Near-Crash Analysis in Smart Transportation

Enhancing Road Safety Through Edge Computing

What is a Near-Crash?

- Near-crash (near-miss or EAR-crash): An incident that could have led to injury or property damage but was avoided through quick reactions or evasive actions
- Involves conflicts between road users (vehicles, pedestrians, etc.) with collision potential
- Reveal underlying safety issues without causing harm
- Occur much more frequently than actual crashes

Heinrich's Safety Ratio

For every:

->1 major injury or fatal accident,

->29 minor injuries,

->300 no-injury incidents (near-misses or near-crashes)

Usefulness of Near-Crash Data Model Training & Corner Cases

APPLICATION	BENEFITS
Model Training for IV/AV Systems	• Larger, more varied dataset than real crashes • Covers high-risk but non-fatal incidents • Trains safety interventions (emergency braking, steering) • Covers wide range of driving conditions
Corner Case Identification	• Captures rare, dangerous scenarios • Enables simulation in virtual environments • Evaluates AV reactions to unpredictable situations • Critical for Level-5 autonomy development

Usefulness of Near-Crash Data

Model Training & Corner Cases

Application zzz

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Threats

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Applications of Near-Crash Data

Traffic Safety Research: Studying hazardous locations and improving road designs

Smart Transportation Systems: Enhancing real-time safety monitoring and traffic management

Autonomous Vehicles (AVs): Training AI, handling corner cases, improving reliability mmodo consequat.

Intelligent Vehicles (IVs): Evaluating onboard safety systems like collision warnings

Urban Planning: Supporting Vision Zero (goal of zero traffic fatalities)

Computer Vision/AI: Developing models to detect and classify risky behavior

DATA COLLECTION SOURCES

WHERE NEAR-CRASH DATA COMES FROM

Source	Characteristics	Examples
Roadside Sensors (Surveillance Cameras)	• Widely deployed in urban areas • Information-rich, captures detailed traffic interactions • Used in smart city initiatives	• City of Bellevue (with Microsoft Research & UW) • Large-scale video-based near-crash detection • Vision Zero initiatives
Vehicle Onboard Sensors (Dashcams)	• Used where roadside cameras unavailable • Act as mobile data collectors	• Pilot project evaluating Mobileye Shield+ • Detecting potential bus-pedestrian collisions

What the Paper Actually Uses:

- The system is based on onboard sensors:
- Dashcams installed on the ego-vehicle
- Data from the CAN bus (speed, braking, acceleration)
- All video processing and near-crash detection are done locally on an edge device (NVIDIA Jetson TX2) inside the vehicle

Key Challenges in Near-Crash Data Collection

Major Obstacles to Overcome

Rarity of Near-Crashes

- Still infrequent events, hard to collect large datasets
- Requires continuous high-resolution monitoring
- Needs intensive computing power, high bandwidth, large storage

Inefficient Detection Methods

- Often relies on manual review of footage
- Post-event analysis is slow and labor-intensive
- Not scalable for real-time or city-wide applications

Cost and Scalability Issues

- Commercial systems like Mobileye Shield+ are expensive
- Not easily transferable across different vehicle types
- Limited scalability due to proprietary design

Why Edge?

The Problem:

- Not feasible to store all raw videos locally (storage limits)
- Not feasible to transmit all raw videos in real-time (bandwidth limits)
- Need for low-cost, transferable, real-time systems

Edge Computing Benefits:

- Processes data close to where it's generated (vehicles or roadside units)
- Reduces latency, bandwidth usage, and storage requirements
- Enhances privacy by keeping sensitive data local
- Enables faster response to potential crashes
- Allows scalable, cost-effective deployment

Privacy Protection:

- Video feeds contain sensitive data (faces, license plates)
- Edge processing keeps this data local when possible

How Edge Computing Solves Challenges

Addressing Each Problem

CHALLENGE	EDGE COMPUTING SOLUTION
Large video data volume	Edge devices process video locally, extracting only useful insights, not raw video
Limited bandwidth for transmission	Only relevant near-crash event data is transmitted, minimizing network load
Cloud computation overload	Offloading to the edge frees up the cloud, allowing scalable deployment
Real-time detection needed	Processing on the edge enables instantaneous detection and alerting
Cost and scalability concerns	Uses affordable dashcams + edge processors, making it scalable and retrofittable
Privacy concerns in video data	Analyzing and anonymizing data locally, sensitive raw data never needs to leave the device

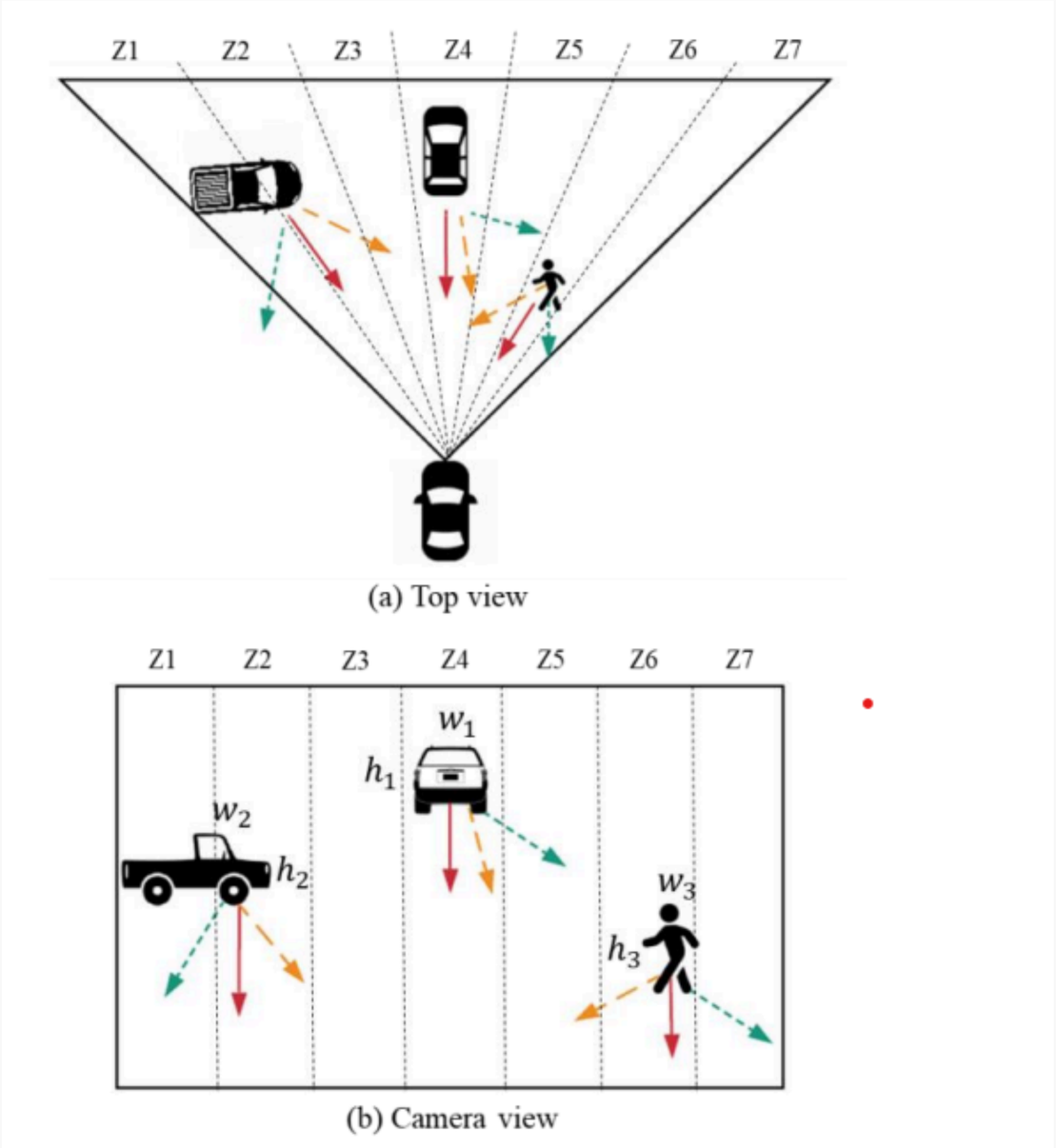
The Physics Behind Near-Crash Detection

Key Concepts:

- **Ego vehicle:** The main vehicle under observation (equipped with sensors/cameras)
- **Target road users**: All other entities that could potentially crash with the ego vehicle

Motion Patterns from Camera View:

- **Relative motion** towards ego-vehicle is a warning sign
- Objects moving toward the bottom center of the frame indicate potential danger
- Object size change helps identify threat level (growing = approaching)



Three Motion Categories:

- 🔴 Potential crashes – Object moves toward the center of the camera
- 🟠 Warning cases – Object is near the center or moving slightly toward it
- 🟢 Safe cases – Object moves away from the center line of sight

SYSTEM ARCHITECTURE

Main Thread (Crash Detection Core):

- Detects objects (SSD-Inception) → tracks (SORT) → generates bounding boxes
- Models width, height, center → estimates TTC and
- motion patterns
- Applies double-threshold rule → detects near-crash
- Triggers data transmission if crash likely

Frame Reading Thread:

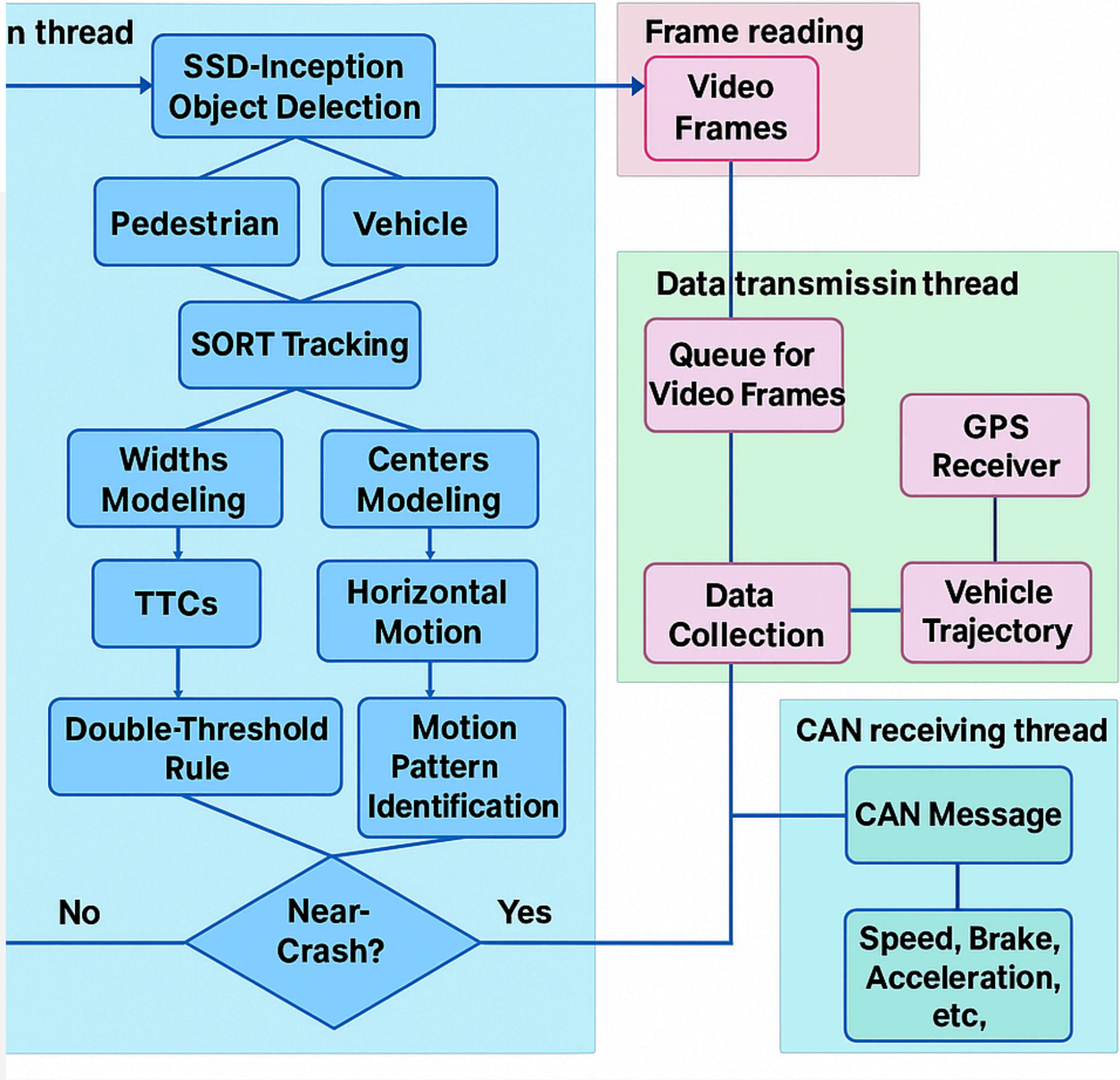
- Continuously fetches fresh video frames
- Maintains queue, drops outdated frames

Data Transmission Thread:

- Triggered by near-crash
- Collects: video frames, GPS trajectory, CAN data
- Sends for logging/analysis

CAN Receiving Thread:

- Independently captures vehicle data (speed, brake)
- Supports future system integrations



CRASH DETECTION ALGORITHM

01

DEEP-LEARNING-BASED ROAD USER DETECTION AND TRACKING

02

MODELING BOUNDING BOXES FOR TTC ESTIMATION

03

CRASH DETECTION ALGORITHM

04

BOUNDING BOX CENTER FOR HORIZONTAL MOTION DETECTION

01.Deep-Learning-Based Road User Detection and Tracking

1. Object Detection with SSD-Inception
   - Analyzes every video frame using deep learning
   - Detects and classifies objects (cars, pedestrians)
   - Runs efficiently at ~30 FPS on NVIDIA Jetson TX2
   - Output: Bounding boxes around detected road users

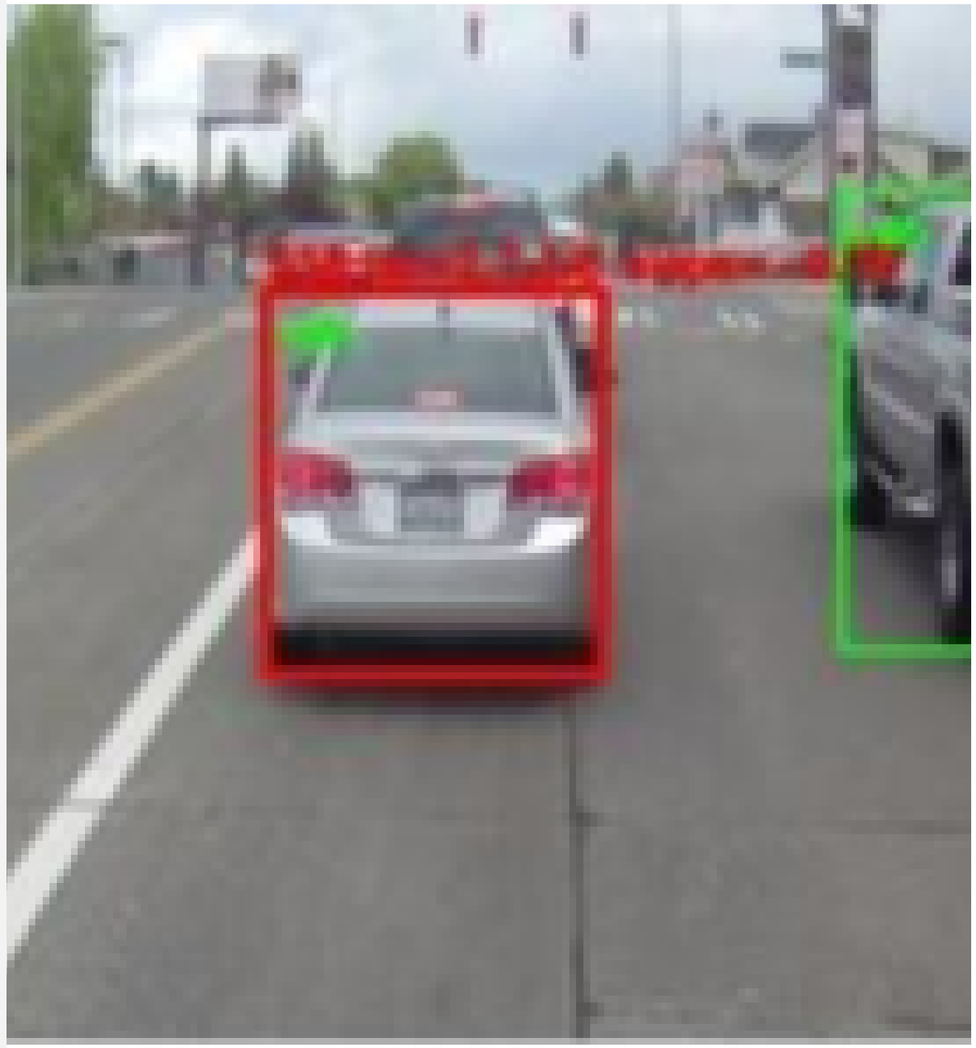
2. Object Tracking with SORT Algorithm
   - Follows objects across frames using bounding box data
   - Reduces detection errors (false positives/negatives)
   - Understands motion over time relative to the vehicle

Why This Approach Is Special:
   - Camera-parameter-free: No calibration required
   - Real-time: Processes video fast enough for instant reactions
   - Low-power edge-compatible: Efficient on devices like Jetson TX2

CHALLENGES & SOLUTIONS:

Challenge	Solution in Paper
High computational cost	SSD-Inception balances accuracy/speed
Detection inaccuracies	SORT tracking reduces noise
Inconsistent bounding boxes	Multi-frame analysis
Tracking errors	Bounding box overlap tracking

02.MODELING BOUNDING BOXES FOR TTC ESTIMATION



What Are Bounding Boxes?

- Rectangular boxes highlighting objects in video frames
- Contain: position (X,Y), size (width/height), class, confidence score

Camera-Independent TTC Estimation:

- As objects approach ego-vehicle, they appear larger in video
- Tracks size changes over time to predict collision risk
- No camera calibration needed - works with any dashcam

Challenge	Solution
Bounding box noise	Linear regression on multiple frames
Minimal size change between frames	Analyze longer frame sequences
Bounding box ≠ exact object size	Focus on relative change over time

CALCULATION OF TTC TIME-TO-COLLIDE

What Each Variable Means:

- s_t: Size of the object (e.g., car) in the video frame at time t (like width or height of the bounding box).
- r_t: Rate at which that size is changing over time (e.g., how fast the object appears to grow/shrink).
- D_t: Actual longitudinal distance from the ego vehicle to the object.
- V_t: Relative longitudinal speed between ego vehicle and object.
- S_t: Real-world size of the object (assumed constant, e.g., height of a pedestrian).
- f: Camera focal length (fixed).

Note: Based on the pinhole camera model

$$\frac{s_t}{f} = \frac{S_t}{D_t} \tag{1}$$

Relative speed is the first derivative of relative distance, and that size change rate is the first derivative of the object size over time

$$V_t = \frac{dD_t}{dt}, \qquad r_t = \frac{ds_t}{dt} \tag{2}$$

Since the real-world target road user's size does not change over time, there is the following equation

$$0 = \frac{dS_t}{dt} = \frac{d\left(\frac{D_t s_t}{f}\right)}{dt} \tag{3}$$

And since the focal length does not change over time, we have

$$0 = \frac{d(D_t s_t)}{dt} = \frac{dD_t}{dt} \cdot s_t + \frac{ds_t}{dt} \cdot D_t = V_t s_t + r_t D_t \tag{4}$$

Thus,

$$TTC = -\frac{D_t}{V_t} = \frac{s_t}{r_t} \tag{5}$$

Parameters that are used in the algorithm

Parameter	Suggested Value/Range	Purpose/Reason
SSD Confidence Threshold	0.3 – 0.5	Reduce false positives without missing real detections
Frames for Size Regression	10 – 15 frames	Smooth bounding box noise; assumes steady target motion
Frames for Center Regression	15 – 20 frames	Capture horizontal motion more effectively
TTC Threshold (δ)	~2 – 3 seconds	Used with height to detect near-crash events
TTC Threshold (φ)	2 – 2.5 × δ (~4 – 7.5 sec)	Used with width; confirms consistent motion
Horizontal Motion Threshold (α)	-1 to -0.5	Identifies warning zone movement
Horizontal Motion Threshold (β)	0.02 to 0.1	Identifies high-risk crash-prone movement
Jetson Power Mode	Max-N (preferred), Max-Q	Max-N: full performance; Max-Q: low-power mode

Keys:

- **Uses multiple frames (10-15) instead of just two**
- **Applies linear regression to smooth size changes**
- **Calculates TTC = size / size change rate**

Note :

TTC = s_t / r_t

Where:

s_t = Object size at time t

r_t = Rate of size change

TTC > 0: Object approaching (potential collision)

TTC < 0: Object moving away (safe)

Challenge	Solution
Bounding box noise	Linear regression on multiple frames
Minimal size change between frames	Analyze longer frame sequences
Bounding box ≠ exact object size	Focus on relative change over time

DOUBLE-THRESHOLD RULE FOR VALIDATION

The Problem with Single-Threshold TTC:

- Height increase alone can be misleading
- Objects moving sideways may appear to approach
- Need to distinguish true collision risk from false alarms

The Solution: Double-Threshold Rule

- Uses **both height and width change rates** to validate approach
- Prevents false alarms from objects just moving sideways

How It Works:

1. **Height Analysis:**
   - **Increasing height suggests approaching object**
   - **But could be misleading if object is crossing**
2. **Width Verification:**
   - **If width decreases while height increases → likely crossing (false alarm)**
   - **If both height and width increase → true approach (valid near-crash)**

Decision Logic:

- IF (height_change_rate > threshold_height) AND (width_change_rate > threshold_width)
- THEN → Valid near-crash event
- ELSE → False alarm (safe)

Benefits:

Reduces false positives by 40-60% in real-world tests

Maintains high detection rate for true collision risks

Computationally lightweight for edge devices

Experiment Design and Hardware

Experiment Design:

- Local video tests (stored videos on Jetson)
- Real-world tests (live video from cars and buses)
- Duration: Over 1000 hours of data collection
- Vehicles tested: 2 Honda cars, 4 Pierce Transit buses (2020-2021)

Hardware Components:

- Main processing: Nvidia Jetson TX2
- Dashcam (USB/IP camera)
- GPS receiver
- Arduino (auto-booting)
- PEAK CAN adapter
- Power inverter, shell, cables
- Internet switch and cloud server

TYPES OF DATA THAT ARE COLLECTED

Data Type	Source	Purpose
Video Frames	Dashcam	Visual analysis of near-crash context
Speed	CAN Bus	Understand driving behavior at the moment
Braking & Acceleration	CAN Bus	Measure reaction to potential collision
Timestamp	Edge device	Mark time of incident
Relative Motion	Edge device	Detect dangerous approach or trajectory
Event Metadata	Edge device	Save, label, and transmit relevant data only

Some common doubts

What is Real-Time?Why Real-Time Matters:

- System's ability to process data and respond immediately as events happen
- Prevention: Enables fast reaction (e.g., triggering alerts)
- Efficient data capture: Only save what's relevant—no delay
- Safety: In AVs or smart transport, milliseconds can mean life or death

Timestamp Importance:

- Records exact date and time when events occur
- Enables data alignment across multiple sources
- Essential for event tracking, post-analysis, ML training, and legal purposes

Why Not Use Custom Time Systems?

- No relation to real-world time
- Harder to sync across devices
- Not compatible with standard tools
- Loss of interoperability in multi-vehicle systems

Implementation Feasibility
Can This Be Deployed in Real Life?

1. Edge Hardware Is Readily Available
    - NVIDIA Jetson TX2 is commercially available
    - Handles video processing, deep learning inference, and real-time computation
2. Selective Data Transmission Is Realistic
    - Detects near-crash events locally
    - Only transmits data when a near-crash is detected
    - Minimizes bandwidth and storage usage
3. Common Components
    - Regular dashcams can be used as input
    - Most modern vehicles have accessible CAN bus data
4. Existing Real-World Use Cases
    - City of Bellevue project with Microsoft and UW
    - Mobileye Shield+ system on transit buses

Vehicle Integration

Fitting the System in Private Vehicles

Compact Components:

- All hardware is small and portable
- Can be installed inside dashboard, under seat, or in trunk

Power Supply Compatibility:

- System runs on 12V DC, compatible with most vehicles
- Uses DC inverter for power

Non-Intrusive Setup:

- Doesn't modify vehicle's core systems
- No compromise to warranty or safety

Installation Locations:

Component	Location Suggestion
Jetson TX2	Under front seat / glovebox / rear trunk
Dashcam	Behind rear-view mirror (standard spot)
Arduino circuit	Inside dashboard or control box
GPS antenna	On dashboard or rear windshield
Power inverter	Under dash / fuse box area

CHALLENGES & CONCERNS

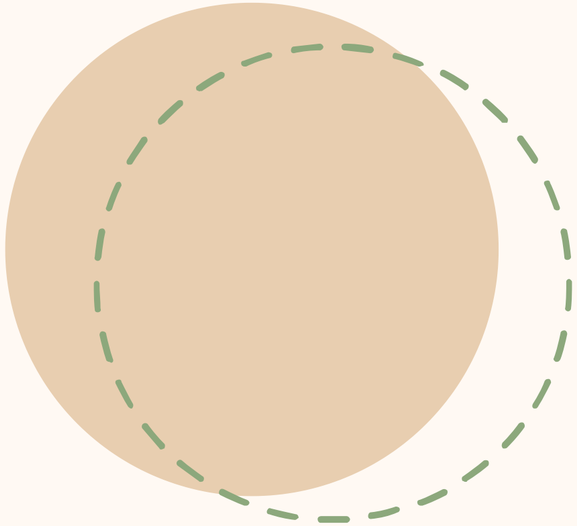
Technical Challenges:

- High-quality model calibration for various traffic scenes
- Data privacy and encryption during selective transmission
- Heat and power management in harsh environments
- Adaptation to different vehicle types and camera angles

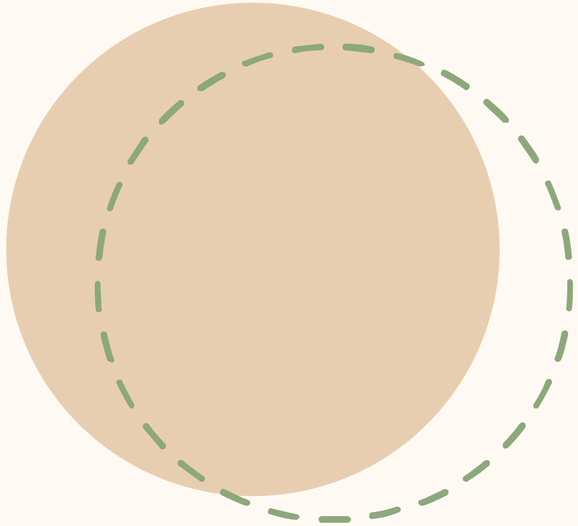
User/Company Concerns:

- Privacy Worries
  - System only stores relevant video clips
  - Data can remain local on edge device
  - No personal data transmitted unless necessary
- Manufacturer Restrictions
  - Some automakers restrict third-party hardware access
  - Solution: Use external video + GPS without tapping internal electronics
- User Trust
  - Must be packaged as safety enhancement, not surveillance
  - User interface or on/off control helps gain trust

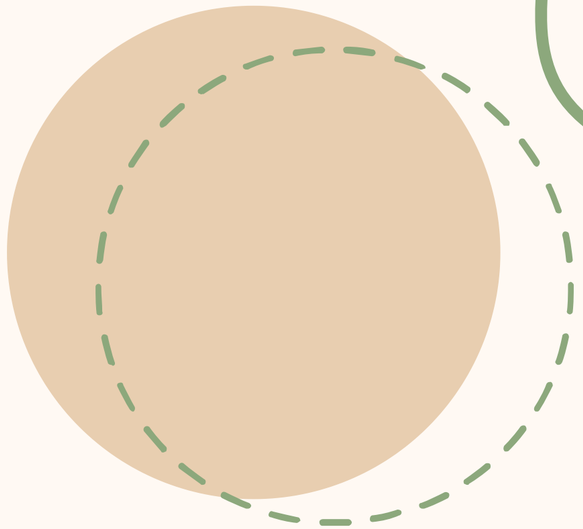
Our Team



Rufus Stewart



Juliana Silva



Samira Hadid