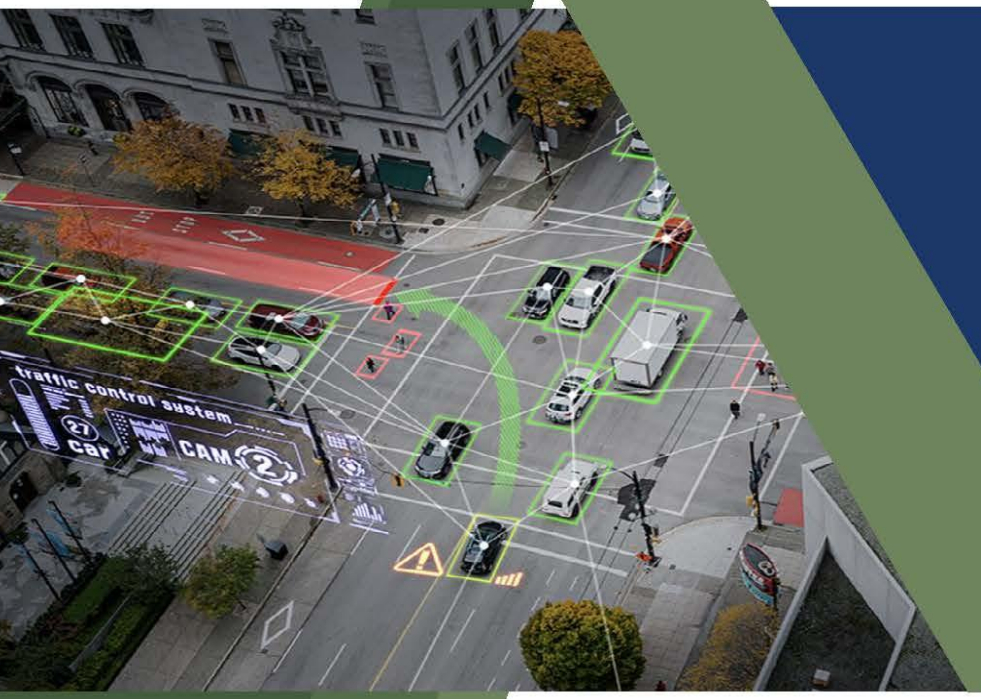


FINAL REPORT

PROJECT 06

DECEMBER 2023

Real-Time Safety Diagnosis for Connected Vehicles with Parallel Computing Architecture

Shuang Tu, Ph.D., Jackson State University
Robert W. Whalin, Ph.D., Jackson State University
Di Wu, Ph.D. Candidate, Jackson State University

STRIDE | Southeastern Transportation Research, Innovation, Development and Education Center
UF | Transportation Institute UNIVERSITY of FLORIDA

TECHNICAL REPORT DOCUMENTATION PAGE

1. Report No. Project O6	**2. Government Accession No.**	**3. Recipient's Catalog No.**
4. Title and Subtitle Real-time Safety Diagnosis System for Connected Vehicles with Parallel Computing Architecture		**5. Report Date** 12/11/2023 **6. Performing Organization Code**
7. Author(s) Shuang Tu, Ph.D., Jackson State University Robert W. Whalin, Ph.D., Jackson State University Di Wu, Ph.D. Candidate, Jackson State University		**8. Performing Organization Report No.** STRIDE Project O6
9. Performing Organization Name and Address Jackson State University Department of Computer Engineering P. O. Box 17098 Jackson State University 1400 J. R. Lynch Street Jackson, MS 39217		**10. Work Unit No.** **11. Contract or Grant No.** Funding Agreement Number 69A3551747104
12. Sponsoring Agency Name and Address University of Florida Transportation Institute/ Southeastern Transportation Research, Innovation, Development and Education Center (STRIDE) 365 Weil Hall, P.O. Box 116580 Gainesville, FL 32611 U.S Department of Transportation/Office of Research, Development & Tech 1200 New Jersey Avenue, SE, Washington, DC 20590		**13. Type of Report and Period Covered** 05/01/2022 to 12/11/2023 **14. Sponsoring Agency Code**

15. Supplementary Notes N/A

16. Abstract - The primary aim of this project is to enhance our system from the previous STRIDE F4 project to a parallel computing version. The original F4 system, designated as Automatic Safety Diagnosis in Connected Vehicle (CV) Environment, established a computational pipeline for diagnosing near-crash events exclusively using Basic Safety Messages (BSMs). It was implemented using a sequential computing paradigm. The O6 project was conceived to expedite the system by transitioning it to a parallel version. The F4 system comprised a driving anomaly detection model (DAD), a conflict identification model (CIM), and the data-path connecting them. The DAD was primarily situated in the core cloud, while the CIM was positioned within the CVs. Throughout the O6 process, notable advancements in in-vehicle computers (IVCs) were uncovered. In order to align our system with real-world operations, we opted to fully migrate the DAD component to the IVCs.Recognizing Domain-Specific Design (DSD) as the future of parallel computing, we propose configuring DSD for IVCs based on three levels of abstractions: selecting the appropriate chip architecture, programming language, and parallelism module. For the CIM of our system, we recommend utilizing ARM architecture, the C programming language, and leveraging the built-in parallelism of the ARM chip. As for the DAD, we advocate for a complete migration to IVC, utilizing ARM architecture, the Python language on the CPU, and employing multiprocessing for parallel computing.

17. Key Words parallel computing, connected vehicle, Python, C, ARM, OpenMP, in-vehicle computer, domain-specific design	**18. Distribution Statement** No restrictions

19. Security Classif. (of this report) N/A	**20. Security Classif. (of this page)** N/A	**21. No. of Pages** 46 pages	**22. Price** N/A

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

ACKNOWLEDGEMENT OF SPONSORSHIP AND STAKEHOLDERS

This work was sponsored by a grant from the Southeastern Transportation Research, Innovation, Development, and Education Center (STRIDE).

Funding Agreement Number - 69A3551747104

LIST OF AUTHORS

Lead PI:

Shuang Z. Tu, Ph.D.
Jackson State University
shuang.z.tu@jsums.edu
https://orcid.org/0000-0002-4506-6447

Co-PI:

Robert W. Whalin, Ph.D.
Jackson State University
Robert.w.whalin@jsums.edu
https://orcid.org/0000-0002-8712-9434

Additional Researchers:

Di Wu, Ph.D. Candidate
Jackson State University
di.wu@students.jsums.edu
https://orcid.org/0000-0003-3169-3041

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The primary aim of this project is to enhance our system from the previous STRIDE F4 project to a parallel computing version. The original F4 system, designated as Automatic Safety Diagnosis in Connected Vehicle (CV) Environment, established a computational pipeline for diagnosing near-crash events exclusively using Basic Safety Messages (BSMs). It was implemented using a sequential computing paradigm. The O6 project was conceived to expedite the system by transitioning it to a parallel version.

The F4 system comprised a driving anomaly detection model (DAD), a conflict identification model (CIM), and the data-path connecting them. The DAD was primarily situated in the core cloud, while the CIM was positioned within the CVs. Throughout the O6 process, notable advancements in in-vehicle computers (IVCs) were uncovered. In order to align our system with real-world operations, we opted to fully migrate the DAD component to the IVCs.

Recognizing Domain-Specific Design (DSD) as the future of parallel computing, we propose configuring DSD for IVCs based on three levels of abstractions: selecting the appropriate chip architecture, programming language, and parallelism module. For the CIM of our system, we recommend utilizing ARM architecture, the C programming language, and leveraging the built-in parallelism of the ARM chip. As for the DAD, we advocate for a complete migration to IVC, utilizing ARM architecture, the Python language on the CPU, and employing multiprocessing for parallel computing.

Keywords:

parallel computing, connected vehicle, Python, C, ARM, OpenMP, in-vehicle computer, Domain-Specific Design.

EXECUTIVE SUMMARY

The aim of this project is to enhance the system from the prior STRIDE F4 project to a parallel computing version. The F4 system, titled as Automatic Safety Diagnosis in the Connected Vehicle Environment, aimed to establish a computational pipeline for diagnosing near-crash events by processing BSMs generated within the CV environment. The F4 system architecture included two components: the DAD, primarily situated in the core cloud, and the CIM, located within the CVs.

A near-crash event was defined as a situation meeting two conditions: (a) the presence of a conflict and (b) at least one of the drivers exhibiting abnormal driving status. The original F4 project utilized a sequential computing paradigm. However, with the growing market penetration of connected vehicles, the demand for faster data processing and transmission has increased, necessitating the adoption of parallel computing.

To initiate the project, an extensive study explored literature and real-world applications of parallel computing. This research unveiled significant advancements in IVCs in recent years and emphasized DSD as the future of parallel computing.

To align our system with real-world operations, we adjusted the system architecture and fully migrated the DAD to the IVC. This required significant effort to determine the appropriate configuration for parallel computing on the IVC. We decided to configure DSD on three levels of abstraction: chip architecture, programming language, and the parallelism module. Based on system performance, we recommend utilizing ARM architecture, C programming language, and leveraging the built-in parallelism of the ARM chip for CIM. For DAD, we suggest employing ARM architecture, Python language on the CPU, and utilizing multiprocessing for parallel computing.

During testing, the O6 system underwent evaluation using various programming languages, including C, Python, and OpenMP, on both Windows and MacOS platforms, specifically with the Apple M1 chip. The testing dataset included BSM data from connected vehicle pilot studies, and system performance was also assessed using the SHARPII naturalistic driving study crash data.

Additionally, to gauge the applicability and effectiveness of our DAD, comparison tests were conducted on selected major Machine Learning (ML) packages for Object Detection (OD) using our working data. The results revealed that these packages were unable to meet our system's requirements.

1.0 INTRODUCTION

Traffic accidents contribute significantly to traffic congestion and travel delays. Issuing a warning message to drivers as a hazardous situation approaching can prompt them to take necessary maneuvers and prevent accidents. Our previous STRIDE project, F4—Automatic Safety Diagnosis in Connected Vehicle Environment—established a cloud-based system capable of delivering timely accident warnings within the CV environment.

As the automotive industry progresses in vehicle connectivity and automation, the distribution of computing tasks among central hubs, roadside infrastructure, and mobile units becomes a critical consideration. The Intelligent Transportation System (ITS) is evolving with the widespread use of CVs, leading to the generation of massive data. The safety diagnosis application must have the capacity to process this Big Data effectively. The adoption of parallel computing technology to expedite data processing and analysis is, therefore, an unavoidable necessity.

The O6 research was initiated to implement parallel computing in both the cloud and the in-vehicle subsystem. This approach aims to enhance the system's capability to handle the substantial data generated by CVs and ensure efficient safety diagnosis in real-time scenarios.

1.1 Objective

The goal of this research is to transition our preceding STRIDE F4 study from a sequential version to a parallel version through the incorporation of cutting-edge parallel computing techniques.

1.2 Scope

The computational pipeline represents an automatic safety diagnosis system in the Connected Vehicle CV environment. In Project F4, the system comprised the DAD in the cloud, edge computing, all CVs under its surveillance, and the datapath connecting them. Figure 1 illustrates the concept of the ASDSCE. The datapath involves communication between the vehicle and the cloud, posing an open research problem and presenting a significant challenge in CV research. However, it's important to note that the datapath is beyond the study scope of this project.

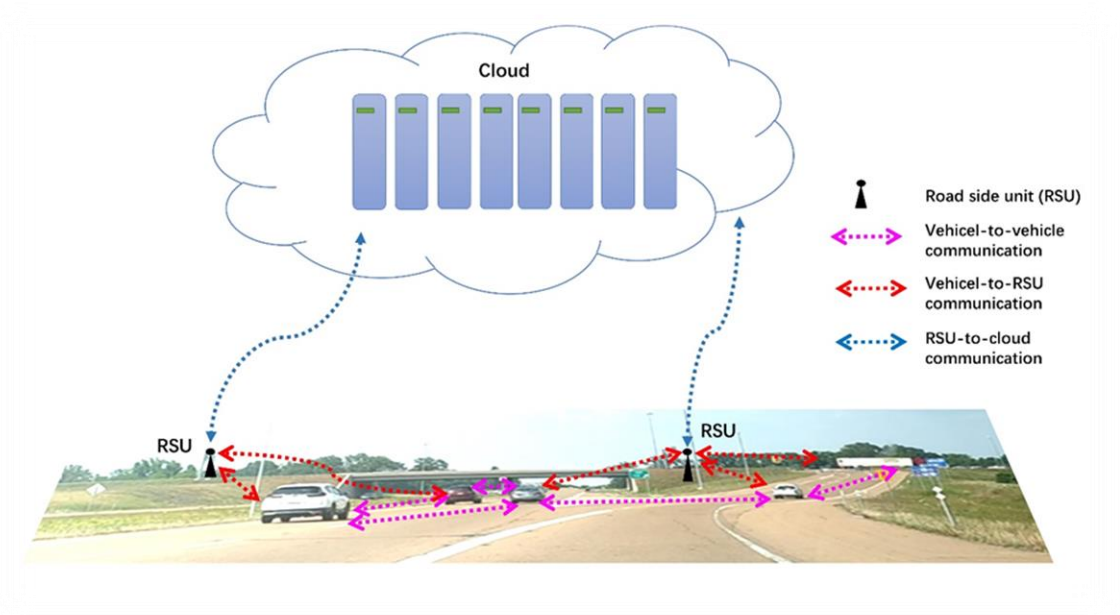


FIGURE 1. THE CONCEPT OF THE ASDSCE

1.3 The Practical Significance of This Study

In a traffic safety diagnosis system, parallel computing is essential to guarantee real-time processing and analysis of data. This research has the potential to elevate the technological capabilities of our system to align with the requirements of future modern transportation systems.

1.4 The Expected Final Products

The final product is a parallelized version of the computational pipeline for the automatic safety diagnosing system. This includes the incorporation of a MySQL database for handling working data, parallel computing utilizing OpenCL in the cloud subsystem, and parallel computing with OpenMP in the in-vehicle subsystem. The outcomes encompass a software package, a user's guide, instructional videos and webinars, publications, a final research report, and support for a PhD student.

2.0 LITERATURE REVIEW

2.1 A Glance of Transportation Big Data Analytics

Decades ago, the focus of transportation shifted from infrastructure expansion to operational efficiency and sustainability. ITS brought forth cutting-edge technologies in information systems, electronics, control, communications, sensing, robotics, and more,

evolving into a global phenomenon crucial for economic and social development (Lin, et al., 2017).The widespread deployment of Global Positioning System (GPS), sensors, CVs and other sources has resulted in the generation of Big Data in transportation, reaching the scale of Petabytes and Terabytes, with projections for continued growth in volume, speed, and complexity (Lin, et al., 2017).This massive influx of Big Data from ITS and CVs necessitates a well-designed computing architecture to support Quality-of-Service (QoS), given the diverse requirements of different types of applications (Wang, et al., 2020).

In the realm of Big Data analytics, computation architectures vary based on the selection of data storage, compression, and processing tools from a pool of options, such as the Hadoop Distributed File System (HDFS), Relational Databases, Apache Parquet, Spark, and more. For instance, a platform with multiple engines was proposed to support various types of traffic data (Mian, et al., 2014). Godzilla introduced a conceptual architecture for real-time traffic data processing, employing a multi-cluster approach to handle substantial data under various workloads and user numbers (Shtern, et al., 2014).. Kafka, a state-of-the-art Big Data tool, was utilized for building data pipelines and stream processing to manage real-time traffic data (Amini, et al., 2017).. Sipresk proposed a platform to process urban transportation data (Khazaei, et al., 2016). A comprehensive review of computing architectures for Big Data of CVs can be found in the paper by Wang (Wang, et al., 2020).

Despite these accomplishments, widely used parallelization algorithms for Big Data, such as peer-to-peer networks, MapReduce, and Spark platforms, have been reported to face significant issues related to speed-up, throughput, and scalability (An, et al., 2011). To address dynamic resource allocation, Big Data workloads were designed to be malleable and task oriented.

2.2 Parallel Computing

Parallel computing, a computational paradigm introduced in the late 20th century, involves the simultaneous execution of numerous calculations or processes to enhance processing speed and problem-solving capabilities (Gottlieb & Almsi, 1989). Positioned as the pinnacle of computing, parallel computing has been extensively applied to computationally intensive problems in science and engineering (Culler, et al., 1999).

The progression of computers from vacuum tubes in the 1950s to present-day nano-scale microchips with Very Large-Scale Integration (VLSI) has been remarkable. These microchips are now ubiquitous in personal computers, mobile devices, control systems, the internet, clouds, clusters, and high-performance computers. Over time, computers have become more user-friendly, evolving from a realm accessible to a few geniuses in the 1950s to highly trained individuals in the 1960s and 1970s, and finally, to almost

anyone since the 1980s. Concurrently, there has been a substantial shift in computer architecture from single processors to parallel processors (pheatt2008intel). This evolution is the result of collaborative efforts across the industry, involving vendors, programmers, and users working on hardware, architecture, algorithms, languages, and applications. Presently, parallel programming has become crucial for further advancements in computing (Stoller, et al., 2019) (Asanovic, et al., 2009).

The objective of computing improvement has been to achieve higher speedup while balancing factors such as cost, heat dissipation, and energy consumption (Moore, 1998). Higher clock speeds translate to faster CPUs, and increased transistor counts result in greater computing power. From the 1950s to the 1970s, significant improvements were made in chip technology, an ongoing trend (Saidu, 2015). Gordon Moore's 1965 prediction, known as Moore's Law, foresaw a doubling of the number of transistors on a microchip every two years. While this law accurately anticipated the consistent growth in transistor density, it is expected to face challenges. The size of a silicon atom being approximately 0.2 nm, the density of transistors on a chip cannot indefinitely increase. Additionally, with fundamental challenges related to heat dissipation and energy consumption, clock rates above 4.0 GHz are considered unsafe (Null & Lobur, 2014), rendering the method of frequency scaling less effective. As a result, further improvements in physical computer builds are likely to slow down unless new materials replace silicon or chip processing technology is updated with new approaches such as quantum computing or molecular computers.

With Moore's Law approaching its limits, computer manufacturers are left with limited options for performance improvements on chips or processors, except for distributing the computation load among several processors using parallel computing (Schauer, 2008). Parallel computing involves using multiple compute resources simultaneously to solve computational problems (Gottlieb & Almsi, 1989). Originating in the early 1950s, parallel computing was initially considered high-end, defense-oriented, and particularly featured for supercomputers. However, after the Cold War in the 1990s, when financial funding for defense decreased, parallel computing faced potential relegation. At this juncture, the high-performance computing community aimed to simplify the writing of parallel applications, realizing that wider user adoption could stimulate industry growth (Osuna, 1994). Fortunately, the crisis was overcome with the widespread adoption of parallel architectures, making parallel and distributed computing fundamental for computing professionals. Today, parallel computing is more prevalent than ever, with expectations of innovative advancements, particularly in applications related to artificial intelligence (AI) and ML, which demand intensive computation. In 2009, after systematically studying the landscape of parallel computing, Krste Asanovic asserted that "the parallel computing industry needs help from research to succeed in its recent dramatic shift to parallel computing" (Asanovic, et al., 2009).

Parallel computers are categorized by build into symmetric multiprocessor parallel computers, multicore parallel computers, distributed parallel computers, cluster parallel computers, massively parallel computers, and grid computers. In terms of processor-memory architecture, parallel computers are categorized into shared memory architecture (SMA) and distributed memory architecture (DMA). SMA involves building parallel computers from the combination of multiple microprocessors connected via specialized high-speed buses. On the cutting-edge, the Apple M1 chip realizes the system on a chip (SoC). DMA involves constructing parallel computers from multiple computers connected via a network such as an Ethernet network.

In terms of forms, computing parallelism can exist at three levels: bit-level, instruction-level, and data-level. Bit-level parallelism is based on the processor's size. Over time, processor word sizes have increased from 4-bit microprocessors to 8-bit, then 16-bit, 32-bit, and, in 1996, the introduction of 64-bit architectures that remain mainstream today. Larger processor word sizes reduce the number of instructions needed to perform tasks on large-sized data, enhancing overall performance. Instruction-level parallelism (ILP) involves optimizing microprocessor architecture. The CPU, the core of a computer, consists of an arithmetic–logic unit, processor registers, and a control unit. Initially, these operations were completed sequentially. By breaking instructions into stages and allowing a thread to run on stages of multiple instructions in parallel within the same clock cycle, the pipelining technique implemented ILP. A microprocessor with an n-stages pipeline can deliver n-times performance over a non-pipelined architecture (Saidu, 2015) (Saidu, 2015).. The number of pipeline stages, however, cannot be increased endlessly due to control dependencies and data dependencies. The amount of ILP in a program is highly application-specific (Hernandez, 2009). (Hernandez, 2009). Data-level parallelism (DLP) involves parallelization across multiple processors to achieve higher throughput. DLP encompasses data parallelism and task parallelism. Data parallelism includes single program multiple data (SPMD), vector processing, and single instruction multiple thread (SIMT) (e.g., with GPUs). Task parallelism decomposes a task into subtasks and allocates each subtask to multiple processors for concurrent execution. The amount of parallelism achievable is program-specific, requiring some control over execution patterns and resource allocation to ensure efficient execution. Reconciling these two conflicting requirements is the goal of parallel computing systems.

A fundamental requirement for any parallel programming system is to support abstraction, relieving users of the low-level complexities of parallel programming to work with familiar concepts from their own domain (Darlington, 1996). However, the achievable level of parallelism is highly program specific. Some control over execution patterns and resource allocation is still necessary to ensure efficient execution. Reconciling these two conflicting requirements remains the goal of parallel computing systems.

2.3 DSD and IVC

DSD has been identified as a pivotal future direction for parallel computing. The 2019 National Science Foundation (NSF) Workshop on Future Directions for Parallel and Distributed Computing emphasized the centrality of parallel and distributed computing in computational innovation. It advocated for exploiting specialized hardware accelerators and adopting computational platforms through DSD to enhance performance. The overarching goal of DSD is to develop comprehensive algorithms-software-hardware solutions that optimally align with the objectives of a given domain. For instance, in the case of convolutional neural networks (CNN), there was consensus on proliferating tensor processing units (ISCA, 2017), GPU tensor cores, new CPU instructions, new CNN chips for hardware, and incorporating frameworks built on common libraries such as PyTorch, Tensorflow, Horovod, etc., for software.

However, the challenge lies in building interfaces that embody appropriate abstractions for specific domains, posing difficulties at both the technological and industry levels. As the potential benefits of DSD are substantial, achieving a balance becomes more challenging due to potential disruptions caused by innovations. Any perturbation to the ecosystem—whether in applications, compilers, operating systems, or hardware—tends to have cascading effects, leading to an "inherent reluctance to change" throughout the industry (Stoller, et al., 2019). In this context, a pilot study that involves developing a DSD on a system in a formative stage could be instrumental, with In-Vehicle Computers (IVCs) serving as a suitable testbed.

IVCs, designed to withstand harsh vehicle environments, including shocks, vibrations, extreme temperatures, and electromagnetic interferences, have become a crucial component with the rise of vehicle telematics and camera-based surveillance systems. The global IVC market is estimated to reach 1.65 billion in 2029, offering hardware and software solutions for various automotive applications. Top players in this market include S&T AG, Lanner Electronics Inc. (Taiwan), Axiomtek (Taiwan), SINTRONES Technology Corporation (Taiwan), NEXCOM International (Taiwan), IBASE Technology Inc. (Taiwan), Acrosser (Taiwan), and Premio Inc (US) (MarketsandMarkets™ Ltd., 2020). As the market penetration and level of automation increase, addressing how to integrate parallel computing into applications becomes imperative for CAVs.

Transportation engineers and data scientists, as users of IVCs, face a dilemma due to inconsistencies in computing algorithms-software-hardware. Traditionally, computer programs were written sequentially, and when transitioning to parallel computing, the common practice was to rewrite the code in languages like C or C++ that have APIs developed for parallel computing, such as MPI, OpenMP, OpenCL, and CUDA.

Python, with its productivity and extensive library support, has become the preferred programming language for many transportation data scientists, especially in machine learning for data analysis. However, rewriting Python code to C involves breaking down

functionalities encapsulated in Python libraries. The C version of the same algorithm might be much longer than its Python counterpart. Complicating matters further, popular frameworks like TensorFlow, often used in artificial intelligence and machine learning applications for traffic and in-vehicle systems, recommend Python as the language of choice. Despite the availability of open-source parallel computing libraries for Python, such as Multiprocessing and Dask, Python's performance is compromised as it is an interpreted language. This uncertainty leads data scientists to question the choice of parallel computing programming languages for transportation engineering problems.

Both C and Python, as general-purpose programming languages, have their pros and cons. C is simple, flexible, and machine-independent, while Python is easy to learn and features numerous libraries with built-in functions. C code is compiled directly to machine code, executed directly by the CPU, making it a low-level language close to machine. In contrast, Python code is first compiled to bytecode and then interpreted by a C program, making it a high-level language closer to humans. While focusing on the intricate aspects of basic C coding, the advantages of abstractions might be lost. Alternatively, assigning a C programmer to perform the rewriting could introduce different coding habits and conventions. A middle ground could involve keeping the sequential parts of Python and using C or C++ for heavy calculations, with libraries like Ctypes and cPython developed for this purpose.

Balancing the trade-off between productivity, portability, and performance poses a significant challenge. The direction parallel computation should take concerning data analysis on in-vehicle computers remains an open question. These issues are expected to be addressed in the context of DSD.

The literature suggests that, before the settlement of DSD, three levels of abstractions within the users' control significantly affect the performance of in-vehicle parallel computing: chip architecture, language, and parallelism module.

2.4 OD

In the field of data science, anomaly detection is also referred to as OD, denoting the identification of abnormal events in data, often termed outliers. Outliers represent data points that significantly deviate from the majority of the dataset. In ML programs, OD serves as an initial step in data cleaning. However, OD itself has evolved into a complex and challenging field with the development of ML algorithms.

ML algorithms are generally categorized into three fundamental types based on the availability of the dependent variable (or label) for the data under examination: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. Supervised ML is applied to data that includes labels, unsupervised ML is designed for data lacking

labels, and reinforcement ML, essentially an unsupervised variant, can learn from the environment over time to create labels.

OD is typically considered unsupervised since outliers are usually rare, leading to a lack of labels for the data (Boukerche, 2020). This inherent nature makes it challenging to define statistical and mathematical measures for deviation. Various packages of OD algorithms are available in different programming languages, each employing a unique method to measure deviation. Basic categories of unsupervised OD algorithms include Angle-Based OD (ABOD) (Kriegel, et al., 2008), Cluster-based Local Outlier Factor (CBLOF) (Duan, et al., 2009), Histogram-based OD (HBOS) (Putrada & Abdurohman, 2021), Isolation Forest (Xu, et al., 2017),, and K Nearest Neighbors (KNN) (Larose & Larose, 2014). For instance, in Python, the PyOD package summarizes various OD algorithms, featuring over forty algorithms and finding application in numerous academic and industrial settings, with over 10 million downloads (Zhao, et al., 2019).

Selecting the most suitable OD ML algorithm is challenging, as datasets may vary in dimensions and features, and users may have different interests. Different OD algorithms employ distinct methods of measuring deviation, making the algorithm selection a critical aspect of OD processing.

3.0 TASK 1: ARCHITECTURAL DESIGN

Both F4 and O6 function as a real-time near-crash warning tool at the individual level, exclusively using BSMs. They define a near-crash as a traffic situation meeting two conditions: firstly, at least one of the vehicles in a driver-vehicle unit (DVU) pair exhibits abnormal driving conditions, and secondly, a conflict is present. The system architecture of O6 was derived from F4.

3.1 The Architecture of the F4 System

The F4 project was structured as a two-tier hierarchical system, comprising a cloud-based DAD and a CIM in each CV. The DAD served as a multi-dimensional anomaly detection model, with Modules 1, 2, 3, and 5 of the DAD processed in the cloud and Module 4 in the IVC alongside the CIM. Illustrated in Figure 2, the F4 system collected and stored BSM from the covered vehicles in the cloud. It determined thresholds for selected key performance indicators (KPIs) and broadcasted these thresholds through BSMs. Within the IVC, as real-time BSMs streamed in, the device compared the new values of each KPI with the received thresholds to identify outliers. The outliers were then analyzed to ascertain if their combination warranted an anomaly event, triggering the transmission of abnormal flags. Periodically, the system assessed impact factors to update thresholds based on the significance of the outliers.

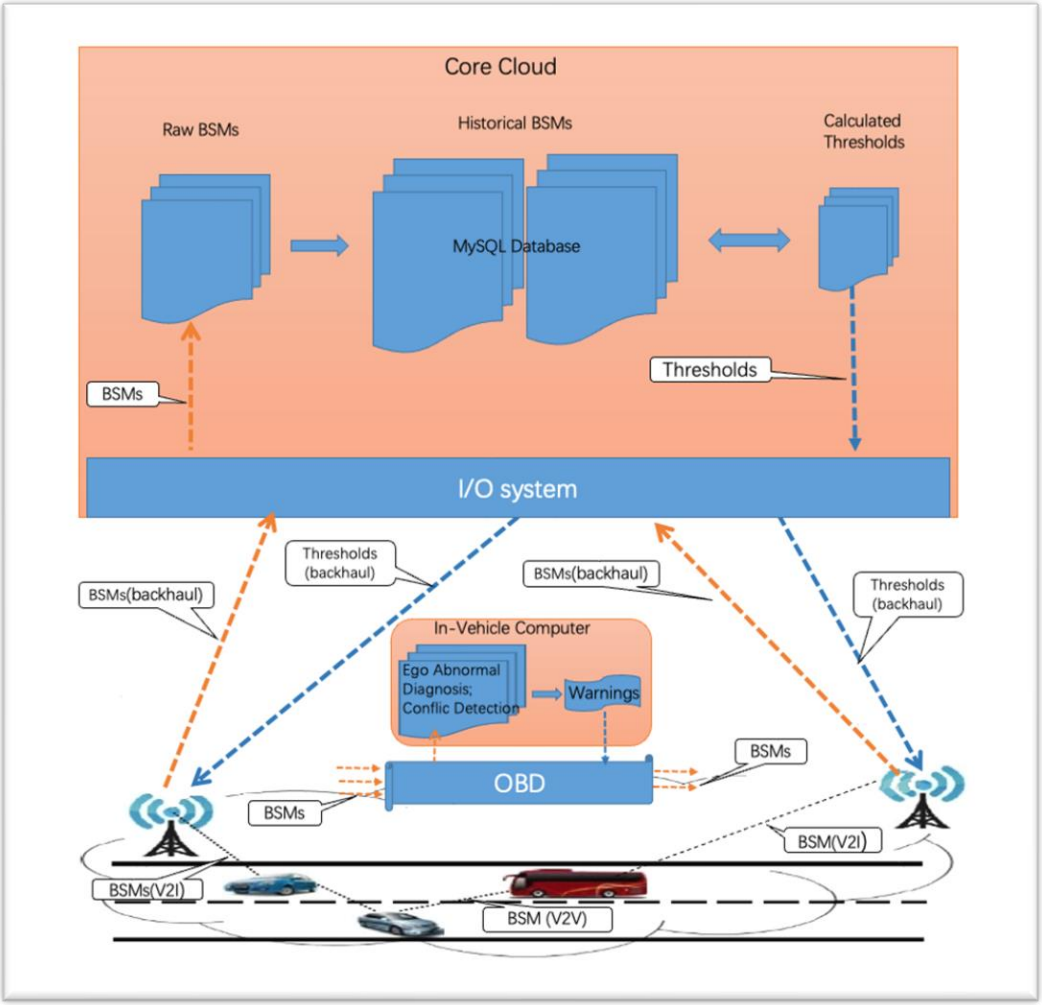


FIGURE 2.THE ARCHITECTURE OF THE F4 PROJECT.

The primary considerations for the F4 architecture were the substantial volume of Big Data from BSMs and the computational capacity limitations of in-vehicle computers. Energy conservation was also a significant consideration, with offloading presenting a substantial reduction in energy usage, particularly beneficial for electric vehicles amid global warming concerns. The architecture's advantages lay in central control of all CVs while keeping the in-vehicle computers lightweight. However, a drawback was the accumulation of massive historical BSMs in the cloud, leading to unnecessary data traffic.

In our O6 project, as we delved into the latest literature on computational platforms, we recognized that the progress in in-vehicle computers surpassed expectations. Consequently, we identified the need to update our system architecture to align with state-of-the-art parallel computing and CV technology.

3.2 The Architecture of the O6 System

The O6 system retains the two-tier architecture from the F4 project, comprising the top tier of the core cloud and the bottom tier of the ad-hoc vehicle cloud. In the core cloud, the flag list of abnormal CV IDs is stored and updated. DAD and CIM are processed in the IVCs. The IVCs broadcast and receive BSMs through On-board devices, forming the ad-hoc vehicle cloud, also known as vehicle-to-everything (V2X) BSMs. Figure 3 illustrates the architecture of the O6 system, which has been updated based on the advancements in IVC technology to meet the system's requirements.

The O6 system architecture brings improvements in performance, including reduced latency and substantial data traffic reduction. However, it is not without its drawbacks:

1. Data Loss in CV Malfunction:

   In the event of a CV malfunction within the O6 architecture, there is a risk of data loss, making it challenging to maintain the accuracy of the flag list. The system may struggle to assess abnormal CV situations accurately if malfunctions result in data loss, impacting the overall effectiveness of the safety diagnosis system.

2. Limitation for Future Development:

   As traffic safety requirements evolve to encompass factors such as roadway geometry, real-time traffic signal control, traffic flow, and travel demand analysis, the O6 architecture may encounter limitations. Integrating these additional elements for comprehensive traffic analysis might prove challenging within the confines of the O6 architecture.

Realizing the F4 system is a complex task that necessitates seamless collaboration between auto manufacturers, BSM central control, and government support. This cooperation is identified as a significant challenge for successful implementation. Concurrently, the prevailing trend in the automotive market emphasizes the shift towards flexible and lightweight CAVs. The O6 architecture, with its advantages in latency performance and reduced data traffic, aligns well with the current trend favoring lightweight solutions in the CAV market.

Given the intricacies of the transportation industry, addressing challenges related to potential data loss during CV malfunctions and adapting to evolving demands for comprehensive traffic analysis require thoughtful consideration and strategic planning. However, it's essential to note that addressing these broader challenges extends beyond the scope of the current project.

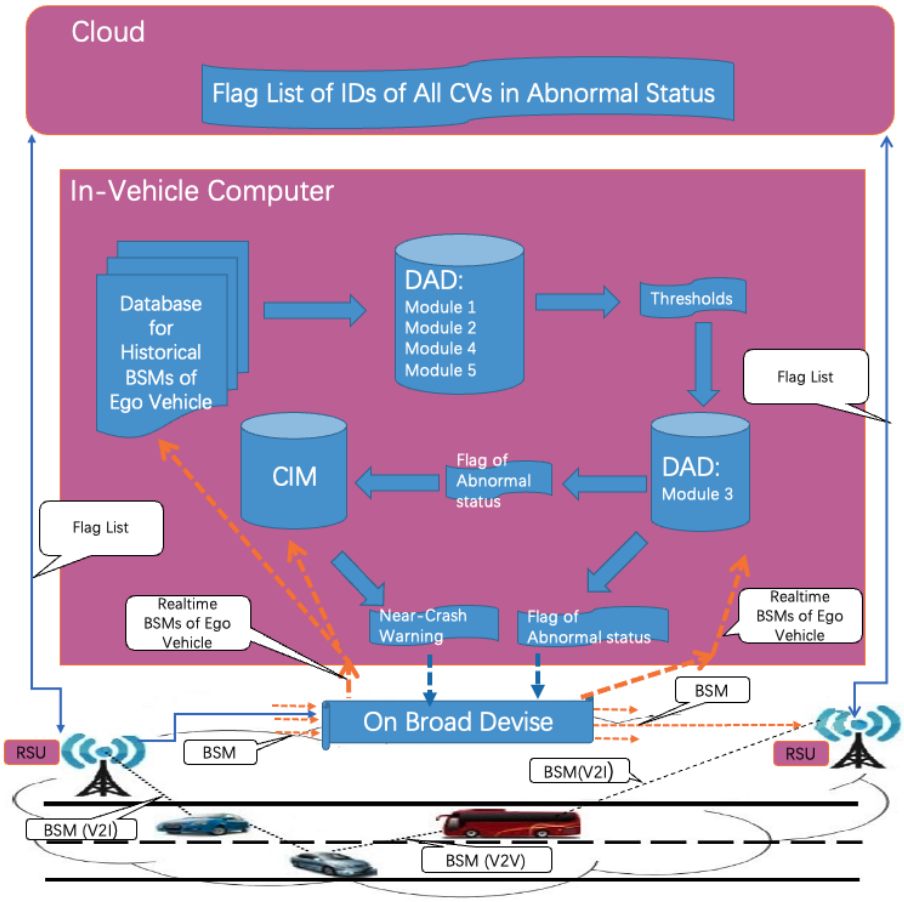


FIGURE 3. THE ARCHITECTURE OF THE O6 PROJECT.

4.0 TASK 2: DATABASE CONSTRUCTION

While the database plays a crucial role in our system, it is not the primary emphasis of the O6 Project. Fundamental stages of constructing the database were undertaken, encompassing tasks such as defining the database's purpose, segmenting the working data into tables, transforming the working data into columns, designating primary keys, establishing table relationships, and refining the database using normalization rules.

4.1 MySQL Database

A Database Management System (DBMS) serves as the repository for storing, accessing, modifying, and overseeing data, contributing to enhanced data integration, consistency, security, and efficiency. Numerous DBMS options exist in the market, including SQL, Oracle, MariaDB, MySQL, and PostgreSQL, with MySQL being notably popular. MySQL stands as a versatile relational DBMS owned by Oracle Corporation, operating as open-source software under the GNU General Public License, while also being available for proprietary licenses. Recognized for its widespread use, MySQL is established as a

preferred choice for various applications.

In our project, MySQL, as a relational database management system (RDBMS), was employed for handling BSMs. The installation of MySQL proved to be straightforward, and comprehensive documentation in the Oracle reference manual (Oracal, 2023) facilitated the process. For script development, the MySQL/Python Connector was generated, and database management was carried out using MySQL Workbench.

4.2 Database Schema

The database schema represents the logical configuration of a relational database, and for the BSM database, it was devised based on the characteristics of the working data and the requirements of the DAD. The working data for the O6 project consisted of the same BSM data collected during the F4 project.

BSM, a fundamental application of CVprograms, serves as the "Here I Am" data message. Originating from OBDs specifically designed for CVs, BSMs are broadcasted in the air at the dedicated 5.9 GHz spectrum with a frequency of 10 Hz (Henclewood, 2014). Nearby CVs and roadside units (RSUs) can receive these BSMs. The format of a BSM is defined by the Society of Automotive Engineers J2735: The Dedicated Short-Range Communications (DSRC) Message Set Dictionary. Typically, a BSM comprises two parts: the main part of the message, containing vehicle ID, epoch time, GPS location, speed, acceleration, yaw rate, and associated accuracy measurements; and the supplementary part providing additional information. Initially, BSMs were regarded as disposable and not intended for reuse.

The BSMs utilized in our project were part of the test data from the Safety Pilot Model Deployment (SPMD) project conducted in Ann Arbor, Michigan, in October 2012. These data were obtained from the ITS DataHub (its.dot.gov/data/). The BsmP1 file, formatted as Comma Separated Values (CSV), had a size of 67GB and stored all BSMs generated by the 1527 test vehicles during the test. The original downloaded data file contained 19 attributes and over 500 million records. During the data pre-processing phase of the F4 project, irrelevant attributes were filtered out, resulting in a data file with 11 attributes, including DevID for vehicle ID, EpochT for timestamp, and attributes for latitude, longitude, accelerations, heading, and yaw-rate. Descriptions of these attributes are detailed in Table 1.

TABLE 1. ATTRIBUTE LIST OF THE BSM DATA OF THE F4 PROJECT

Attributes Name	Type	Units	Description
DevID	Integer	None	Test vehicle ID assigned by the CV program
EpochT	Integer	seconds	Epoch time, the number of seconds since the January 1 of 1970 Greenwich Mean Time (GMT)
Latitude	Float	Degrees	Current latitude of the test vehicle
Longitude	Float	Degrees	Current longitude of the test vehicle
Elevation	Float	Meters	Current elevation of test vehicle according to GPS
Speed	Real	m/sec	Test vehicle speed
Heading	Real	Degrees	Test vehicle heading/direction
Ax	Real	m/sec^2	Longitudinal acceleration
Ay	Real	m/sec^2	Lateral acceleration
Az	Real	m/sec^2	Vertical acceleration
Yawrate	Real	Deg/sec	Vehicle yaw rate

For the O6 project, a table named ID_flag was established to store vehicle IDs along with corresponding driving status flags, utilizing ID as the primary key. Another table named BSM was created to house historical BSM data, encompassing vehicle ID, time, latitude, longitude, speed, and longitudinal and lateral accelerations. In the BSM table, DevID serves as the primary key. The attributes of these tables are detailed in Table 2 and Table 3. The ID in ID_flag and DevID in the BSM table function as foreign keys interchangeably. The entity-relationship (ER) diagram depicting these tables is presented in Figure 4.

To facilitate data insertion into the database, Oracle's standardized API, MySQL Connector/Python, was employed. Additionally, MySQL Workbench, an all-encompassing visual tool catering to data modeling, SQL development, and administration, was utilized for generating the ER diagram and managing the data.

TABLE 2. DESCRIPTION OF TABLE ID_FLAG OF O6

Attributes Name	Type	key	Units	Description
ID	Integer	PRI	None	Test vehicle ID assigned by the CV program
Flag	Integer		None	Epoch time, the number of seconds since the January 1 of 1970 Greenwich Mean Time (GMT)

TABLE 3. DESCRIPTION OF TABLE BSM OF O6

Attributes Name	Type	key	Units	Description

DevID	Integer	PRI	None	Test vehicle ID assigned by the CV program
EpochT	Integer		seconds	Epoch time, the number of seconds since the January 1 of 1970 Greenwich Mean Time (GMT)
Latitude	Float		Degrees	Current latitude of the test vehicle
Longitude	Float		Degrees	Current longitude of the test vehicle
Speed	Real		m/sec	Vehicle speed
AccX	Real		m/sec^2	Longitudinal acceleration
AccY	Real		m/sec^2	Lateral acceleration

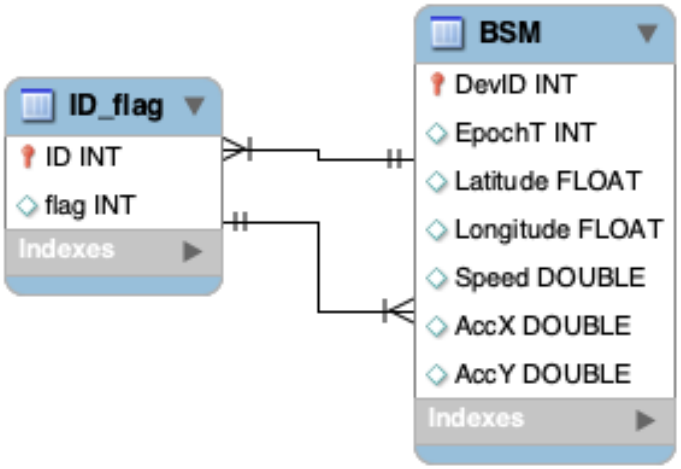


FIGURE 4. ER DIAGRAM OF THE DATABASE.

5.0 TASK 3: PARALLEL COMPUTING IMPLEMENTATION

Similar to the F4 project, the O6 system triggers a near-crash warning when a conflict arises between the ego CV and a neighboring CV, provided that any CV in the pair exhibits abnormal driving behavior. In the updated O6 architecture, both the DAD and CIM are executed within the IVC, leaving only the flag list of abnormal CVs stored in the core cloud. Consequently, the implementation of parallel computing is bifurcated into two distinct components: the CIM within the IVC, where the IVC serves as the primary hardware for parallel computing tasks of both DAD and CIM. While DAD involves predominantly offline processing, CIM demands real-time computations and places a higher emphasis on computing speed. Hence, the careful selection of a parallel computing platform for the IVC emerges as a pivotal consideration for the success of the O6 system.

5.1 Parallel Computing of CIM

In the O6 system, once a CV's engine starts running, its Collision Impact Mitigation (CIM) module becomes operational and examines the flag list containing identification numbers of CVs identified with abnormal driving status. Upon receiving BSM of a new CV_B, CV_A checks the ID of CV_B to determine if CV_B is listed in the flag list. If either

CV_A or CV_B is found on the list, the CIM proceeds to assess whether the CV pair (CV_A and CV_B) warrants a conflict. This process occurs at the same frequency as BSM generation and is applied to all CVs.

For the CIM to effectively operate, it must have the capacity to process the maximum number of BSMs generated by nearby CVs. Given that BSMs are generated at a frequency of 10 Hz, the CIM risks overload if the entire computing time for one BSM per CV exceeds 0.1 seconds. Therefore, our research goal was to identify the optimal computational setup for the CIM, balancing capacity and execution speed, while considering factors such as market availability, energy consumption, and the global trends in computing technology.

5.1.1 Test Data

As mentioned earlier, the test data utilized in this project were obtained from the Naturalistic Driving Study (NDS) conducted for the second Strategic Highway Research Program (SHRP 2). The NDS is a research initiative aimed at understanding the influence of driver performance and behavior on traffic safety. The Virginia Tech Transportation Institute (VTTI) serves as the technical coordination and study design contractor for the NDS and manages the InSight Data Access Website (Jafari, 2017). A sample dataset is presented in Table 4:

TABLE 4. SAMPLE INPUT DATA OF PROCESSED BSMS

vtti_timestamp	vtti.file_id	vtti.speed_network	x_position	y_position	x_ego	y_ego
199500	18539287	3	408	-1073	292	-1067
113400	44909777	0	0	0	73	-171
17000	44909777	0	0	0	73	-171
10567500	41894439	46	14779	27624	14665	27464
9783800	26026997	34	232345	39413	232490	39364
1027000	39534577	32	16996	-1145	16995	-1419
1871800	61805034	0	-9754	7404	-9771	7386
6000	44909777	0	0	0	73	-171
2324200	55152798	0	-32326	-9433	-32252	-9205

The maximum number of CVs were estimated using counting the CVs in the roadway network in the effective range of BSMs. Considering in the most congested condition, suppose the effective range of BSM is 1000 meters in radio, the area it covers3,140,000m^2. In the condition of high density of road network, the road grids are of the size of 300 meters long, so every gird covers 300 *300 = 9,000m^2 and 90000 m^2 and can have road of 600 meters long. Therefore, the maximum road length in the effective range is about to be 21,000 m, as in Equation (1).

$$3140000/90000 * = 21000\ (m) \qquad (1)$$

Each CV occupy a street length of 40 feet /12 meters (20 feet for vehicle length and another 20 feet for safety spacing). Assuming all the roads are 4-lane road, the maximum number vehicle around 7000 CVs, as calculated in Equation (2).

$$21{,}000/12 * 4 = 7000\ (\text{CVs}) \qquad (2)$$

Based on the assumptions, as shown in Figure 5, when the CV penetration rate reaches 0.14, in the most congested scenario the CIM on a single thread will be overloaded and experience malfunction.

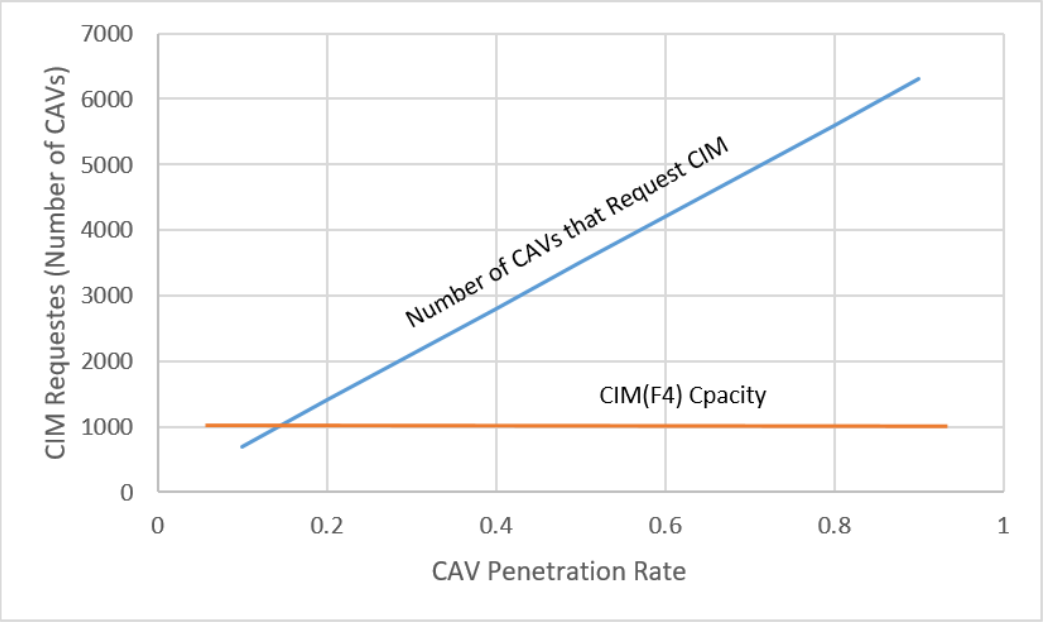


FIGURE 5. THE RELATIONSHIP OF CIM CAPACITY (SEQUENTIAL) AND VARIOUS CV MARKET PENETRATION RATES.

To assess the consistency of scenario performance, we generated 70 input files simulating varying numbers of Connected Vehicles (CVs) within the effective Basic Safety Message (BSM) range, ranging from 100 CVs to 7000. Assuming that 10% of them carried an abnormal flag, triggering the CIM, we randomly selected 10 to 700 BSMs from the available BSM data to form the 70 input files. Taking into account a 25% capacity reserve, the runtime was set to be less than 0.075 seconds.

For testing purposes, a MacBook Pro and a NUC were chosen as hardware platforms. The MacBook Pro was equipped with an Apple M1 Pro chip featuring 10 cores, 32GB memory, and macOS Ventura 13.1, used for testing on the ARM_64 architecture. The NUC, equipped with an Intel chip boasting 7 cores,

8GB memory, and running Windows 11, was utilized for testing on a different architecture. All codes were executed in the Visual Studio Code IDE version 1.74.2. The compiler used for Mac was Apple clang version 14.0.0 (Target: arm64-apple-darwin22.2.0), and for the NUC, Ming64 was employed. Python version 3.9 was used.

When the number of CVs exceeded 1000 (resulting in a CV penetration rate exceeding 0.14, as shown in Appendix C – Sample Results), the capacity of the NUC was no longer sufficient. On ARM_64, performance issues only occurred with Pandas sequential when the number of CVs exceeded 2500 (CV penetration rate exceeded 0.25, as shown in the figure), and with Pandas multiprocessing when the number of CVs exceeded 3500 (CV penetration rate exceeded 0.5, as shown in Figure 5).

5.1.2 Scenario Configuration

As mentioned earlier, the performance of IVC is influenced by factors such as chip architecture, programming language, and the parallelism module, all within the users' control range. Accordingly, testing scenarios were configured based on different selections of these abstractions. For chip architecture, the primary types for PCs and mobile devices currently include ARM_64 and X86_64. Regarding programming languages, Python and C were chosen, with Python being used for the sequential program of Collision Avoidance System (CIM) and C being known for its speed and widespread use.

Given the need to handle tabular data, particularly in CIM's sequential program, two widely-used Python libraries—Pandas (for data frames and series) and Numpy (for numerical data stored in arrays)—were selected for performance comparison. Numpy, known for its memory efficiency, enables C libraries to operate on the same memory. To explore the performance of Pandas versus Numpy, both were included in the scenarios.

For parallelism modules, Python's multiprocessing and C's OpenMP were included in the scenarios to leverage parallel programming capabilities. The testing scenarios are detailed in Table 5, with the aim of utilizing Python parallel programming libraries and extending heavy computations to C.

TABLE 5. SCENARIO SETUP OF IVC PARALLEL COMPUTING TESTS

Scenario	Chip Architecture	Language	Parallelism Module
1	ARM_64	Python: Pandas	None
2	ARM_64	Python: Pandas	Multiprocessing
3	ARM_64	Python: Numpy	None
4	ARM_64	Python: Numpy	Multiprocessing
5	ARM_64	C	None
6	ARM_64	C	OpenMP

7	X_86_64	Python: Pandas	None
8	X_86_64	Python: Pandas	Multiprocessing
9	X_86_64	Python: Numpy	None
10	X_86_64	Python: Numpy	Multiprocessing
11	X_86_64	C	None
12	X_86_64	C	OpenMP

5.1.3 Test Results

Excluding the scenarios deemed incapable (S1 and S2), as illustrated in 9.3 Appendix C – Sample Results, the candidate scenarios were refined to Scenario 3 through 6. Subsequent tests were conducted to ascertain the fastest scenario among the capable options. The results of 15 runs for scenarios 3 to 6 were averaged and presented in as Figure 7and Table 6. Notably, the outcomes reveal that Scenario 5 exhibited the shortest running time, indicating that employing C on ARM architecture represents the fastest hardware-software solution for the CIM model.

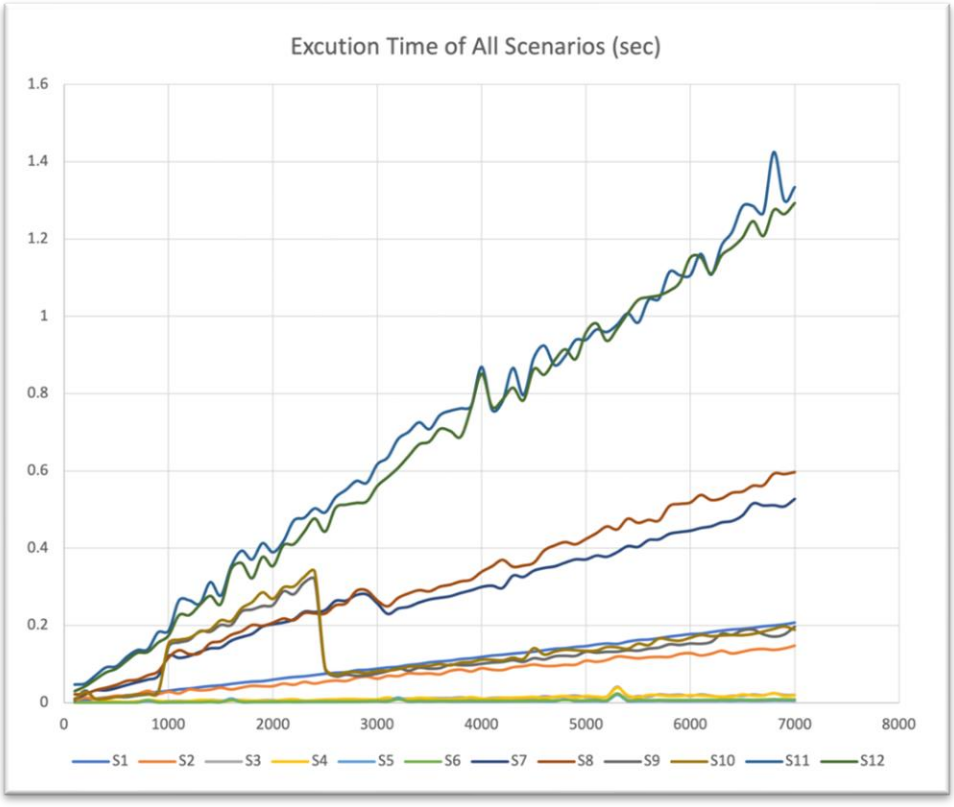


FIGURE 6. EXECUTION TIME OF ALL SCENARIOS.

FIGURE 7. AVERAGE EXECUTION TIME OF SCENARIO 3 TO 6.

TABLE 6. AVERAGE EXECUTION TIME OF SCENARIO 3 TO 6.

N_CV	S3_Numpy_Sequencial	S4_Numpy_Multithreading	S5_C_Sequencial	S6_C_OpenMP
100	0.00037618	0.00039701	0.00021555	0.00048691
200	0.0006278	0.00066274	0.00040401	0.00068768
300	0.00091147	0.00092614	0.00042075	0.00068067
400	0.00116693	0.00242577	0.00059257	0.00080706
500	0.00120098	0.00281097	0.00059716	0.00086295
600	0.00146801	0.00147554	0.00069116	0.00107881
700	0.00324532	0.00286123	0.00095795	0.00118876
800	0.00422285	0.0069939	0.00499573	0.00736597
900	0.00339616	0.00459367	0.00103703	0.00145898
1000	0.00375341	0.00239828	0.0012018	0.00193734
1100	0.00421241	0.0029459	0.00120867	0.00189303
1200	0.00412151	0.00370491	0.00151504	0.00201797
1300	0.00413102	0.00689872	0.00135786	0.00194823
1400	0.00709357	0.00636341	0.00229295	0.00293517
1500	0.00505778	0.00361713	0.00161853	0.00206482
1600	0.00659817	0.0060202	0.01086928	0.00784216
1700	0.00701656	0.0064436	0.00184089	0.00256166
1800	0.00386368	0.00505366	0.00177991	0.00270373
1900	0.00638371	0.00524933	0.00204767	0.00253048
2000	0.00642845	0.00453205	0.00185722	0.00300344
2100	0.00452328	0.00744322	0.00214489	0.00283408
2200	0.00877123	0.00895325	0.00207504	0.00334992
2300	0.00519767	0.0050756	0.00208824	0.00313733
2400	0.00523845	0.00619318	0.0022426	0.00350714
2500	0.00815277	0.00801045	0.00232366	0.00365728
2600	0.00814398	0.00828927	0.00231625	0.00373872
2700	0.00931892	0.00973814	0.00247366	0.0038118
2800	0.00910473	0.00882055	0.00259752	0.00375961
2900	0.00735723	0.00848298	0.00326271	0.00414349
3000	0.00811946	0.00851868	0.00399947	0.00490901

3100	0.00679641	0.01324968	0.0030122	0.00404466
3200	0.0099538	0.00861047	0.01322333	0.00875519
3300	0.01048598	0.0095562	0.00293357	0.00452545
3400	0.01119514	0.01259559	0.00383368	0.00522283
3500	0.01170475	0.01117643	0.00308412	0.0047664
3600	0.01174002	0.0087731	0.00424633	0.0051192
3700	0.01174207	0.00915731	0.00332821	0.00502855
3800	0.01365021	0.0106485	0.00338833	0.00508666
3900	0.01089614	0.01427571	0.00337915	0.00523122
4000	0.00950856	0.01040309	0.00349126	0.00533462
4100	0.01213601	0.01013807	0.00355094	0.00556459
4200	0.01278113	0.01237793	0.00362884	0.00548917
4300	0.01229172	0.0129449	0.00399906	0.00530767
4400	0.01071588	0.01375863	0.00396601	0.0052048
4500	0.01163241	0.01495471	0.00424627	0.00581616
4600	0.0167098	0.01179102	0.00395652	0.00624658
4700	0.01407057	0.01542074	0.00415324	0.00608387
4800	0.01624705	0.01727096	0.00902251	0.00764454
4900	0.01855477	0.01355395	0.00445479	0.00618682
5000	0.01434414	0.0172863	0.00462782	0.00623991
5100	0.01474716	0.01537808	0.00522788	0.00786392
5200	0.01601187	0.01572582	0.00482818	0.00688977
5300	0.04119802	0.03925424	0.02044813	0.02353403
5400	0.0152188	0.01872145	0.00488667	0.00679941
5500	0.01685316	0.01377551	0.00521146	0.00812666
5600	0.01523064	0.02000707	0.00492196	0.00700733
5700	0.02127318	0.01949736	0.00568476	0.00753218
5800	0.01958623	0.01707579	0.00491247	0.00717864
5900	0.02087779	0.01757984	0.00509493	0.00730913
6000	0.01695126	0.01942582	0.0051034	0.00719953
6100	0.02139595	0.01875671	0.00525125	0.00755448
6200	0.01806939	0.01703672	0.00519245	0.00782825
6300	0.01482445	0.01563784	0.00545565	0.00784276
6400	0.01789223	0.01613706	0.00528502	0.00824817
6500	0.0173605	0.02133476	0.00585688	0.00822589
6600	0.02113012	0.01844281	0.00543628	0.0077957
6700	0.01932419	0.01844972	0.00560497	0.00832198
6800	0.02394835	0.02482243	0.00617112	0.00920216
6900	0.0201206	0.01718718	0.00610511	0.00889285
7000	0.0197974	0.01916796	0.00573654	0.00858847

5.1.4 Performance Evaluation

The test results reveal distinctions in computation setups for the CIM. Specifically:

- The execution times of ARM scenarios are faster than the corresponding X86 scenarios.
- For the same ARM-based PC, Python Pandas scenarios are considerably slower than C, and Python Numpy scenarios are slightly slower than C.
- On the same ARM-based PC, scenarios with parallel versions of Python Pandas and NumPy are faster than their sequential counterparts, while C with OpenMP is slower than without.

It is logical that the execution times of ARM scenarios are faster than their corresponding X86 scenarios, as depicted in Table 6, because the Mac M1 Pro is well-equipped and currently stands as one of the fastest PCs on the market. Our test indicates that this type of X86 is not sufficiently fast for the CIM.

As illustrated in Figure 8, for the same ARM-based PC, Python Pandas scenarios are significantly slower than C, and Python Numpy scenarios are slightly slower than C. Python is an interpreted programming language, implying that the source code of a Python program is converted/interpreted into bytecode, which is then executed one instruction after another. In contrast, compiled languages like C and C++ require the entire program to be built and compiled ahead of time before execution. Consequently, Python is slower in execution than C. Another factor contributing to Python's slowness is its Global Interpreter Lock (GIL), which reduces the chances of race conditions with multiple threads but also prevents multiple threads from running in parallel. The Multiprocessing library can circumvent the GIL, thereby accelerating speedup. Ultimately, while Python is written in C and can be close to but not faster than C, Numpy, the fundamental Python library, is more memory-efficient and much faster in indexing than Pandas. However, Pandas is easier to use and has higher industry application.

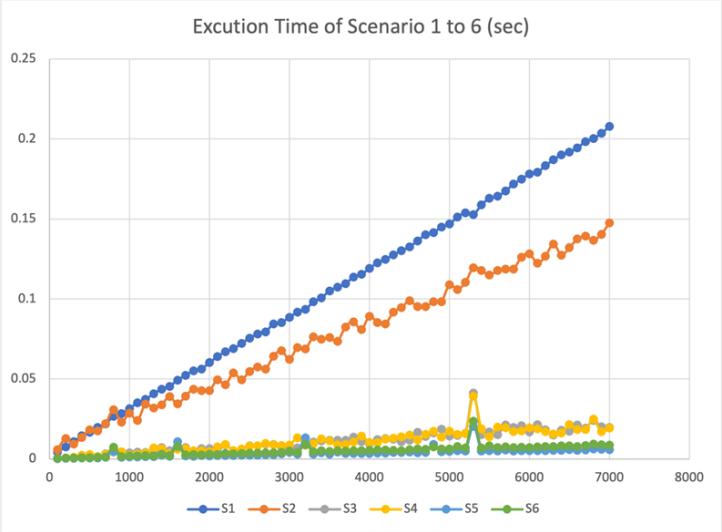


FIGURE 8.EXECUTION TIME OF SCENARIO 1 TO 6.

For both Pandas and NumPy, utilizing multiprocessing results in faster execution compared to the sequential approach. On the other hand, the OpenMP paradigm represents one of the most widely employed parallel programming models on desktop machines, particularly with C or C++. OpenMP operates under the single program multiple data (SPMD) parallelism model, assuming shared memory between threads and introducing overhead to the execution. The benefits of Data Level Parallelism (DLP) on speedup are contingent on the specific program. Users must identify the parallelizable sections of the program in advance. Amdahl's law articulates that "the overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used" (Mian, et al., 2014),as expressed in Equation 1, where f is the fraction of operations in a computation that must be performed sequentially, and p is the speedup of the part of the task that benefits from improved system resources.

$$\mathbf{speedup\_overall} \leq \frac{\mathbf{1}}{\mathbf{f} + \frac{\mathbf{1} - \mathbf{f}}{\mathbf{p}}} \qquad (\mathbf{3})$$

Moreover, an additional factor is the advancement of the Apple M1 chip, which is a system-on-a-chip that already incorporates built-in optimized parallelism. Enforcing OpenMP may lead to less optimized parallelism in this context.

ARM architecture is extensively utilized in smartphones, offering advantages such as low energy consumption and minimal heat generation. Coupled with its shorter running time, ARM can be an ideal choice for IVCs. Consequently, the

recommended computational setup for the CIM is determined to be ARM architecture with the C programming language, leveraging the M1 chip's inherent parallelism.

5.2 Parallel Computing of DAD

Exclusively utilizing the BSM data, the DAD determines whether a CV is in abnormal driving status. The DAD consists of five modules: Module 1: Data Preprocessing and Selecting KPIs; Module 2: Learning Normal Behavior; Module 3: Detecting Outliers; Module 4: Determining Abnormal Driving Events; and Module 5: System Updating, as illustrated in Figure 9. In the O6 project, all these modules will be executed on the IVC. As discussed in in 5.1 Parallel Computing of CIM, ARM architecture is recommended as the hardware for the CIM model. This section presents the parallel computing implementation of the DAD running on ARM architecture. Similar to the CIM, a Mac Pro with an M1 chip and macOS Ventura 13.3.1 was employed for these computations.

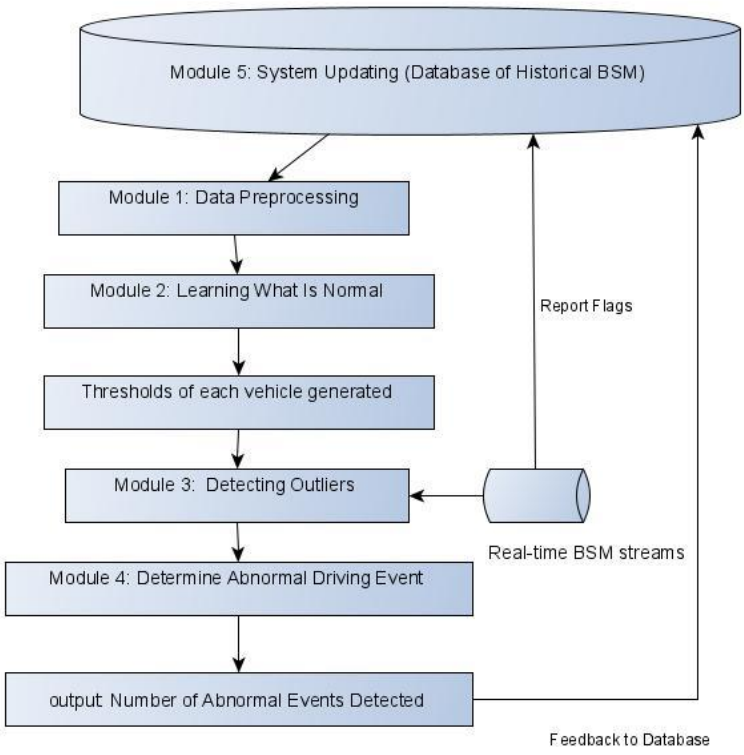


FIGURE 9. THE FLOWCHART OF DAD.

Since the DAD can operate offline, and the O6 system has mitigated the extensive data transfer observed in the F4, this section concentrates on the integration of GPU for parallel computing and assesses the suitability of OD ML packages.

5.2.1 Test Data

The Test Data used the historical BSMs of a selected CV used in F4 project, the attributes are as described in section 4.0 TASK 2: DATABASE CONSTRUCTION. As in F4, the longitudinal acceleration and lateral acceleration were found have some relationship with speed, as shown in the visualization of the raw data of Figure 10.

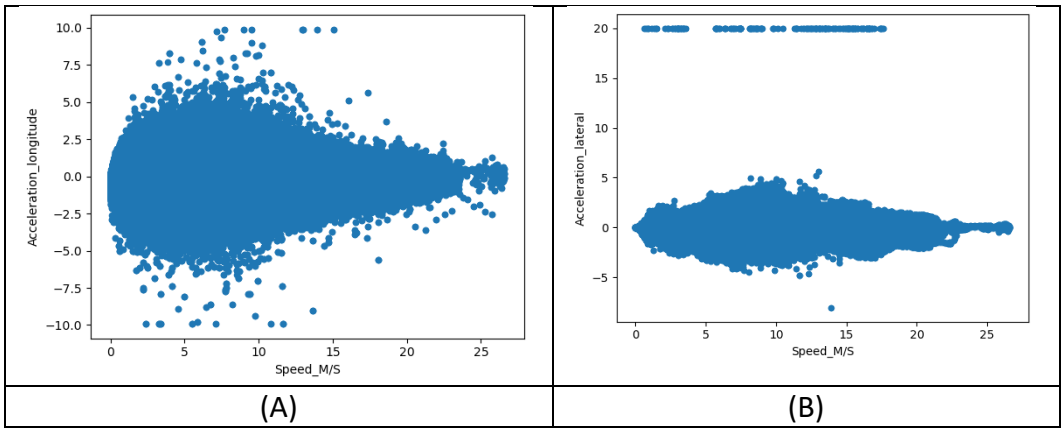


FIGURE 10. THE SCATTER PLOT OF SPEED AND ACCELERATION. (A) LONGITUDINAL ACCELERATION AND SPEED. (B) LATERAL ACCELERATION AND SPEED

5.2.2 Scenario Configuration

Apple's M1 chip incorporates a built-in graphics GPU that facilitates parallel computing, utilizing the Metal Performance Shaders (MPS) framework as the Graphics and Compute API. PyTorch, an open-source ML framework based on the Python programming language and the Torch library, employs MPS as a backend for GPU acceleration on Mac systems with the M1 chip. PyTorch utilizes tensors to represent model inputs, outputs, and parameters, with the ability to run on GPUs and share memory with NumPy arrays, eliminating the need for data copying. For this task, PyTorch was installed on a Mac Pro with an M1 chip running macOS Ventura 13.3.1, and Jupyter Lab served as the integrated development environment (IDE).

The test scenarios for this section were configured as follows: DAD on CPU, DAD on GPU, and the application of OD ML algorithms, including Angle-based Outlier Detector (ABOD), Cluster-based Local Outlier Factor (CBLOF), Histogram-based Outlier Score (HBOS), Isolation Forest (IF), and K Nearest Neighbors (KNN).

TABLE 7. SCENARIO SETUP OF DAD PARALLEL COMPUTING TESTS

Algorithm Name	Algorithm Type

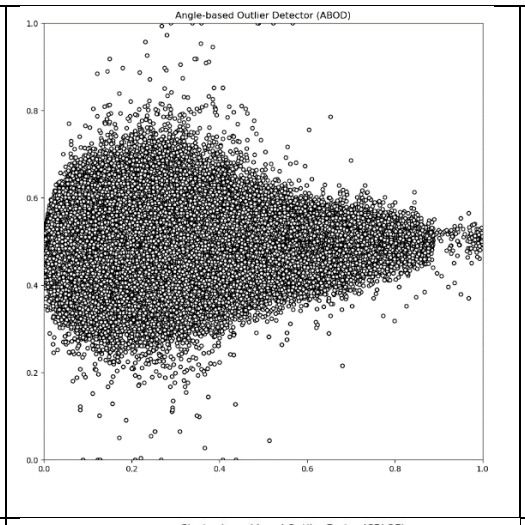
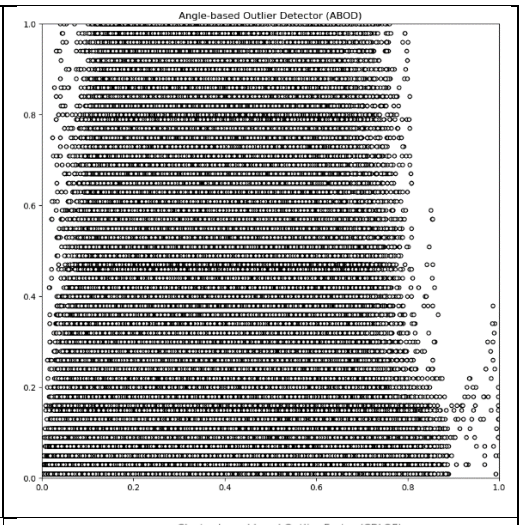
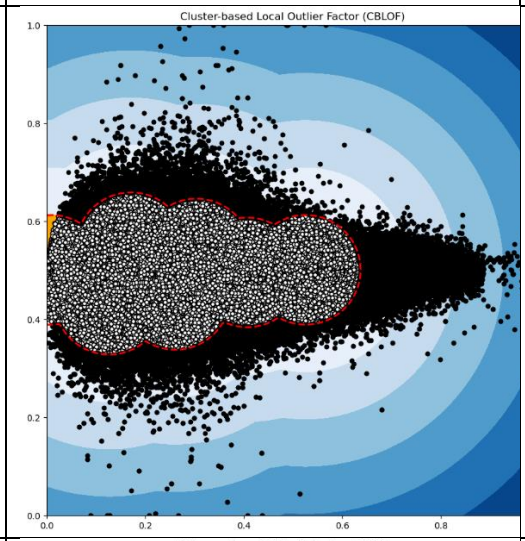
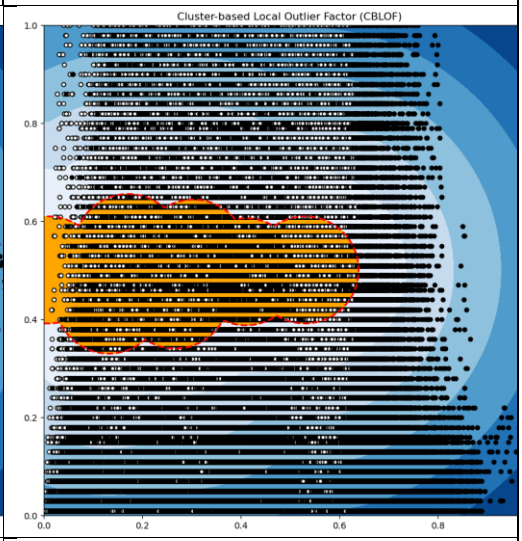
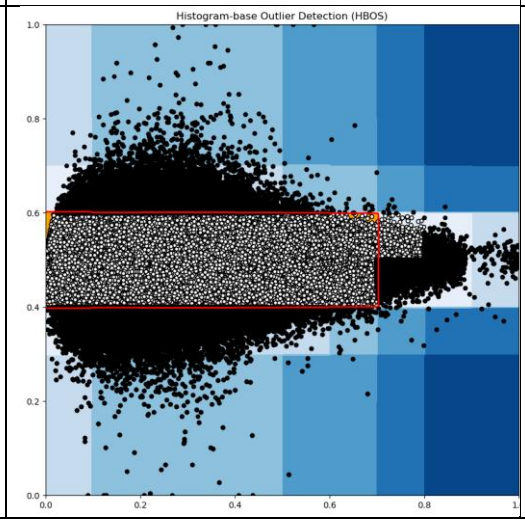
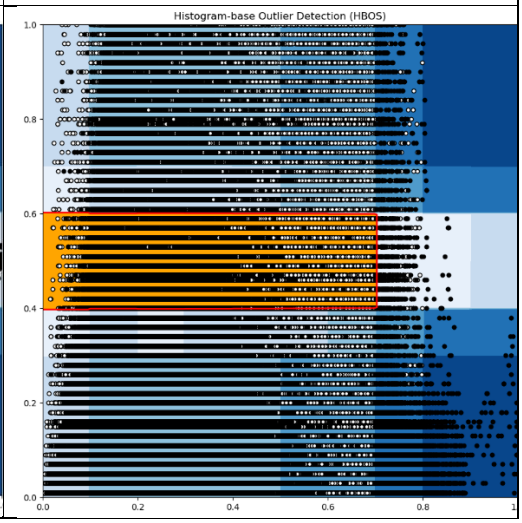
1	ABOD	ML open source
2	CBLOF	ML open source
3	HBOS	ML open source
4	IF	ML open source
5	KNN	ML open source
6	DAD on CPU	STRIDE F4
7	DAD on GPU	STRIDE O6

5.2.3 Test Results

As the OD ML algorithms output the number of outliers, thresholds and plots to show the outlier, as shown in Table 8 and Figure 11.

TABLE 8. OUTPUT OF OD ALGORITHMS AND DAD MODELS

	Algorithm Name	**Outliers**		**Threshold**	
		Longitudinal	Lateral	Longitudinal	Lateral
1	ABOD	0	0	nan	nan
2	CBLOF	158345	158344	-0.11175434977913001	-0.10919071757369557
3	HBOS	153140	135949	-1.9078634333717992	0.2580508062387792
4	IF	158348	0 158335	-2.0801210125544493e-17	0.0
5	KNN	142490	142490	-0.0001503609022556196	-0.000505561180569658
6	DAD on CPU with multiprocessing	Output Alarm Once Detected		Threshold Pannal	
7	DAD on GPU	Output Alarm Once Detected		Threshold Pannal	

(1) ABOD		
(2) CBLOF		
(3) HBOS		

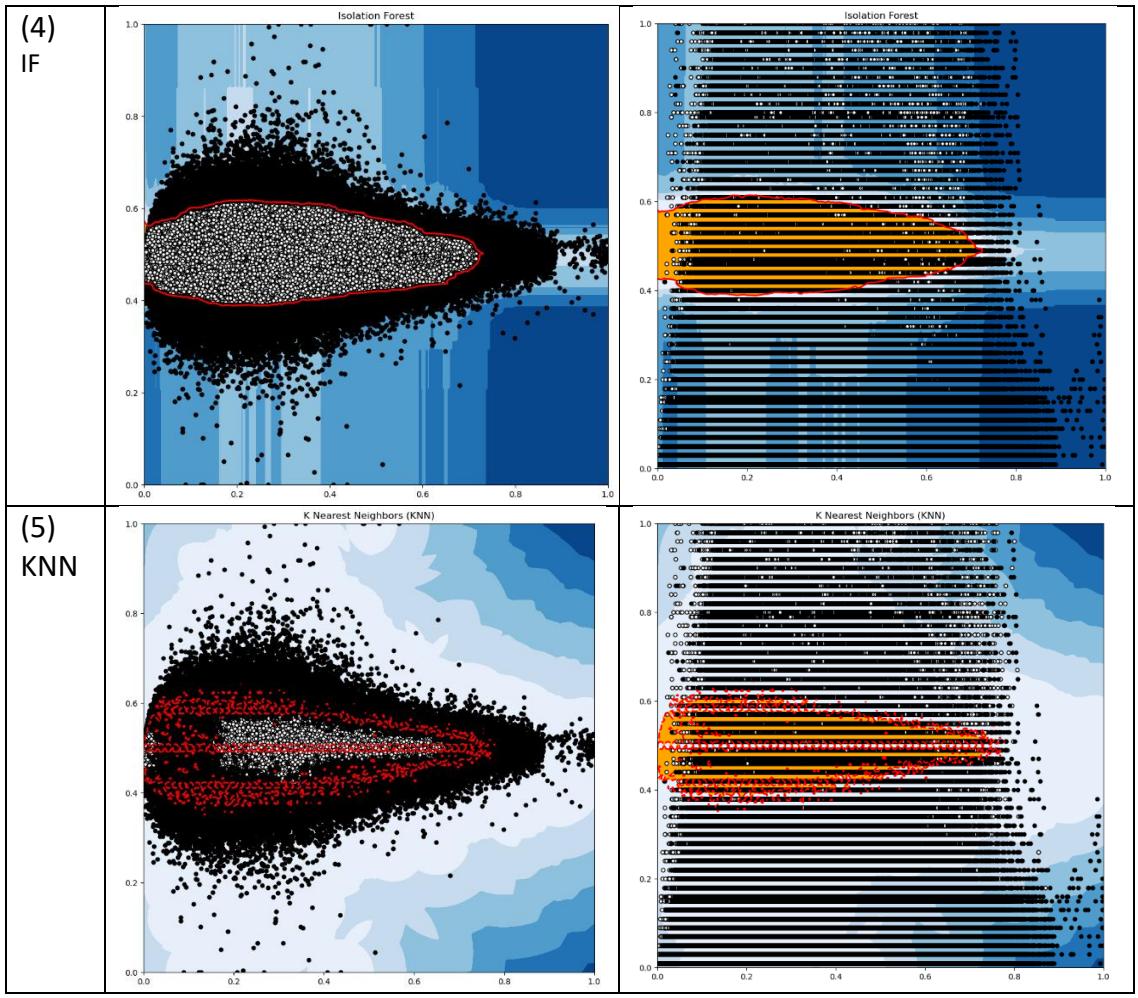


FIGURE 11. THE OUTPUT PLOTS OF OD ML ALGORITHMS

The average execution times for DAD were approximately 13 seconds on CPU and 23 seconds on GPU when processing a dataset with 3 million instances. While calculations on the GPU demonstrate faster performance, the data loading time was significantly slower. This disparity raised from storing the data in CSV format, with loading CSV data to a tensor being notably slower than loading it to a Pandas DataFrame with multiprocessing.

TABLE 9. THE EXECUTION TIME OF DADS

	Total Execution Time	Data Loading	
DAD on CPU with multiprocessing	13 seconds	CSV to Pandas DataFrame	2 seconds
DAD on GPU	23 seconds	CSV to Tensor of Torch	20 seconds

5.2.4 Performance Evaluation

The OD ML algorithms yielded approximately 5 percent outliers of the total instances, as indicated in Table 10. This percentage resulted from setting the parameter "outliers_fraction" to 0.05. Similar to F4, the output thresholds

served as the threshold panel for real-time detection of abnormal driving. However, the thresholds generated by the ODs are not applicable to our system, lacking any association with transportation terms or phenomena.

TABLE 10. THE OUTPUT OF OD ALGORITHMS

	Algorithm Name	Total Instance	Outliers				Threshold	
			Longitudinal	Longintudinal Persentage	Lateral	lateral Persentage	Longitudinal	Lateral
1	ABOD	3166950	0	0	0		nan	nan
2	CBLOF	3166950	158345	0.04999921	158344	0.04999889	-0.1117543	-0.1091907
3	HBOS	3166950	153140	0.04835567	135949	0.04292742	-1.9078634	0.25805081
4	IF	3166950	158348	0.05000016	158335	0.04999605	-2.08E-17	0
5	KNN	3166950	142490	0.04499282	142490	0.04499282	-0.0001504	-0.0005056

6.0 CONCLUSIONS

The O6 project marks a substantial improvement over F4, driven by advancements in both system architecture and computing paradigm. Our journey commenced with an extensive literature review on parallel computing, revealing the prevailing trend in the automotive market towards flexible and lightweight CAVs. Recognizing the significant advancements in IVCs, we identified DSD as the future of parallel computing.

Both F4 and O6 feature a two-tier hierarchical structure with an upper-tier core cloud and a lower tier consisting of CVs monitored by the core cloud. In F4, the core cloud manages the flag list of abnormal CVs and major DAD modules, while the IVC handles a portion of DAD in conjunction with the CIM. In contrast, O6 relocates the entire DAD to the IVC, assigning the core cloud exclusive responsibility for the flag list.

F4's success relies on seamless cooperation between auto manufacturers, BSM central control, and government support—an identified challenge for the near future. Meanwhile, O6, while potentially susceptible to minor data loss and unsuitability for comprehensive traffic analysis, presents significant benefits in reducing data traffic and improving latency performance. Considering O6's advantages over its drawbacks and its alignment with the prevailing trend towards flexible and lightweight solutions, we adapted the O6 architecture to fully migrate DAD to the IVC.

Moving from F4's sequential computation paradigm, O6 underwent a crucial upgrade to a parallel version, resulting in notable improvements in processing speed, efficiency, and scalability. We designed the DSD process with considerations at three levels of abstraction: chip architecture, programming language, and parallelism module. Testing configurations included

C, Python, and OpenMP on both Windows and MacOS platforms, specifically targeting the M1 chip for MacOS, using Visual Studio Code.

Our working datasets comprised BSM data from CV pilot studies, with performance evaluation utilizing crash data from the SHRPII NDS. Both datasets were in CSV format. In evaluating our DAD, we compared its performance in F4 and O6 with various established OD packages designed for outlier detection. The findings indicated that existing OD models fell short of meeting our system requirements. Focused on minimizing processing time and based on our working data, we concluded to employ ARM architecture, C programming language, and leverage the built-in parallelism of the ARM chip for CIM. For DAD, ARM architecture and Python language on the CPU with multiprocessing were deemed suitable for parallel computing.

7.0 RECOMMENDATIONS AND FURTURE WORK

This project's primary contribution lies in its innovative approach to configuring DSD for IVCs across three levels of abstraction: chip architecture, programming language, and the parallelism module. For the CIM of our system, we recommend utilizing ARM architecture, the C programming language, and leveraging the built-in parallelism of the ARM chip. For the DAD, we propose fully migrating DAD to IVC, employing ARM architecture, and using Python language on the CPU with multiprocessing for parallel computing. It is important to note that these recommendations are based on datasets in CSV format, and if binary format data is used, which is the format of the BSM in real-world operation, additional testing is necessary.

Several significant challenges lie ahead for future work on DSD and IVC. These challenges include understanding emerging trends in the IVC market, exploring the integration of automated and connected vehicles, assessing the impact of connected and automated vehicles on intelligent transportation systems, and examining how market players would adopt DSD. Regarding technologies, thoughtful consideration is needed to address challenges related to potential data loss in the event of CV malfunctions and evolving demands for comprehensive traffic analysis in the future.

There is also considerable future work anticipated for DAD. Recognizing that driving behaviors are complex processes involving actions controlled by both conscious and subconscious aspects of the human brain, relying solely on the vehicle's footprint to determine behavior status may be insufficient. When scoring outliers, the relative impacts of different key performance indicators lack clarity, and auto-tuning was not possible due to a lack of data.

The scope of this project does not encompass the data path, which involves the vehicle cloud, and remains an open research problem representing one of the most significant challenges for CAV development. Further research and development in the data path are anticipated for the

realization of our system. While this project only scratches the surface, it serves as a case that showcases the initial research conducted on DSD for IVC.

8.0 REFERENCE LIST

ACM, 2017. *ISCA '17: Proceedings of the 44th Annual International Symposium on Computer Architecture.* New York, NY, USA, Association for Computing Machinery.

Amini, S., Gerostathopoulos, I. & Prehofer, C., 2017. *Big data analytics architecture for real-time traffic control.* s.l., s.n., p. 710–715.

An, S.-h., Lee, B.-H. & Shin, D.-R., 2011. *A survey of intelligent transportation systems.* s.l., s.n., p. 332–337.

Asanovic, K. et al., 2009. A view of the parallel computing landscape. *Communications of the ACM,* Volume 52, p. 56–67.

Boukerche, A. a. Z. L. a. A. O., 2020. Outlier detection: Methods, models, and classification. *ACM Computing Surveys (CSUR),* Volume 53, pp. 1--37.

Brown, E. N., 2010. *Type oriented parallel programming.* s.l.:Durham University.

Culler, D., Singh, J. P. & Gupta, A., 1999. *Parallel computer architecture: a hardware/software approach.* s.l.:Gulf Professional Publishing.

Darlington, J., 1996. Structured parallel programming: parallel abstract data types.

Duan, L., Xu, L., Liu, Y. & Lee, J., 2009. Cluster-based outlier detection. *Annals of Operations Research,* Volume 168, p. 151–168.

Gottlieb, A. & Almsi, G., 1989. Highly parallel computing.

Hernandez, O. a. N. R. C. a. C. B. a. B. V. a. K. R., 2009. *Open source software support for the openmp runtime api for profiling.* s.l., IEEE.

ISCA, 2017. *ISCA '17: Proceedings of the 44th Annual International Symposium on Computer Architecture.* New York, NY, USA, Association for Computing Machinery.

Khazaei, H., Zareian, S., Veleda, R. & Litoiu, M., 2016. Sipresk: A big data analytic platform for smart transportation. In: *Smart City 360°.* s.l.:Springer, p. 419–430.

Kriegel, H.-P., Schubert, M. & Zimek, A., 2008. *Angle-based outlier detection in high-dimensional data.* s.l., s.n., p. 444–452.

Larose, D. T. & Larose, C. D., 2014. k-nearest neighbor algorithm.

Lin, Y., Wang, P. & Ma, M., 2017. *Intelligent transportation system (ITS): Concept, challenge and opportunity.* s.l., s.n., p. 167–172.

MarketsandMarkets™ Ltd., 2020. *In-Vehicle Computer System Market,* s.l.: s.n.

Mian, R. et al., 2014. *A data platform for the highway traffic data.* s.l., s.n., p. 47–52.

Moore, G. E., 1998. Cramming more components onto integrated circuits. *Proceedings of the IEEE,* Volume 86, p. 82–85.

Null, L. & Lobur, J., 2014. *Essentials of Computer Organization and Architecture.* s.l.:Jones \& Bartlett Publishers.

Oracal, 2023. *MySQL 8.0 Reference Manual.* [Online]
Available at: https://dev.mysql.com/doc/refman/8.0/en/
[Accessed 2023].

Osuna, J. A., 1994. COMPUTING RESEARCH NEWS. *COMPUTING.*

Putrada, A. G. & Abdurohman, M., 2021. *Anomaly detection on an iot-based vaccine storage refrigerator temperature monitoring system.* s.l., s.n., p. 75–80.

Saidu, C. I. a. O. A. a. O. P. O., 2015. Overview of Trends Leading to Parallel Computing and Parallel Programming. 7(British Journal of Mathematics \& Computer Science), p. 40.

Schauer, B., 2008. Multicore processors–a necessity. *ProQuest discovery guides,* p. 1–14.

Shtern, M. et al., 2014. *Towards a multi-cluster analytical engine for transportation data.* s.l., s.n., p. 249–257.

Stoller, S. D. et al., 2019. *Future directions for parallel and distributed computing: SPX 2019 workshop report.* s.l., s.n.

Wang, H. et al., 2020. Architectural design alternatives based on cloud/edge/fog computing for connected vehicles. *IEEE Communications Surveys & Tutorials,* Volume 22, p. 2349–2377.

Xu, D., Wang, Y., Meng, Y. & Zhang, Z., 2017. *An improved data anomaly detection method based on isolation forest.* s.l., s.n., p. 287–291.

Zhao, Y., Nasrullah, Z. & Li, Z., 2019. Pyod: A python toolbox for scalable outlier detection. *arXiv preprint arXiv:1901.01588.*

Zhu, L. et al., 2018. Big data analytics in intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems,* Volume 20, p. 383–398.

9. APPENDICES

9.1 Appendix A – Acronyms, abbreviations, etc.

AASHTO -- American Association of State Highway and Transportation Officials

ACC -- adaptive cruise control

ADAS -- advanced driver assistance systems

AV -- autonomous vehicle

BSM -- basic safety message

CIM -- conflict detection model

CPU – central processing unit

CSV -- comma-separated values

CV -- connected vehicle

DA -- driving anomaly

DVU -- driver-vehicle unit

ESA -- emergency steering assistance

FCW -- forward collision warning

FHWA -- Federal Highway Administration

GPS -- Global Positioning System

GPU -- graphic processing unit

ITS -- intelligent transportation system

ITS -- intelligent transportation system

IVC – in-vehicle computer

KPI -- key performance indicator

LDW -- lane departure warning

MTC -- margin to collision

ML – machine learning

NDS -- Naturalistic Driving Study

NHTSA -- National Highway Traffic Safety Administration

OBU -- on-board unit

OD – outlier detection

SHRP II -- the Strategic Highway Research Program

SPMD -- Safety Pilot Model Deployment (SPMD)

US DOT -- United States Department of Transportation

V2X -- vehicle-to-everything

9.2 Appendix B – Associated websites, data, etc., produced

https://insight.shrp2nds.us/login/auth

https://www.its.dot.gov/pilots/

9.3 Appendix C – Sample Results

TABLE 11. SAMPLE RESULTS OF CIM TESTS

Senario	1	2	3	4	5	6	7	8	9	10	11	
Chip Architec	ARM_64	ARM_64	ARM_64	ARM_64	ARM_64	ARM_64	X_86_64	X_86_64	X_86_64		X_86_64	X_86_64
Language	Python: Pand	Python: Pand	Python: Num	Python: Num	C	C	Python: Pand	Python: Pand	Python: Num	Python: Nump	C	C
Parallelism Module	None	Multi-processing	None	Multi-processing	None	OpenMP	None	Multi-processing	None	Multi-processing	None	OpenMP
Number of CVs												
100	0.00429201	0.00332403	0.00040698	0.00067616	0.00040698	0.00033522	0.01000428	0.42237091	0.0049665	0.007448196	0.04725337	0.02999187
200	0.00748396	0.0053091	0.00056028	0.00081587	0.00056028	0.00047421	0.02094364	0.02293873	0.03153634	0.027921677	0.04986644	0.04240489
300	0.01085591	0.00771785	0.00082994	0.00211692	0.00082994	0.00041032	0.03191423	0.03291106	0.00997353	0.011114597	0.07082629	0.0603807
400	0.0144062	0.00942016	0.00096703	0.0013268	0.00096703	0.00052214	0.03195238	0.03937244	0.01132965	0.010575056	0.09146738	0.07830667
500	0.01669312	0.01131201	0.00116611	0.00198483	0.00116611	0.00051785	0.0379293	0.04665184	0.01492238	0.015585899	0.09450936	0.0882802
600	0.01969886	0.01348829	0.00134683	0.00166297	0.00134683	0.00074196	0.04585123	0.05714822	0.01486421	0.018205404	0.11897254	0.10922432
700	0.0219121	0.01606083	0.00185299	0.00181794	0.00185299	0.00141501	0.05291629	0.06013083	0.01892138	0.022853136	0.13623095	0.12899113
800	0.02668405	0.02652001	0.00253701	0.14537883	0.00253701	0.002424	0.05982566	0.07143044	0.02166271	0.023399353	0.1381712	0.13116503
900	0.02828598	0.02003813	0.00205708	0.00246119	0.00205708	0.00090909	0.06910825	0.08158398	0.02876735	0.025264263	0.18258572	0.15561771
1000	0.03151202	0.02217722	0.002069	0.00315118	0.002069	0.00094604	0.12100649	0.1173737	0.14495349	0.153775215	0.18631101	0.17424822
1100	0.03502989	0.02435803	0.00229812	0.00328922	0.00229812	0.00105023	0.11597252	0.13612103	0.15599346	0.163504839	0.26481223	0.2264719
1200	0.03716707	0.02656794	0.00609803	0.03028297	0.00609803	0.00109482	0.1209352	0.12613964	0.16299796	0.168321133	0.26494193	0.22693276
1300	0.04067302	0.02855802	0.00272107	0.03237104	0.00272107	0.00131679	0.13162899	0.12879753	0.18548393	0.184026003	0.25668073	0.25335836
1400	0.04358602	0.03031683	0.00308394	0.02391315	0.00308394	0.00235295	0.14062428	0.15455627	0.18382478	0.189732075	0.31196404	0.27647924
1500	0.0453701	0.03273988	0.0034492	0.00369716	0.0034492	0.00145817	0.14261174	0.1595521	0.20149493	0.214132547	0.27720904	0.25530648
1600	0.04923391	0.03453326	0.00548077	0.00366902	0.00548077	0.00181794	0.16057396	0.17607522	0.20114112	0.211987972	0.35357213	0.34597898
1700	0.05234599	0.03706384	0.00485778	0.00403214	0.00485778	0.00159693	0.17008185	0.18514919	0.23613167	0.245057344	0.39316535	0.36128139
1800	0.05502319	0.03887391	0.04587793	0.00403118	0.04587793	0.00157213	0.17810822	0.20143437	0.24134612	0.260811567	0.37075448	0.32193041
1900	0.05624127	0.04099798	0.003865	0.00404096	0.003865	0.0016439	0.19704866	0.19946647	0.25033164	0.286137342	0.41273069	0.37731099
2000	0.06018209	0.04334688	0.02271175	0.00474286	0.02271175	0.00202894	0.20320582	0.20744467	0.25274277	0.269272089	0.38930583	0.35352397
2100	0.06390095	0.04583287	0.00509501	0.00546098	0.00509501	0.00185394	0.20765328	0.2179873	0.28843451	0.299110174	0.41801858	0.40740204
2200	0.06696868	0.04690409	0.02555323	0.03523827	0.02555323	0.00209403	0.21573567	0.21374846	0.28095841	0.301294327	0.47151041	0.41031218
2300	0.06888318	0.04974008	0.00499296	0.00559282	0.00499296	0.00197506	0.2354722	0.23235106	0.31105065	0.326141357	0.47870421	0.44242859
2400	0.07221198	0.05184412	0.00591779	0.00553012	0.00591779	0.00213003	0.23475385	0.2315731	0.31867313	0.339980602	0.50282359	0.47680211
2500	0.07550097	0.05370903	0.00506902	0.00529909	0.00506902	0.00221801	0.23958158	0.23113537	0.08731699	0.091241121	0.49235654	0.44285512
2600	0.07803607	0.05612493	0.00566602	0.04267502	0.00566602	0.00234413	0.26354957	0.25238514	0.06856537	0.072802067	0.53151226	0.50442815
2700	0.0794251	0.05800605	0.00529623	0.00573301	0.00529623	0.00255013	0.26410007	0.25739884	0.07271957	0.078794241	0.55112195	0.51250148
2800	0.08438015	0.05989313	0.00580907	0.00655913	0.00580907	0.00261712	0.27882576	0.28970337	0.06973648	0.079763651	0.57380581	0.51705551
2900	0.08534122	0.06212306	0.0058639	0.00626516	0.0058639	0.00581717	0.28030658	0.29051495	0.07081032	0.077032328	0.56930375	0.52127504
3000	0.08858895	0.06606412	0.00646615	0.00709009	0.00646615	0.0060401	0.25731134	0.26477313	0.07552624	0.082663059	0.61696768	0.56082106
3100	0.09173393	0.06655526	0.00858307	0.00681686	0.00858307	0.00291395	0.22984457	0.24993563	0.08187342	0.083248854	0.63488913	0.58375812
3200	0.09356594	0.0682528	0.00805092	0.00680614	0.00805092	0.0027051	0.24381566	0.27155805	0.0897646	0.08727622	0.68250275	0.60802078
3300	0.09818721	0.07070637	0.00664878	0.00696206	0.00664878	0.00271606	0.24830914	0.28316426	0.0829246	0.093607187	0.70081687	0.63822627
3400	0.100703	0.072649	0.00769687	0.0073328	0.00769687	0.00361109	0.25926304	0.29145908	0.09075427	0.094763041	0.7255621	0.6684916
3500	0.10507798	0.07501388	0.02608299	0.00716496	0.02608299	0.00351906	0.26723337	0.28853798	0.08828092	0.096373558	0.70801377	0.67594814
3600	0.10738468	0.07719016	0.04316306	0.00761127	0.04316306	0.00336099	0.27160835	0.30055833	0.08967853	0.10043931	0.74398541	0.70848989
3700	0.10955191	0.07916379	0.00743914	0.00840497	0.00743914	0.00308108	0.27563572	0.30564761	0.09872246	0.097474813	0.7554512	0.70304918
3800	0.11369991	0.08129668	0.03263998	0.0426681	0.03263998	0.00350094	0.28411269	0.31417251	0.09733391	0.104615927	0.76152325	0.68849373
3900	0.11545897	0.083606	0.00978899	0.00844002	0.00978899	0.00343895	0.29102874	0.31859159	0.09759212	0.105899811	0.76780462	0.76397443
4000	0.11907315	0.0855701	0.00822473	0.00804925	0.00822473	0.00410819	0.29966378	0.33915305	0.10105801	0.111937523	0.86891389	0.85162234
4100	0.12264919	0.08797002	0.00916791	0.04037595	0.00916791	0.00425196	0.3029201	0.35322428	0.10399985	0.111816883	0.75805068	0.76595831
4200	0.12462425	0.08988309	0.00908804	0.00901413	0.00908804	0.0040791	0.29707432	0.36963272	0.10570931	0.110432625	0.7802484	0.78466702
4300	0.12759399	0.09251213	0.01055384	0.00898623	0.01055384	0.00407195	0.32875896	0.35204911	0.11034155	0.117016792	0.86627841	0.81511188
4400	0.13010311	0.09371114	0.00902677	0.03996921	0.00902677	0.00418901	0.32552004	0.35523558	0.10591626	0.113755703	0.79626894	0.78267741
4500	0.13257718	0.09589291	0.00952721	0.03696585	0.00952721	0.00394797	0.34182978	0.36232257	0.11538696	0.141619682	0.8925364	0.86353683
4600	0.13624001	0.09802318	0.00944614	0.00980711	0.00944614	0.00378489	0.34892058	0.39407372	0.11236882	0.125045538	0.923944	0.84899068
4700	0.14007616	0.10049701	0.0134201	0.00936484	0.0134201	0.00389886	0.3529129	0.40711021	0.1204412	0.132828236	0.87340689	0.88663673
4800	0.14149213	0.10162592	0.0394671	0.02774191	0.0394671	0.00488186	0.36257029	0.41577744	0.12171197	0.137428761	0.89674306	0.91496825
4900	0.14495802	0.10385919	0.03758597	0.01044297	0.03758597	0.0040462	0.37099195	0.41001868	0.12138677	0.136131763	0.9383595	0.88949871
5000	0.14679599	0.10741115	0.01227808	0.04067397	0.01227808	0.00449109	0.37085032	0.4239018	0.13227248	0.134638071	0.93896103	0.95801926
5100	0.1512742	0.11247373	0.01173186	0.0113678	0.01173186	0.00536299	0.38076854	0.43873739	0.12986469	0.135502338	0.96587181	0.98173356
5200	0.15387392	0.11148	0.03759933	0.01101494	0.03759933	0.00485611	0.37793159	0.45615578	0.13190722	0.144584179	0.95931697	0.93644738
5300	0.15280819	0.115062	0.03757811	0.01586986	0.03757811	0.06055498	0.38988042	0.44854212	0.13215137	0.143692732	0.97886515	0.96882653
5400	0.1587131	0.11388111	0.01094103	0.03784871	0.01094103	0.00559688	0.40532088	0.47618318	0.13596606	0.140133619	1.00755906	1.00774169
5500	0.16299295	0.11633992	0.04127789	0.01283407	0.04127789	0.00503397	0.4033289	0.46542835	0.13468075	0.153412342	0.98364139	1.04245353
5600	0.16415787	0.11785865	0.01223993	0.01204395	0.01223993	0.0047431	0.4210515	0.47314048	0.14017391	0.149520397	1.04245067	1.04961753
5700	0.16743612	0.12098384	0.01203108	0.01326585	0.01203108	0.00701404	0.42287135	0.47164035	0.1426754	0.166781425	1.04531908	1.05425119
5800	0.17176795	0.12434387	0.03269196	0.01301599	0.03269196	0.00511193	0.43713689	0.50895405	0.15139365	0.163255692	1.11455274	1.06656885
5900	0.17484426	0.12538004	0.04345083	0.03776574	0.04345083	0.01516008	0.441751	0.51438498	0.14966464	0.161153793	1.10618424	1.08864737
6000	0.17807102	0.12839222	0.01264	0.01274085	0.01264	0.00505972	0.44510603	0.51799464	0.15346432	0.168765545	1.10620809	1.15201902
6100	0.17917895	0.13066387	0.04608011	0.01269579	0.04608011	0.00503993	0.4521265	0.5374012	0.15260363	0.175770044	1.16146684	1.15271282
6200	0.18329501	0.13208795	0.0470252	0.0139792	0.0470252	0.00517106	0.45650554	0.52445793	0.1576221	0.172509909	1.10769057	1.11055589
6300	0.18703008	0.13431597	0.03139806	0.01318908	0.03139806	0.0054841	0.46672559	0.52861714	0.17852926	0.17461133	1.18343663	1.15809059
6400	0.19002318	0.13579321	0.01304889	0.01421428	0.01304889	0.00546813	0.47048235	0.54313517	0.17580032	0.175651312	1.21833158	1.17812037
6500	0.1917429	0.13875389	0.01356316	0.03743935	0.01356316	0.01007414	0.48494887	0.54627204	0.18731451	0.175204992	1.28438306	1.20401907
6600	0.19441915	0.13994384	0.0144999	0.01343727	0.0144999	0.00572705	0.51566339	0.56142211	0.18941545	0.178333759	1.28530121	1.24633169
6700	0.19832706	0.14231396	0.01450396	0.0138979	0.01450396	0.00567389	0.5103991	0.56246567	0.17708969	0.184883356	1.26786065	1.20823288
6800	0.20033693	0.14410186	0.01378894	0.03809595	0.01378894	0.00780392	0.51115751	0.59249139	0.17135286	0.191464663	1.42492151	1.27553916
6900	0.20346498	0.14701581	0.04324889	0.01472902	0.04324889	0.0061152	0.50815797	0.59149575	0.17668724	0.197807789	1.29974866	1.26460314
7000	0.20789409	0.14873075	0.03858423	0.01499701	0.03858423	0.00689411	0.52736259	0.59645963	0.19692445	0.189439058	1.3340497	1.29321718

9.4 Appendix C – Summary of Accomplishments

Date	**Type of Accomplishment** *(select from drop down list)*	**Detailed Description** *Provide name of person, name of event, name of award, title of presentation, location and any links to announcements if available Please attach any abstracts, summaries, high quality photos, or additional details as an appendix.*
11/01/2022	Conference Paper	We submitted the abstract of a paper titled "***Parallel Computing on the In-vehicle Subsystem for Safety Diagnosis in the Connected Vehicle Environment*** " to the International Conference on Transportation and Development (ICTD) 2023.
06/25/2023	Journal Paper	Submitted to Vehicles a paper titled "Adaptive Individual-Level Cognitive Driving Anomaly Detection Model Exclusively Using BSMs". **Accepted: 18 September 2023 /Published: 26 September 2023**

Abstract of the paper submitted to the International Conference on Transportation and Development (ICTD) 2023. https://www.asce-ictd.org/

ASCE INTERNATIONAL CONFERENCE ON TRANSPORTATION & DEVELOPMENT | INTERNATIONAL AIRFIELD & HIGHWAY PAVEMENTS CONFERENCE

In Partnership with the Texas Department of Transportation

Austin, Texas | June 14–17, 2023

Dear Di Wu :

On behalf of the Steering Committee of ICTD 2023 we are pleased to notify you that your abstract ***ID# 1370273 titled, "Parallel Computing on the In-vehicle Subsystem for Safety Diagnosis in the Connected Vehicle Environment "*** has been ***tentatively selected for inclusion in the conference program.*** Final approval is contingent upon fulfillment of the following requirements:

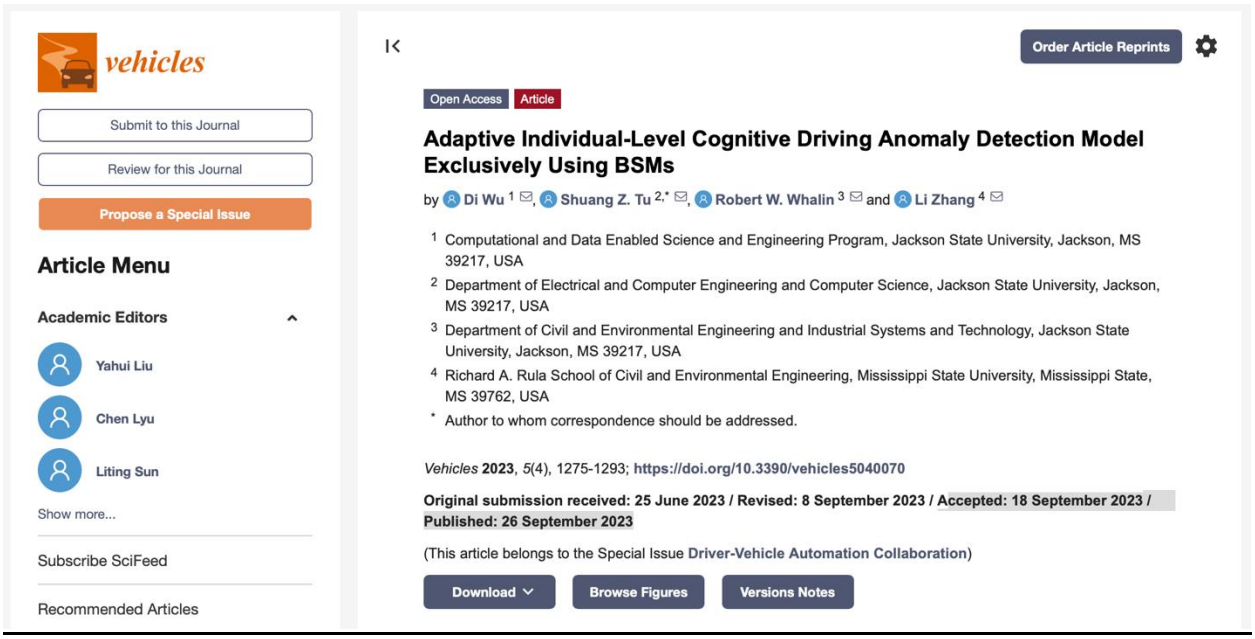
- Submission of your paper in accordance with ASCE's paper formatting guidelines by November 15, 2022, and approval of your paper by the Conference Steering Committee. *Instructions and access information for submitting your paper will be sent at a later date.*
- Submission of your final paper incorporating any comments from the review committee by January 16, 2023, and approval of your final paper by the proceeding's editor.
- Registration by the speaker registration deadline of February 8, 2023.

DECISION ON PODIUM VERSUS POSTER PRESENTATION

While your submission was selected for inclusion in the conference program, **the Steering Committee has not made the determination on whether it will be included as a podium or a poster presentation at the conference at this time.** The committee will make that determination following a review of all the draft submissions and creation of technical sessions based on submissions of similarly-themed topics. Final presentation assignments will be communicated in mid-January. All presentations (podium or poster) must be delivered in person in Austin, Texas at the assigned date and time, between the dates of June 15 - 17, 2023.

SUBMISSION OF PAPER IN THE CONFERENCE PROCEEDINGS

- Your paper must adhere strictly to ASCE's paper formatting guidelines. The paper should be from 8 to 12 pages in MS Word format and should be absent of any comments or edits in MS Word's track changes mode. Note that one of the most common reasons for paper rejection is the failure to adhere to the ASCE formatting guidelines and paper length restrictions.
- Deadline for submission of a paper for consideration in the conference proceedings is November 15, 2022.
- Deadline for the final paper incorporating any comments from the review committee is January 16, 2023.
- Instructions and login access information to upload your paper will be sent at a later date.



vehicles

Submit to this Journal

Review for this Journal

Propose a Special Issue

Article Menu

Academic Editors

Yahui Liu

Chen Lyu

Liting Sun

Show more...

Subscribe SciFeed

Recommended Articles

Order Article Reprints

Open Access Article

Adaptive Individual-Level Cognitive Driving Anomaly Detection Model Exclusively Using BSMs

by Di Wu [1], Shuang Z. Tu [2,*], Robert W. Whalin [3] and Li Zhang [4]

[1] Computational and Data Enabled Science and Engineering Program, Jackson State University, Jackson, MS 39217, USA

[2] Department of Electrical and Computer Engineering and Computer Science, Jackson State University, Jackson, MS 39217, USA

[3] Department of Civil and Environmental Engineering and Industrial Systems and Technology, Jackson State University, Jackson, MS 39217, USA

[4] Richard A. Rula School of Civil and Environmental Engineering, Mississippi State University, Mississippi State, MS 39762, USA

* Author to whom correspondence should be addressed.

Vehicles **2023**, *5*(4), 1275-1293; https://doi.org/10.3390/vehicles5040070

Original submission received: 25 June 2023 / Revised: 8 September 2023 / Accepted: 18 September 2023 / Published: 26 September 2023

(This article belongs to the Special Issue Driver-Vehicle Automation Collaboration)

Download Browse Figures Versions Notes