Graph Literacy Analysis Using Eye Tracking Documentation

California State Polytechnic University, Pomona - hapii Lab

Last Modified: Dec 17, 2025

Research Advisor: Dr. Ben Steichen, bsteichen@cpp.edu

Researchers: Eli Tolentino elitolentino2004@gmail.com, Tony Gonzalez tony.gonzalez9868@gmail.com

This document provides a comprehensive summary of the work completed for the Graph Literacy Analysis Using Eye Tracking Project, conducted in collaboration with CPP's hapii Lab.

See the code here. All data mentioned can be found in the hapii Lab Teams in the Eye Tracking Analysis Channel under Files.

Table of Contents

Fall 2024 - Eli Tolentino

Data Integration

This study was conducted on two separate PCs, each running a database containing responses and gaze data from different users. These two databases were merged into a single dataset.

BEACH-Gaze Data Files

For each user's study session (start to finish), Beach-GAZE generated two CSV files:

1. `user_x_all_gaze.csv`
    - Contains raw eye-gaze data recorded at 120 FPS (one row per frame).
2. `user_x_fixations.csv`
    - Contains processed eye-gaze data (one row per fixation event).
    - These were used as input when splitting data into individual questions, since they are more concise.

MySQL Database Structure

This study used four MySQL tables to store question data and user responses.

1. **`master_table`**
    - Tracks all user interactions during the study.
    - Columns:
        - `user_id`
        - `button_name`
            - Possible values:
                - Begin Study
                - Claim User ID
                - Begin Prestudy
                - Submit
                - Start Calibration
                - Begin Main Study
        - `question_id`

    - `question_text`
    - `user_answer`
    - `timestamp`
  - Each user's sequence of 50 question timestamps begins after **"Begin Main Study"** is pressed. These timestamps were used to segment the CSV gaze recordings.

2. **`prestudy_questions`**
  - Stores user performance before the main study.
  - Columns:
    - `user_id`
    - `question_text`
    - `user_answer`
    - `timestamp`
  - A user's literacy label (literate or illiterate) is determined by the number of pre-study questions they answer correctly.

3. **`test_questions`**
  - Metadata for each test question.
  - Columns:
    - `question_id`
    - `question_text`
    - `chart_image`
    - `options`
    - `correct_answer`
  - Used to determine the type of question (bar, line, pie) associated with each timestamp.

4. **`test_responses`**
  - Detailed records of each user's answers.
  - Columns:
    - `user_id`
    - `question_id`
    - `question`
    - `user_answer`

- `is_correct`
- `timestamp`

Data Processing - Original Study

Each user's original spreadsheet (raw gaze session) was split into 50 per-question spreadsheets (e.g., `user_1_question_1.csv`). This was done by:

1. Referencing the question timestamps in the MySQL `master_table`.
2. Cutting the fixation CSV into segments using the elapsed time between each question.

NOTE: Due to a bug in the survey interface during the study, User 11 only received 49 questions.

Question Mapping

A separate mapping spreadsheet was created to list the exact sequence of question IDs answered by each user. This can be used for future analysis, such as splitting questions by chart type (bar, line, pie) or analyzing overall performance per question.

You can find this user-to-question mapping in the Teams Channel under Code -> Utilities -> YennhiThoaQuestionMapping.csv

Spring 2025 - Tony Gonzalez

We started by exploring the data produced from Fall 2024. We first noticed that user 5 and user 20 had abnormal data compared to the rest of the users. Upon review of the video footage, it was clear that the eye track was malfunctioning during their study and rendered the data unusable. **Therefore, users 5 and 20 are excluded from all further model training pipelines.**

BEACH-Gaze

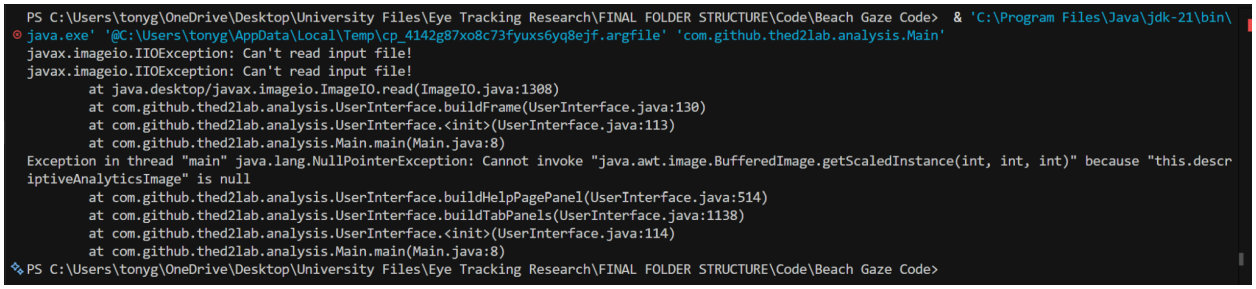
BEACH-Gaze is an open-source software toolkit designed to analyze human eye-tracking data, supporting both **descriptive analytics and predictive analytics**. It is compatible with

Gazepoint GP3/GP3HD eye trackers and processes raw gaze recordings to produce a wide range of scientifically grounded gaze-behavior metrics. The system performs data cleaning, fixation extraction, AOI (Area of Interest) analysis, entropy calculations, saccade and fixation profiling, blink analysis, and optional window-based temporal slicing. It can also train machine-learning models through WEKA to classify or predict outcomes based on gaze patterns

Descriptive Gaze Measures (DGMs) are the core outputs of BEACH-Gaze's descriptive analytics module. They are numeric summaries that characterize how **participants look at visual stimuli**, derived from cleaned fixation, saccade, pupil, blink, and AOI-related data. DGMs include measures such as fixation counts and durations, saccade lengths and velocities, gaze entropy, blink rates, convex hull area, pupil dilation statistics, transition probabilities between AOIs, and more. These metrics provide a quantitative description of visual attention, information processing strategies, and viewer behavior during a task

We suggest that you use the Beach Gaze code that is in the Teams and GitHub repo rather than pulling from the original Beach Gaze repo. If the latter is done, be sure to adjust the code as we did (below), so that you produce the same DGM data as we did.

If you want to run BEACH-Gaze, open the "BEACH-Gaze" folder specifically. The UserInterface.java module has strange behavior if the project is opened in any other way. If you get the error below, you did not open the correct folder.



Important note for Beach Gaze: We noticed that Beach Gaze was classifying a large amount of fixations as invalid. Upon review of the code, we realized Beach Gaze's Data Filter parameters were too strict with the data we had. We adjusted the "*dataFilter.java*" file for lines 67, 69, and 78-85.

Line 67: Changed MIN_DIAMETER from 2 to 1

Line 69: Changed MAX_PUPIL_DIFF from 1 to 3

Lines 78-85: Commented out this section to ignore the Pupil Validity Flag from Gazepoint.

Important note about the DGM files: We immediately noticed that the "mean_peak_saccade_velocity", "median_peak_saccade_velocity", "std_peak_saccade_velocity", "min_peak_saccade_velocity", "max_peak_saccade_velocity", "mean_mean_saccade_velocity", "median_mean_saccade_velocity", "std_mean_saccade_velocity", "min_mean_saccade_velocity", "max_mean_saccade_velocity", "average_blink_rate_per_minute", "stdev_of_relative_degrees" columns for all DGM files were NaN. **The exact reason for this remains unknown, however we decided to manually drop these 17 columns from all model training pipelines, resulting in 50 total features.**

Learn more about BEACH-Gaze here:
BEACH Gaze User Manual | **BEACH Gaze Code Documentation**

Defining the Classes

The ultimate goal of our models was to develop a binary classifier to identify if a user had High or Low graph literacy. Each user completed 4 pre-study questions. If the user got 2 or more questions correct, they were assigned the label of “1” for High Literacy (Literate). And if 1 or less was correct, they were labeled as Low Literacy (Illiterate).

The mapping for the users’ class can be found in the “*user_literacy_results*” CSV file in Code -> Utilities. Where “MEDIA_ID” is the user id and “LITERACY” is their class. This file serves as the Source of Truth for all further models.

We then investigated if defining the classes using a **median split based on the 50-question study** would yield better results. We determined that 34 correct questions was the proper threshold to evenly split the users. Those with 34 or more correct were grouped in the high literacy (literate) group and those with 33 or fewer were grouped in the low literacy (illiterate) group. After running the models and comparing the results, we saw that there was **no significant difference between using the 4 prestudy questions and the full-study median split**. See the LOUO Cross Validation - Full Study section to see the results. **The mapping for the users’ median split class can be found in the “*user_literacy_results_full_study*” CSV file in Code -> Utilities**.

Logistic Regression

Expected Folder Structure

The logistic regression and random forest models expect the following folder structure at the project root:

```
Project Root/
├── Chopped DGMs Unorganized/
│   ├── user_1/
│   │   ├── user_1_question_1_DGMs.csv
│   │   ├── user_1_question_2_DGMs.csv
│   │   ├── ...
│   │   └── user_1_question_50_DGMs.csv
│   ├── user_2/
│   │   ├── user_2_question_1_DGMs.csv
│   │   ├── ...
│   │   └── user_2_question_50_DGMs.csv
│   └── ...
│
├── Full DGMs Unorganized/
│   ├── user_1/
│   │   └── User 1_fixations_DGMs.csv
│   ├── user_2/
│   │   └── User 2_fixations_DGMs.csv
│   └── ...
│
└── Code/
    ├── Logistic Regression/
    │   ├── log_reg_beach_gaze_chopped.ipynb
    │   └── log_reg_beach_gaze_full.ipynb
    ├── Random Forest/
    │   ├── RF_beach_gaze_chopped.ipynb
    │   └── RF_beach_gaze_full.ipynb
    └── Utilities/
        └── users_literacy_results.csv
```

Important Caveat:

The data used for the following models were “contaminated.” Meaning, users’ data contributed to both training and testing groups when 10-fold cross validation was done. The results that follow should be taken with a grain of salt. See the Leave-One-User-Out Cross Validation section for more accurate results that removed the contamination issue.

log_reg_beach_gaze_full.ipynb

The first model we trained with the DGMs was a simple Logistic Regression Model using Scikit-Learn with the Beach Gaze-produced DGMs for the users' entire study. That is, one DGM vector per user that summarized their eye movement activity from the full duration of the study. This model did not perform very well, it had a tendency to completely bias one class or the other. It also suffered from a lack of data instances. Since we used the users' entire study, we only had 30 total instances for training and testing.

Results (10-Fold Contaminated Cross Validation):

```
Cross-validation scores: [0.66666667 0.66666667 0.66666667 0.33333333 0.66666667 0.66666667
 1.         0.         1.         0.66666667]
Mean accuracy: 0.6333
Standard deviation: 0.2769
```

log_reg_beach_gaze_chopped.ipynb

Because of the lack of training instances from using the full study DGMs, we opted to use the "chopped" data instead, which separated the eye tracking data per question. This resulted in 1499 instances. All users have 50 questions, with the exception of user 11, who is missing question 50 because of a software glitch, resulting in 1499 rather than 1500.

Results (10-Fold Contaminated Cross Validation):

```
Cross-validation scores: [0.71333333 0.8        0.59333333 0.34666667 0.48666667 0.64666667
 0.22666667 0.50666667 0.7        0.71812081]
Mean accuracy: 0.5738
Standard deviation: 0.1724
```

Please see the *log_reg_DGM_LOUO.ipynb* section for the Leave-One-User-Out Cross Validation model results.

Random Forest

RF_beach_gaze_full.ipynb

The next algorithm we tried was the Random Forest using Scikit-Learn. Like Logistic Regression, first tried using the DGMs from the users' entire study. And like Logistic Regression,

this approach suffered from a lack of training instances and performed poorly as a result, tending to always bias and predict one class.

Results (10-Fold Contaminated Cross Validation):

```
Cross-validation scores: [0.66666667 1.         0.33333333 1.         0.66666667 0.66666667
 0.66666667 0.         1.         1.        ]
Mean accuracy: 0.7000
Standard deviation: 0.3145
```

RF_beach_gaze_chopped.ipynb

Like Logistic Regression, we trained another Random Forest Classifier using the users' chopped questions. We saw an improvement in model performance due to the increase in training data.

We also performed a hyperparameter search and determined the best parameters according to accuracy: 'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': 10, 'bootstrap': False

Results (10-Fold Contaminated Cross Validation):

```
Cross-validation scores: [0.77333333 0.88666667 0.8        0.51333333 0.58666667 0.66
 0.35333333 0.62666667 0.78       0.70469799]
Mean accuracy: 0.6685
Standard deviation: 0.1488
```

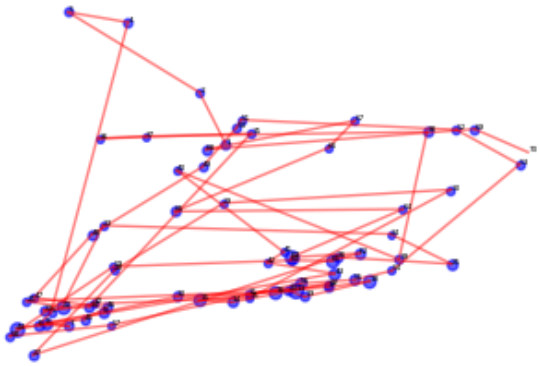
Please see the *RF_DGM_LOUO.ipynb* section for the Leave-One-User-Out Cross Validation model results.

Convolutional Neural Network (CNN)

Scanpaths

We then turned our focus to deep learning models. Our first attempt was converting the raw eye tracking data (X and Y gaze coordinates) into a visual representation (Scanpaths). Using the *generate_scanpath.py* script, we produced the following images for each question.

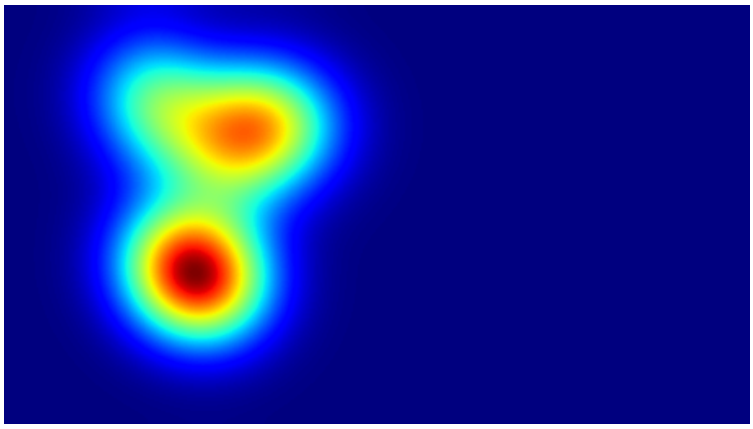
Example Scanpath 512 x 288 pixels (user_2_question_1_scanpath):



Heatmaps

Another way you can convert the raw eye tracking data into a visual representation is through a heatmap. We compared the two representations and decided to stick with scanpaths, but heatmaps may be used in future work.

Example Heatmap 512 x 288 pixels (user_13_question_10_heatmap):



CNN_Tony.ipynb

The first CNN model we tried was a model with an arbitrarily chosen architecture of:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 286, 510, 32)      896

 max_pooling2d (MaxPooling2D  (None, 143, 255, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 141, 253, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 70, 126, 64)      0
 2D)

 conv2d_2 (Conv2D)           (None, 68, 124, 128)      73856

 max_pooling2d_2 (MaxPooling  (None, 34, 62, 128)      0
 2D)

 flatten (Flatten)           (None, 269824)            0

 dense (Dense)               (None, 128)               34537600

 dropout (Dropout)           (None, 128)               0

...
Total params: 34,630,977
Trainable params: 34,630,977
Non-trainable params: 0
```

The model was fed all the scanpath images at their original size of 512 x 288 and in full color.

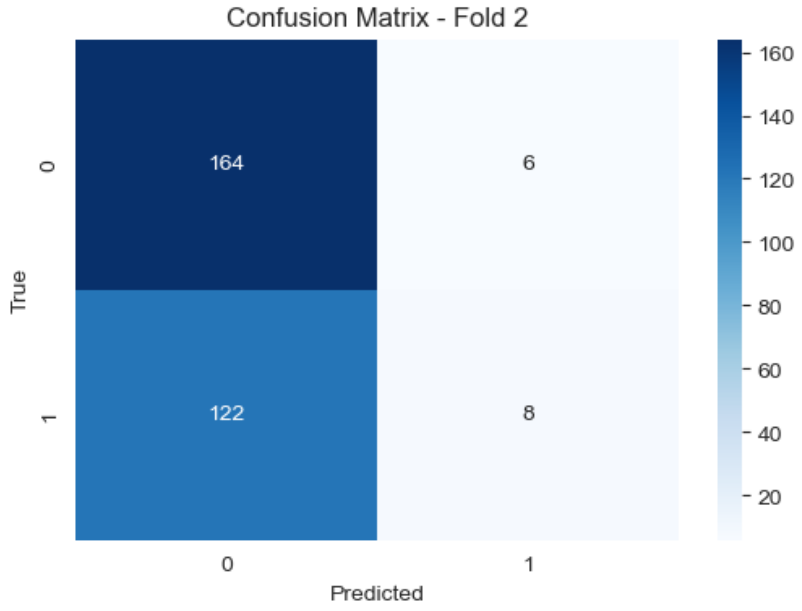
CV_C_CNN.ipynb

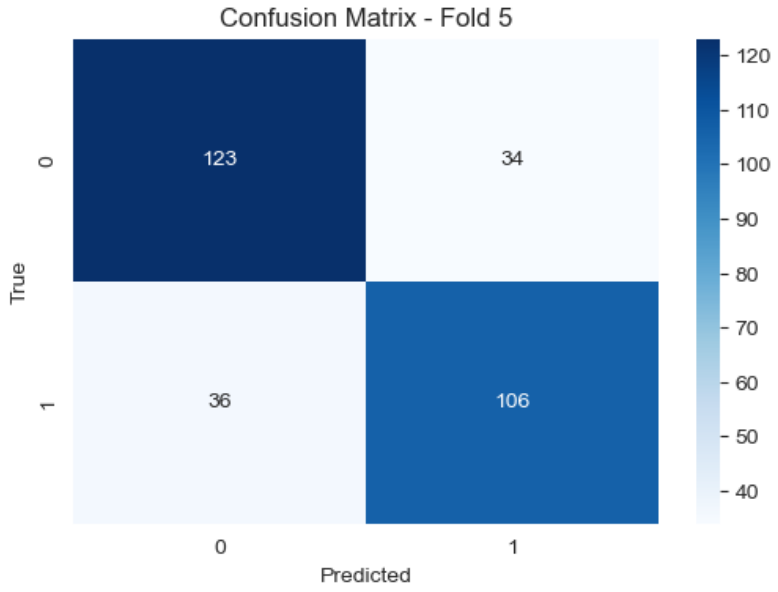
We then performed a 5-fold cross validation on this model architecture above. The second “C” in the title refers to “contaminated,” that is, users’ data overlapped between training and testing sets, which we speculated will alter the results.

Results:
Cross-validation results:
Average Accuracy: 0.6572 ± 0.0794
Average Loss: 0.6788 ± 0.0691

Important: One interesting behavior became apparent when training the CNN. On some folds, the model would find distinction between the two classes and on others, it would completely bias one class. This behavior continued throughout all future models (CNN, VTNet, Transformers, etc)

Example confusion matrices from the 5-fold CV:





CV_NC_CNN.ipynb

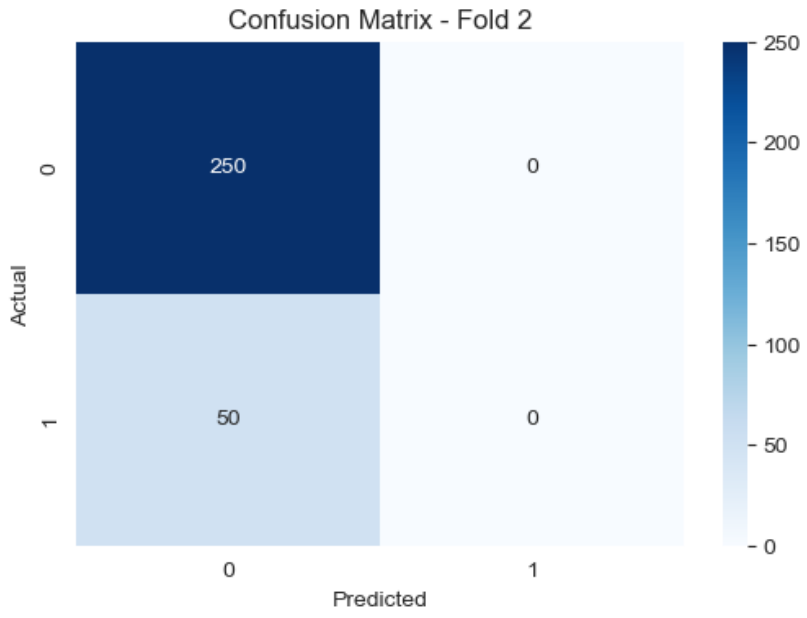
We then performed the same 5-fold cross validation, but ensured that users that contributed to the testing group did not contribute to the testing group using Scikit-Learn's Grouped K-Folds class.

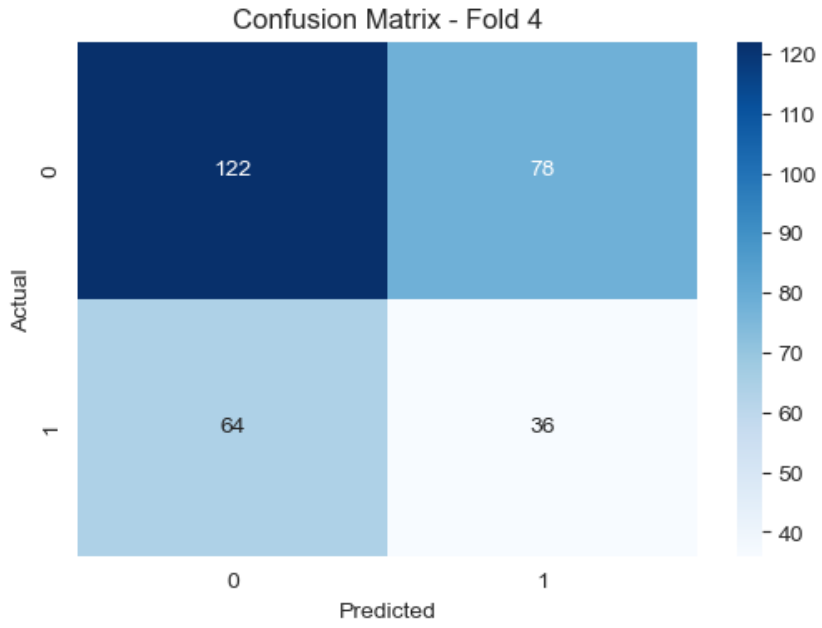
Like the previous CV run, we saw that the model would sometimes completely bias one class over the other.

Results:
Cross-validation results:
Average Accuracy: 0.6671 ± 0.0621
Average Loss: 0.6251 ± 0.0543

While the results seem comparable to the contaminated CV, this could be caused by the model getting lucky and entirely predicting the one class that contained the majority of the training data, as seen in Fold 2 below. As we'll see in the Leave-One-User-Out CV, the models performed far worse when contamination was eliminated.

Example confusion matrices from the 5-fold CV:





Convolutional Neural Network - Conati Architecture

We then looked to other previous deep learning work that also used scanpaths to achieve a similar binary classification goal. We took inspiration from Dr. Cristina Conati from her paper "A Neural Architecture for Detecting User Confusion in Eye-tracking Data." The CNN architecture used in this paper is as follows:

Input Layer

- Single-channel grayscale scanpath image (color is irrelevant for gaze paths).
- Images are **downsampled by a factor of 6** to reduce resolution and avoid unnecessary parameters.

Convolutional Layers

- **Two convolutional layers only** (kept shallow to avoid overfitting on the small dataset):
  - **Layer 1:** 16 channels, kernel size **5×5** (larger than the common 3×3 to increase receptive field directly).
  - **Layer 2:** 6 channels, kernel size **5×5**.
- The choice of just two layers reflects the simplicity of scanpath images (dots and lines, not complex textures).

Hidden Layer

- A **50-unit fully connected hidden layer** connects the convolutional outputs to the classifier.
- This acts as a bridge between extracted visual features and the final decision.

Output Layer

- Two units (confused vs. not confused, **high vs low literacy for our use-case**).
- Softmax activation for classification.

Graph-type Grouped CNN

We also decided to train separate models for each graph type (bar, line, pie). This separation was necessary because eye movement patterns vary significantly between graph types, allowing each model to focus on literacy-related distinctions within a consistent task context rather than across different visualization types.

We developed 4 different CNN models that took inspiration from Conati's architecture:

1. **CNN with Conati's architecture (using a 5x5 convolutional kernel) trained on scanpath images**
    a. Filename: "conati_grouped_CNN_scanpaths_5x5.ipynb"
2. **CNN with Conati's architecture (using a 5x5 convolutional kernel) trained on heatmap images**
    a. Filename: "conati_grouped_CNN_heatmaps_5x5.ipynb"
3. **CNN with a modified Conati architecture (using a 3x3 convolutional kernel) trained on scanpath images**
    a. Filename: "conati_grouped_CNN_scanpaths_3x3.ipynb"
4. **CNN with a modified Conati architecture (using a 3x3 convolutional kernel) trained on heatmap images**
    a. Filename: "conati_grouped_CNN_heatmaps_3x3.ipynb"

We also developed the same 4 CNN models using the arbitrary chosen CNN architecture, see here.

5. **CNN with our architecture (using a 5x5 convolutional kernel) trained on scanpath images**
    a. Filename: "grouped_CNN_scanpaths_5x5.ipynb"
6. **CNN with our architecture (using a 5x5 convolutional kernel) trained on heatmap images**
    a. Filename: "grouped_CNN_heatmaps_5x5.ipynb"
7. **CNN with our architecture (using a 3x3 convolutional kernel) trained on scanpath images**

   a. Filename: “grouped_CNN_scanpaths_3x3.ipynb”
8. **CNN with our architecture (using a 3x3 convolutional kernel) trained on heatmap images**
   a. Filename: “grouped_CNN_heatmaps_3x3.ipynb”

We then trained and compared all 8 variations using 5-fold cross validation with contamination, on each graph type. The results are as follows:

Best-Performing Architectures (per Graph Type)

Graph Type	Best Train Model	Train Acc	Best Validation Model	Val Acc
Bar	Conati’s Arch. (3x3 kernel)	0.9135	Conati’s Arch. (3x3 kernel)	0.6116
Line	Conati’s Arch. (5x5 kernel)	0.8263	My Arch. (3x3 kernel)	0.6561
Pie	Conati’s Arch. (3x3 kernel)	0.9083	Conati’s Arch. (3x3 kernel)	0.6533

Initially we wanted to continue with the Conati Architecture with the 3x3 kernel with Scanpaths as it had the best balance of performance and training speed. **But instead we decided to keep the 5x5 kernel with the scanpaths to exactly replicate Conati’s architecture.**

See the full comparison analysis here or in the Team’s channel titled “Spring 2025-CNN Training Report”

This is where the work from Spring 2025 concluded. Our next goal was to implement a sequential layer alongside the CNN to also be able to process the eye tracking data.

Fall 2025 - Eli Tolentino and Tony Gonzalez

VTNet Hybrid Architecture

We again drew inspiration from Dr. Conati’s work from “A Neural Architecture for Detecting User Confusion in Eye-tracking Data” and “Classification of Alzheimer’s Disease with Deep Learning on Eye-tracking Data.” Here they introduce a hybrid spatial-temporal hybrid architecture they named VTNet.

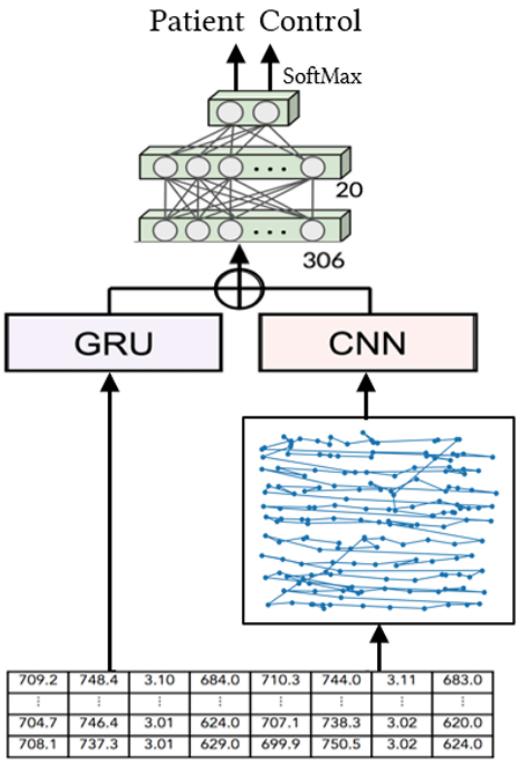


Figure 4: The VTNet architecture.

VTNet is a deep learning architecture designed to classify user states (like confusion) and health conditions (like Alzheimer's Disease) from **eye-tracking (ET) data**.
It combines **two perspectives of ET data**:

- **Temporal (T):** raw sequential gaze samples processed by a **GRU** (a type of recurrent neural network).
- **Visual (V):** scanpath images (showing fixations and transitions) processed by a **CNN**.

It works as follows:

GRU branch

- Takes raw ET sequences (gaze coordinates, pupil size, head distance).
- Learns temporal patterns over thousands of timesteps.

CNN branch

- Takes scanpath images (visual representation of gaze paths).
- Learns spatial patterns of fixations and saccades.

Fusion layer

- Outputs from GRU (256 units) and CNN (50 units) are concatenated into a single vector.

- Passed through a small fully connected network with a Softmax output (AD vs. control).

The original model code was written using PyTorch and can be found here. It can also be found in the GitHub Repo and Teams under Code -> VTNet -> Conati's Model

vtnet_replication.ipynb

We used Conati's VTNet code "*vtnet_experiments.ipynb*" and translated it from PyTorch to TensorFlow. We tried our best to exactly replicate every model detail and data pipeline.

Data Preprocessing

The VTNet model requires preprocessing of two types of input data: **scanpath images and time-series gaze coordinate data.**

IMAGE PREPROCESSING:

- Images are loaded from directories organized by graph type (bar, line, pie) and class (illiterate, literate)
- Images are resized to 150x150 pixels to match the model's input requirements
- RGB images are converted to grayscale, resulting in single-channel images
- Pixel values are normalized to the range [0, 1] by dividing by 255.0
- Final image shape: (150, 150, 1)

TIME-SERIES DATA PREPROCESSING:

- Raw gaze coordinate data is loaded from CSV files containing FPOGX and FPOGY columns
- Invalid or NaN values are removed from the coordinate sequences
- Sequences longer than 150 timesteps are truncated by taking the last 150 timesteps
- Sequences shorter than 150 timesteps are zero-padded at the beginning (pre-padding)
- Final time-series shape: (150, 2) representing 150 timesteps with x and y coordinates

DATA SPLITTING:

- The dataset is split into training and validation sets using an 80/20 split
- Stratified splitting is used to maintain class distribution in both sets
- A random seed of 42 is used for reproducibility

LABEL ENCODING:

- Class labels (illiterate, literate) are converted to one-hot encoded vectors
- Illiterate is encoded as [1, 0] and literate as [0, 1]
- Final label shape: (2,)

DATA LOADING:

- Image paths are matched with corresponding CSV files based on filename patterns
- The filename matching removes the '_scanpath' suffix from image filenames to find corresponding CSV files

- CSV files are located in separate directories (illiterate/literate) within the raw data path
- Both image and time-series data are loaded together for each sample during training

BATCH PROCESSING:

- Data is organized into batches of size 32 for efficient training
- TensorFlow's data pipeline is used with prefetching and parallel processing for optimal performance

Expected Folder Structure

The VTNet replication notebook automatically searches for data directories by traversing up the directory tree from the notebook's location. The notebook looks for either "**Scanpaths**" or "**contaminated datasets**" directories within 6 levels of parent directories.

REQUIRED DIRECTORY STRUCTURE:

1. SCANPATH IMAGES DIRECTORY:

The scanpath images must be organized in the following structure:

```
Scanpaths/
└── organized_by_graph_type_contaminated/
    ├── bar/
    │   ├── illiterate/
    │   │   ├── user_X_question_Y_scanpath.png
    │   │   ├── user_X_question_Z_scanpath.png
    │   │   └── ... (more PNG files)
    │   └── literate/
    │       ├── user_X_question_Y_scanpath.png
    │       ├── user_X_question_Z_scanpath.png
    │       └── ... (more PNG files)
    ├── line/
    │   ├── illiterate/
    │   │   └── ... (PNG files)
    │   └── literate/
    │       └── ... (PNG files)
    └── pie/
        ├── illiterate/
        │   └── ... (PNG files)
        └── literate/
            └── ... (PNG files)
```

Key points:

- Graph types: bar, line, pie (required subdirectories)
- Classes: illiterate, literate (required subdirectories under each graph type)
- File naming: user_X_question_Y_scanpath.png format

- File format: PNG images

2. RAW EYE TRACKING DATA DIRECTORY:

The CSV files containing gaze coordinates must be organized as follows:

```
contaminated datasets/
└── Raw Eye Tracking Data/
    ├── illiterate/
    │   ├── user_X_question_Y.csv
    │   ├── user_X_question_Z.csv
    │   └── ... (more CSV files)
    └── literate/
        ├── user_X_question_Y.csv
        ├── user_X_question_Z.csv
        └── ... (more CSV files)
```

Key points:

- Classes: illiterate, literate (required subdirectories)
- File naming: user_X_question_Y.csv format (matches scanpath names but without _scanpath suffix)
- File format: CSV files with FPOGX and FPOGY columns
- Note: CSV files are NOT organized by graph type, only by class

CSV FILE REQUIREMENTS:

Each CSV file must contain at minimum the following columns:

- FPOGX: X-coordinate of gaze fixation point (normalized 0-1)
- FPOGY: Y-coordinate of gaze fixation point (normalized 0-1)

IMAGE FILE REQUIREMENTS:

Scanpath images should be:

- PNG format
- Any size (will be resized to 150x150 pixels)
- RGB or grayscale (will be converted to grayscale)
- Pixel values will be normalized to [0, 1] range

Results

This model showed very promising results on some training runs, but we also observed the same behavior as seen in the previous CNN models. Precisely, on some training runs, the model would find a very clear distinction between the two classes, and on other runs, it would completely bias one class. *It should also be noted that this model suffered from contamination as training instances from some users contributed to both training and testing sets.*

See the Leave-One-User-Out Cross Validation results here.

vtnet_replication_6col.ipynb

We were then curious to see if adding additional eye tracking data would improve the results seen from the previous VTNet version. Instead of just using the X and Y coordinates (FPOGX and FPOGY), we also included:

- **LPS, RPS:** Distance of the left and right pupils from the eye tracker from calibration time, as a scale factor
- **LPMM, RPMM:** Pupil diameter of the left and right pupil in millimeters

Learn more about the features BEACH-Gaze produces here.

This file also follows the same data preprocessing and expected folder structure from the *vtnet_replication.ipynb* file. This version also saw the same performance behavior as the previous version. See the Leave-One-User-Out Cross Validation results here.

Tumbling Window DGMs

The tumbling window DGM approach (generated using BEACH-Gaze's Tumbling Window Feature) provides a temporal segmentation of eye-tracking data by dividing each question session into **non-overlapping 3-second windows**. This differs from the standard DGM files which aggregate all eye-tracking data from an entire question into a single summary.

Key Differences:

1. Temporal Segmentation:
   - **Standard DGM:** One row per question, summarizing the entire session
   - **Tumbling Window DGM:** Multiple rows per question, one for each 3-second window

2. File Structure:
   - Standard DGM files contain 61 feature columns covering fixation, saccade, scanpath, and pupil metrics
   - Tumbling Window DGM files contain the same 61 feature columns plus 5 additional temporal metadata columns:
     - **beginning_timestamp:** Start timestamp of the window
     - **ending_timestamp:** End timestamp of the window
     - **window_duration:** Duration of the window in seconds
     - **initial_seconds_elapsed_since_start:** Seconds elapsed from question start to window start
     - **final_seconds_elapsed_since_start:** Seconds elapsed from question start to window end

3. Data Characteristics:
   - Windows are non-overlapping and sequential (e.g., 0-3s, 3-6s, 6-9s)

- Each window contains DGM features calculated only from eye-tracking data within that time segment
- The number of windows varies per question based on total question duration
- Feature values in each window reflect the eye movement patterns during that specific 3-second period

Example:
A question lasting 25.9 seconds would produce approximately 8-9 tumbling window rows (each covering 3 seconds), compared to a single row in the standard DGM file. Each window row contains the same feature types (fixation counts, saccade metrics, etc.) but calculated only from the eye-tracking data within that specific 3-second segment.

The tumbling window datasets can be found in the Teams Channel under non-contaminated datasets -> Tumbling Window DGMs (3s)/Organized Normalized Tumbling Window DGMs (3s)

vtnet_replication_DGM_3s.ipynb

This notebook implements the same VTNet model as vtnet_replication.ipynb, but instead of using the raw eye tracking data (X and Ys) it uses the tumbling window DGMs as input to the sequential portion of the model. However, the same scanpaths are used as input to the CNN portion of the model.

The motivation behind this was to compare the performance between the raw eye tracking data and the featured engineered DGMs for our use case.

See the Leave-One-User-Out Cross Validation results here.

BERT Transformer Architecture

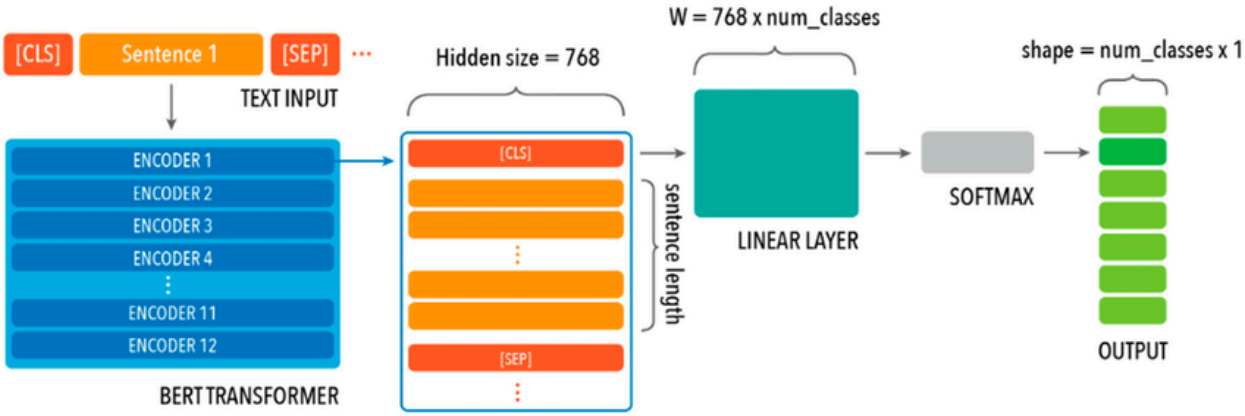
BERT (Bidirectional Encoder Representations from Transformers) is a deep learning model that encodes input sequences bidirectionally, allowing each element in a sequence to attend to both preceding and subsequent elements. We selected BERT for this work because our eye-tracking data can be represented as a sequential string of numerical values, and we were interested in evaluating whether transformer-based self-attention mechanisms could capture differences in gaze patterns across users. Specifically, the model is trained to predict **literate versus illiterate reading ability** from numerical data.

Model Configuration

We use the *bert-base-uncased* variant, an encoder-only architecture with the following specifications:

- **Model Variant:** bert-base-uncased (encoder-only BERT architecture; text is lowercased during tokenization)

- **Transformer layers:** 12 encoder (Transformer) blocks
- **Hidden size:** 768 (embedding and hidden state dimensionality)
- **Attention heads:** 12 (each head has dimension 64)
- **FFN / intermediate size:** 3072 (feed-forward inner layer; 4× hidden size)
- **Max positions:** 512 (maximum sequence length supported by positional embeddings)



For classification, the final hidden representation of the `[CLS]` token is used as an aggregate sequence embedding and passed to a classification head to predict the binary literacy label. As the input consists solely of numerical values and special tokens, the uncased tokenizer does not affect semantic content.

Data Preprocessing

Each input file is consolidated into a single string representation. Tabular eye-tracking data are converted into a linearized text format using the following preprocessing rules:

- Column values within each row are comma-separated
  - Example (6-col): `x,y,lps,rps,lpmm,rpmm`
- All numerical values are rounded to the nearest 0.01.
- Each sequence begins with the special token `[CLS]`.
- Rows are separated using a custom special token `[ROW]`.
  - This token was explicitly added to the tokenizer vocabulary and is used to distinguish individual fixation points, as each row in the raw data corresponds to a single fixation.
- Each sequence ends with the special token `[SEP]`.
- Sequences longer than 512 tokens are truncated, and shorter sequences are padded to the maximum sequence length.

Training Specifications

The model was fine-tuned for binary classification using the following hyperparameters:

- **Optimizer:** AdamW
- **Learning rate:** 2e-5
- **Batch size:** 4
- **Number of epochs:** 5 (no early stopping)
- **Loss function:** binary cross-entropy

The final `[CLS]` hidden representation serves as input to the classification head. All experiments were implemented using the Hugging Face Transformers library with PyTorch.

Limitations and Future Work

While BERT provides a strong baseline for modeling sequential eye-tracking data, its fixed maximum sequence length of 512 tokens limits the amount of fixation information that can be encoded within a single input. Longer gaze sequences must therefore be truncated, potentially discarding temporally distant but behaviorally relevant fixation patterns. Additionally, BERT's WordPiece tokenizer was originally designed for natural language and may not optimally represent continuous numerical values.

Future work could explore:

- Transformer architectures with extended context windows, such as **ModernBERT**, to allow longer sequences and capture more comprehensive fixation patterns.
- Preprocessing enhancements, including normalization, standardization, or missing-value handling, to improve input representation for numerical data.
- Alternative architectures tailored to numerical or time-series data, including hybrid models that combine transformer attention with recurrent layers.

Leave One User Out (LOUO) Cross Validation

Once we finished developing all the different models we were interested in (Logistic Regression, Random Forest, VTNet, and BERT) we wanted to rigorously test and compare all variations without any data contamination. We did this by performing a Leave-One-User-Out Cross Validation (LOUO CV), ensuring no data from the testing user contributed to the training data.

We performed LOUO CV, training separate models for each graph type (bar, line, pie). This separation was necessary because eye movement patterns vary significantly between graph types, allowing each model to focus on literacy-related distinctions within a consistent task context rather than across different visualization types.

Pupil Normalization

Additionally, to account for individual differences in baseline pupil size and to better capture moment-to-moment cognitive or emotional changes, we implemented a pupil normalization process based on each participant's initial gaze behavior.

For each participant, **we used the first 10 seconds of data from Question 1 to establish a personalized baseline.** During this window, we calculated the average pupil diameter for the left eye (LPMM), right eye (RPMM), and the combined value for both eyes. These baseline metrics were then used to compute change-from-baseline (delta) values for all subsequent DGMs and raw gaze files.

For each DGM file, we added three new columns representing normalized pupil change:

- delta_average_pupil_size_of_left_eye
- delta_average_pupil_size_of_right_eye
- delta_average_pupil_size_of_both_eyes

Each value reflects the difference between the current DGM's pupil metric and the baseline metric.

At the raw gaze level, we added three analogous columns that quantify pupil deviation on a per-sample basis:

- delta_LPMM
- delta_RPMM
- delta_BPMM (the average of left and right deviations)

This normalization procedure ensures that all pupil-related features reflect relative changes within participants rather than absolute pupil size, reducing inter-individual variability and increasing the interpretability of pupil dynamics across tasks and participants.

The modified datasets can be found in the Teams Channel under non-contaminated datasets -> Organized Normalized {Dataset name}.

The new delta columns were added to the end of each CSV file as the names listed above.

We compared the performance of all our models with and without the normalized during the LOUO CV process. **Models trained on the modified datasets are suffixed with "_normalized."**

Pre-Study Defined Classes (PSC)

The following models used the classes determined by the 4 pre-study questions asked before the start of the 50-question study (PSC).

log_reg_DGM_LOUO.ipynb

This notebook implements the same Logistic Regression model, using the 1-DGM per question dataset, as log_reg_beach_gaze_chopped.ipynb, but now without contamination by performing LOUO CV.

Please see the *log_reg_DGM_LOUO.ipynb* section for the Leave-One-User-Out Cross Validation model results.

log_reg_DGM_LOUO_normalized.ipynb

This notebook implements the same LOUO Logistic Regression model as log_reg_DGM_LOUO.ipynb, but with the DGM dataset containing the normalized pupil data.

Please see the *log_reg_DGM_LOUO_normalized.ipynb* section for the Leave-One-User-Out Cross Validation model results.

RF_DGM_LOUO.ipynb

This notebook implements the same Random Forest model, using the 1-DGM per question dataset, as RF_beach_gaze_chopped.ipynb, but now without contamination by performing LOUO CV.

Please see the *RF_DGM_LOUO.ipynb* section for the Leave-One-User-Out Cross Validation model results.

RF_DGM_LOUO_normalized.ipynb

This notebook implements the same LOUO Random Forest model as RF_DGM_LOUO.ipynb, but with the DGM dataset containing the normalized pupil data.

Please see the *log_reg_DGM_LOUO_normalized.ipynb* section for the Leave-One-User-Out Cross Validation model results.

vtnet_LOUO_XY.ipynb

This notebook implements the same VTNet model, using the raw eye tracking data (X and Ys) and scanpaths, as vtnet_replication.ipynb, but now without contamination by performing LOUO CV.

Please see the *vtnet_LOUO_XY.ipynb* section for the Leave-One-User-Out Cross Validation model results.

vtnet_LOUO_XY_6col.ipynb

This notebook implements the same VTNet model, using the raw eye tracking data (X and Ys), head distance scale factors (LPS and RPS), pupil size measurements (LPMM and RPMM), and scanpaths, as vtnet_replication_6col.ipynb, but now without contamination by performing LOUO CV.

Please see the *vtnet_LOUO_XY_6col.ipynb* section for the Leave-One-User-Out Cross Validation model results.

vtnet_LOUO_XY_6col_normalized.ipynb

This notebook implements the same VTNet model as vtnet_LOUO_XY_6col.ipynb, but uses the normalized pupil measurements (delta_LPMM and delta_RPMM) instead of the original ones.

Please see the *vtnet_LOUO_XY_6col_normalized.ipynb* section for the Leave-One-User-Out Cross Validation model results.

vtnet_LOUO_XY_7col_normalized.ipynb

This notebook implements the same VTNet model as vtnet_LOUO_XY_6col_normalized.ipynb, but also included the normalized pupil measurement from both eyes (delta_BPMM).

Please see the *vtnet_LOUO_XY_7col_normalized.ipynb* section for the Leave-One-User-Out Cross Validation model results.

vtnet_LOUO_DGM_3s.ipynb

This notebook implements the same VTNet model, using the 3-second tumbling window DGMs and scanpaths, as vtnet_replication_DGM_3s.ipynb, but now without contamination by performing LOUO CV.

Please see the *vtnet_LOUO_DGM_3s.ipynb* section for the Leave-One-User-Out Cross Validation model results.

vtnet_LOUO_DGM_3s_normalized.ipynb

This notebook implements the same VTNet model as vtnet_LOUO_DGM_3s.ipynb, but uses the normalized pupil measurements (delta_LPMM, delta_RPMM, and delta_BPMM) instead of the original ones.

Please see the *vtnet_LOUO_DGM_3s_normalized.ipynb* section for the Leave-One-User-Out Cross Validation model results.

Expected Folder Structure

Expected folder structure for all Leave-One-User-Out (LOUO) evaluation notebooks. The notebooks are organized into two main categories: DGM-based traditional models and VTNet-based models.

PART 1: TRADITIONAL DGM LOUO NOTEBOOKS

The DGM LOUO evaluation code (logistic regression and random forest) expects a specific folder structure for the DGM data. The code automatically locates the project root by searching for the "Code" folder marker.

There are four DGM LOUO notebooks:

- log_reg_DGM_LOUO.ipynb (uses "DGMs" folder)
- log_reg_DGM_LOUO_normalized.ipynb (uses "Organized Normalized DGMs" folder)
- RF_DGM_LOUO.ipynb (uses "DGMs" folder)
- RF_DGM_LOUO_normalized.ipynb (uses "Organized Normalized DGMs" folder)

```
project_root/
   non-contaminated datasets/
      DGMs/
         bar/
            illiterate/
               user_2/
```

```
            user_2_question_X_DGMs.csv
            ...
        user_3/
            ...
        user_N/
    literate/
        user_1/
            user_1_question_X_DGMs.csv
            ...
        ...
    SAME FOR LINE AND PIE
            ...
Organized Normalized DGMs/
    bar/
        illiterate/
            user_2/
                user_2_question_X_DGMs.csv
                ...
            user_3/
                ...
            user_N/
        literate/
            user_1/
                user_1_question_X_DGMs.csv
                ...
            ...
        SAME FOR LINE AND PIE
                ...
Code/
    Utilities/
        users_literacy_results.csv
```

PART 2: VTNET LOUO NOTEBOOKS

The VTNet LOUO evaluation code expects a specific folder structure for organizing eye-tracking data. The code automatically locates the project root by searching for the "non-contaminated datasets" directory marker.

VTNet LOUO notebooks include:

- vtnet_LOUO_XY.ipynb (uses scanpath images and raw gaze coordinates)
- vtnet_LOUO_XY_6col.ipynb (uses scanpath images and raw gaze coordinates, 6-column format)
- vtnet_LOUO_XY_7col_normalized.ipynb (uses scanpath images and raw gaze coordinates, 7-column normalized format)

- vtnet_LOUO_DGM_3s.ipynb (uses scanpath images and 3-second tumbling window DGMs)
- vtnet_LOUO_DGM_3s_normalized.ipynb (uses scanpath images and 3-second tumbling window DGMs, normalized)

```
project_root/
   non-contaminated datasets/
      Scanpaths/
         bar/
            illiterate/
               user_2/
                  user_2_question_X_scanpath_ID_Y.png
                  ...
               user_3/
                  ...
               user_N/
            literate/
               user_1/
                  user_1_question_X_scanpath_ID_Y.png
                  ...
               ...
         SAME FOR LINE AND PIE
                  ...
      Raw Eye Tracking Data/
         bar/
            illiterate/
               user_2/
                  user_2_question_X_ID_Y.csv
                  ...
               ...
            literate/
               user_X/
                  user_X_question_Y_ID_Z.csv
                  ...
         SAME FOR LINE AND PIE
                  ...
      Tumbling Window DGMs (3s)/
         bar/
            illiterate/
               user_2/
                  user_2_question_X_ID_Y_tumbling_all_window_DGMs.csv
                  ...
               user_3/
                  ...
```

```
        user_N/
    literate/
        user_1/
            user_1_question_X_ID_Y_tumbling_all_window_DGMs.csv
            ...
        ...
```

SAME FOR LINE AND PIE

```
            …
```

SAME FOR Organized Normalized DGMs, Organized Normalized Raw Eye Tracking Data, AND Organized Normalized Raw Eye Tracking Data

```
            …
Code/
    Utilities/
        users_literacy_results.csv
```

FILE NAMING CONVENTIONS

Scanpath Images:

- Format: user_{user_id}_question_{question_id}_scanpath_ID_{original_question_id}.png
- Example: user_11_question_23_scanpath_ID_3.png

Raw Data CSV Files:

- Format: user_{user_id}_question_{question_id}_ID_{original_question_id}.csv
- Example: user_11_question_23_ID_12.csv
- Note: The CSV filename should match the scanpath filename (without the "_scanpath" suffix and with ".csv" extension)

Tumbling Window DGM CSV Files:

- Format: user_{user_id}_question_{question_id}_ID_{original_question_id}_tumbling_all_window_DGMs.csv
- Example: user_2_question_1_ID_34_tumbling_all_window_DGMs.csv
- Contains multiple rows, one for each 3-second time window

DATA REQUIREMENTS

1. Matching Pairs: Each scanpath image must have a corresponding CSV file (raw gaze or DGM) with matching user ID and question ID
2. User Folders: User folders must follow the pattern "user_{ID}" where {ID} is a numeric identifier
3. Graph Types: Supported graph types are "bar", "line", and "pie"
4. Literacy Classes: Two classes are expected: "illiterate" (label 0) and "literate" (label 1)

Full-Study Defined Classes (FSC)

We also experimented with the classes determined by the median split threshold determined by the results from the entire 50-question study.

We modified and compared the performance of 3 VTNet versions: **vtnet_LOUO_XY**, **vtnet_LOUO_XY_6col_normalized**, and **vtnet_LOUO_DGM_3s**.

After running the models and comparing the results, we saw that there was **no significant difference between using the 4 prestudy questions and the full-study median split**. After concluding there was no improvement in performance, we concluded the use of this data split and kept with the original data split.

The dataset for the Full-Study Defined Classes can be found in the Teams Channel under non-contaminated datasets - full study.

LOUO Results - Prestudy Defined Classes (PSC)

The following models used the classes determined by the 4 pre-study questions asked before the start of the 50-question study.

DGM Cross Validation - PSC

Original DGMs

Model	Bar Metrics	Line Metrics	Pie Metrics
Logistic Regression log_reg_DGM_LOUO	**Accuracy (overall):** 0.508 **Accuracy (user):** .500 **Precision:** 0.436 **Recall:** 0.464 **F1 Score:** 0.450	**Accuracy (overall):** 0.503 **Accuracy (user):** .467 **Precision:** 0.429 **Recall:** 0.437 **F1 Score:** 0.433	**Accuracy (overall):** 0.493 **Accuracy (user):** .400 **Precision:** 0.429 **Recall:** 0.508 **F1 Score:** 0.465
Random Forrest	**Accuracy (overall):** 0.497 **Accuracy (user):** .433	**Accuracy (overall):** 0.504 **Accuracy (user):** .467	**Accuracy (overall):** 0.367 **Accuracy (user):** .333

RF_DGM_LOUO	**Precision:** 0.404 **Recall:** 0.334 **F1 Score:** 0.366	**Precision:** 0.405 **Recall:** 0.304 **F1 Score:** 0.347	**Precision:** 0.250 **Recall:** 0.231 **F1 Score:** 0.240
VTNet vtnet_LOUO_DGM_3s	**Accuracy (overall):** 0.519 **Accuracy (user):** .500 **Precision:** 0.429 **Recall:** 0.312 **F1 Score:** 0.361	**Accuracy (overall):** 0.513 **Accuracy (user):** .533 **Precision:** 0.353 **Recall:** 0.138 **F1 Score:** 0.198	**Accuracy (overall):** 0.125 **Accuracy (user):** .133 **Precision:** 0.111 **Recall:** 0.143 **F1 Score:** 0.125

DGMs with Normalized Pupil Data

Model	Bar Metrics	Line Metrics	Pie Metrics
Logistic Regression log_reg_DGM_LOUO_normalized	**Accuracy (overall):** 0.437 **Accuracy (user):** 0.400 **Precision:** 0.372 **Recall:** 0.432 **F1 Score:** 0.399	**Accuracy (overall):** 0.408 **Accuracy (user):** 0.400 **Precision:** 0.332 **Recall:** 0.360 **F1 Score:** 0.346	**Accuracy (overall):** 0.413 **Accuracy (user):** 0.433 **Precision:** 0.354 **Recall:** 0.431 **F1 Score:** 0.389
Random Forest RF_DGM_LOUO_normalized	**Accuracy (overall):** 0.545 **Accuracy (user):** 0.467 **Precision:** 0.475 **Recall:** 0.473 **F1 Score:** 0.474	**Accuracy (overall):** 0.504 **Accuracy (user):** 0.533 **Precision:** 0.427 **Recall:** 0.413 **F1 Score:** 0.420	**Accuracy (overall):** 0.460 **Accuracy (user):** 0.433 **Precision:** 0.362 **Recall:** 0.323 **F1 Score:** 0.341
VTNet vtnet_LOUO_DGM_3s_normalized	**Accuracy (overall):** 0.512 **Accuracy (user):** 0.400 **Precision:** 0.421 **Recall:** 0.361 **F1 Score:** 0.381	**Accuracy (overall):** 0.561 **Accuracy (user):** 0.533 **Precision:** 0.357 **Recall:** 0.251 **F1 Score:** 0.356	**Accuracy (overall):** 0.312 **Accuracy (user):** 0.467 **Precision:** 0.231 **Recall:** 0.312 **F1 Score:** 0.251

Raw Eye Tracking Data Cross-Validation - PSC

Original X & Ys

Model	Bar Metrics	Line Metrics	Pie Metrics
VTNet vtnet_LOUO_XY	**Accuracy (overall):** 0.456 **Accuracy (user):** 0.500 **Precision:** 0.331 **Recall:** 0.243 **F1 Score:** 0.280	**Accuracy (overall):** 0.523 **Accuracy (user):** 0.500 **Precision:** 0.413 **Recall:** 0.218 **F1 Score:** 0.286	**Accuracy (overall):** 0.500 **Accuracy (user):** 0.467 **Precision:** 0.444 **Recall:** 0.571 **F1 Score:** 0.500

6 col Original - X, Y, Head Distance (LPS, RPS), Original Pupil Size (LPMM, RPMM)

Model	Bar Metrics	Line Metrics	Pie Metrics
VTNet vtnet_LOUO_XY_6col	**Accuracy (overall):** 0.461 **Accuracy (user):** 0.400 **Precision:** 0.325 **Recall:** 0.220 **F1 Score:** 0.262	**Accuracy (overall):** 0.491 **Accuracy (user):** 0.500 **Precision:** 0.491 **Recall:** 0.322 **F1 Score:** 0.389	**Accuracy (overall):** 0.438 **Accuracy (user):** 0.467 **Precision:** 0.417 **Recall:** 0.714 **F1 Score:** 0.526

6 col Normalized - X, Y, Head Distance (LPS, RPS), delta_LPMM, delta_RPMM

Model	Bar Metrics	Line Metrics	Pie Metrics
VTNet vtnet_LOUO_XY_6col_normalized	**Accuracy (overall):** 0.504 **Accuracy (user):** 0.500 **Precision:** 0.435 **Recall:** 0.468 **F1 Score:** 0.451	**Accuracy (overall):** 0.503 **Accuracy (user):** 0.400 **Precision:** 0.440 **Recall:** 0.506 **F1 Score:** 0.471	**Accuracy (overall):** 0.438 **Accuracy (user):** 0.467 **Precision:** 0.375 **Recall:** 0.429 **F1 Score:** 0.400

7 col - X, Y, Head Distance (LPS, RPS), delta_LPMM, delta_RPMM, delta_BPMM

Model	Bar Metrics	Line Metrics	Pie Metrics
VTNet vtnet_LOUO_XY_7col_normalized	**Accuracy (overall):** 0.547 **Accuracy (user):** 0.533 **Precision:** 0.481 **Recall:** 0.509 **F1 Score:** 0.494	**Accuracy (overall):** 0.548 **Accuracy (user):** 0.500 **Precision:** 0.480 **Recall:** 0.414 **F1 Score:** 0.444	**Accuracy (overall):** 0.312 **Accuracy (user):** 0.267 **Precision:** 0.300 **Recall:** 0.429 **F1 Score:** 0.353

LOUO Results - Full-Study Defined Classes (FSC)

We also experimented with the classes determined by the median split threshold determined by the results from the entire 50-question study.

DGM Cross Validation - FSC

Original DGMs

Model	Bar Metrics	Line Metrics	Pie Metrics
VTNet vtnet_LOUO_DGM_3s	**Accuracy (overall):** 0.504 **Accuracy (user):** 0.500 **Precision:** 0.435 **Recall:** 0.468 **F1 Score:** 0.451	**Accuracy (overall):** 0.503 **Accuracy (user):** 0.467 **Precision:** 0.440 **Recall:** 0.506 **F1 Score:** 0.471	**Accuracy (overall):** 0.438 **Accuracy (user):** 0.433 **Precision:** 0.375 **Recall:** 0.429 **F1 Score:** 0.400

Raw Eye Tracking Data Cross-Validation - FSC

Original X & Ys

Model	Bar Metrics	Line Metrics	Pie Metrics
VTNet vtnet_LOUO_XY	**Accuracy (overall):** 0.513 **Accuracy (user):** 0.467 **Precision:** 0.472 **Recall:** 0.343 **F1 Score:** 0.397	**Accuracy (overall):** 0.489 **Accuracy (user):** 0.500 **Precision:** 0.439 **Recall:** 0.338 **F1 Score:** 0.382	**Accuracy (overall):** 0.440 **Accuracy (user):** 0.433 **Precision:** 0.365 **Recall:** 0.271 **F1 Score:** 0.311

6 col Normalized - X, Y, Head Distance (LPS, RPS), delta_LPMM, delta_RPMM

Model	Bar Metrics	Line Metrics	Pie Metrics
VTNet	**Accuracy (overall):** 0.452	**Accuracy (overall):** 0.525	**Accuracy (overall):** 0.502

vtnet_LOUO_XY_6col_normalized	**Accuracy (user):** 0.500 **Precision:** 0.331 **Recall:** 0.246 **F1 Score:** 0.280	**Accuracy (user):** 0.500 **Precision:** 0.411 **Recall:** 0.218 **F1 Score:** 0.296	**Accuracy (user):** 0.467 **Precision:** 0.454 **Recall:** 0.577 **F1 Score:** 0.500

Contaminated Data Analysis

In addition to analyzing model performance by question type, we also evaluated performance when question types (bar, line, and pie) were combined rather than analyzed separately. This mixed setting allowed each user's dataset to include all 50 questions, increasing the amount of data available per user. Model evaluation in this condition used a **leave-one-user-out (LOUO)** architecture. Overall, mixing question types did not result in any significant changes in model performance compared to the per-question-type analysis. A table summarizing model performance under this setting is provided below.

Model	2-col (X and Y)	6-col (X and Y, Pupil Size, Screen Proximity)	DGMs
VTNet	**Accuracy:** 0.558 **Precision:** 0.500 **Recall:** 0.279 **F1 Score:** 0.344	**Accuracy:** 0.489 **Precision:** 0.517 **Recall:** 0.261 **F1 Score:** 0.312	**Accuracy:** 0.567 **Precision:** 0.567 **Recall:** 0.567 **F1 Score:** 0.567 Across all users, the model only predicted the majority class (illiterate).
BERT	**Accuracy:** 0.4960 **Precision:** 0.5333 **Recall:** 0.3660 **F1 Score:** 0.4035	**Accuracy:** 0.5333 **Precision:** 0.5333 **Recall:** 0.4180 **F1 Score:** 0.4573	**Accuracy:** 0.5427 **Precision:** 0.4000 **Recall:** 0.2047 **F1 Score:** 0.2467