptor (R2D2) algorithm is a state-of-the-art approach to the problem of identifying and describing such points with neural network models. R2D2 models are jointly trained to identify and describe key points *as well as* provide a predictor of discriminativeness (how likely a unique point is to exist). Therefore a sparse set of R2D2 points are unique and uniquely described; they serve well for finding matches between images that share similar interest point features [3].

**Summary**

Like all the above approaches, our project utilizes finding key points from an image and using those key points for matching images of certain classes. We have decided to use R2D2 as the literature has shown it outperforms other approaches in finding key features [3]. However, we will be utilizing similar search algorithms described in the scale-invariant approach. Furthermore, we will compare feature vectors using cosine similarity. This outperforms euclidean distance in high dimension space like that of R2D2 descriptors.

## References


[1] Lowe, D. G. (1999, September). Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision* (Vol. 2, pp. 1150-1157). Ieee.

[2] Matas, J., Chum, O., Urban, M., & Pajdla, T. (2004). Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, *22*(10), 761-767.

[3] Revaud, J., Weinzaepfel, P., De Souza, C., Pion, N., Csurka, G., Cabon, Y., & Humenberger, M. (2019). R2D2: repeatable and reliable detector and descriptor. *arXiv preprint arXiv:1906.06195*

[4] *Facts about human trafficking in the US*. DeliverFund.org. (2021, April 30). Retrieved December 10, 2021, from https://deliverfund.org/blog/facts-about-human-trafficking-in-the-us/.

## *WA2, Team 8*

### Introduction to Concepts

Precision and recall are two common metrics used in evaluated algorithm performance in the task of information retrieval. If every object among the results of the retrieval algorithm can be classified exclusively as “correct” or exclusively as “incorrect,” then we can define precision and recall as follows. Precision is the proportion of correct objects among the retrieved objects (ie the search results). In statistics, precision is referred to as the positive predictive value of the sample, where the correct results are the true positives, and all others are true negatives.

Recall is the proportion of all correct objects that were retrieved. Again, the analogous concept in statistics is called sensitivity and refers to the proportion of true positives to false negatives. With variance depending on the specific characteristics of the application, these two metrics are often useful in evaluating how well an algorithm is performing at retrieving desired objects.

The maximum coverage problem is another classic concept in computer science, and it can be described informally as follows. Given a collection of sets and a value k, how can you choose at most k sets so that your choice of sets contains as many elements as possible? Note that for this problem to be hard, we assume that the initial sets overlap to some degree. That is, the intersection of the original sets is nonempty, otherwise a simple greedy algorithm of choosing the largest remaining set until k sets are chosen would find the correct solution quickly. The general problem is known to be NP-hard, and thus the maximum coverage problem is frequently used when teaching about approximation algorithms.

### Our problem and how it can be modeled

Our task, at a high level, is to use a query image of a hotel room to predict the specific hotel room in which that image was taken. This task can be performed by returning the k nearest images each of which has a known hotel, or it can be performed by returning the k nearest hotels directly. Regardless, we utilize key points for our approach. Thus, the search problem can be viewed as: *given a set of query points $Q$, a set of database points $D$, and a distance metric on points, return the set $K \subset D$ with $|K| = k$, such that, for each point $p \in K$, the distance from $p$ to the nearest point in $D$ is minimal.* In particular, points are 128-dimension vectors, and the distance metric is cosine distance (one minus cosine similarity).

While there may be multiple ways to frame our problem as a maximum coverage problem, perhaps the most obvious one is as follows. Consider the full database $D$ of all points. Now, the crucial step for transforming this into a maximum coverage problem is to recognize that some points are close, by the metric of cosine similarity, even among those in different hotels. Thus, we project the points of $D$ into a smaller space where all points within a certain threshold of closeness are now a single point (this could be modeled as a subset of those corresponding points, or a new point altogether). This allows us to view the sets of hotels as overlapping sets. Consider that this universe of projected points, call it $D'$ is partitioned into sets $H_1, H_2, \ldots, H_m$ corresponding to the total $m$ hotels in the database. Now, it is clear that some $h \in D'$ could appear in distinct $H_i$. (Note we are leaving out details on this project, since the particular method used does not impact the high level conceptual task and how it is useful to

think this way to see an application of the maximum coverage problem. Now simply consider a query image of points, $Q \subseteq D'$, and notice that finding the $k$ hotels (choices among subsets of $H_1, H_2, \ldots, H_m$) which contain as many points as possible in $Q$ is a maximum coverage problem.

**Discussion of application of the described concepts to our problem**

In our problem, precision corresponds to the ratio of the number of points returned by our search algorithm from the correct hotel to the number of points from an incorrect hotel. A search algorithm for this problem that has higher precision than another will lead to the correct hotel being selected from the search results more frequently and efficiently. This being the goal of the search utility, we see that high precision is a, perhaps *the*, desirable trait of our search algorithm.

Recall, on the other hand, corresponds to the ratio of points among those returned by the search which are from the correct hotel to the total of all points in the database $D$ that are from the correct hotel. Notice that this is not necessarily a desirable trait. It may be reasonable to expect that an increase in recall leads to an increase in precision (more of the correct points are being returned), but recall on its own does not necessarily help fulfill the goal of the system. Identifying the correct hotel for any particular image does not require returning a high proportion of the total points from that hotel in the database because many such points could be descriptors of unique features which do not appear in the query image.

The premise that motivates the use of key points is simply that images of hotel rooms have patches (referred to as points) of pixels that are unique in some way to that hotel. A painting, for instance, with especially uncommon colors or a highly contrasting peculiar shape is the kind of point that we expect to be described by a point(s) in the database. That said, such a point is likely to be returned only when that painting appears in the query image. Of course, query images from this same hotel may not always have said painting, and therefore this returning this point is not always indicative of quality search results. For this reason, we do not expect recall to be, for our problem, as useful a metric as precision, with precision being the metric that better measures the quality of the output as conceived of with the goal of our system in mind.

Fitting our problem into the more general maximum coverage problem is a useful intellectual exercise for considering the various ways in which we can abstract our task. That said, there are a few reasons that the maximum coverage problem may not perfectly resemble hotel recognition with keypoints, but there are also some reasons that it may fit. Fundamentally, a maximum coverage problem does not strictly obtain sets of large size, sets that cover many points. Rather, the problem favors sets which cover many points *that are not common among other sets*. This could suggest that a hotel set that covers many of the points in our image is not selected despite it covering, perhaps even more points in our image than any other hotel. On the other hand, initial conjecture but more importantly preliminary testing results suggest that the types of points that will lead to good matches in hotels are those points which match unusually well from a query to their hotel. That is, there are many points which match similarly as well among all the hotels (they have similar, poorly separated similarity density distributions), but there seem to usually be relatively few (on the order of 10 to 100) points in an image that match particularly well just to their hotel.

If these results hold true on the full scale, it may be the case that the results of a good approximation of the k-maximum coverage result will often contain the correct hotel, since this hotel may be the only hotel to have some of these peculiar points. This is even more likely to be true if we successfully discriminate between and ultimately remove points which tend to be redundant among hotels. There are, as mentioned before, points like those found on grey walls or white sheets that are common to many or all hotels, and if these are the kinds of points that are successfully removed, then the k-maximum coverage results may tend to contain the correct result.

To conclude our discussion of the maximum coverage problem, we will point out that, more than anything else, framing our problem as a maximum coverage problem has expanded the ways in which we can think about abstracting our problem, a useful result in and of itself. Finally, we reemphasize the observation that, as we have conceptualized it, a kind of projection of the search space is necessary to induce overlap between the hotel sets (which, on their own, partition the space). This projection makes the maximum coverage abstraction possible but also suggest that it may not be the most natural one.