Censorship Detection

John Doros
john.doros10@bcmail.cuny.edu

Supervisor <Professor Chen, chen@sci.brooklyn.cuny.edu>

Abstract

Censorship detection is a ambitious challenge in understanding internet freedom as organizations use various forms of content restriction and blocking policies. This project aims to focus on preprocessing and data aggregation on censorship measurement data from Censored Planet to achieve a more accurate form of detecting censorship events. I will focus on processing raw censorship measurement data integrating metadata to it and applying aggregation methods to identify patterns in censorship behavior. My goal for this project is to develop an efficient data processing method and or approach that enables to identify potential censorship events while differentiating from false positives for example such as network outages. The understanding and methods developed from this project will contribute to a broader approach in efforts for censorship circumvention and or reliability in the future internet. This project will focus on supporting the CenDTect framework focusing on step 1 of CenDTect Architecture which is Raw data preprocessing, which transforms censorship measurement data into a more suitable structured format for a better approach to censorship analysis.Throughout this project the main goal is to advance the effectiveness of censorship detection methods and to contribute to a more reliable internet.

Research Focus & Hypothesis

This project focuses on how censorship measurement data can be processed and aggregated to improve the accuracy of detecting censorship while reducing the amount of false positives. By analyzing a large Censored Planet dataset I aim to develop a structured approach for classifying censorship events and be able to differentiate between actual censorship events and non censorship events such as network failures, my goal is to as well see how effective my proposed approach will be.

Research Question

How can censorship data be effectively aggregated and preprocessed to improve censorship detection accuracy while reducing false positives?

Hypothesis

If censorship data is preprocessed with aggregation techniques and with metadata integration then censorship events can become more accurately identified while reducing false positives.

Background on Censorship Mechanisms

Background on censorship mechanisms.

DNS Blocking: Preventing domain name lookups such as failing or redirecting

IP Blocking : Preventing connections to a IP address

HTTP/HTTPS Filtering: Blocking specific content on webpage

Network Disruptions: Packet injections or network outages

Previous Research in Censorship Detection

OONI : Volunteer based censorship measurement gathering organization

Censored Planet : Censorship measurement database using global vantage points

CenDTect : Decision tree based censorship event detection

Research Methodology

Data Collection

Extract censorship data from Censored Planted

Preprocessing and feature extraction

Cleaning the extracted data and extract metadata (ASN, HTTP response codes, ISP, TLS details)

Classification and testing

Categorize response as normal censorship, likely censorship or unlikely censorship, compare these detected events with OONI reports for validation

Aggregation and trend analysis

Grouping data by ISP, country or censorship type, and assess false positives (such as network outage)

Finally measure accuracy of censorship detection (by comparing with ground truth)

Tools

GitHub - Used for uploading all project related files, used for backup as well.

SQL - Used for storing and querying censorship data efficiently supporting data aggregation analysis.

Dropbox - Used for retrieving censorship data provided by my supervisor.

PySide2 (Python Library) - Used for creating simple interactive interfaces for visualizing censorship analysis if needed.

Matplotlib - Used for visualizing censorship analysis if needed.

OONI/Censys - Has real world censorship data trends used for cross checking my own data set to determine its accurateness.

Python - Programming language used for data processing/validation/aggregation makes it easier for data filtering

Data Sources

The data sources that will be used are as followed

Censored Planted “https://censoredplanet.org/” - Used for extracting raw censorship data measurement (Professor has provided a dropbox link where he had extracted this data so I can use it)

OONI “https://ooni.org/” - Possibly used for cross checking/validating censorship events for further accurateness

Citizen Lab “https://citizenlab.ca/” - Can be used for further checking/validating censorship events

ICANN “https://www.icann.org/” - Can be used for metadata extraction on IP address

WHOIS “https://who.is/” - Can be used for metadata extraction on IP address

OONI Measurement Toolkit “https://explorer.ooni.org/chart/mat”- Can be utilized for checking/validating censorship events

Use Cases

Some use cases for this technical implementation can be as follows

Identifying government website blocking : Governments in some areas block access to website contents to which can include news, social media and other websites to which governments do not like people to access at a particular time. This is important as people in society would form a bias towards the government and whether the government should be trusted or not.

Distinguishing between network outage and real censorship events : When it comes to detecting censorship events some events can create false positives to which the access to certain content are not caused by a censorship event but rather a network failure and or outage. This use case can minimize those false positives by distinguishing from a network error and a censorship event, this can have an impact on our data that we are processing to which can make our data more reliable and more accurate when it comes to detecting a censorship event.

Tracking censorship events over time : When it comes to a censorship event taking place such as during a political event, protests and or elections tracking censorship during these periods can be helpful for monitoring censorship trends over time to detect patterns and obtain further analysis. This can be useful to push for transparency during these periods and further can help detecting new censorship events and patterns.

Expectations and Future Work

-Research expected contributions

Develop a structured data processing approach for detecting censorship

Created an improvement to reduce false positive

Provide trend analysis on censorship patterns over time

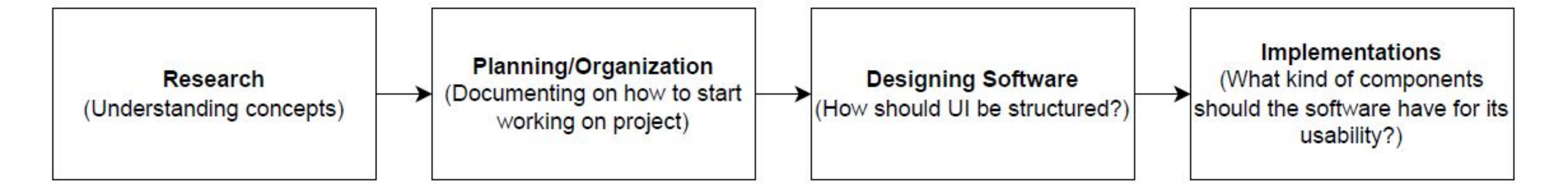
-Future considerations

Using machine learning models for censorship classification

Improve accuracy through combination of multiple datasets

Expand to focus on detecting new types of censorship

Evolution



Research
(Understanding concepts)

Planning/Organization
(Documenting on how to start working on project)

Designing Software
(How should UI be structured?)

Implementations
(What kind of components should the software have for its usability?)

Started the project by researching and understanding concepts that were related to the censorship detection research report. Next was to plan and organize how project materials were to be organized and how I should start working on the project itself. The planning portion was about first about designing an intuitive interface for the software and how components should be laid out onto the UI. In the implementation phase I had to connect and work on components that were necessary for software data processing/usability which is the current phase.

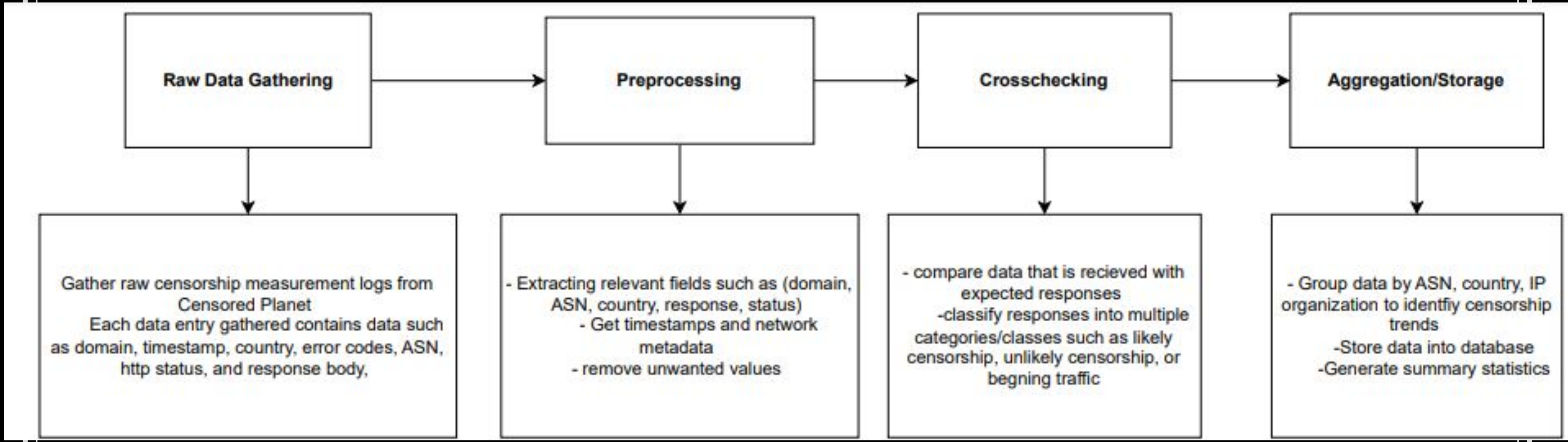
Research Phase

In the research phase I had to understand concepts that were presented in the “Cndetect” research paper, some concepts that I researched were what types of censorship events occur such as for example on the DNS level, TCP level/Http level. Why are there challenges in validating censorship events as organizations do not report on censorship policies so validating censorship events is a difficult challenge. How different censorship detection methods can lead to false positives as there can be a detection on censorship while there is no actual event happening leading to a false positive such as utilizing time series based anomaly detection, or stationary data analysis. How “Cndetect” utilizes their own methods for censorship discovery by utilizing decision trees to find blocking policies in events and then are organized into different censorship categories. And most importantly how “Cndetect” utilizes their approach in the aggregate measurement process in order to process raw data more effectively, enriching each raw data event with additional metadata to improve detection accuracy which is the focus of this project by implementing this approach in order to classify raw data events as a censored vs uncensored event. The challenge is to come up with the aggregate measurement approach as “Cndetect” does not describe their implementation.

Organization Phase:

Logical flow of data processing (Sequence of operations on data)

Showing the what happens on processing pipeline in detail



Raw Data Gathering

Gather raw censorship measurement logs from Censored Planet
Each data entry gathered contains data such as domain, timestamp, country, error codes, ASN, http status, and response body,

Preprocessing

- Extracting relevant fields such as (domain, ASN, country, response, status)
- Get timestamps and network metadata
- remove unwanted values

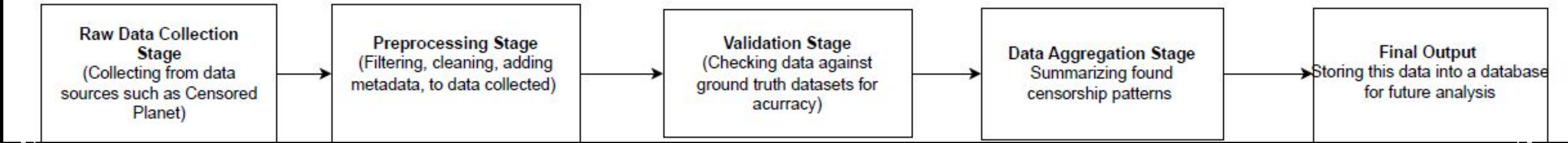
Crosschecking

- compare data that is recieved with expected responses
-classify responses into multiple categories/classes such as likely censorship, unlikely censorship, or begning traffic

Aggregation/Storage

- Group data by ASN, country, IP organization to identfiy censorship trends
-Store data into database
-Generate summary statistics

Stage Flow Diagram For Data Processing



Raw Data Collection Stage
(Collecting from data sources such as Censored Planet)

Preprocessing Stage
(Filtering, cleaning, adding metadata, to data collected)

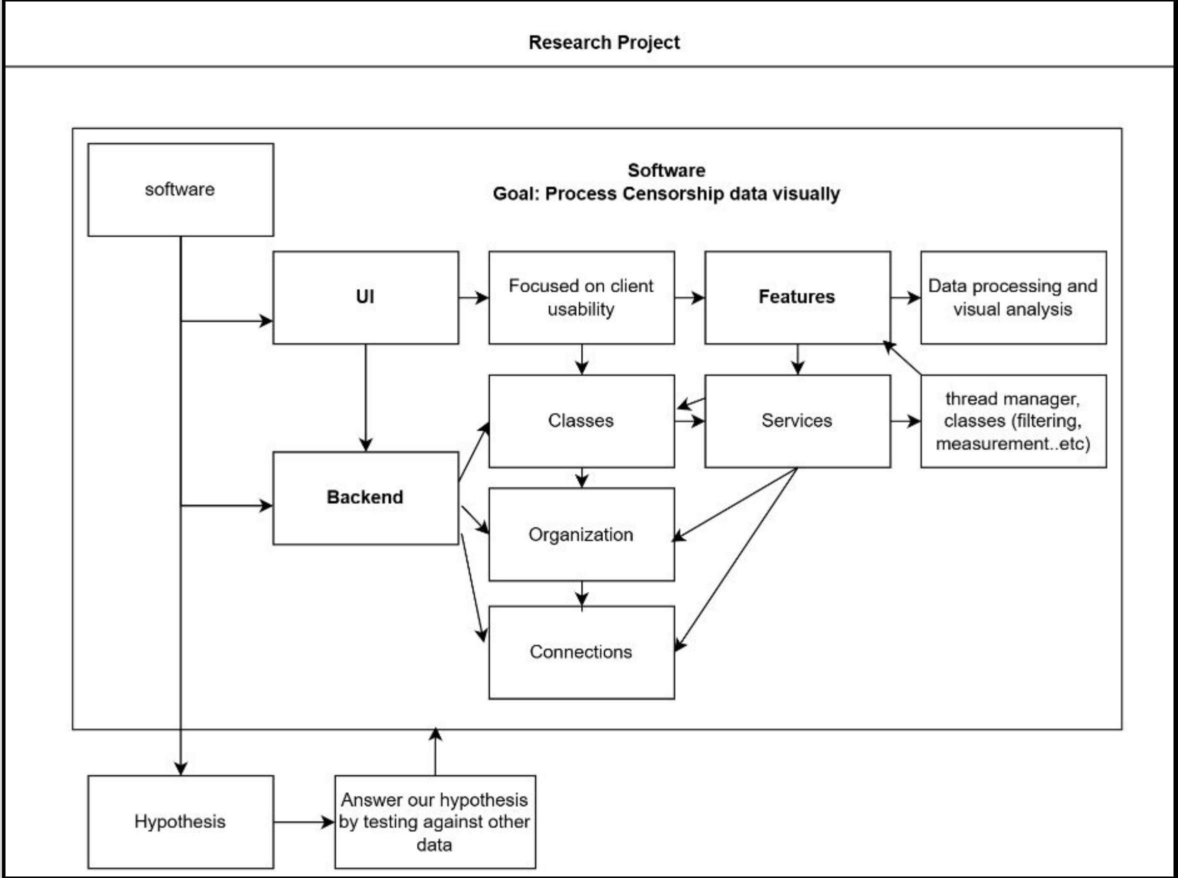
Validation Stage
(Checking data against ground truth datasets for acurracy)

Data Aggregation Stage
Summarizing found censorship patterns

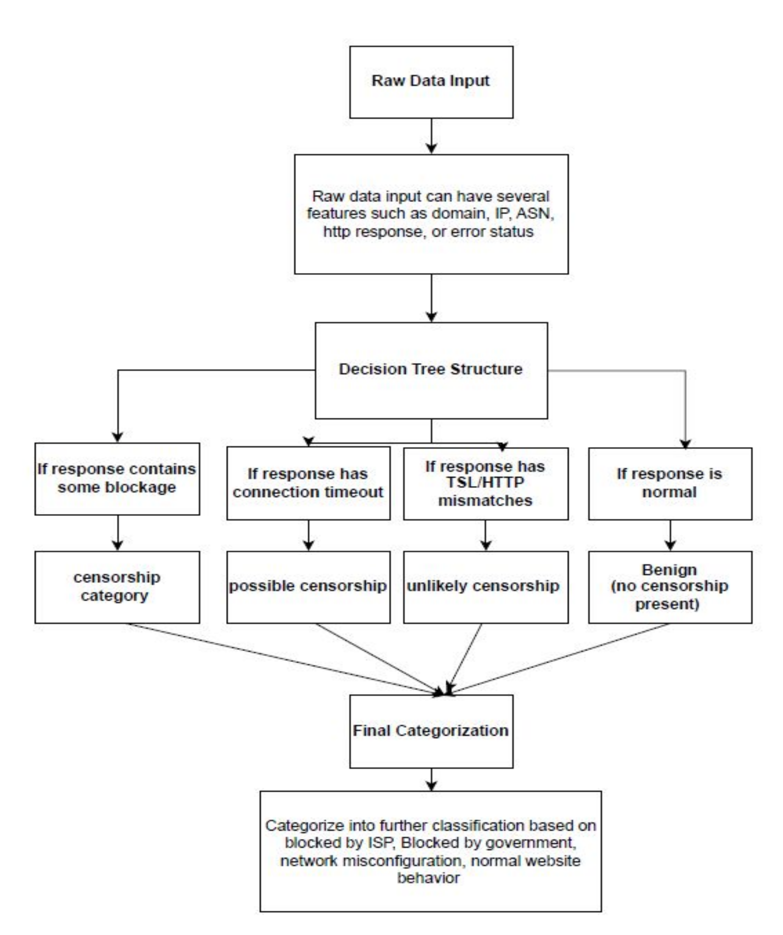
Final Output
Storing this data into a database for future analysis

Showing the data processing states at each stage and what will happen to the raw data at each stage.

Organization Phase: Structured Design Plan


Research Project
software
Software
Goal: Process Censorship data visually
UI
Focused on client usability
Features
Data processing and visual analysis
Classes
Services
thread manager, classes (filtering, measurement..etc)
Backend
Organization
Connections
Hypothesis
Answer our hypothesis by testing against other data

We can design the software using a modular structured approach, dividing areas into distinct components such as UI, backend, classes and services. This allows users to easily navigate and interact with the software itself. This type of approach ensures efficient data processing and visual analysis by allowing components to communicate seamlessly such as the UI sending user inputs to the backend, which then works with classes for tasks such as data filtering or thread management.By keeping these connections organized the software can handle complex operations such as censorship data analysis more effectively while maintaining a smooth and responsive user experience which we have to keep in mind as we are developing the software.


Raw Data Input
Raw data input can have several features such as domain, IP, ASN, http response, or error status
Decision Tree Structure
If response contains some blockage
If response has connection timeout
If response has TSL/HTTP mismatches
If response is normal
censorship category
possible censorship
unlikely censorship
Benign (no censorship present)
Final Categorization
Categorize into further classification based on blocked by ISP, Blocked by government, network misconfiguration, normal website behavior

Organization Phase:

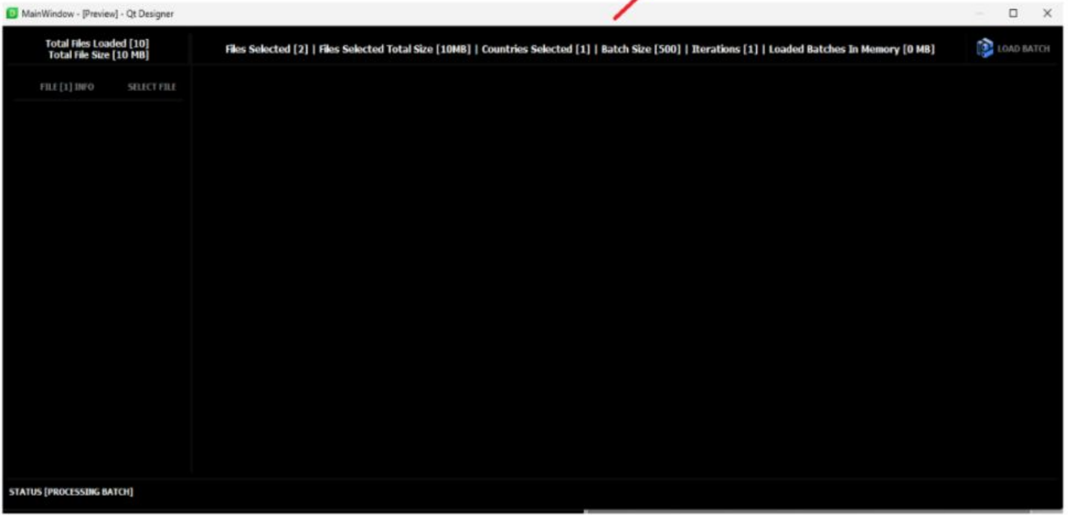
How can censorship events can be classified?

A censorship event can be classified using a decision tree flow approach, to which evaluates different conditions and based from those conditions an answer can be produced, such as sorting the events into categories for example "likely censorship" or "not censored". We can get inspiration by the Cndetects research report to start with this approach, and as we continue building the software we can refine it to make classifications more accurate overtime with using this decision tree approach or by different methods.

Designing Software

User Interface (Beginning)

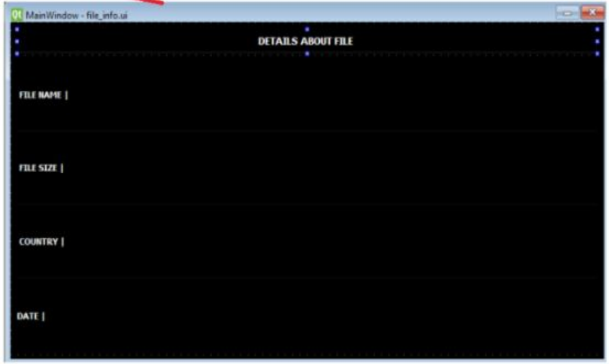
The current software UI design making the data processing easily shown to the user which can give us a better understanding of the data we are processing



Total Files Loaded [10]
Total File Size [10 MB]

Files Selected [2] | Files Selected Total Size [10MB] | Countries Selected [1] | Batch Size [500] | Iterations [1] | Loaded Batches In Memory [0 MB]

STATUS [PROCESSING BATCH]



DETAILS ABOUT FILE

FILE NAME |

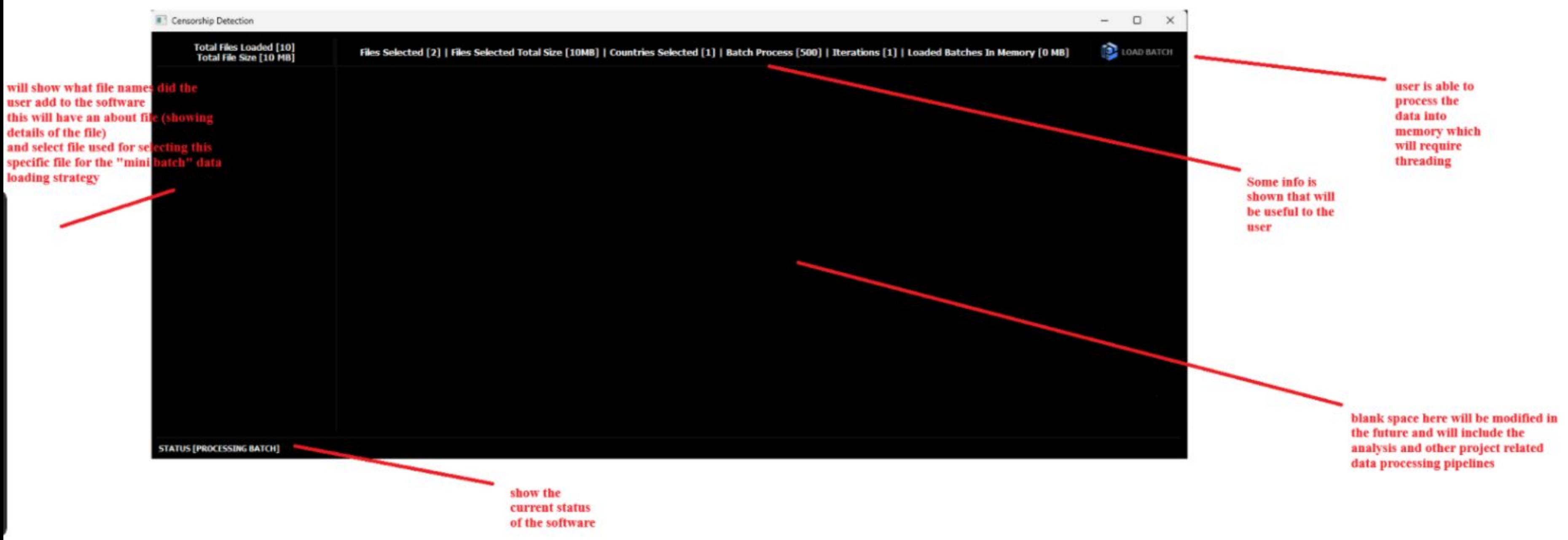
FILE SIZE |

COUNTRY |

DATE |

Designing Software

User Interface (Beginning)



Total Files Loaded [10]
Total File Size [10 MB]

Files Selected [2] | Files Selected Total Size [10MB] | Countries Selected [1] | Batch Process [500] | Iterations [1] | Loaded Batches In Memory [0 MB]

STATUS [PROCESSING BATCH]

will show what file names did the user add to the software
this will have an about file (showing details of the file)
and select file used for selecting this specific file for the "mini batch" data loading strategy

Some info is shown that will be useful to the user

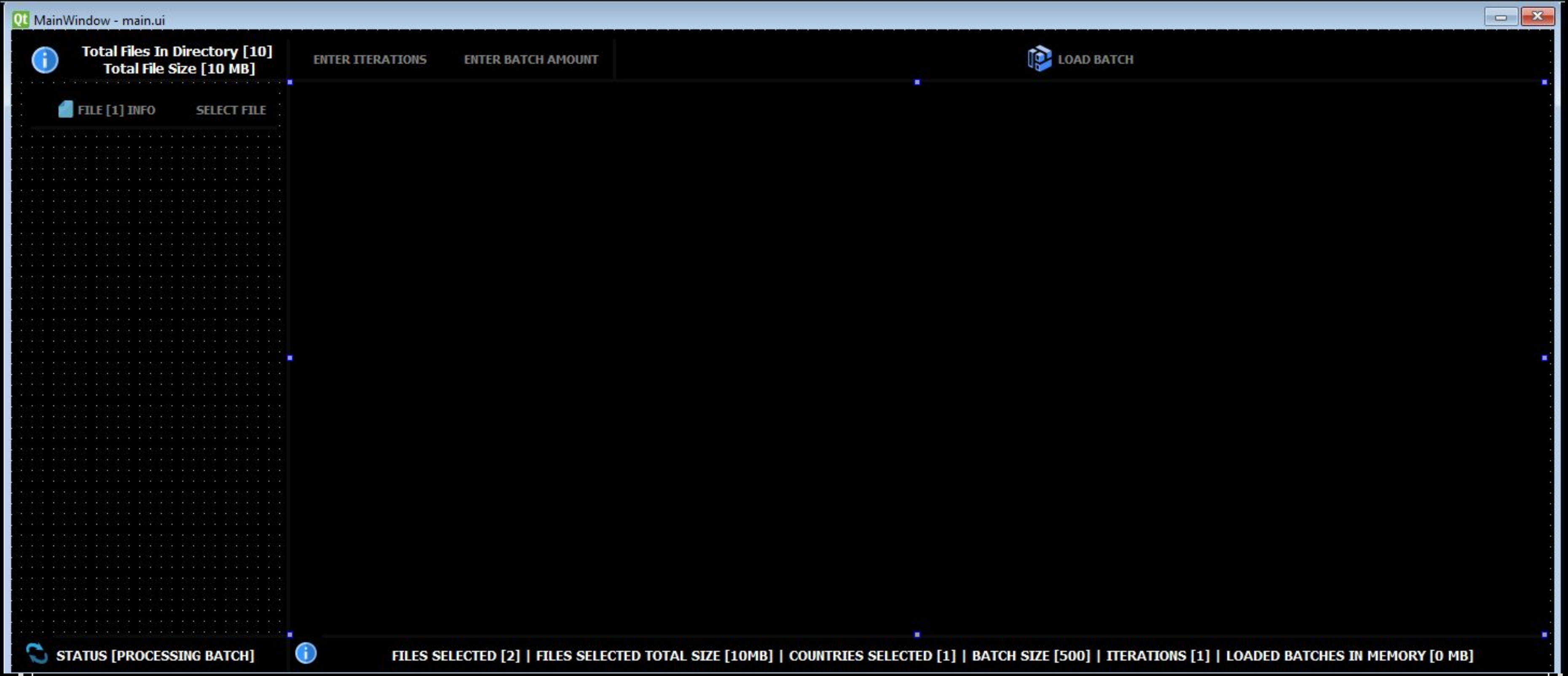
user is able to process the data into memory which will require threading

blank space here will be modified in the future and will include the analysis and other project related data processing pipelines

show the current status of the software

Designing Software

User Interface (Halfway)



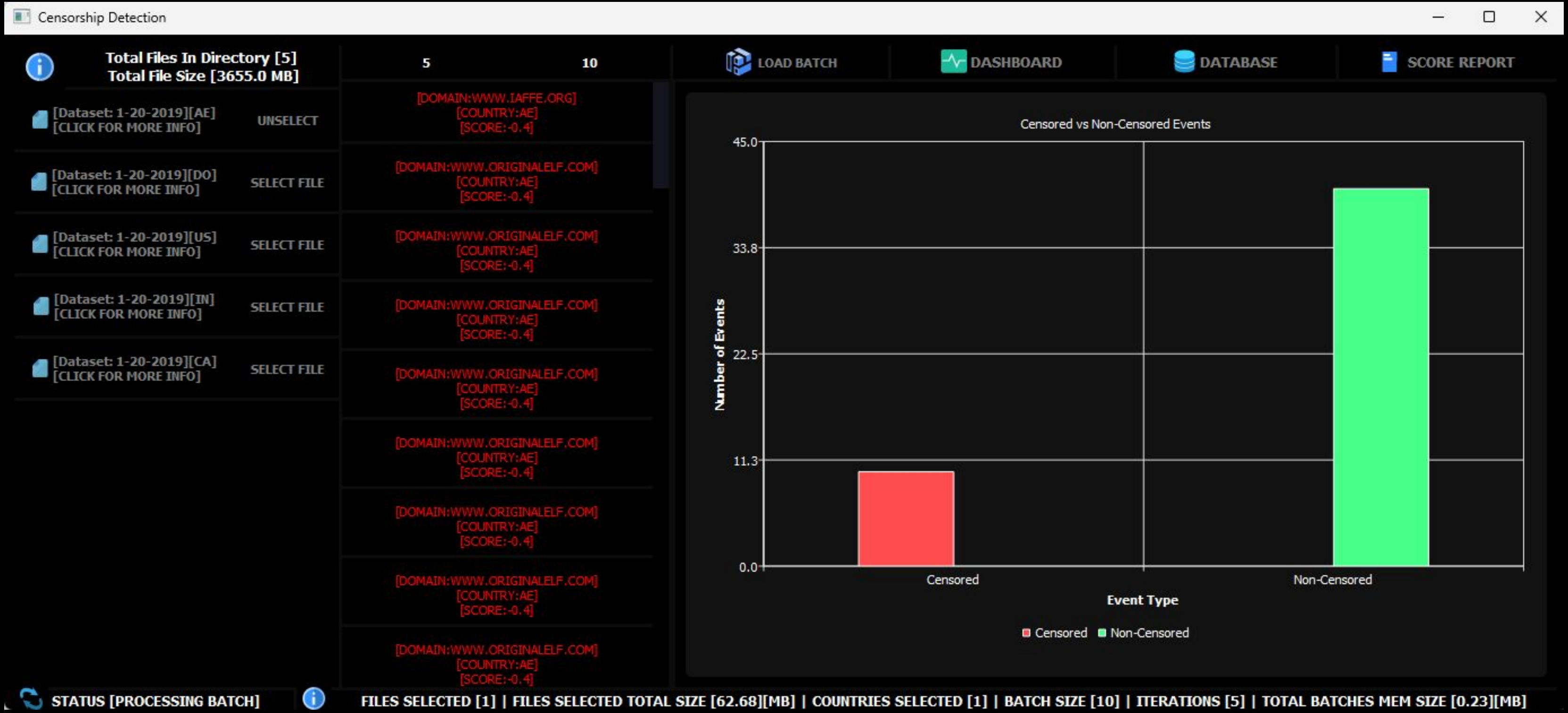
Total Files In Directory [10]
Total File Size [10 MB]

STATUS [PROCESSING BATCH]

FILES SELECTED [2] | FILES SELECTED TOTAL SIZE [10MB] | COUNTRIES SELECTED [1] | BATCH SIZE [500] | ITERATIONS [1] | LOADED BATCHES IN MEMORY [0 MB]

Designing Software

User Interface (Current)



Total Files In Directory [5]
Total File Size [3655.0 MB]

[Dataset: 1-20-2019][AE]
[CLICK FOR MORE INFO]

[Dataset: 1-20-2019][DO]
[CLICK FOR MORE INFO]

[Dataset: 1-20-2019][US]
[CLICK FOR MORE INFO]

[Dataset: 1-20-2019][IN]
[CLICK FOR MORE INFO]

[Dataset: 1-20-2019][CA]
[CLICK FOR MORE INFO]

[DOMAIN:WWW.IAFFE.ORG]
[COUNTRY:AE]
[SCORE:-0.4]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]
[SCORE:-0.4]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]
[SCORE:-0.4]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]
[SCORE:-0.4]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]
[SCORE:-0.4]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]
[SCORE:-0.4]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]
[SCORE:-0.4]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]
[SCORE:-0.4]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]
[SCORE:-0.4]

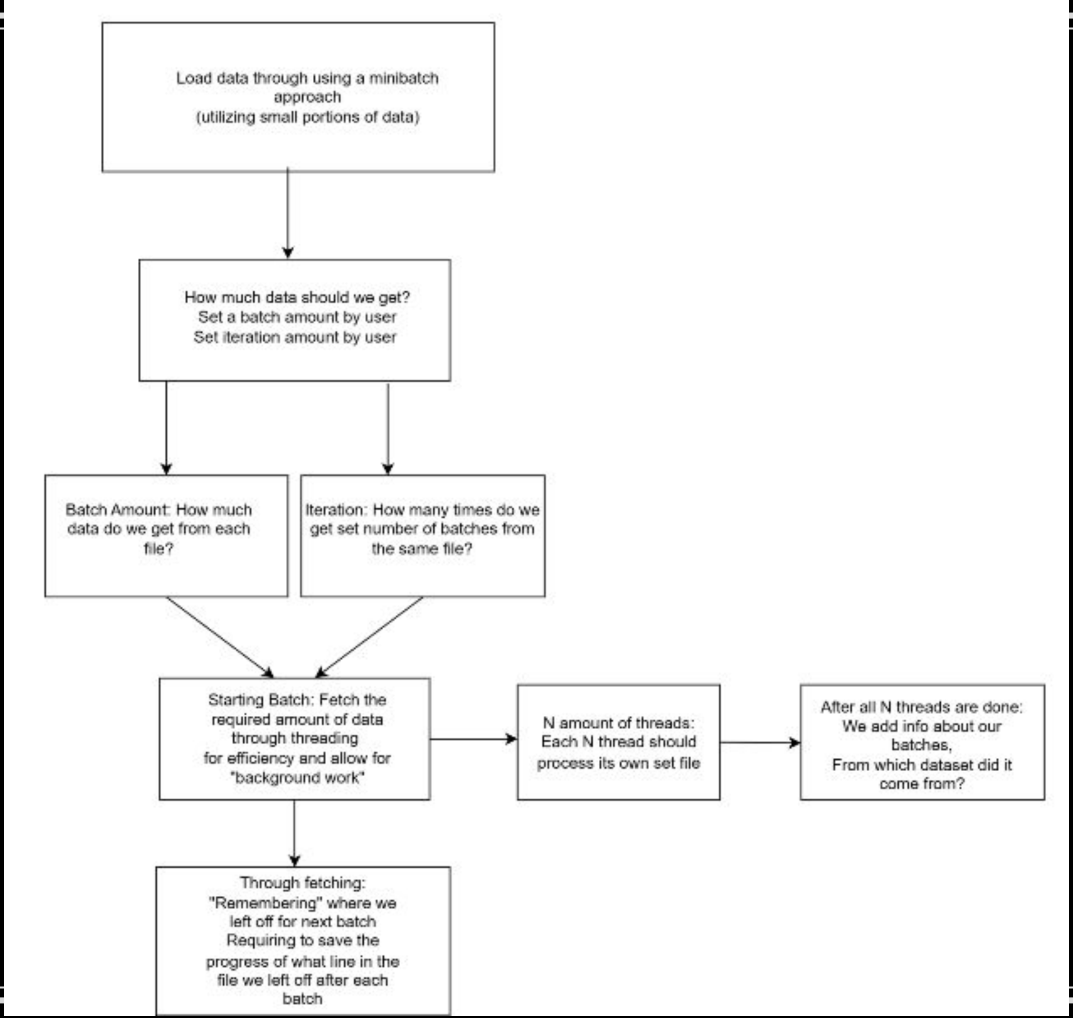
STATUS [PROCESSING BATCH]

FILES SELECTED [1] | FILES SELECTED TOTAL SIZE [62.68][MB] | COUNTRIES SELECTED [1] | BATCH SIZE [10] | ITERATIONS [5] | TOTAL BATCHES MEM SIZE [0.23][MB]

Implementation Phase

Backend Components

(Example: miniBatch Class)



Load data through using a minibatch approach (utilizing small portions of data)

How much data should we get? Set a batch amount by user Set iteration amount by user

Batch Amount: How much data do we get from each file?

Iteration: How many times do we get set number of batches from the same file?

Starting Batch: Fetch the required amount of data through threading for efficiency and allow for "background work"

N amount of threads: Each N thread should process its own set file

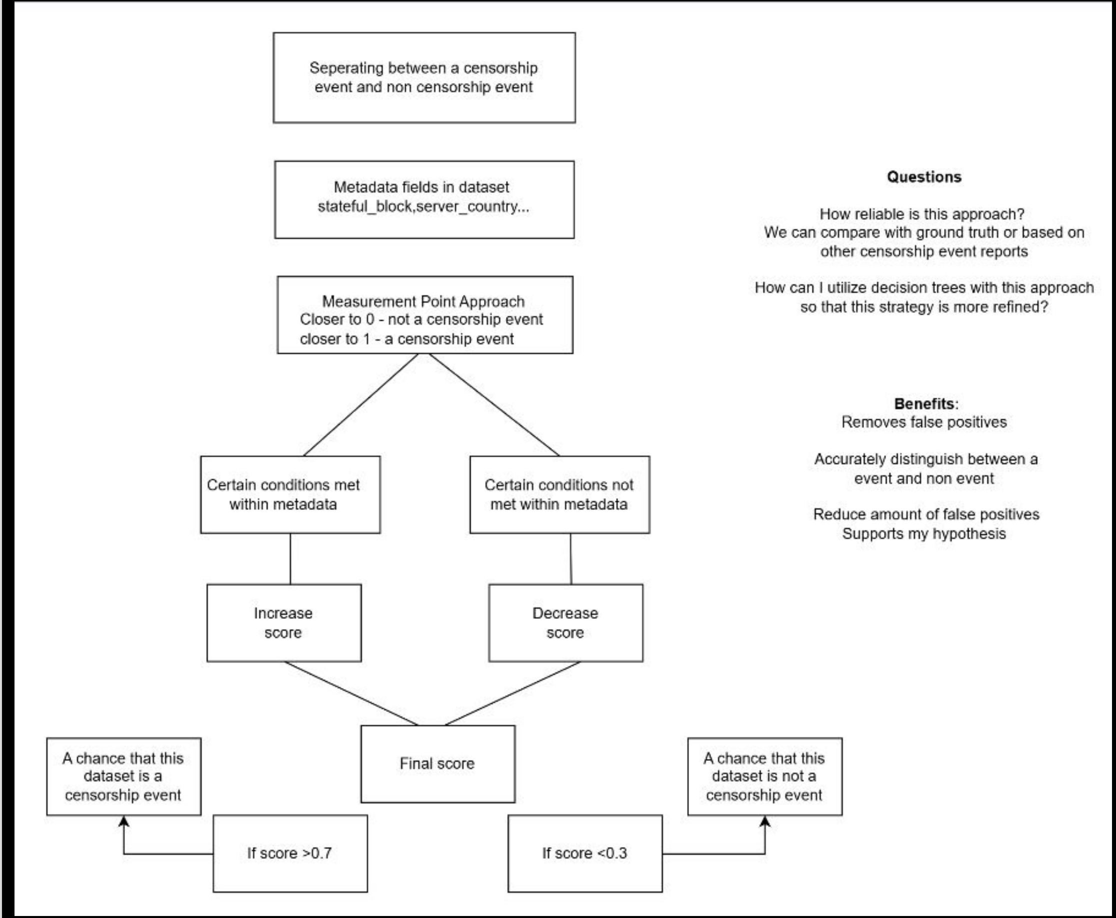
After all N threads are done: We add info about our batches, From which dataset did it come from?

Through fetching: "Remembering" where we left off for next batch Requiring to save the progress of what line in the file we left off after each batch

Overview of one of the classes used in the backend of software for example **miniBatch** class in which is an implementation of an approach used to load in dataset data, as the software cannot just load in the entire dataset into memory as the software will slowdown and will not work, so a minibatch approach is implemented and used to let the user select how much data they are willing to process in their machine. The dataset is split into small chunks so only a small amount of data is processed with the help of threading.

Implementation Phase Backend Components

(Example: PointMeasurement Class)



Seperating between a censorship event and non censorship event

Metadata fields in dataset stateful_block,server_country...

Measurement Point Approach
Closer to 0 - not a censorship event
closer to 1 - a censorship event

Certain conditions met within metadata

Certain conditions not met within metadata

Increase score

Decrease score

Final score

If score >0.7

A chance that this dataset is a censorship event

If score <0.3

A chance that this dataset is not a censorship event

Questions

How reliable is this approach?
We can compare with ground truth or based on other censorship event reports

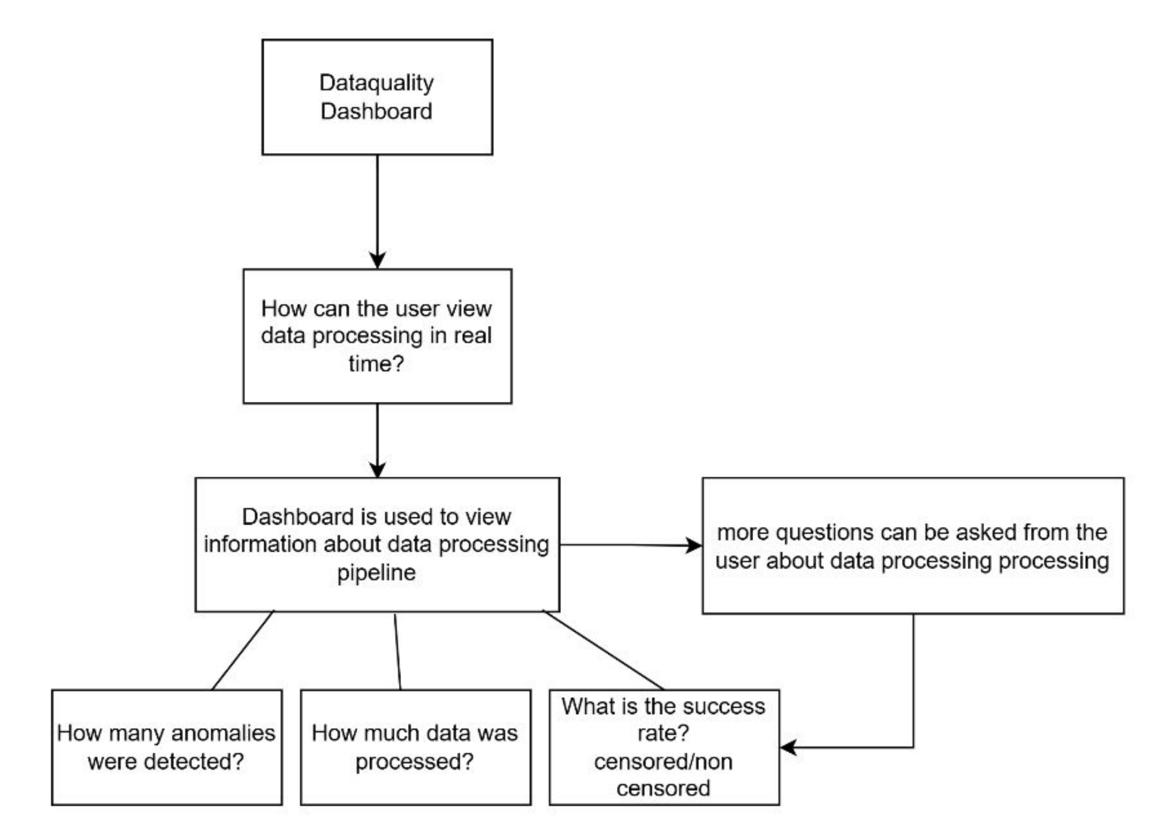
How can I utilize decision trees with this approach so that this strategy is more refined?

Benefits:
Removes false positives

Accurately distinguish between a event and non event

Reduce amount of false positives
Supports my hypothesis

Overview of one of the classes used in the backend of software for example **PointMeasurement** class in which is an implementation of an approach used to determine whether an event is either a censored or non-censored event by utilizing a score for each dataset after examining the metadata fields presented in the dataset each metadata field with a certain condition met can really influence whether this particular dataset is a censorship event or not


Dataquality Dashboard
How can the user view data processing in real time?
Dashboard is used to view information about data processing pipeline
more questions can be asked from the user about data processing processing
How many anomalies were detected?
How much data was processed?
What is the success rate? censored/non censored

Updated Designs

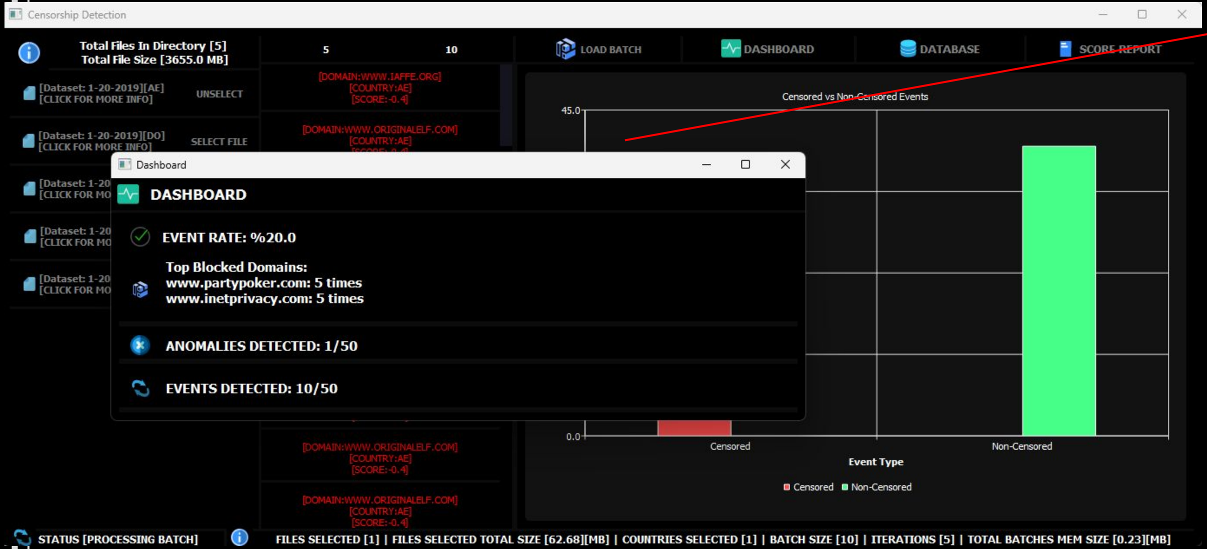
(Dataquality Dashboard)

A new implementation that is used to show the user more insights into the data processing pipeline, some questions that this dashboard would answer is
Were there any anomalies or skipped events?
How much data was processed?
What is the rate of censored vs uncensored events?

There might be more questions to be answered but we can implement these later for now we should implement an intuitive dashboard that gives the most important insights.

Updated Designs

(Dataquality Dashboard)



Total Files In Directory [5]
Total File Size [3655.0 MB]

[Dataset: 1-20-2019][AE]
[CLICK FOR MORE INFO]

[Dataset: 1-20-2019][DO]
[CLICK FOR MORE INFO]

[Dataset: 1-20
[CLICK FOR MO

[Dataset: 1-20
[CLICK FOR MO

[Dataset: 1-20
[CLICK FOR MO

[DOMAIN:WWW.IAFFE.ORG]
[COUNTRY:AE]
[SCORE:-0.4]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]
[SCORE:-0.4]

[DOMAIN:WWW.ORIGINALELF.COM]
[COUNTRY:AE]
[SCORE:-0.4]

EVENT RATE: %20.0

Top Blocked Domains:
www.partypoker.com: 5 times
www.inetprivacy.com: 5 times

ANOMALIES DETECTED: 1/50

EVENTS DETECTED: 10/50

STATUS [PROCESSING BATCH]

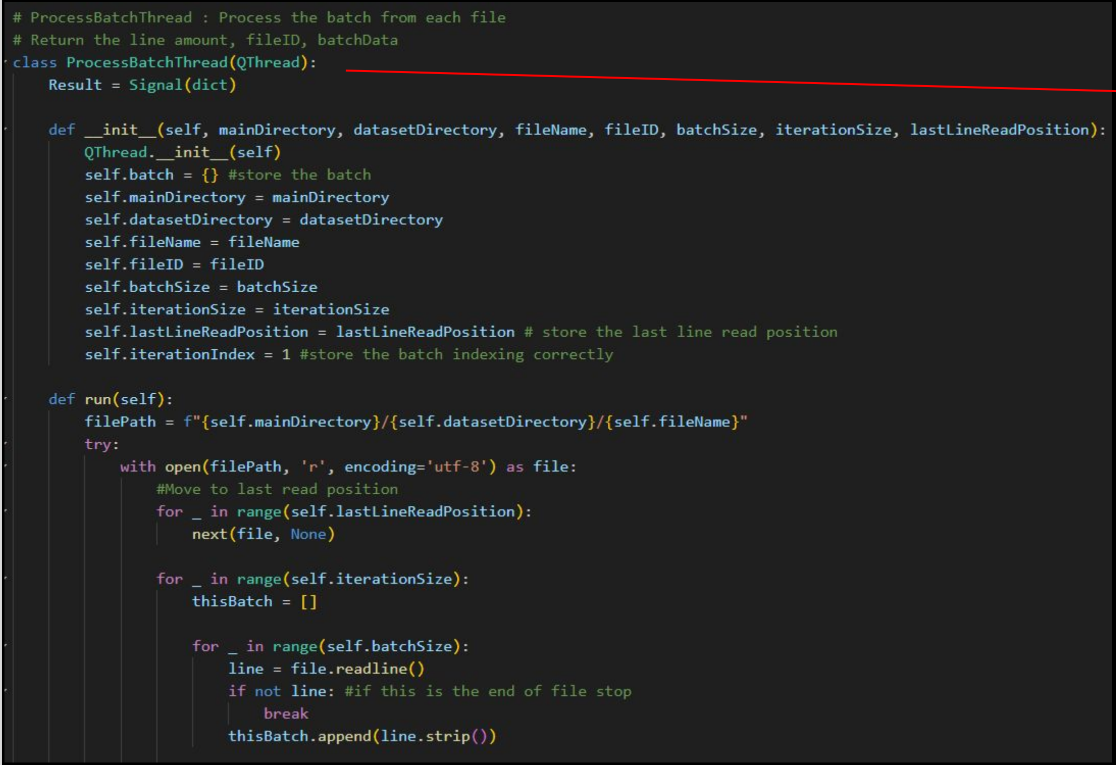
FILES SELECTED [1] | FILES SELECTED TOTAL SIZE [62.68][MB] | COUNTRIES SELECTED [1] | BATCH SIZE [10] | ITERATIONS [5] | TOTAL BATCHES MEM SIZE [0.23][MB]

We can see this current implemented feature here which I am excited about since we can get more insights into our data that is processed, we can continue working on this feature to incorporate dataset extraction after the data processing is done, but for now it only answers the most important questions about the datasets processed.

Event Rate: Shows us how many events were considered to be censored based out of all of it
Anomalies : Shows how many datasets produced errors in the metafields themselves (Missing metadata)

MiniBatch Class

(Challenges)

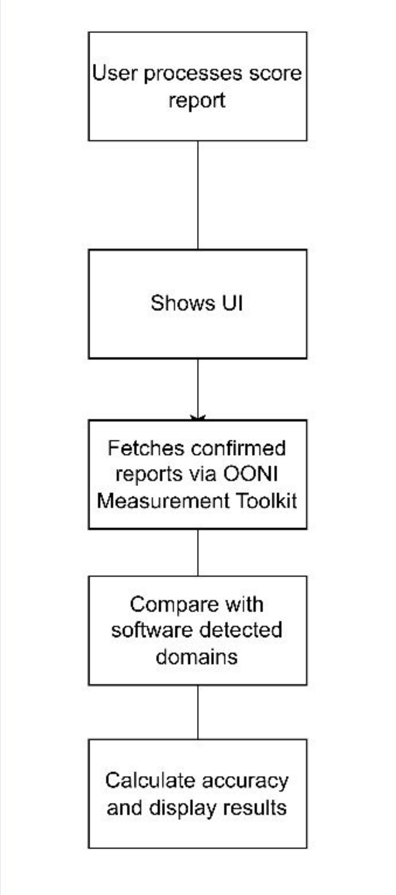


```
# ProcessBatchThread : Process the batch from each file
# Return the line amount, fileID, batchData
class ProcessBatchThread(QThread):
    Result = Signal(dict)

    def __init__(self, mainDirectory, datasetDirectory, fileName, fileID, batchSize, iterationSize, lastLineReadPosition):
        QThread.__init__(self)
        self.batch = {} #store the batch
        self.mainDirectory = mainDirectory
        self.datasetDirectory = datasetDirectory
        self.fileName = fileName
        self.fileID = fileID
        self.batchSize = batchSize
        self.iterationSize = iterationSize
        self.lastLineReadPosition = lastLineReadPosition # store the last line read position
        self.iterationIndex = 1 #store the batch indexing correctly

    def run(self):
        filePath = f"{self.mainDirectory}/{self.datasetDirectory}/{self.fileName}"
        try:
            with open(filePath, 'r', encoding='utf-8') as file:
                #Move to last read position
                for _ in range(self.lastLineReadPosition):
                    next(file, None)

                for _ in range(self.iterationSize):
                    thisBatch = []

                    for _ in range(self.batchSize):
                        line = file.readline()
                        if not line: #if this is the end of file stop
                            break
                        thisBatch.append(line.strip())
```

One of the challenges during the implementation phase of the software was processing large amounts of data efficiently so we utilize the minibatch approach but at the same time the problem was that the software cannot do other work while it is processing this data, so one solution I had to come up with and through reading documentation was to utilize qthreads in which will help run data processing in the “background” while the software/UI can do other work. So implementing Qthreading (threading via interface) helped in a sense that the user is now able to do other operations on the software without having to worry about waiting until data processing is complete. We can see this threading implementation in our thread manager class.

Confirming Censored Data

(Challenges)

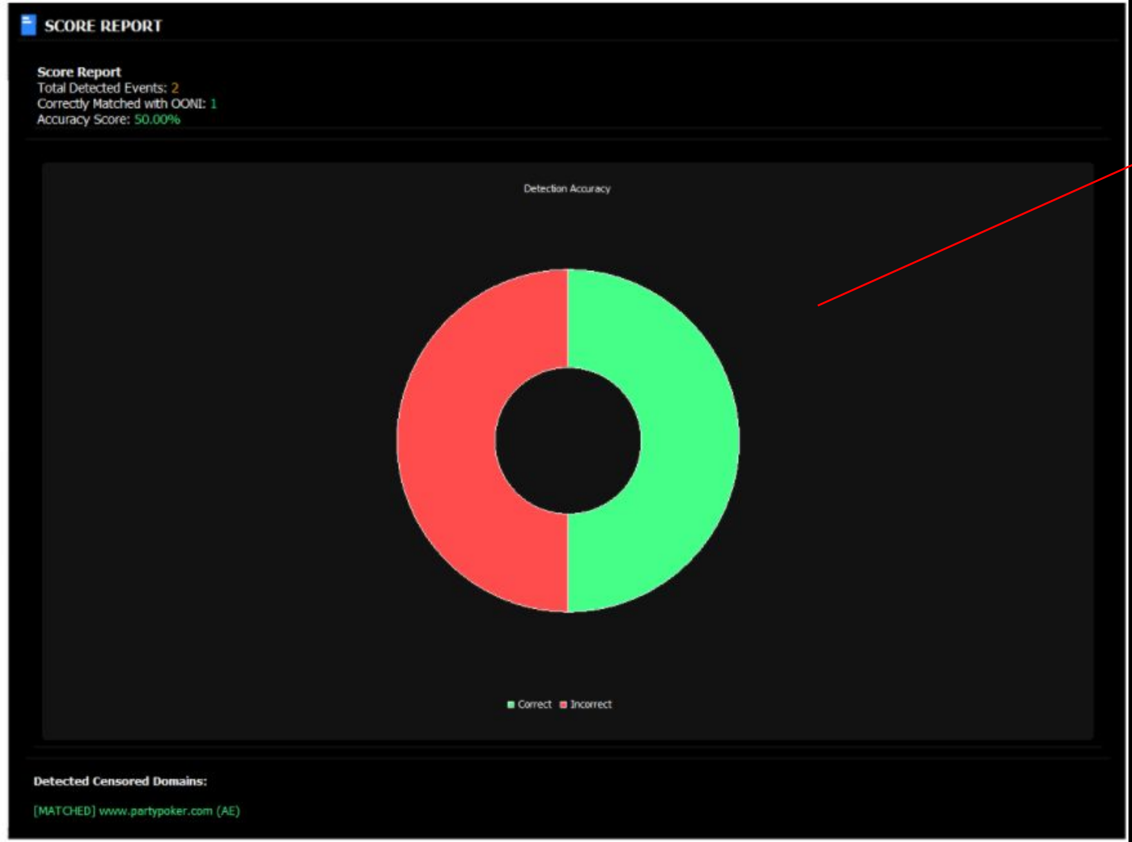

User processes score report
Shows UI
Fetches confirmed reports via OONI Measurement Toolkit
Compare with software detected domains
Calculate accuracy and display results

One of the main challenges in the project was to verify whether our proposed approach of measuring censorship via measurement score is accurate. To address this I decided to use a well known source of censored event data which helps confirm the reliability of the findings. We can simply break this solution down into steps:

User Interaction: The process starts with the user who generates censored events through data processing via the software.

Fetching Confirmed Data: The software then connects to the OONI Measurement Toolkit through api requests which collects real time global network measurements from probes worldwide. The toolkit provides detailed data such as blocked websites - which is what we need

Comparison : Once we fetched the data we can compare it with domains the software has identified as censored then analyzes and calculates an accuracy score. The results are displayed to the user showing how well the measurement approach matches with the OONI data, with more details on the results provided on the next slide



SCORE REPORT

Score Report
Total Detected Events: 2
Correctly Matched with OONI: 1
Accuracy Score: 50.00%

Detected Censored Domains:

[MATCHED] www.partypoker.com (AE)

Updated Designs

(ScoreReport)

Here we can see a very intuitive and approachable accuracy test, with a score report summary interface. I tried to be as simple as possible in visualizing the detection accuracy to which I did accomplish by implementing a simple pie chart, a short score report, and the matched detected censored domains, those domains that our software detected matched with the OONI report. The purpose of keeping this report as intuitive and understandable as possible is so that the user can easily identify the accuracy score. We can see in this example our software detected 2 events in a specific dataset in which one of them were confirmed by the OONI toolkit, meaning that our measurement approach is 50% accurate.

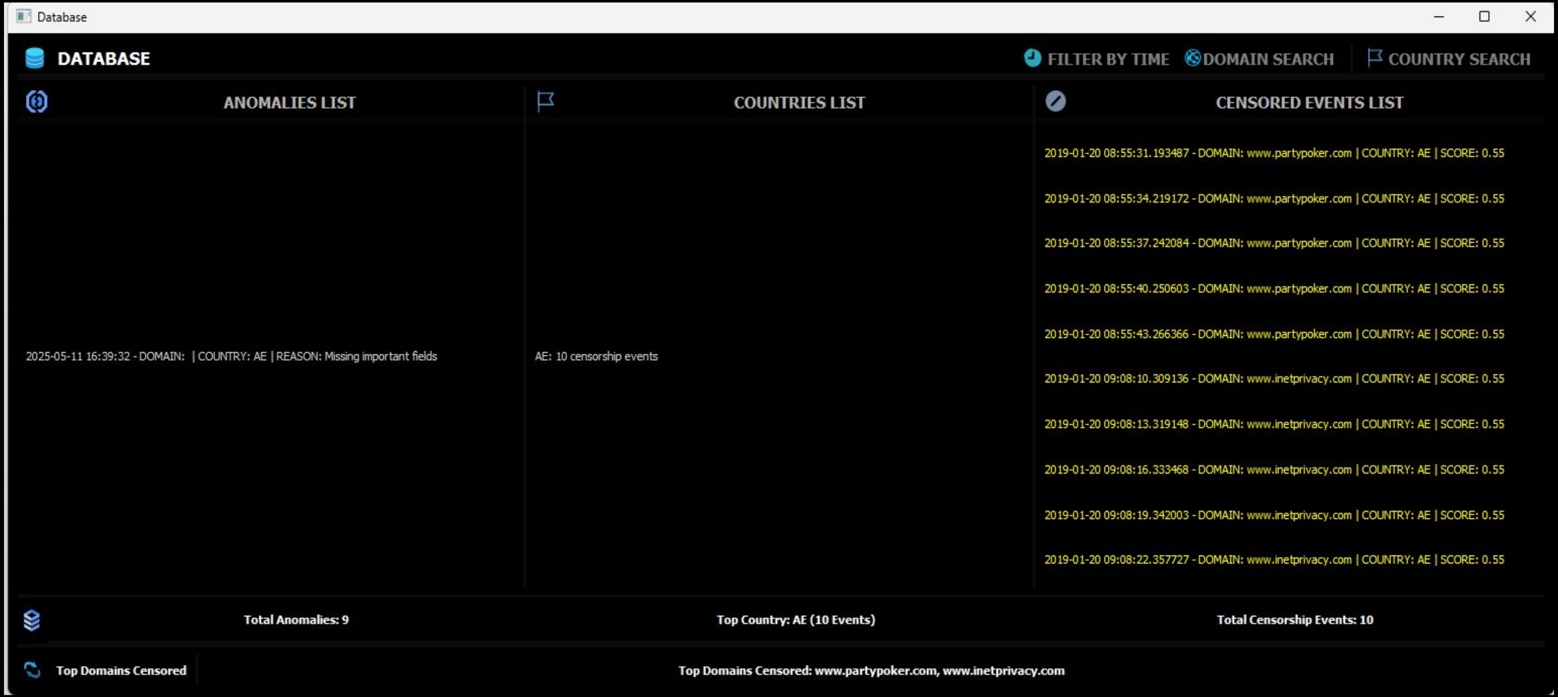
Updated Designs

(Database Challenge)

Another main challenge was storing the softwares processed data as after processing the data itself it is only stored into memory and once the software is closed this processed data is cleared in memory. One solution was to create a database in which we can use for further data analysis and to store this processed data in which we can use for the future as well. We can utilize SQL in which I am familiar with as I am taking a class on database this semester. But we need to implement this into our software so I needed some clarifications on the implementation and utilization process as well so I read the required Sql documentation to implement this feature into the software. I created a simple database interface in which shows us a list of anomalies detected in the datasets, which countries had the censored events and a censored events list. The user can simply filter these lists by time, country and more specifically the type of domain. The goal was to make this feature as intuitive as possible for data processing and usability for the user, we can see the design of the feature on the next slide.

Updated Designs

(Database Interface)



ANOMALIES LIST

```
2025-05-11 16:39:32 - DOMAIN:  | COUNTRY: AE | REASON: Missing important fields
```

COUNTRIES LIST

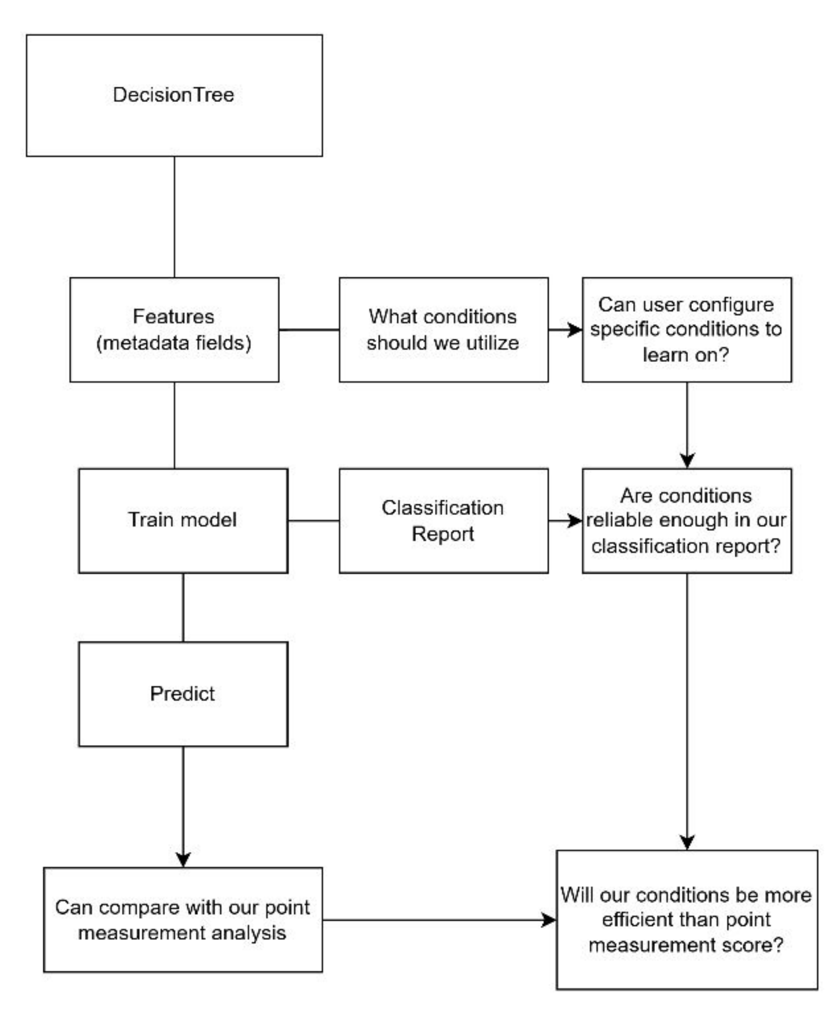
```
AE: 10 censorship events
```

CENSORED EVENTS LIST

```
2019-01-20 08:55:31.193487 - DOMAIN: www.partypoker.com | COUNTRY: AE | SCORE: 0.55
2019-01-20 08:55:34.219172 - DOMAIN: www.partypoker.com | COUNTRY: AE | SCORE: 0.55
2019-01-20 08:55:37.242084 - DOMAIN: www.partypoker.com | COUNTRY: AE | SCORE: 0.55
2019-01-20 08:55:40.250603 - DOMAIN: www.partypoker.com | COUNTRY: AE | SCORE: 0.55
2019-01-20 08:55:43.266366 - DOMAIN: www.partypoker.com | COUNTRY: AE | SCORE: 0.55
2019-01-20 09:08:10.309136 - DOMAIN: www.inetprivacy.com | COUNTRY: AE | SCORE: 0.55
2019-01-20 09:08:13.319148 - DOMAIN: www.inetprivacy.com | COUNTRY: AE | SCORE: 0.55
2019-01-20 09:08:16.333468 - DOMAIN: www.inetprivacy.com | COUNTRY: AE | SCORE: 0.55
2019-01-20 09:08:19.342003 - DOMAIN: www.inetprivacy.com | COUNTRY: AE | SCORE: 0.55
2019-01-20 09:08:22.357727 - DOMAIN: www.inetprivacy.com | COUNTRY: AE | SCORE: 0.55
```

Total Anomalies: 9 **Top Country: AE (10 Events)** **Total Censorship Events: 10**

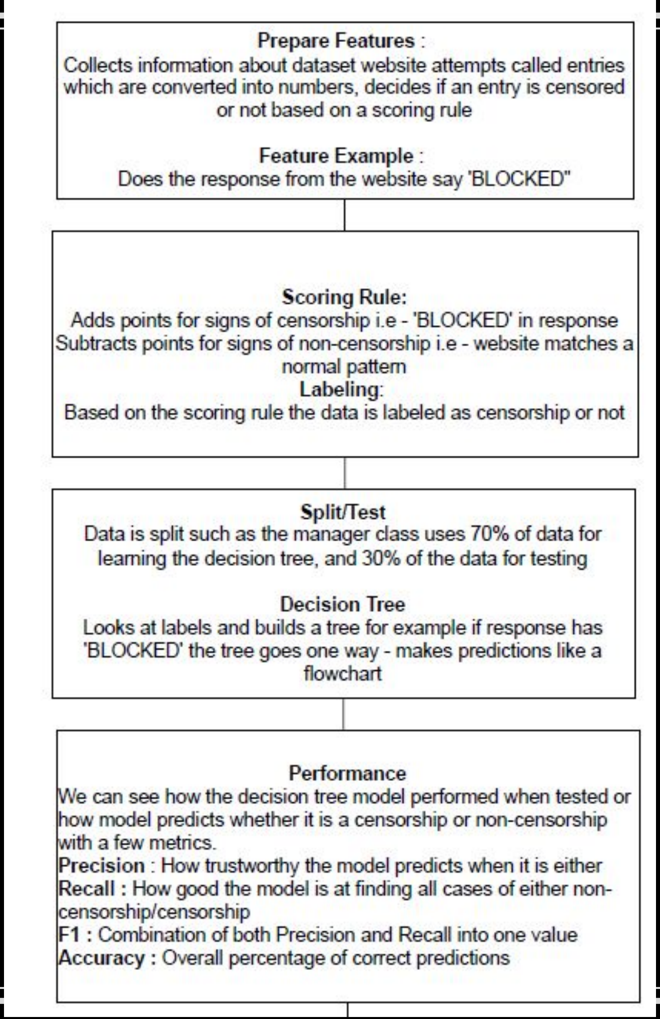
Top Domains Censored **Top Domains Censored: www.partypoker.com, www.inetprivacy.com**


DecisionTree
Features (metadata fields)
What conditions should we utilize
Can user configure specific conditions to learn on?
Train model
Classification Report
Are conditions reliable enough in our classification report?
Predict
Can compare with our point measurement analysis
Will our conditions be more efficient than point measurement score?

Updated Designs

(DecisionTree Implementation)

A new implementation that will use decision trees in order to classify datasets as censored vs non-censored this can help us in a sense that we can compare these results with our point measurement approach and see if one is more reliable than the other. But at the same time we have to figure out a way to prove if our pointMeasurement score is reliable by looking at other censorship event reports, which will be done soon.



Prepare Features :
Collects information about dataset website attempts called entries which are converted into numbers, decides if an entry is censored or not based on a scoring rule

Feature Example :
Does the response from the website say 'BLOCKED"

Scoring Rule:
Adds points for signs of censorship i.e - 'BLOCKED' in response
Subtracts points for signs of non-censorship i.e - website matches a normal pattern
Labeling:
Based on the scoring rule the data is labeled as censorship or not

Split/Test
Data is split such as the manager class uses 70% of data for learning the decision tree, and 30% of the data for testing

Decision Tree
Looks at labels and builds a tree for example if response has 'BLOCKED' the tree goes one way - makes predictions like a flowchart

Performance
We can see how the decision tree model performed when tested or how model predicts whether it is a censorship or non-censorship with a few metrics.
Precision : How trustworthy the model predicts when it is either
Recall : How good the model is at finding all cases of either non-censorship/censorship
F1 : Combination of both Precision and Recall into one value
Accuracy : Overall percentage of correct predictions

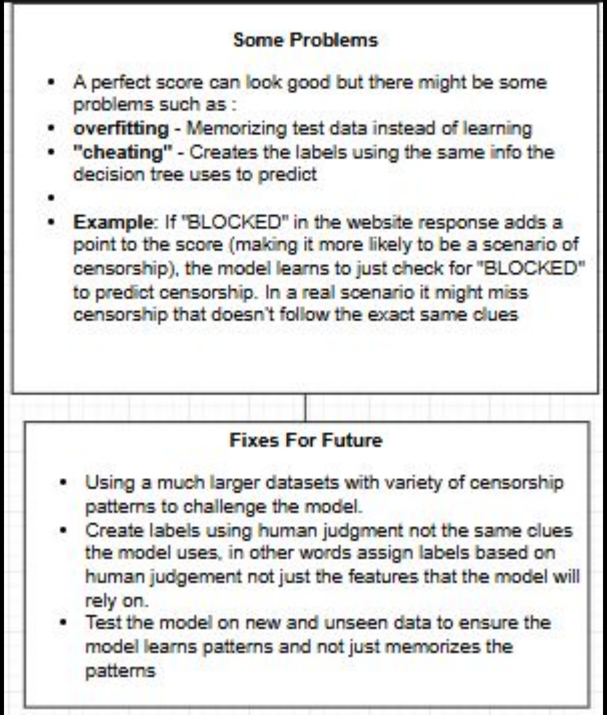
Updated Designs

(DecisionTree Details)

We can view how this DecisionTree manager class implementation works in a much more detailed manner, for example how it works by utilizing features, scoring, labeling and a decision tree to make a final prediction. The way we see how the model has performed is by looking at the performance report with key metrics

Future Considerations

(DecisionTree Details)



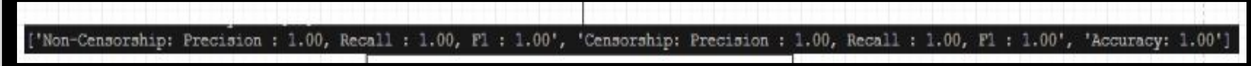
Some Problems

- A perfect score can look good but there might be some problems such as :
- **overfitting** - Memorizing test data instead of learning
- **"cheating"** - Creates the labels using the same info the decision tree uses to predict
-
- **Example**: If "BLOCKED" in the website response adds a point to the score (making it more likely to be a scenario of censorship), the model learns to just check for "BLOCKED" to predict censorship. In a real scenario it might miss censorship that doesn't follow the exact same clues

Fixes For Future

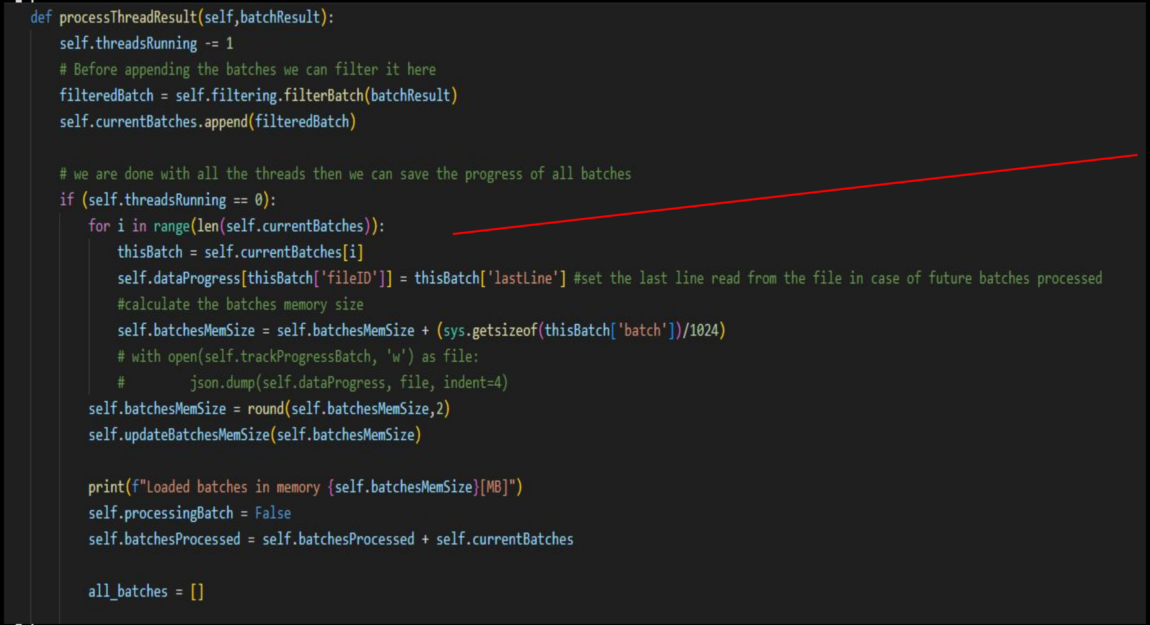
- Using a much larger datasets with variety of censorship patterns to challenge the model.
- Create labels using human judgment not the same clues the model uses, in other words assign labels based on human judgement not just the features that the model will rely on.
- Test the model on new and unseen data to ensure the model learns patterns and not just memorizes the patterns

A performance report based on the model can look good but there can be some red flags in the prediction such as overfitting and cheating. Some future considerations are important to keep in mind to further improve our decision tree approach in our software. Some solutions can be using larger datasets with a variety of censorship patterns, we can use human judgement to create the labels and not just utilize the features that the model will rely on. We can as well test the model on new and unseen data to avoid overfitting.



```
['Non-Censorship: Precision : 1.00, Recall : 1.00, F1 : 1.00', 'Censorship: Precision : 1.00, Recall : 1.00, F1 : 1.00', 'Accuracy: 1.00']
```

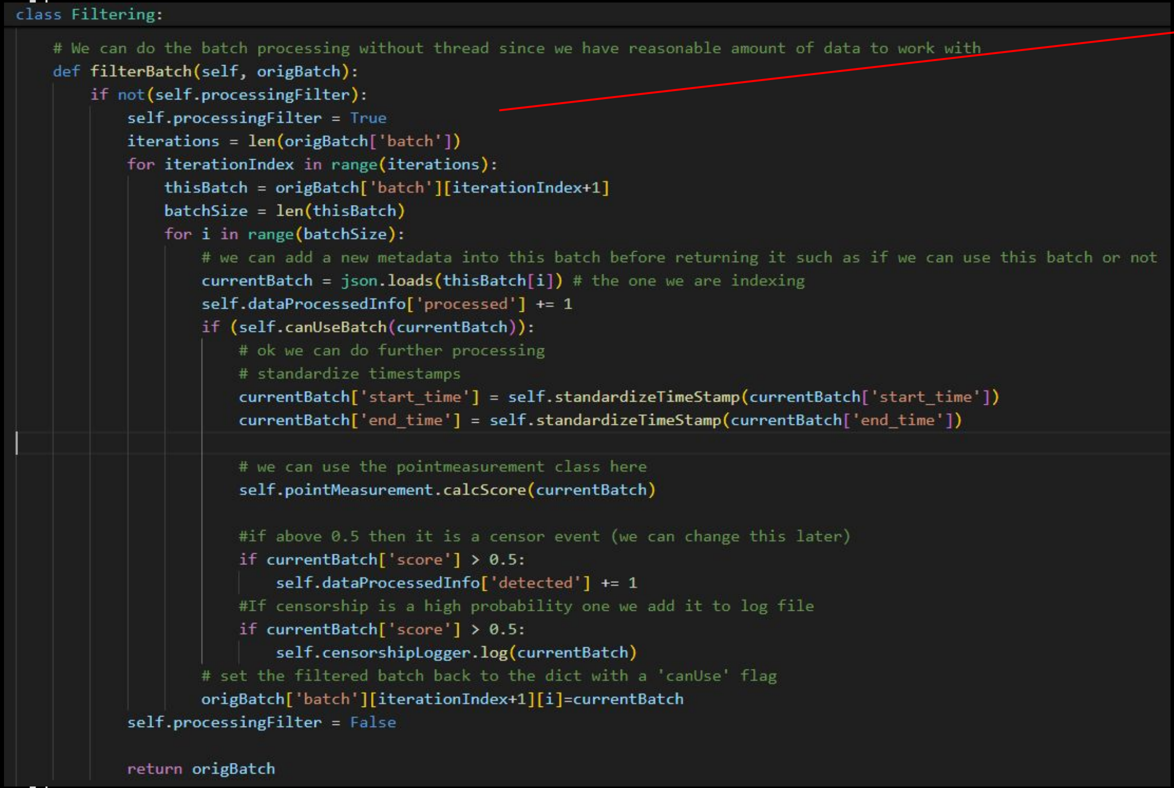
Algorithms



```python
def processThreadResult(self,batchResult):
    self.threadsRunning -= 1
    # Before appending the batches we can filter it here
    filteredBatch = self.filtering.filterBatch(batchResult)
    self.currentBatches.append(filteredBatch)

    # we are done with all the threads then we can save the progress of all batches
    if (self.threadsRunning == 0):
        for i in range(len(self.currentBatches)):
            thisBatch = self.currentBatches[i]
            self.dataProgress[thisBatch['fileID']] = thisBatch['lastLine'] #set the last line read from the file in case of future batches processed
            #calculate the batches memory size
            self.batchesMemSize = self.batchesMemSize + (sys.getsizeof(thisBatch['batch'])/1024)
            # with open(self.trackProgressBatch, 'w') as file:
            #       json.dump(self.dataProgress, file, indent=4)
        self.batchesMemSize = round(self.batchesMemSize,2)
        self.updateBatchesMemSize(self.batchesMemSize)

        print(f"Loaded batches in memory {self.batchesMemSize}[MB]")
        self.processingBatch = False
        self.batchesProcessed = self.batchesProcessed + self.currentBatches

        all_batches = []
```

Some components required to develop unique algorithms in order to process the data efficiently, one example is the batch processing algorithm which not only filters out unusable dataset events but also performs final processing logic in a more modular way. This is the core of the software allowing us to easily adjust how the software interprets censorship events. The learning outcome out of this was that I can improve the way I think about implementing algorithms in a way that will be more modular based.

Algorithms



```
class Filtering:

    # We can do the batch processing without thread since we have reasonable amount of data to work with
    def filterBatch(self, origBatch):
        if not(self.processingFilter):
            self.processingFilter = True
            iterations = len(origBatch['batch'])
            for iterationIndex in range(iterations):
                thisBatch = origBatch['batch'][iterationIndex+1]
                batchSize = len(thisBatch)
                for i in range(batchSize):
                    # we can add a new metadata into this batch before returning it such as if we can use this batch or not
                    currentBatch = json.loads(thisBatch[i]) # the one we are indexing
                    self.dataProcessedInfo['processed'] += 1
                    if (self.canUseBatch(currentBatch)):
                        # ok we can do further processing
                        # standardize timestamps
                        currentBatch['start_time'] = self.standardizeTimeStamp(currentBatch['start_time'])
                        currentBatch['end_time'] = self.standardizeTimeStamp(currentBatch['end_time'])

                        # we can use the pointmeasurement class here
                        self.pointMeasurement.calcScore(currentBatch)

                        #if above 0.5 then it is a censor event (we can change this later)
                        if currentBatch['score'] > 0.5:
                            self.dataProcessedInfo['detected'] += 1
                        #If censorship is a high probability one we add it to log file
                        if currentBatch['score'] > 0.5:
                            self.censorshipLogger.log(currentBatch)
                    # set the filtered batch back to the dict with a 'canUse' flag
                    origBatch['batch'][iterationIndex+1][i]=currentBatch
            self.processingFilter = False

            return origBatch
```

Some components required to develop unique algorithms in order to process the data efficiently, one example is filtering batches. This is one of the core algorithms of the software in which filters the datasets when doing data processing, the filterBatch method handles if this batch is usable or not, if it is then filtering can begin by standardizing timestamps, calculating the measurement score for this batch and handling logging the censored events to the file all in one function call. Again this algorithm was designed to be as modular as possible since we can just modify those components used without modifying the algorithm itself, or in other words if a component goes wrong we can just modify that specific component such as the calcScore method in the pointMeasurement class

Documentation

The most important thing was learning and reading through the QT documentation about designing and working with different interface components that were added to the software interface, such as list views, charts, and how to style the interface itself. Learning and reading documentation helped me implement useful services in the backend of the software such as implementing threading, connecting the UI interface with the backend and implementing a database. So the outcome of learning the documentation really helped out in designing and building the software from the ground up. The learning outcome was that I can further improve my ability to program since with programming I can connect components into the backend making a full stack application such as this project.

Documentation used:

https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/index.html
https://doc.qt.io/
stackoverflow.com
https://docs.python.org/3/library/sqlite3.html

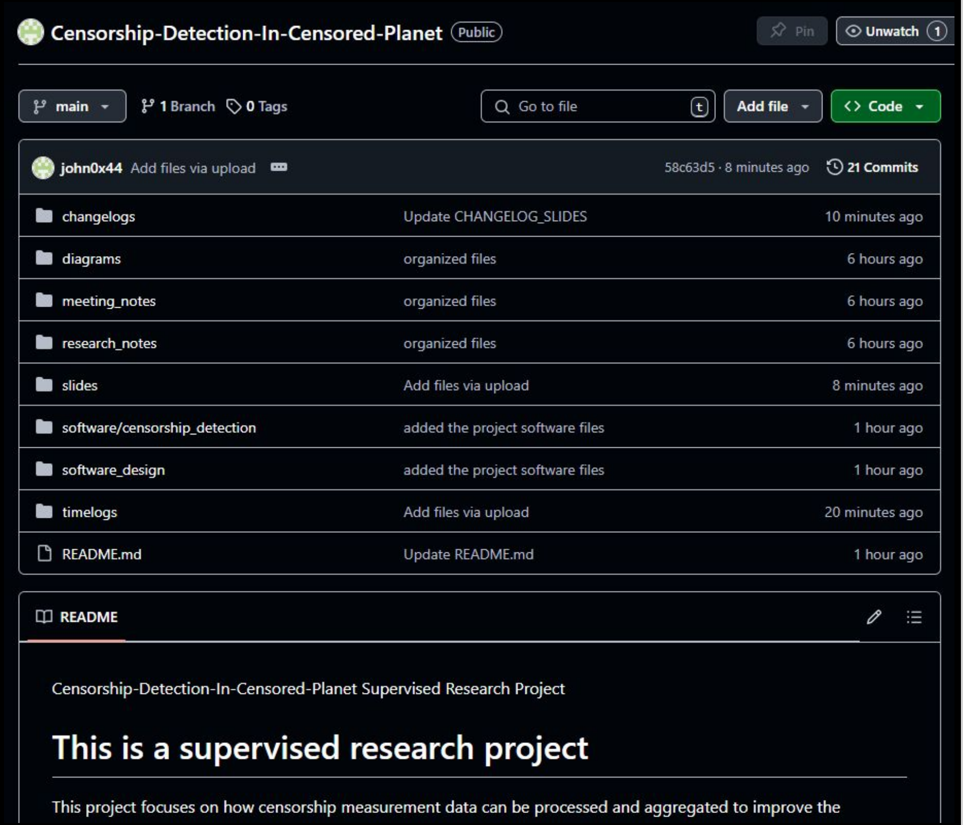
Conclusion

Working on this supervised project was an exciting journey for me as I genuinely enjoyed coding it from scratch and watching it progress from the ground up. I learned many new skills, such as software and algorithm design, creating user interfaces and most importantly full stack development, all at the same time putting my academic knowledge to work by using efficient data structures, good coding practices and database techniques to build a complete full software from the ground up. What made it even more enjoyable was UI design such as figuring out appealing layouts, and themes, while exploring and learning tools such as QT for UI design and documentation which turned my project into an intuitive software that anyone can utilize. The best part was organizing my codebase so that all the components such as the UI, backend and classes worked together in a seamless manner making it easy to add new features as I went. I started with nothing and ended up with a software that I am proud of to which as well proves my hypothesis to which was:

If censorship data is preprocessed with aggregation techniques and with metadata integration then censorship events can become more accurately identified while reducing false positives. I tested this hypothesis by comparing my own measurement score approach with OONI datasets achieving a 50% accuracy rate - but at the same time I see there is room to improve - such as adding in more metadata fields and techniques into to specific classes, and configuring measurement scoring rules, using a variety of different dated datasets than the ones that were provided to me, I think this could boost the accuracy and make my measurement score approach more reliable.

Github Repository

https://github.com/john0x44/Censorship-Detection-In-Censored-Planet/tree/main



john0x44 Add files via upload 58c63d5 · 8 minutes ago 21 Commits

changelogs	Update CHANGELOG_SLIDES	10 minutes ago
diagrams	organized files	6 hours ago
meeting_notes	organized files	6 hours ago
research_notes	organized files	6 hours ago
slides	Add files via upload	8 minutes ago
software/censorship_detection	added the project software files	1 hour ago
software_design	added the project software files	1 hour ago
timelogs	Add files via upload	20 minutes ago
README.md	Update README.md	1 hour ago

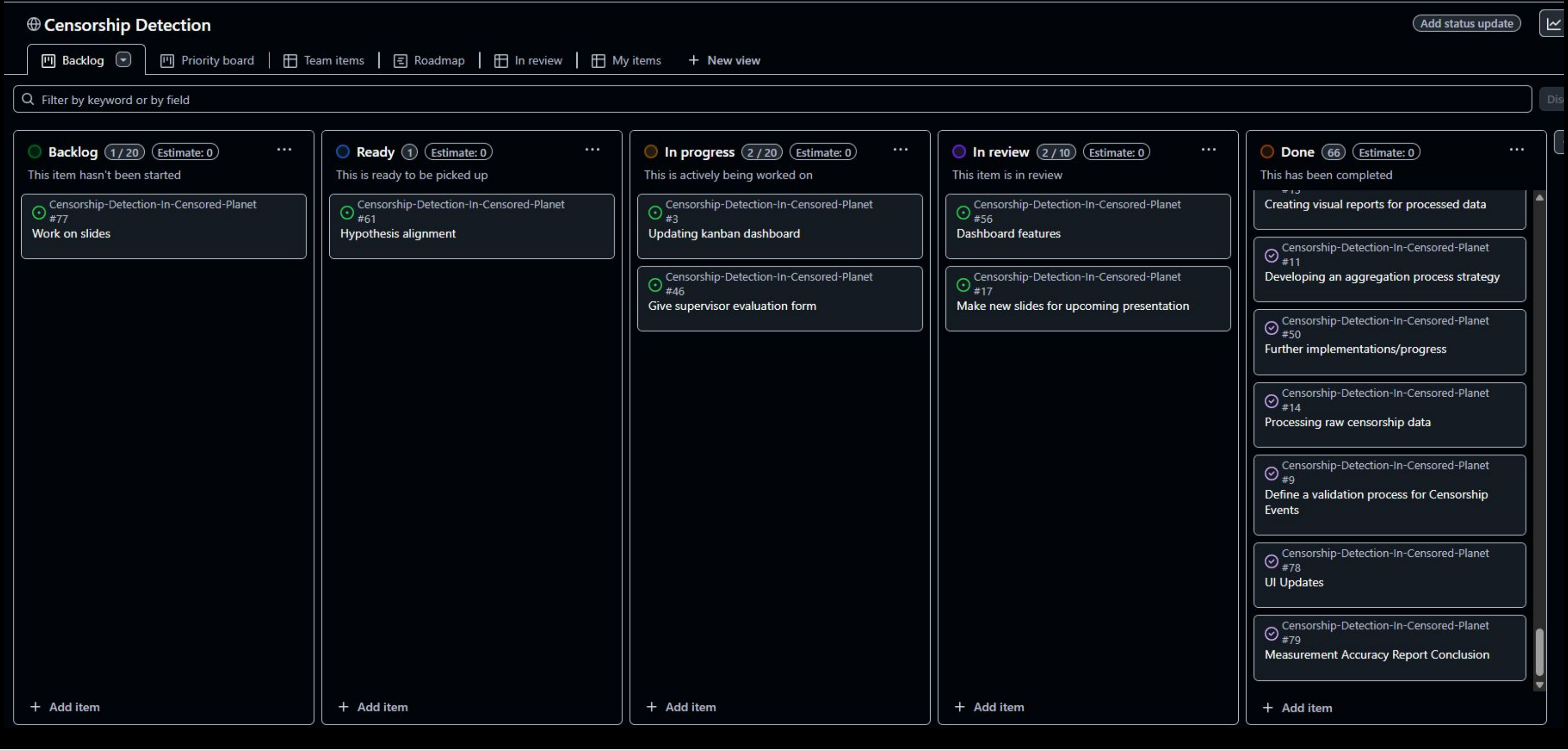
Censorship-Detection-In-Censored-Planet Supervised Research Project

This is a supervised research project

This project focuses on how censorship measurement data can be processed and aggregated to improve the

Project Management

https://github.com/users/john0x44/projects/1/views/1



Censorship Detection

Backlog 1 / 20 Estimate: 0
This item hasn't been started

- Censorship-Detection-In-Censored-Planet #77
  Work on slides

Ready 1 Estimate: 0
This is ready to be picked up

- Censorship-Detection-In-Censored-Planet #61
  Hypothesis alignment

In progress 2 / 20 Estimate: 0
This is actively being worked on

- Censorship-Detection-In-Censored-Planet #3
  Updating kanban dashboard
- Censorship-Detection-In-Censored-Planet #46
  Give supervisor evaluation form

In review 2 / 10 Estimate: 0
This item is in review

- Censorship-Detection-In-Censored-Planet #56
  Dashboard features
- Censorship-Detection-In-Censored-Planet #17
  Make new slides for upcoming presentation

Done 66 Estimate: 0
This has been completed

- Creating visual reports for processed data
- Censorship-Detection-In-Censored-Planet #11
  Developing an aggregation process strategy
- Censorship-Detection-In-Censored-Planet #50
  Further implementations/progress
- Censorship-Detection-In-Censored-Planet #14
  Processing raw censorship data
- Censorship-Detection-In-Censored-Planet #9
  Define a validation process for Censorship Events
- Censorship-Detection-In-Censored-Planet #78
  UI Updates
- Censorship-Detection-In-Censored-Planet #79
  Measurement Accuracy Report Conclusion

Project Details

Github : https://github.com/john0x44/Censorship-Detection-In-Censored-Planet

Project Management Board: https://github.com/users/john0x44/projects/1

Timelog(s) :
https://github.com/john0x44/Censorship-Detection-In-Censored-Planet/tree/main/timelogs

Latest Timelog :
https://github.com/john0x44/Censorship-Detection-In-Censored-Planet/blob/main/timelogs/4900_time_log_week_15.pdf