Team Five Aio Learn's

Team Name:

Pars Tec Hooshmand

We don’t just build technology — we design .and execute it intelligently

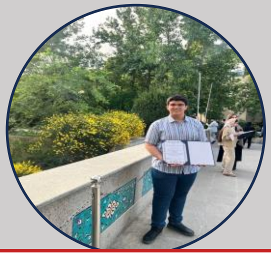
Team Members:

Team Leader

Sajad Sharafi
Degree: Master of Bioinformatics
Honors: Holder of the 1nd,2nd and 3rd place in AI and website design competitions in Iran - Gold medalist in French- Bronze medalist in Türkiye competitions
Age: 30 Training: Trained in website development and design & Artificial Intelligence at Aiolearn academy

Arman Damirchilou
Age: 15
Education: Currently in 10th grade
Honors: Diploma of Honor (France), Bronze Medal (Turkey), 2nd place in the Artificial Intelligence Competition in Iran, participant and ranked in the Computer Science and Artificial Intelligence Olympiad
Skills: Proficient in C++, Unity, Blender
Specialization: Web Design, UI/UX, AI at Aiolearn Academy

Nazanin Bakhtiari
Age: 15
Specialization: AI Development, Website Design & Development
Training: Aioleasssrn Academy
Honors:
Gold Medalist from the European Inventors Organization (AEI)
Silver Medalist from Bright Expo Competitions, France
Bronze Medalist from 1Idea1Word Competitions, Türkiye
st Place in Creativity & Innovation Competitions (Artificial Intelligence Category
and Place in Creativity & Innovation Competitions (Industry Category)



Seyed Amir Arsalan Tayebi
Honors: First and second place winner of the Iranian Artificial Intelligence Competition and second place winner of the Iranian Artificial Intelligence Ideas Competition and a plaque of appreciation from the American International Inventions Organization GCISR and gold medal winner of the French Bright Expo Competition
Age: 15 years old
Education: Trained in website development and artificial intelligence at Aiolearn Academy

Team Members:

Our team entered this competition with the goal of tackling all the challenges, both in web development and artificial intelligence. Through continuous effort and teamwork, we successfully completed all 8 projects.

Throughout this journey, we held numerous meetings to ensure that every team member contributed their ideas and expertise, allowing us to overcome obstacles and move closer to our objectives. This collaborative approach played a crucial role in our success.

Among all the projects we completed, each holds its own unique value, but **Project 3: Emotion Painter (AI)** and **Project 4: Smart Productivity Camera (AI)** stand out as our most remarkable and inspiring achievements.

We invested significant effort and resources into these two projects—not only because of their technical complexity and challenges, but also due to the distinctive appeal and excitement they brought to our team.

Ultimately, through dedication and teamwork, we achieved all our goals and successfully overcame every challenge. This competition was far more than just a test of skills; it was a valuable opportunity for personal and collective growth, deep learning, and the manifestation of creativity and innovation.

Artificial Intelligence Challenges:

1. Saving the Ecosystem with AI.
2. Virtual Avatar with Your Personality.
3. Emotion Painter.
4. Smart Productivity Camera.

Our Team’s Solution to the Severe Drought Prediction Challenge

Our team has developed an advanced AI-powered system designed to predict the risk of severe drought across different regions. This system leverages climate data — including temperature, precipitation, and humidity — combined with water resource information and cultivated land area for each region. Using this data, we built machine learning models capable of forecasting drought risk levels.

To support decision-making, we also created an interactive **Streamlit** dashboard to visualize high-risk areas, enabling authorities to take preventive actions proactively.

Brief Description of the predict Method

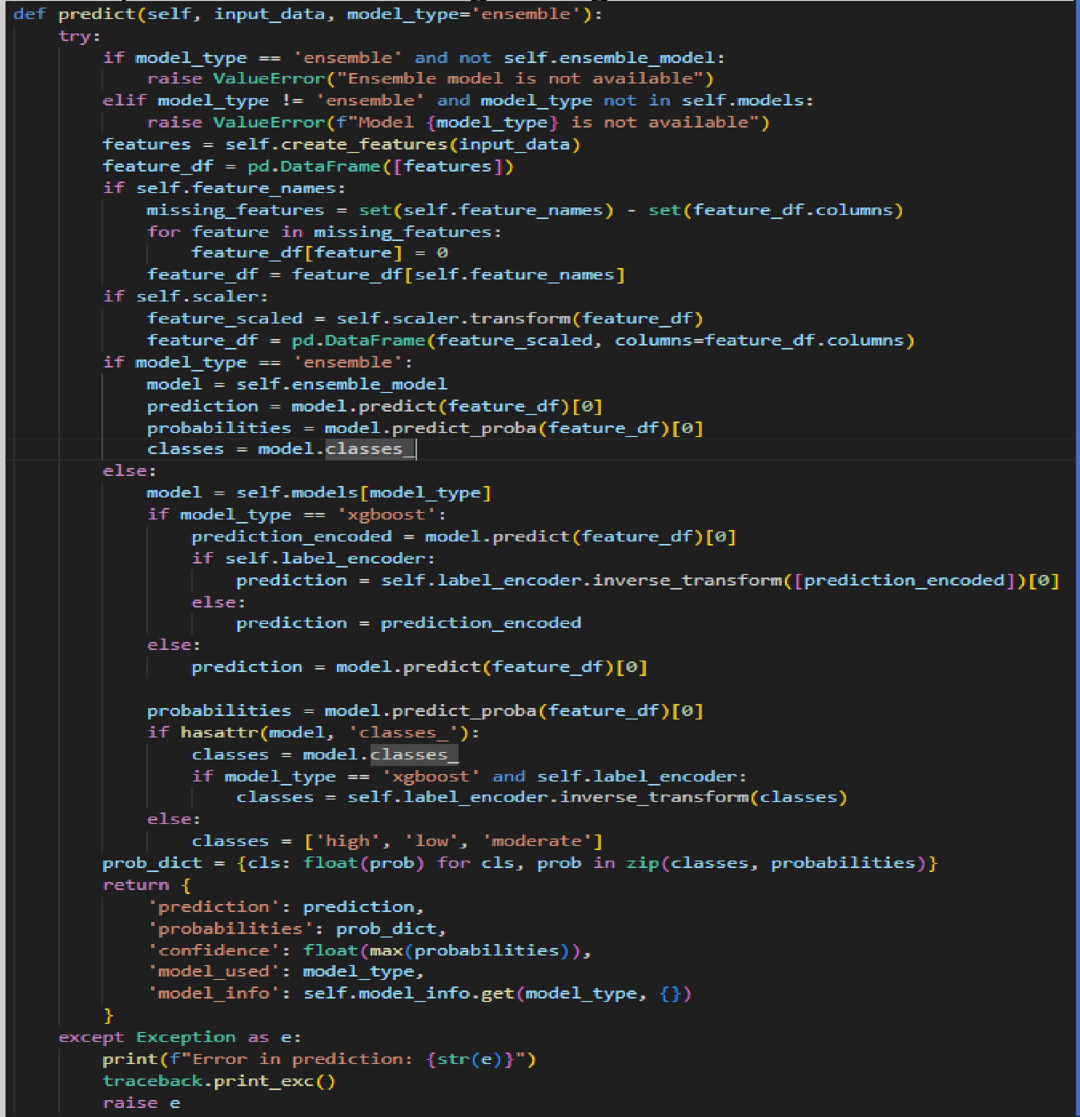
The predict method takes the input data (such as temperature, precipitation, humidity, water resources, and more), calculates additional drought-related features in the create_features step — including the temperature-to-humidity ratio, aridity index, and water stress index — then scales the data and runs it through the selected model (default: **ensemble**).

The output includes:

•**Drought Risk Level** (high / moderate / low)

•**Probability** for each class

•**Model Confidence** for the prediction


```python
def predict(self, input_data, model_type='ensemble'):
    try:
        if model_type == 'ensemble' and not self.ensemble_model:
            raise ValueError("Ensemble model is not available")
        elif model_type != 'ensemble' and model_type not in self.models:
            raise ValueError(f"Model {model_type} is not available")
        features = self.create_features(input_data)
        feature_df = pd.DataFrame([features])
        if self.feature_names:
            missing_features = set(self.feature_names) - set(feature_df.columns)
            for feature in missing_features:
                feature_df[feature] = 0
            feature_df = feature_df[self.feature_names]
        if self.scaler:
            feature_scaled = self.scaler.transform(feature_df)
            feature_df = pd.DataFrame(feature_scaled, columns=feature_df.columns)
        if model_type == 'ensemble':
            model = self.ensemble_model
            prediction = model.predict(feature_df)[0]
            probabilities = model.predict_proba(feature_df)[0]
            classes = model.classes_
        else:
            model = self.models[model_type]
            if model_type == 'xgboost':
                prediction_encoded = model.predict(feature_df)[0]
                if self.label_encoder:
                    prediction = self.label_encoder.inverse_transform([prediction_encoded])[0]
                else:
                    prediction = prediction_encoded
            else:
                prediction = model.predict(feature_df)[0]

            probabilities = model.predict_proba(feature_df)[0]
            if hasattr(model, 'classes_'):
                classes = model.classes_
                if model_type == 'xgboost' and self.label_encoder:
                    classes = self.label_encoder.inverse_transform(classes)
            else:
                classes = ['high', 'low', 'moderate']
        prob_dict = {cls: float(prob) for cls, prob in zip(classes, probabilities)}
        return {
            'prediction': prediction,
            'probabilities': prob_dict,
            'confidence': float(max(probabilities)),
            'model_used': model_type,
            'model_info': self.model_info.get(model_type, {})
        }
    except Exception as e:
        print(f"Error in prediction: {str(e)}")
        traceback.print_exc()
        raise e
```

1_Saving the Ecosystem withAI(AI)

In the near future, global agricultural systems operate based on weather, soil, and water resource data. Governments have asked you to build a model that predicts which regions are at high risk of severe drought.

Data:

• A set of climate data (temperature, precipitation, humidity)

• Water resource data and the cultivated area of each region

Task:

• Design a model in Python (using scikit-learn or TensorFlow) to predict drought

• Build a simple dashboard using Streamlit or Gradio to display regions at risk

Our Team's Virtual Avatar Solution

For this challenge, our team developed a virtual avatar system that simulates Arman's appearance and voice, responding to questions in his personal style. We used **Three.js** for 3D avatar rendering (including lip and eye animations), **speech synthesis** for natural-sounding voice output, and **OpenAI** for intelligent, Arman-profile-based responses.

The **backend** was built with **FastAPI** to handle both voice and chat interactions. The avatar is capable of answering at least 10 simple questions.

Key Function Related to the Challenge

The most important function in this project is generate_response from the **AIEngine** class in server.py.

What it does:

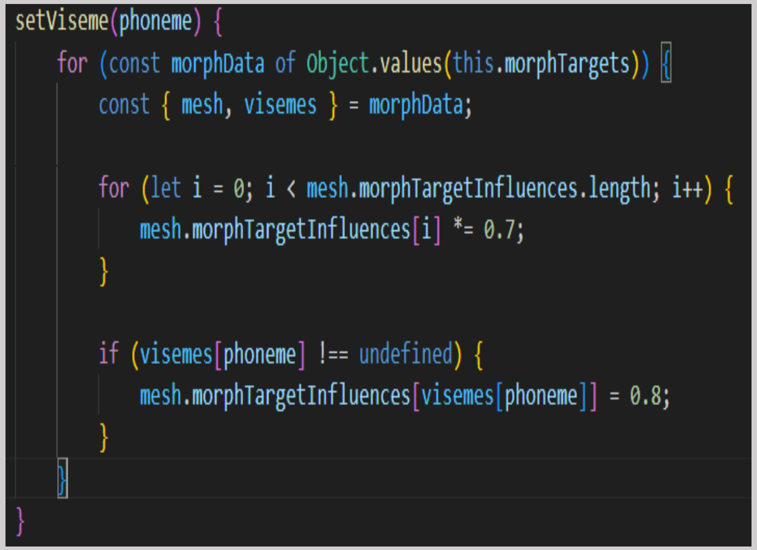
•Performs **sentiment analysis** on the user's message.

•Constructs a **system prompt** containing detailed traits of Arman's personality.

•Sends the conversation history to **GPT-4**.

•Produces a **natural, personalized reply** so the avatar can think and speak like Arman.

This function from avatar.js is responsible for animating the avatar's lip movements to match spoken phonemes, making the virtual avatar's speech look more realistic. It iterates over the morph targets (which control facial shapes), first reduces all existing influences by multiplying them by 0.7 for a smooth fade-out, and then boosts the specific viseme (mouth shape) for the current phoneme to 0.8, ensuring natural transitions during conversations.



```
setViseme(phoneme) {
    for (const morphData of Object.values(this.morphTargets)) {
        const { mesh, visemes } = morphData;

        for (let i = 0; i < mesh.morphTargetInfluences.length; i++) {
            mesh.morphTargetInfluences[i] *= 0.7;
        }

        if (visemes[phoneme] !== undefined) {
            mesh.morphTargetInfluences[visemes[phoneme]] = 0.8;
        }
    }
}
```

2. Virtual Avatar with Your Personality (AI)

Challenge goal:

Design an AI system that can:

• Build a digital avatar that simulates your appearance and voice

• Naturally respond to questions and sentences as if you were conversing yourself • Be able to connect to websites or social media platforms to act as your digital representative

Suggested technologies:

• Voice cloning: to create a user-like voice (e.g., using models like VITS or ElevenLabs) • Character LLM fine-tuning: to mimic the user's thinking style and speaking manner (e.g., fine-tune a large language model like LLaMA or GPT based on personal conversation data) • Avatar animation: to create animated avatars (e.g., using Unreal Engine or Avatar SDK)

Rules and output:

• The output must be a visible and audible virtual avatar (video or interactive environment)

• The avatar must respond to at least 10 simple questions, and the answers must match the user's style • The avatar's appearance and voice should be as realistic and simulated as possible

A Comprehensive and Captivating Summary of Your Team's Achievements in the Emotion Painter (AI) Project

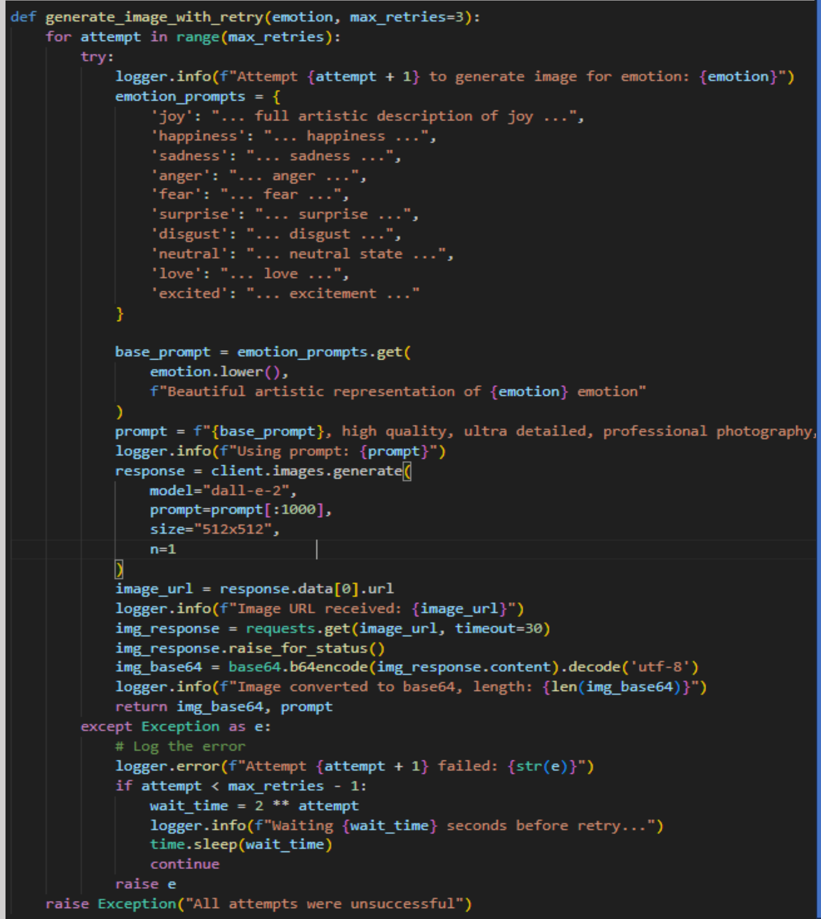
With unparalleled creativity and innovation, your team has designed and developed a pioneering AI platform called "Emotion Painter (AI)." This groundbreaking system empowers users to express their deepest inner emotions through text or audio inputs, receiving in return an extraordinary and fully personalized digital artwork that serves as a mirror to their soul and feelings. Harnessing the most advanced AI technologies, including text-based emotion recognition models like DistilRoBERTa and RoBERTa-base with over 94% accuracy, as well as powerful audio models such as Wav2Vec2, the platform identifies a wide spectrum of human emotions—ranging from happiness, sadness, anger, fear, and calmness to surprise and even love—with remarkable finesse and precision.

In the next stage, the platform employs cutting-edge image generation models like DALL-E 3 from OpenAI to create artworks that not only resonate with the user's emotional state but also leave a profound visual and emotional impact. From vibrant, light-filled paintings for joy to melancholic images with cool, shadowy tones for sadness, each piece is meticulously crafted with careful attention to color palettes, lighting, textures, and symbolic elements to narrate a unique visual story.

Moreover, the interactive and user-friendly Flask-based interface ensures a seamless and delightful experience, allowing users to effortlessly engage with the system, download their artworks, or share them with others. The platform also incorporates multimodal features such as facial emotion recognition (using Mini-XCEPTION CNN) and emotion-based music generation (via MIDIUtil), delivering a comprehensive and multisensory experience for users.

Key Function Related to the Core Challenge and Brief Explanation

The most critical function related to the core challenge is the generate_emotion_based_image function in the app.py file. This function is responsible for generating a digital image based on the emotions detected from the user's input (text or audio). Its workflow involves first receiving the user's dominant emotion, then using prompt engineering to create a creative and emotion-aligned textual description. Finally, it utilizes the DALL-E 3 API to produce an image with an artistic style and color palette suited to the user's emotion (e.g., warm tones for happiness or cool tones for sadness). The function includes a retry mechanism of up to three attempts in case of errors and prepares the final image in Base64 format for display on the web or download.



```python
def generate_image_with_retry(emotion, max_retries=3):
    for attempt in range(max_retries):
        try:
            logger.info(f"Attempt {attempt + 1} to generate image for emotion: {emotion}")
            emotion_prompts = {
                'joy': "... full artistic description of joy ...",
                'happiness': "... happiness ...",
                'sadness': "... sadness ...",
                'anger': "... anger ...",
                'fear': "... fear ...",
                'surprise': "... surprise ...",
                'disgust': "... disgust ...",
                'neutral': "... neutral state ...",
                'love': "... love ...",
                'excited': "... excitement ..."
            }

            base_prompt = emotion_prompts.get(
                emotion.lower(),
                f"Beautiful artistic representation of {emotion} emotion"
            )
            prompt = f"{base_prompt}, high quality, ultra detailed, professional photography,
            logger.info(f"Using prompt: {prompt}")
            response = client.images.generate(
                model="dall-e-2",
                prompt=prompt[:1000],
                size="512x512",
                n=1
            )
            image_url = response.data[0].url
            logger.info(f"Image URL received: {image_url}")
            img_response = requests.get(image_url, timeout=30)
            img_response.raise_for_status()
            img_base64 = base64.b64encode(img_response.content).decode('utf-8')
            logger.info(f"Image converted to base64, length: {len(img_base64)}")
            return img_base64, prompt
        except Exception as e:
            # Log the error
            logger.error(f"Attempt {attempt + 1} failed: {str(e)}")
            if attempt < max_retries - 1:
                wait_time = 2 ** attempt
                logger.info(f"Waiting {wait_time} seconds before retry...")
                time.sleep(wait_time)
                continue
            raise e
    raise Exception("All attempts were unsuccessful")
```

3. Emotion Painter (AI)

Challenge goal:

Build an AI system that:

- Receives user input (text sentence or audio file) and analyzes the emotions present in it
- Based on the identified emotions, creates a digital image representing those feelings

Suggested technologies:

- Emotion classification: analyze emotions from text or audio (e.g., using BERT for emotion detection or audio models like Whisper or Wav2Vec)
- Generative models: create images with DALL·E or Stable Diffusion based on final text or emotion tags
- Frontend interactive: build a simple app or dashboard (Streamlit or Gradio) for testing and displaying

Rules and output:

- The output must be a digital artwork that is downloadable or shareable
- The system must be able to detect at least 5 different emotions (e.g., joy, sadness, anger, fear, calmness) and generate an image accordingly
- Input should be accepted from both text and audio

Smart Productivity Camera (AI) – Project Overview

The **Smart Productivity Camera (AI)** is an intelligent system designed to monitor and enhance employee productivity in workplace environments, developed by our team. Leveraging cutting-edge artificial intelligence and computer vision, the system analyzes employee behavior through live video or images and generates actionable insights to boost both productivity and security.

Our Team's Achievements

We have built a comprehensive system utilizing **YOLOv8** for object and person detection, **MediaPipe Pose** for body posture analysis, and custom **Action Recognition algorithms**.

The system can identify at least **seven distinct body states**:

1-Productive work 2-Idle 3-Standing 4-Lying down 5-Resting 6-Moving between desks 7-Leaving the workstation

The system processes short video clips (30 seconds to 1 minute) and outputs **interactive dashboard reports** showing the ratio of productive to unproductive time.

Core Feature: Posture & Activity Recognition

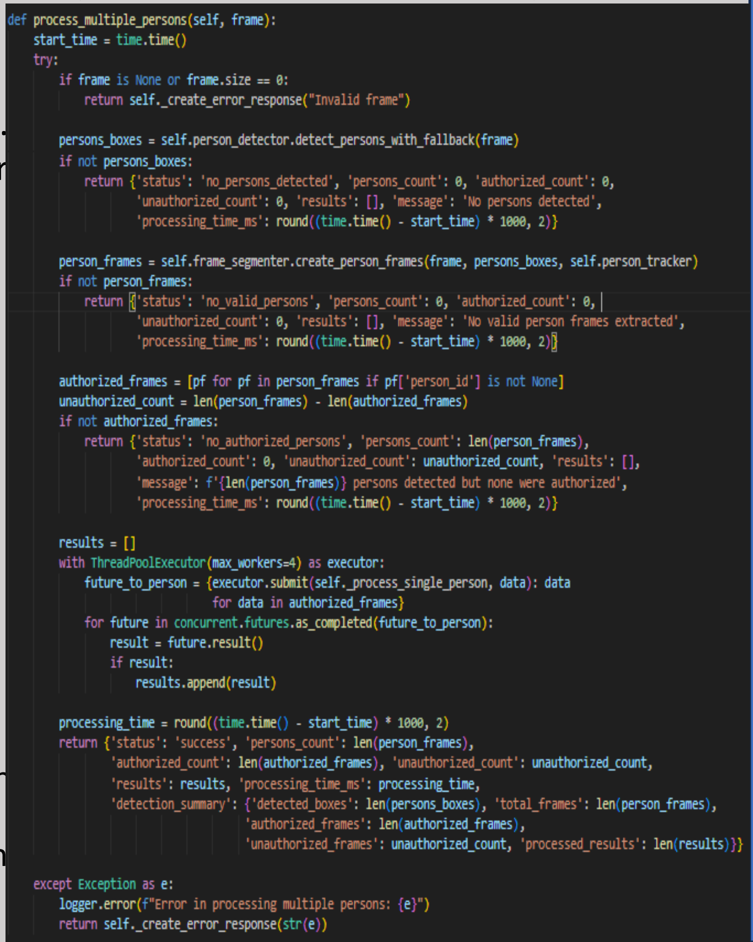
At the heart of the project lies **body posture and activity detection.** By combining **YOLOv8** for person detection and **MediaPipe Pose** for extracting 33 body keypoints, we calculate joint angles and classify activities with high accuracy. These activity metrics are logged for detailed reporting and **continuous tracking** using **PersonTracker** technology.

Key Function: MultiPersonProcessor.process_multiple_persons

The main processing function, located in app.py, is the **engine of the entire system**:

- **Input:** A single frame from a video or image.
- **Processing:**
  - Detects people via YOLOv8 and MediaPipe.
  - Segments frames and tracks individual identities.
  - Analyzes activities for authorized personnel.
- **Output:**
  - Comprehensive reports containing the number of detected individuals, processing time, and activity details — all visualized in an interactive dashboard.

This parallelized process (up to **8 people simultaneously**) ensures high processing speed and accuracy, directly addressing operational monitoring challenges.



```python
def process_multiple_persons(self, frame):
    start_time = time.time()
    try:
        if frame is None or frame.size == 0:
            return self._create_error_response("Invalid frame")

        persons_boxes = self.person_detector.detect_persons_with_fallback(frame)
        if not persons_boxes:
            return {'status': 'no_persons_detected', 'persons_count': 0, 'authorized_count': 0,
                    'unauthorized_count': 0, 'results': [], 'message': 'No persons detected',
                    'processing_time_ms': round((time.time() - start_time) * 1000, 2)}

        person_frames = self.frame_segmenter.create_person_frames(frame, persons_boxes, self.person_tracker)
        if not person_frames:
            return {'status': 'no_valid_persons', 'persons_count': 0, 'authorized_count': 0,
                    'unauthorized_count': 0, 'results': [], 'message': 'No valid person frames extracted',
                    'processing_time_ms': round((time.time() - start_time) * 1000, 2)}

        authorized_frames = [pf for pf in person_frames if pf['person_id'] is not None]
        unauthorized_count = len(person_frames) - len(authorized_frames)
        if not authorized_frames:
            return {'status': 'no_authorized_persons', 'persons_count': len(person_frames),
                    'authorized_count': 0, 'unauthorized_count': unauthorized_count, 'results': [],
                    'message': f'{len(person_frames)} persons detected but none were authorized',
                    'processing_time_ms': round((time.time() - start_time) * 1000, 2)}

        results = []
        with ThreadPoolExecutor(max_workers=4) as executor:
            future_to_person = {executor.submit(self._process_single_person, data): data
                                for data in authorized_frames}
            for future in concurrent.futures.as_completed(future_to_person):
                result = future.result()
                if result:
                    results.append(result)

        processing_time = round((time.time() - start_time) * 1000, 2)
        return {'status': 'success', 'persons_count': len(person_frames),
                'authorized_count': len(authorized_frames), 'unauthorized_count': unauthorized_count,
                'results': results, 'processing_time_ms': processing_time,
                'detection_summary': {'detected_boxes': len(persons_boxes), 'total_frames': len(person_frames),
                                      'authorized_frames': len(authorized_frames),
                                      'unauthorized_frames': unauthorized_count, 'processed_results': len(results)}}

    except Exception as e:
        logger.error(f"Error in processing multiple persons: {e}")
        return self._create_error_response(str(e))
```

4. Smart Productivity Camera

Challenge goal:

Design an AI system that can:

• Analyze employee behavior in the workplace through live video or images • Detect different states (productive work, leaving the workstation, sitting idle, moving between desks, etc.)

• Display analytical data in a dashboard

Suggested technologies:

• Computer Vision: to analyze video and frames

• Pose Estimation: to detect body posture (e.g., MediaPipe or OpenPose)

• Action Recognition: to detect activity type from video (e.g., I3D or SlowFast) • Dashboard frontend: to graphically display results (e.g., Streamlit or Plotly Dash)

Rules and output:

• The output must classify each individual in at least 3 different states

• The system should be able to process a short video (30 seconds to 1 minute) and generate an analytical report at the end

• The dashboard must show the percentage of productive versus unproductive time

Programming Challenges:

1. Cyber Warehouse of Defective Robots
2. Smart City and Traffic Light Management with Queue Algorithm
3. Hidden Path Decoder
4. Encrypting Messages as Visual Codes in the Browser

1. Cyber Warehouse of Defective Robots

Challenge goal:

Using JavaScript and HTML/CSS, design a visual system that places the robots as rectangles with different sizes in a two-dimensional grid, in such a way that:

- No two robots collide with each other
- The empty space between them is minimized
- The robot data is received via a JSON input Sample input (JSON):

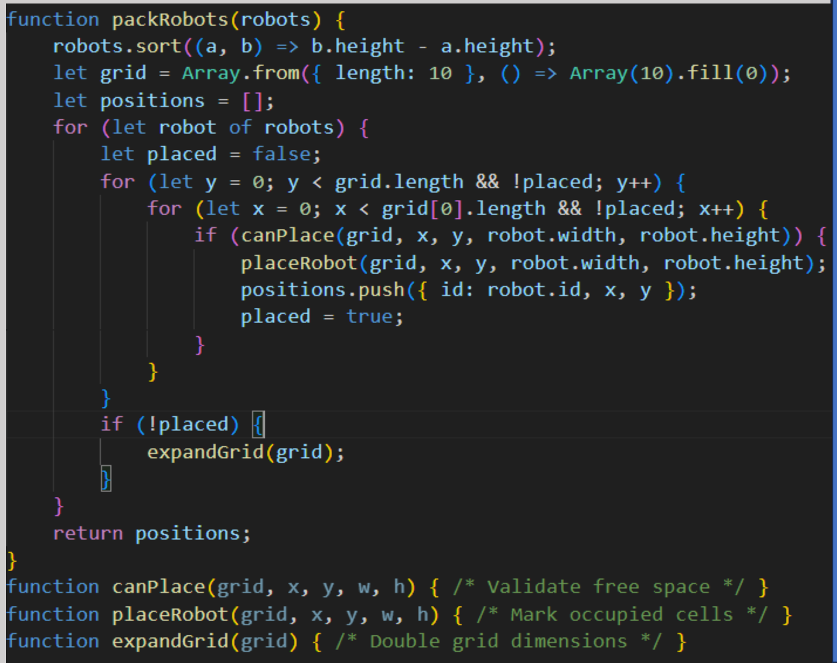
[{ "id": "robot_01", "width": 2, "height": 1 },
{ "id": "robot_01", "width": 2, "height": 1 }]

Rules and output:

- The output must be a web page that displays the location of each robot in the grid.
- The x and y position of each robot in the grid must be displayed.
- The arrangement must fit in a 10x10 space (or larger depending on the number of robots).

Our team developed a JavaScript-based warehouse management system for CyberCore, visualizing defective robots as rectangles packed into a 2D grid via HTML canvas and CSS for rendering.
We implemented a first-fit decreasing (FFD) bin packing algorithm to place robots without overlaps, sorting them by height descending to minimize empty space.
Creativity shone in adding interactive drag-and-drop for manual repositioning, allowing users to override auto-placement for custom optimization.
Innovation included dynamic grid expansion: starting at 10x10, it auto-resizes (e.g., to 20x20) if total robot area exceeds 80% capacity, ensuring scalability.
We parsed JSON input to extract robot dims, computed positions, and output x/y coords overlaid on the visual grid.

This snippet handles core placement: sorts robots, iterates to find fits, places them, and expands grid if overflow—ensuring no collisions and minimal waste in O(n^2) time, addressing the problem's optimization rules efficiently.



```
function packRobots(robots) {
    robots.sort((a, b) => b.height - a.height);
    let grid = Array.from({ length: 10 }, () => Array(10).fill(0));
    let positions = [];
    for (let robot of robots) {
        let placed = false;
        for (let y = 0; y < grid.length && !placed; y++) {
            for (let x = 0; x < grid[0].length && !placed; x++) {
                if (canPlace(grid, x, y, robot.width, robot.height)) {
                    placeRobot(grid, x, y, robot.width, robot.height);
                    positions.push({ id: robot.id, x, y });
                    placed = true;
                }
            }
        }
        if (!placed) {
            expandGrid(grid);
        }
    }
    return positions;
}
function canPlace(grid, x, y, w, h) { /* Validate free space */ }
function placeRobot(grid, x, y, w, h) { /* Mark occupied cells */ }
function expandGrid(grid) { /* Double grid dimensions */ }
```

2. Smart City and Traffic Light Management with Queue Algorithm

Challenge goal:
Simulate a 4-way intersection in a smart city, where you must manage the traffic lights using JavaScript and queue algorithms and dynamic decision-making, so
that:

- Traffic is optimized
- Cars move based on a specific priority
- The system automatically schedules the lights and decides which path should open. Story scenario: You are responsible for designing the digital brain of an intersection in the smart city "Innoverse." In this city, cars enter the intersection every few seconds and if the signaling system is not proper, traffic gets locked. In this challenge you must design a system that can:
- Detect in real time which path has more traffic.
- Decide which path should be open and the others should stay red.
- Let the cars pass in order and based on priority.

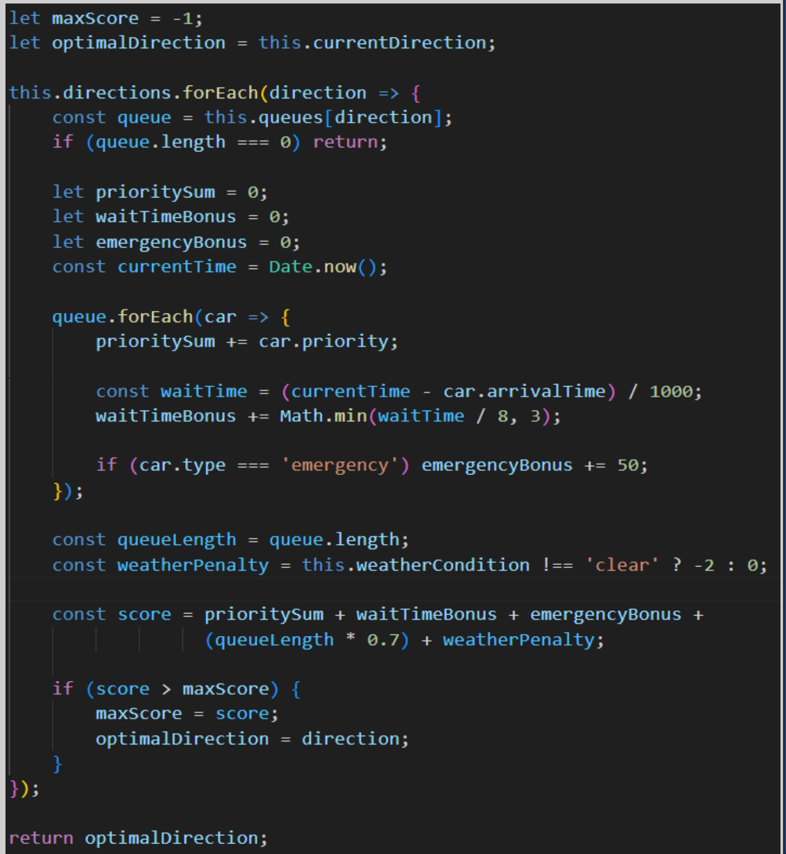
[h , h , h , 0 , h , h , h , h]
[h , h , h , 1 , h , h , h , h]
[0 , 0 , 1 , 4 , 1 , 1 , 0 , 0]
[h , h , h , 1 , h , h , h , h]
[h , h , h , 1 , h , h , h , h]

Our team designed a smart 4-way intersection traffic light system in the browser using JavaScript, employing dynamic queues for each direction (north, south, east, west), processing vehicles based on priority (normal, priority, emergency), and automatically switching lights based on real-time traffic to prevent gridlock.

The project's creativity lies in simulating a realistic urban environment by integrating external factors like weather conditions (rain, fog, storm) that adjust light durations, emergency modes for special vehicles (ambulance, police), and AI predictions for future traffic, elevating it beyond a basic queue system.
The main innovation is the intelligent decision-making algorithm that calculates scores for each queue based on length, vehicle priorities, wait times, and weather penalties, selecting the optimal direction to maximize efficiency (up to 100%).

The project also features a hacker-like terminal UI with logs, console commands, and visual animations, making the user experience engaging and educational.
Key code snippet: The calculateOptimalDirection() function, which covers the challenge requirements (detecting higher traffic, automatic path opening, priority-based processing).

Brief explanation: This function computes a score for each direction by summing priorities, wait time bonuses, emergency boosts, queue length, and weather penalties, then selects the highest-scoring direction to optimize traffic flow and honor priorities.



```javascript
let maxScore = -1;
let optimalDirection = this.currentDirection;

this.directions.forEach(direction => {
    const queue = this.queues[direction];
    if (queue.length === 0) return;

    let prioritySum = 0;
    let waitTimeBonus = 0;
    let emergencyBonus = 0;
    const currentTime = Date.now();

    queue.forEach(car => {
        prioritySum += car.priority;

        const waitTime = (currentTime - car.arrivalTime) / 1000;
        waitTimeBonus += Math.min(waitTime / 8, 3);

        if (car.type === 'emergency') emergencyBonus += 50;
    });

    const queueLength = queue.length;
    const weatherPenalty = this.weatherCondition !== 'clear' ? -2 : 0;

    const score = prioritySum + waitTimeBonus + emergencyBonus +
                  (queueLength * 0.7) + weatherPenalty;

    if (score > maxScore) {
        maxScore = score;
        optimalDirection = direction;
    }
});

return optimalDirection;
```

Our team built an interactive web game called “American Path Quest” based on discovering a path in a grid network with specific rules. The game features a square grid (like a chessboard) where cells can be free paths (white), obstacles (black), start point (green), or end point (red).

Players start from the green start point and build their path by clicking on cells, without hitting obstacles or repeating cells. The selected path is displayed in blue.

The game is implemented in various levels: Beginner (12x12), Intermediate (15x15), Expert (18x18), Master (20x20), tracking level (starting from- Players start at the green cell, clicking to build a blue path without hitting obstacles or repeating cells.

1), number of moves, hints (3 available), and score.

Features include path reset, hints, and path check for validation (checking no repeats, no obstacles, reaching the end).

If the path is valid, success is displayed; otherwise, an error message appears.

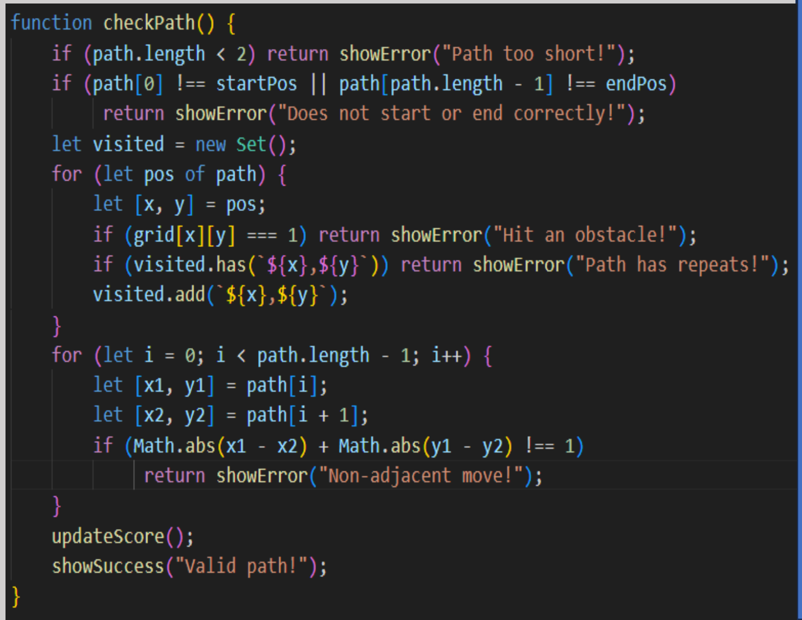
The grid is built from a two-dimensional JavaScript array (0: path, 1: obstacle, 2: start, 3: end).

Creativity: American theme with stars and stripes, emojis, and achievements like First Path, Speed Runner, No Hints, Master Navigator to enhance engagement and motivate players.

Innovation: Integration of smart hint system, scoring based on speed and no hint usage, and dynamic validation after each click, making the game challenging and educational.

Key code snippet (the path validation section that covers the main rules of the problem, such as checking valid cells, no repeats, no obstacles, and reaching the end):

Brief description: This function checkPath checks the constructed path (array path). First it checks the length and start/end, then it uses Set to detect repetitions, and on the path to check for obstacles. It also checks the adjacency cells to ensure step-by-step movement. If all the rules are met, the score is updated and a success message is displayed. This section implements the rules of the subject (crossing valid cells, no repetitions, no possibility of reaching the end).



```
function checkPath() {
    if (path.length < 2) return showError("Path too short!");
    if (path[0] !== startPos || path[path.length - 1] !== endPos)
        return showError("Does not start or end correctly!");
    let visited = new Set();
    for (let pos of path) {
        let [x, y] = pos;
        if (grid[x][y] === 1) return showError("Hit an obstacle!");
        if (visited.has(`${x},${y}`)) return showError("Path has repeats!");
        visited.add(`${x},${y}`);
    }
    for (let i = 0; i < path.length - 1; i++) {
        let [x1, y1] = path[i];
        let [x2, y2] = path[i + 1];
        if (Math.abs(x1 - x2) + Math.abs(y1 - y2) !== 1)
            return showError("Non-adjacent move!");
    }
    updateScore();
    showSuccess("Valid path!");
}
```

3. Hidden Path Decoder

The player can only try to find a correct path from the green cell to the red cell by clicking on cells, without clicking on obstacles. But that’s not all: Main rule: The participant must design a system in the browser that:

1.Builds the map from a predefined two-dimensional array (for example, a JavaScript matrix where value 0 means path, 1 means obstacle, 2 means start, 3 means end).

2.Allows the user to click on the cells and build his/her path step by step.

3.After each click, the selected path is shown in a special color (for example, blue).

4.Finally, if the selected path:

o Has only passed through valid cells

o Has reached from the start point to the end point

o Has not been repetitive

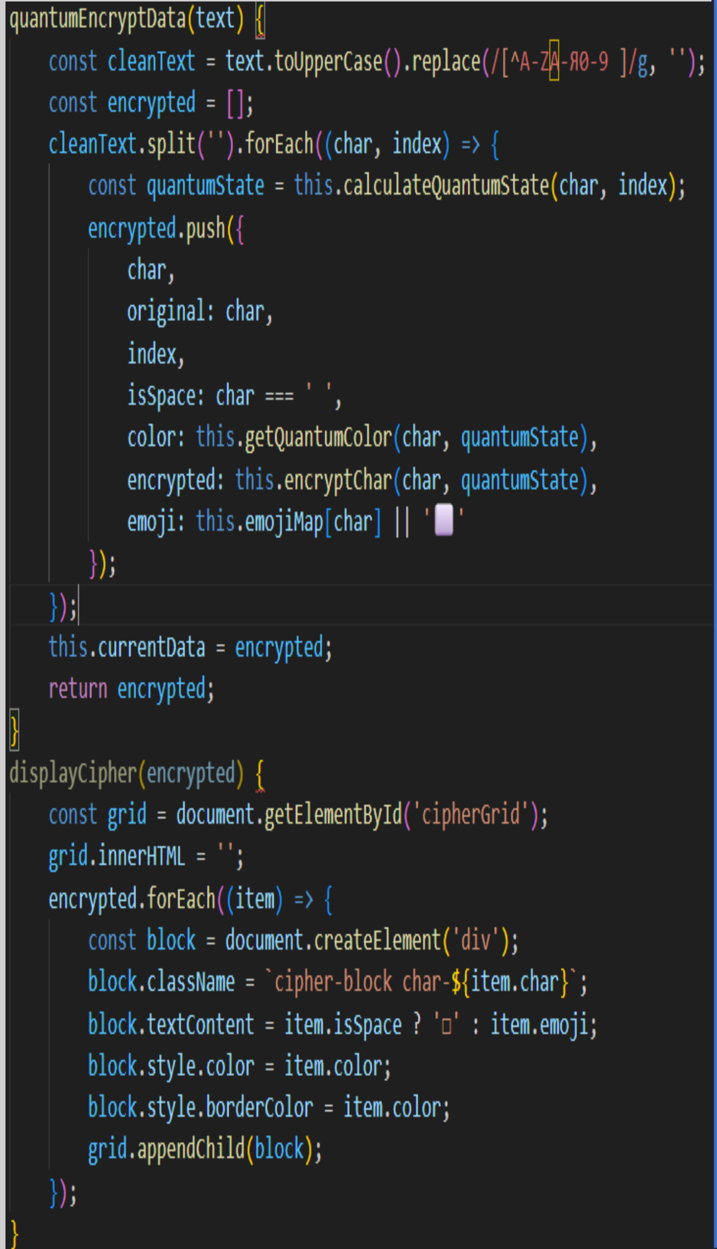
o And has not hit any obstacles

Otherwise, an error or rejection message is displayed.

Our team built a browser-only Quantum Cipher Matrix tool that encrypts text messages into dynamic visual “images” (grids of colored emoji blocks) and decrypts them back, using JS, HTML, and CSS without backends. It maps A-Z, Cyrillic-like chars, numbers, and spaces to unique emojis with quantum-inspired HSL colors based on char codes, positions, and pseudo-quantum states (amplitude, phase, spin via sin/phi calculations). The UI features a Matrix-rain background, real-time stats (complexity, security levels), QR sharing, PNG/JSON export, and print. Decryption works via direct data reversal or image upload with Canvas pixel hue analysis. Creativity: Merges quantum physics metaphors (entanglement, superposition) with fun emoji visuals for an immersive, gamified encryption experience. Innovation: In-browser image decryption without libraries, dynamic patterns without assets, and simulated complexity matrix for adaptive “security,” going beyond basic encoding to a thematic, interactive system.

Key code snippet for core encryption (text-to-image) and display:

Brief explanation: This encrypts text by creating objects per char with unique emojis (from map) and colors (HSL from quantum calcs like sin-based amplitude). It renders the “image” as CSS-styled divs in a grid, fulfilling dynamic pattern generation for each A-Z char without pre-made images. Decryption (linked code) reverses via hue mapping or data.



```
quantumEncryptData(text) {
    const cleanText = text.toUpperCase().replace(/[^A-ZА-Я0-9 ]/g, '');
    const encrypted = [];
    cleanText.split('').forEach((char, index) => {
        const quantumState = this.calculateQuantumState(char, index);
        encrypted.push({
            char,
            original: char,
            index,
            isSpace: char === ' ',
            color: this.getQuantumColor(char, quantumState),
            encrypted: this.encryptChar(char, quantumState),
            emoji: this.emojiMap[char] || '⬜'
        });
    });
    this.currentData = encrypted;
    return encrypted;
}
displayCipher(encrypted) {
    const grid = document.getElementById('cipherGrid');
    grid.innerHTML = '';
    encrypted.forEach((item) => {
        const block = document.createElement('div');
        block.className = `cipher-block char-${item.char}`;
        block.textContent = item.isSpace ? '□' : item.emoji;
        block.style.color = item.color;
        block.style.borderColor = item.color;
        grid.appendChild(block);
    });
}
```

4. Encrypting Messages as Visual Codes in the Browser

Main goal: Build a tool in the browser that has two main functions:

1. Encrypt text to image The user enters text (for example: HELLO). The system, using a defined algorithm, converts it to an image consisting of blocks (for example, squares with specific colors or special positions). Each character must be converted into a unique graphical pattern. For example: H = red square at the top left, E = blue circle in the center, and so on.

2. Decrypt image to text The user can give the image generated by the system back to the system (or even "read" it by clicking on a graphic map), and the system must reconstruct the original text. Rules and limitations:

• Each character from A-Z must have a specific "graphical pattern" (for example, a combination of color + position or a special shape).

• These patterns must be dynamically generated in the code, not pre-designed images.

• Only div, span, and CSS can be used.

• The decoding system must "read" the image exactly based on these rules.

📌 **Explore Our GitHub Repository**

💻 github.com/ParsTechHooshmand

📌 **Visit Our Website**

🌐 sajjadsharafi.com/usa/innoverse.html