UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL DE CALCULATOARE



# PROIECT DE DIPLOMĂ

Platformă interactivă pentru simularea mesajelor CAN

Florea Radu

**Coordonator științific:**

Sl. dr. ing. Jan Alexandru Văduva

**BUCUREȘTI**

2025

# CUPRINS

## SINOPSIS

Mașinile moderne devin din ce în ce mai complexe, având în interior numeroase "unități electronice de control„ (ECU-uri) care se ocupă de diverse funcții: de la frâne și direcție până la faruri și aer condiționat. Această comunicare internă este crucială pentru asigurarea siguranței și funcționării corecte, dar devine tot mai complicată și expusă riscurilor, mai ales pe măsură ce mașinile devin din ce în ce mai conectate între ele. În această lucrare am creat o platformă virtuală care simulează comunicarea între componentele unui autovehicul, într-un mediu sigur și ușor de înțeles. Folosind un simulator auto și o aplicație cu interfață grafică, putem observa cum circulă informația prin mașină, cum reacționează sistemele la comenzi sau alerte și cum putem testa idei noi fără să avem nevoie de o mașină reală. Această platformă este utilă pentru studenți, cercetători sau ingineri, care vor să înțeleagă și să îmbunătățească siguranța mașinilor viitorului, într-un mod clar, sigur și accesibil.



## ABSTRACT

Modern cars are becoming increasingly complex, containing numerous Electronic Control Units (ECUs) that manage various functions, from brakes and steering to headlights and air conditioning. This internal communication is essential for safety and functionality, but it is also becoming more complicated and exposed to risks, especially as cars connect to the internet and become autonomous. In this paper, we have created a virtual platform that simulates how these components communicate with each other in a safe and easy-to-understand environment. Using a car simulator and a graphical user interface application, we can observe how information flows through the vehicle, how systems respond to commands or alerts, and how we can test new ideas without needing a real car. This platform is useful for students, researchers, or engineers who want to understand and improve the safety of future vehicles in a clear, safe, and accessible way.

# 1 INTRODUCERE

În ultimele decenii, automobilele au cunoscut o transformare profundă, ca urmare a integrării rapide a componentelor electronice complexe și a software-ului avansat în proiectarea și operarea vehiculelor. Acestea nu mai sunt simple mijloace mecanice de transport, ci platforme inteligente, capabile să prelucreze cantități mari de informații în timp real.

Sistemele vehiculului, precum tracțiunea, frânarea, navigația sau climatizarea, funcționează printr-o colaborare continuă între unitățile electronice de control (ECU-uri), realizată cu ajutorul unor rețele interne specializate. Acestea permit transmiterea rapidă a datelor și asigură funcționarea coordonată a componentelor esențiale, într-un ecosistem tehnologic din ce în ce mai integrat și complex.

Această evoluție tehnologică a adus beneficii majore în ceea ce privește siguranța, confortul și eficiența vehiculelor, însă a generat în paralel și o serie de provocări noi. Extinderea conectivității vehiculului cu medii externe, precum infrastructura rutieră, cloud-ul sau alte autovehicule, a transformat rețelele interne într-un punct sensibil din perspectiva securității cibernetice. Inițial proiectate pentru un mediu izolat și predictibil, aceste rețele trebuie astăzi să răspundă unor cerințe moderne de securitate, scalabilitate și fiabilitate, care nu au fost luate în considerare în etapa de dezvoltare.

## 1.1 Context

Protocolul CAN (Controller Area Network) reprezintă una dintre cele mai utilizate soluții de comunicație în rețelele interne ale vehiculelor. Acesta a fost conceput pentru a facilita transferul eficient de date între unitățile electronice de control, oferind un echilibru optim între viteză, simplitate și fiabilitate.

Însă, protocolul CAN, dezvoltat inițial pentru rețele auto izolate, nu include funcționalități de securitate specifice mediilor conectate. Astăzi, este nevoit să opereze în ecosisteme auto moderne, caracterizate printr-o expunere crescută la amenințări cibernetice.

## 1.2 Problema

Deși aceste protocoale de comunicare precum CAN sunt esențiale pentru siguranța vehiculelor, în prezent nu există o metodologie formală sau un cadru clar pentru a testa și analiza comportamentul lor în scenarii periculoase sau în timpul atacurilor. În practică, securitatea

lor este asumată mai degrabă decât dovedită, iar multe sisteme implementate se bazează pe ideea că rețelele de vehicule sunt izolate și implicit sigure (*security by obscurity*), mai degrabă decât să fie proiectate cu mecanisme de protecție încă de la început (*security by design*).

Ca urmare, devine esențială regândirea modului în care aceste rețele pot fi studiate și evaluate, atât din perspectiva funcțională, cât și din cea a securității. În special în mediul educațional și de cercetare, lipsa unor unelte accesibile și interactive limitează posibilitatea de a reproduce și înțelege comportamentul acestor rețele în condiții variate, inclusiv în prezența unor anomalii sau atacuri.

## 1.3 Obiective

Această lucrare are ca scop dezvoltarea unei platforme interactive, unde utilizatorii pot observa și analiza în timp real fluxul complet de informații ce circulă prin rețeaua internă a unui vehicul.

Printre obiectivele specifice se numără:

1. **Simularea comunicației CAN** într-un mediu virtual, controlat, fără a fi necesar hardware real;
2. **Crearea unei interfețe grafice intuitive** care să permită vizualizarea pachetelor CAN care se transmit între module;
3. **Integrarea cu un simulator auto** (precum CARLA) pentru generarea unor scenarii de test realiste;
4. **Posibilitatea de testare a unor situații de risc** (erori, atacuri, anomalii) și observarea impactului acestora asupra sistemului;
5. **Utilizarea unor formate standardizate** (ex. JSON) pentru definirea pachetelor CAN, cu accent pe extensibilitate și reutilizare;
6. **Facilitarea înțelegerii și predării conceptelor** legate de rețelele vehiculului, pentru scopuri educaționale și de cercetare.

## 1.4 Soluția propusă

Această lucrare își propune să răspundă la o întrebare esențială: Cum putem analiza și testa, într-un mod clar și accesibil, modul în care ECU-urile comunică într-un vehicul modern, inclusiv în situații de risc sau eroare? E o provocare complexă, deoarece comunicația auto depinde adesea de implementările producătorilor, procesul de reverse engineering este complicat, iar simularea realistică a rețelei necesită unelte specializate, greu de găsit în mediile educaționale.

Pentru a aborda această problemă, propunem o soluție clară și aplicabilă: o platformă virtuală care reproduce comportamentul rețelei CAN în scenarii controlate și cât mai apropiate de realitate. Aceasta este construită peste simulatorul CARLA și permite injectarea, interceptarea

și vizualizarea pachetelor CAN într-un mod care poate fi adaptat ușor și utilizat în contexte educaționale diverse.

Un aspect esențial pe care îl evidențiem este că multe dintre aceste protocoale pornesc de la presupunerea că rețeaua vehiculului este sigură. Totuși, în practică, această presupunere se poate dovedi falsă, mai ales în cazul în care măsurile de protecție – cum ar fi criptarea sau izolarea – nu funcționează corespunzător.

## 1.5 Rezultatele obținute

Lucrarea a produs o platformă complet funcțională care:

- Simulează fidel traficul CAN și permite vizualizarea sa în timp real.
- Permite corelarea comenzilor CAN cu reacții vizibile în simulatorul CARLA.
- Reproduce scenarii de atac din literatura de specialitate.
- Este scalabilă și rezistentă la stres, testată cu până la un milion de mesaje.
- Se adresează mediului educațional, fiind intuitivă și ușor de extins.

Platforma poate fi utilizată cu succes atât în scop didactic, cât și ca punct de plecare pentru experimente avansate în domeniul securității vehiculelor conectate.

# 2 ANALIZA CERINȚELOR / MOTIVAȚIE

Motivația din spatele realizării acestui proiect este aceea de a oferi o platformă independentă de soluțiile închise ale producătorilor auto, care să fie open-source, să nu necesite hardware suplimentar pentru a fi utilizată și în care să poată fi reproduse atacurile cibernetice reale cu care se confruntă mașinile din ziua de azi. Scopul este ca un număr cât mai mare de utilizatori să poată înțelege riscurile reale la care sunt expuse sistemele electronice ale mașinilor și să conștientizeze consecințele potențiale ale compromiterii acestor componente critice. Înțelegerea acestor riscuri de către o audiență extinsă, nu doar de către experți, reprezintă un pas important în dezvoltarea unor practici de securitate mai solide în sectorul auto.

# 3 STATE OF THE ART

## 3.1 Context și fundamente teoretice

Rețeaua Controller Area Network (CAN) este un protocol de comunicație serială utilizat pe scară largă în industria auto, având rolul de a facilita schimbul de mesaje între unitățile electronice de control (ECU – Electronic Control Unit). Aceasta funcționează într-un mediu de transmisie partajat, în care fiecare nod trimite mesaje folosind identificatori ce stabilesc prioritatea, iar accesul la rețea este reglementat printr-un mecanism de arbitrare non-distructivă.

Un ECU este o unitate electronică responsabilă de controlul unuia sau mai multor subsisteme din vehicul, cum ar fi motorul, frânele sau sistemul de infotainment. Aceste unități comunică între ele prin intermediul mesajelor CAN.

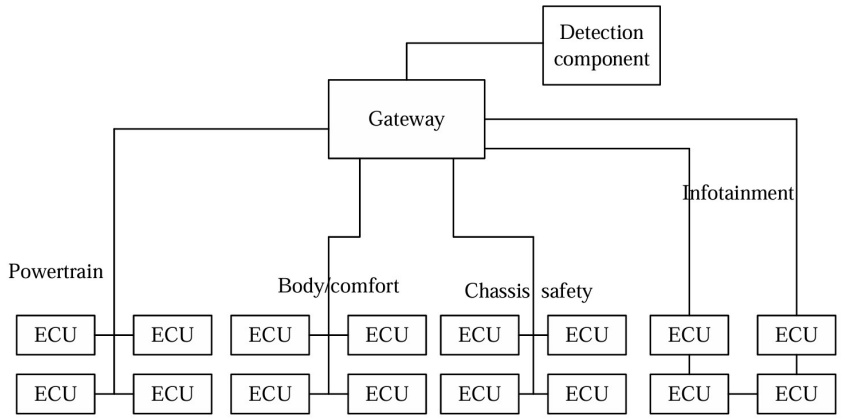


Figura 1: Structura topologică a sistemului electronic de control al vehiculului. Sursă [1].

## 3.2 Contextul securității în rețelele vehiculului

Pe fondul progresului tehnologic al vehiculelor, caracterizat printr-o complexitate tot mai mare și interconectivitate sporită, securizarea comunicațiilor interne între ECU-uri capătă o importanță majoră. Domeniul acesta a devenit un punct de interes major atât pentru mediul academic, cât și pentru industria auto. Studiile timpurii, precum cele ale lui Wolf et al [2], au fost printre primele care au identificat vulnerabilități serioase în rețelele interne ale vehiculului, precum lipsa autentificării mesajelor CAN sau posibilitatea de control de la distanță a funcțiilor sensibile (ex: blocarea ușilor). Ulterior, Koscher et al [3] au demonstrat, prin reverse engineering , generare de mesaje invalide și interceptarea traficului de date, că vehiculele moderne sunt expuse la riscuri din cauza protecției slabe a magistralei, lipsa autentificării și utilizarea

unor protocoale sensibile la atacuri. Un exemplu concret este prezentat de Zhou și Li [1], care descriu un atac de tip Denial of Service asupra sistemului de avertizare al vehiculului, realizat prin transmiterea de mesaje CAN malițioase imediat după cele legitime, cu scopul de a le anula efectul. Sistemul propus de autori detectează astfel de anomalii prin analiză temporală și semantică, clasificând alertele (figura 2) în trei niveluri de severitate - non-critical, critical și severe - și permițând o reacție proporțională în funcție de gravitatea situației, în care decizia finală revine șoferului. Structura logică a sistemului este ilustrată în figura 1, unde se evidențiază poziționarea componentului de detecție în gateway, ceea ce permite monitorizarea întregului trafic CAN fără a interveni direct asupra ECU-urilor. Raportul făcut de Computest [4] evidențiază un decalaj: deși vulnerabilitățile descoperite au fost remediate în modelele ulterioare de mașini, vehiculele deja comercializate nu au primit actualizări, rămânând permanent expuse la aceleași breșe de securitate. Această situație evidențiază necesitatea unui ciclu de suport software mai responsabil din partea producătorilor auto.

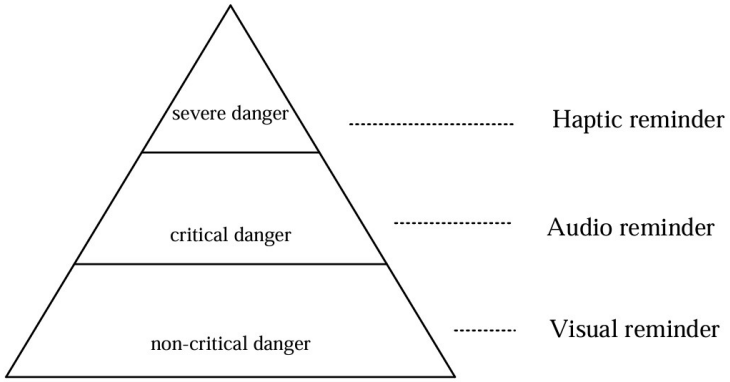


Figura 2: Mecanism de răspuns în trei niveluri. Sursă [1].

## 3.3 Protocoale de comunicație și soluții de protecție existente

În funcție de producătorul și de tipul mașinii, rețelele interne folosesc mai multe tehnologii de comunicație, cum ar fi Local Interconnect Network (LIN), Controller Area Network (CAN), Media Oriented Systems Transport (MOST), Automotive Ethernet sau FlexRay. Se evidențiază principalele caracteristici și diferențe dintre aceste protocoale, în Figura 3. Pentru a combate vulnerabilitățile identificate frecvent în aceste rețele, au fost dezvoltate o serie de mecanisme de protecție. Printre soluțiile consacrate se numără codurile de autentificare a mesajelor (MAC), firewall-urile și sistemele de detecție a intruziunilor [5], menite să protejeze schimbul de date în rețeaua internă. Progresele ulterioare includ mecanisme de autentificare a mesajelor către mai mulți destinatari, grupuri de comunicație securizate, sisteme de gestiune a datelor și mecanisme de control al accesului plasate la niveluri mai înalte în arhitectură. De asemenea, Muter și Asaj [6] clasifică sistemele de detecție a intruziunilor în rețele auto în două categorii principale: bazate pe specificații (specification-based) și bazate pe anomalii (anomaly-based), analizând avantajele și limitele fiecăreia. Autorii propun utilizarea entro-

piei ca metrică pentru identificarea comportamentului anormal în traficul CAN, împreună cu strategii pentru gestionarea alertelor generate de aceste sisteme.

| **Bus** | **LIN** | **CAN** | **FlexRay** | **MOST** | **Bluetooth** |
|---|---|---|---|---|---|
| **Adapted For** | Low-level Subnets | Soft Real-Time | Hard Real-Time | Multimedia Telematics | External Communication |
| **Target Application Examples** | Door locking<br>Climate regulation<br>Power windows<br>Light, rain sensor | Antilock break system<br>Driving assistants<br>Engine control<br>Electronic gear box | Break-by-Wire<br>Steer-by-Wire<br>Shift-by-Wire<br>Emergency systems | Entertainment<br>Navigation<br>Information services<br>Mobile Office | Telematics<br>Electronic toll<br>Internet<br>Telediagnosis |
| **Architecture** | Single-Master | Multi-Master | Multi-Master | Multi-Master | Multi-Master |
| **Access Control** | Polling | CSMA/CA | TDMA<br>FTDMA | TDM<br>CSMA/CA | TDMA<br>TDD |
| **Transfer Mode** | Synchronous | Asynchronous | Synchronous<br>Asynchronous | Synchronous<br>Asynchronous | Synchronous<br>Asynchronous |
| **Data Rate** | 20 kBit/s | 1 MBit/s | 10 MBit/s | 24 MBit/s | 720 kBit/s |
| **Redundancy** | None | None | 2 Channels | None | 79 Frequencies |
| **Error Protection** | Checksum<br>Parity bits | CRC<br>Parity bits | CRC<br>Bus Guardian | CRC<br>System Service | CRC<br>FEC |
| **Physical Layer** | Single-Wire | Dual-Wire | Optical Fiber<br>Dual-Wire | Optical Fiber | Air |

Figura 3: Caracteristici esențiale ale protocoalelor auto pentru comunicații interne și externe în vehicule. Sursa: [2]

## 3.4 CAN FD – Extinderea protocolului CAN clasic

Pe măsură ce industria auto devine tot mai complexă, necesitatea transmiterii unor volume mai mari de date și a unei viteze sporite de comunicare a condus la extinderea standardului CAN clasic. Astfel a fost introdus CAN FD (Flexible Data-Rate), o versiune îmbunătățită a protocolului original, propusă de Bosch în 2012.

Spre deosebire de CAN-ul tradițional, care limitează câmpul de date la 8 octeți și funcționează cu o rată fixă de transmisie, CAN FD introduce o serie de îmbunătățiri semnificative. Permite transmiterea de până la 64 de octeți în cadrul unui singur mesaj, precum și utilizarea unei rate de transfer crescute în faza de date (data phase), prin activarea bitului BRS (Bit Rate Switch). Aceste caracteristici contribuie la reducerea latenței și la creșterea eficienței în cazul transmisiilor de mesaje mari. De asemenea, protocolul este compatibil cu nodurile CAN clasice. Această extensie este documentată în specificația tehnică publică oferită de TekEye [7], care explică în mod clar diferențele structurale față de CAN 2.0, introducerea câmpurilor ESI și BRS, precum și detalii despre arbitrarea și codurile CRC folosite. În figura 4 vedeți o comparație vizuală între cele două formate.

Sistemele moderne de asistență la condus și alte funcții ale vehiculului utilizează tot mai frecvent CAN FD pentru a asigura o comunicare rapidă și fiabilă. Însă, odată cu avantajele transmisiei extinse, crește și suprafața de atac, mai ales în fața atacului de tip flooding, necesitând mecanisme de protecție adaptate noilor dimensiuni de cadru.

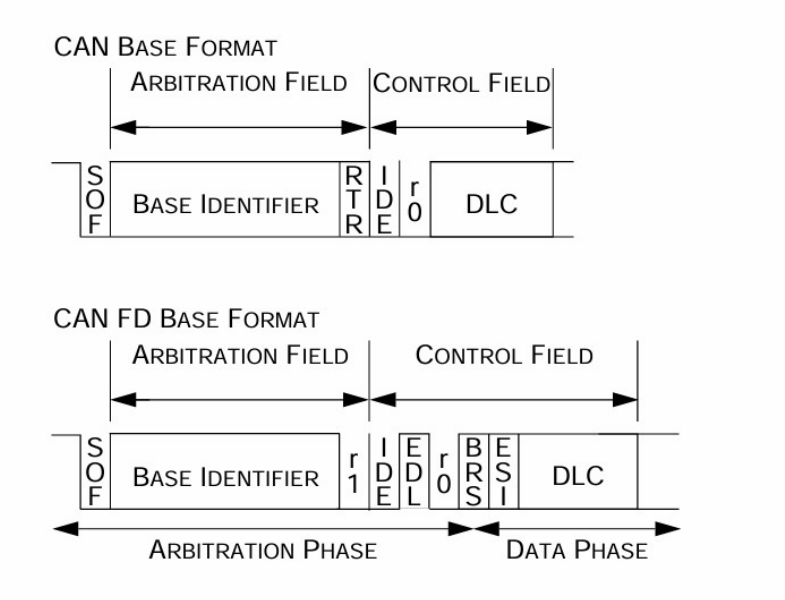


Figura 4: Compararea formatului de cadru CAN clasic cu cel CAN FD. Sursa: [7]

## 3.5 Standardizare în securitatea vehiculului

Paralel cu dezvoltările tehnice din domeniul automotive, au fost inițiate și numeroase demersuri de standardizare care stabilesc cerințe clare privind securitatea cibernetică a vehiculelor. Printre cele mai importante se numără ISO/SAE 21434 [8], care reglementează ingineria securității auto, SAE J3061 [9], axat pe ciclul de viață sigur al software-ului, și standardele ETSI pentru comunicațiile V2X între vehicule și infrastructură. Proiectul HoliSec [10] contribuie la acest efort prin furnizarea de specificații detaliate pentru schimbul de date între ECU-uri, însă fără a include și scenarii practice de aplicare. În completare, CyReV [11] propune o arhitectură auto rezilientă, oferind o bază de proiectare pentru vehicule sigure și indicatori clari pentru evaluarea soluțiilor noi.

## 3.6 Testbed-uri și limitările soluțiilor comerciale

Aceste eforturi reflectă și o schimbare de paradigmă în procesul de proiectare a vehiculelor moderne, care trebuie să atingă simultan obiective precum reducerea costurilor, creșterea fiabilității și scurtarea duratei de dezvoltare.

În acest context, platformele experimentale de testare – așa-numitele testbed-uri joacă un rol din ce în ce mai important. Acestea oferă un cadru controlat în care pot fi simulate atacuri, evaluate contramăsuri și testate funcții de siguranță fără a pune în pericol persoane sau echipamente reale. Conform analizei realizate de Mahmood et al [12], metode precum testarea prin penetrare, fuzzing sau scanarea vulnerabilităților sunt din ce în ce mai folosite, însă infrastructura necesară este adesea dificil de accesat. Tabelul 5 sintetizează principalele testbed-uri identificate în literatura de specialitate, evidențiind suportul oferit pentru protocoalele CAN,

LIN, FlexRay (FLR) și MOST. Se observă o predominanță a testbed-urilor dedicate protocolului CAN, acestea reprezentând infrastructura de bază pentru analiza securității vehiculului. De asemenea, multe platforme comerciale sau specializate au o acoperire limitată sau lipsesc în ceea ce privește protocoalele mai noi sau specifice.

| Testbed Name | CAN | LIN | FLR* | MOST | Ref.* |
|---|---|---|---|---|---|
| Open Car Testbed and Network Experiments (OCTANE) | ✓ | ✓ | ✓ | ✓ | [6] |
| Mobile Testing Platform | ✓ | N/A | N/A | N/A | [36] |
| Cyber Assurance Testbed for Heavy Vehicle Electronic Controls | ✓ | N/A | N/A | N/A | [14] |
| Testbed for Automotive Cybersecurity | ✓ | N/A | N/A | N/A | [22] |
| Testbed for Security Analysis of Modern Vehicle Systems | ✓ | N/A | N/A | N/A | [59] |
| Portable Automotive Security Testbed with Adaptability (PASTA) | ✓ | N/A | N/A | N/A | [56] |
| Hardware-in-loop based Automotive Embedded Systems Cybersecurity Evaluation Testbed | ✓ | N/A | N/A | N/A | [43] |

Figura 5: Compararea testbed-urilor existente în domeniul securității comunicațiilor auto. Sursa: [12]

Deși în prezent există soluții comerciale avansate pentru testarea și analiza rețelelor auto (precum Vector CANoe sau ETAS INCA), acestea sunt de cele mai multe ori costisitoare, greu de configurat și dependente de hardware specializat. Mai mult, accesul la sursa codului este restricționat, ceea ce limitează posibilitatea de extindere sau adaptare la scenarii noi. Astfel de soluții sunt adecvate pentru validarea industrială, dar nu răspund nevoilor din mediul educațional sau cercetare deschisă, unde flexibilitatea și costul redus sunt prioritare. În plus, ele nu sunt întotdeauna potrivite pentru simularea comportamentului sub atac sau în condiții de eroare controlată.

O alternativă notabilă la soluțiile comerciale este testbed-ul PASTA (Portable Automotive Security Testbed with Adaptability) [13]. Acesta oferă o infrastructură portabilă și flexibilă pentru simularea atacurilor cibernetice asupra vehiculelor, fiind adaptabil la diferite configurații ECU. Totuși, deși este open-source și accesibil din punct de vedere financiar, PASTA presupune utilizarea unor componente hardware dedicate, ceea ce îl face mai dificil de utilizat în contexte strict software sau în medii educaționale cu resurse limitate.

## 3.7 Simulatoare open-source și aplicații educaționale

Există un interes tot mai mare în comunitatea academică pentru dezvoltarea unor instrumente open-source care să permită simularea și testarea rețelelor auto într-un mod accesibil și reproductibil. Platforme precum CARLA [14] sau SUMO [15] s-au impus deja ca repere în simularea traficului și a conducerii autonome, datorită naturii lor deschise, structurii modulare și comunităților active care contribuie constant la dezvoltarea lor. Un exemplu notabil în această direcție este lucrarea lui Wu et al [16], în care simulatorul CARLA este folosit pentru a evalua modul în care sistemul de detecție vizuală reacționează la camuflajele aplicate pe suprafața mașinii. Autorii propun un cadru de testare pentru evaluarea riscurilor cibernetice într-un mediu virtual controlat, demonstrând potențialul acestor platforme în cercetarea aplicată din domeniul securității auto. Astfel de abordări evidențiază utilitatea simulatoarelor virtuale în explorarea vulnerabilităților comunicației vehiculului și în testarea reacțiilor sistemelor autonome în prezența unor factori perturbatori controlați.

În mod similar, instrumentele destinate testării comunicației CAN ar trebui să urmeze aceeași filosofie, oferind posibilitatea de personalizare a scenariilor, integrare facilă cu alte simulatoare și o experiență de utilizare intuitivă, adaptată atât începătorilor, cât și utilizatorilor avansați.

Platformele virtuale de simulare CAN pot juca un rol esențial în procesul educațional, oferind studenților oportunitatea de a învăța concepte avansate de comunicație auto, diagnosticare electronică și securitate cibernetică fără riscuri sau costuri ridicate. În acest context, The Car Hacker's Handbook [17] oferă exemple concrete de testbench-uri CAN educaționale, ușor de construit și utilizat în scenarii de simulare și analiză a comunicației auto. Aceste soluții pot fi folosite pentru învățarea interactivă, susținerea laboratoarelor practice și chiar pentru dezvoltarea de prototipuri software aplicabile în proiecte reale. Mai mult, prin simularea comportamentului sub atac sau în condiții anormale, astfel de instrumente pregătesc utilizatorii pentru provocările reale ale industriei auto de mâine.

Contextul actual evidențiază o lipsă acută de soluții accesibile, deschise și flexibile care să permită testarea comunicației CAN într-un mod realist și sigur. Cele mai multe platforme existente fie sunt greu de accesat din punct de vedere tehnic și financiar, fie nu permit o explorare detaliată a comportamentului sistemului în scenarii de atac sau defecțiuni. Astfel, se conturează clar nevoia unei soluții inovatoare care să ofere un echilibru între ușurința utilizării, modularitate și capacitatea de integrare cu simulatoare moderne.

# 4 SOLUȚIA PROPUSĂ

Lucrarea de față răspunde acestei nevoi prin propunerea unei platforme virtuale educaționale, care simulează comunicarea între ECU-uri într-un mediu controlat, vizual și adaptabil, adresând direct limitările observate în testbed-urile tradiționale.

## 4.1 Obiectivele Platformei

Platforma de față își propune să creeze un mediu interactiv unde se pot simula și testa rețelele Controller Area Network (CAN), care să permită utilizatorilor să dezvolte și să testeze scenarii de comunicare între diverse ECU-uri, să monitorizeze traficul CAN în timp real și să evalueze vulnerabilitățile rețelelor în condiții controlate. Astfel, utilizatorii pot simula atât comportamentele normale, cât și atacurile cibernetice, observând semnele pe care le are un sistem când este compromis. Un alt obiectiv important al platformei constă în rolul său educativ. Studenți, cât și profesioniști din domeniul IT și inginerie auto vor putea să înțeleagă în profunzime arhitectura rețelelor CAN, să aplice concepte teoretice în scenarii practice și să dobândească abilități esențiale în testarea și protejarea comunicațiilor auto.

## 4.2 Tehnologii Utilizate

CARLA [14] (Car Learning to Act) este un simulator open-source, larg utilizat în cercetarea privind conducerea autonomă și sistemele avansate de asistență la condus (ADAS). Acesta oferă un mediu realist, complet personalizabil, unde se pot simula comportamente specifice automobilului, senzori virtuali precum GNSS, IMU, LiDAR sau camere, și interacțiuni cu infrastructura rutieră. În această lucrare, CARLA servește drept mediu de simulare pentru vehiculul virtual, furnizând date în timp real care sunt utilizate pentru generarea pachetelor CAN în cadrul platformei. Integrarea cu CARLA permite testarea logicii de comunicare în contexte dinamice, fără riscuri fizice și fără a necesita echipamente specializate, facilitând validarea în condiții apropiate de realitate.

O parte esențială a unei aplicații software moderne o reprezintă interfața grafică. Alegerea tehnologiei potrivite influențează experiența utilizatorului final, flexibilitatea și scalabilitatea programului. Printre cele mai folosite tehnologii pentru dezvoltarea unei interfețe grafice (GUI) se numără Tkinter, wxPython și PyQt5. Datorită integrării cu Qt, care oferă o gamă largă de widget-uri și un model orientat pe obiect, PyQt5 a devenit o variantă optimă pentru dezvoltarea de software pentru multe aplicații comerciale și inițiative de cercetare [18].

Aceasta facilitează dezvoltarea rapidă a interfețelor sofisticate, permițând ajustarea și extinderea funcționalităților după necesități. De asemenea aduce un avantaj considerabil prin integrarea facilă cu alte biblioteci Python pentru prelucrarea și vizualizarea datelor, aspecte fundamentale pentru monitorizarea și analiza traficului CAN în timp real. Totodată, datorită suportului multiplatformă (Windows, Linux, macOS), aplicațiile dezvoltate pot funcționa pe diverse sisteme, fără a necesita ajustări suplimentare, ceea ce crește flexibilitatea și accesibilitatea acestora. În studiul realizat de Yin et al [18], PyQt5 a fost utilizat pentru dezvoltarea unei interfețe grafice intuitive care susține un sistem de detecție a obiectelor în timp real. Acest sistem combină algoritmul YOLOv8 îmbunătățit pentru detecția vehiculelor, cu o interfață vizuală bazată pe PyQt5, care permite utilizatorilor să încarce imagini sau video-uri, să vizualizeze rezultatele detecției și să ajusteze parametrii în mod dinamic. Astfel, PyQt5 reprezintă o alegere optimă pentru dezvoltarea unei interfețe grafice în cadrul platformelor de simulare CAN, datorită echilibrului său între performanță, flexibilitate și ușurință în utilizare.

## 4.3 Caracteristici și Funcționalități Cheie

Această secțiune descrie componentele esențiale ale platformei, evidențiind atât arhitectura modulară, cât și funcționalitățile implementate pentru a simula și analiza comunicațiile CAN în vehicule. Platforma a fost proiectată să fie interactivă, extensibilă și ușor de utilizat, integrând atât un simulator realist, cât și o interfață grafică intuitivă pentru observarea în timp real a fluxului de mesaje CAN.

### 4.3.1 Simularea comunicației ECU în CARLA

Platforma comunică bidirecțional cu simulatorul CARLA (figura 6): toate comenzile transmise către mașina virtuală trec prin platformă, iar răspunsurile și pachetele emise de platformă modifică comportamentul vehiculului din simulare.

Figura 6: Simulatorul Carla

De asemenea, platforma oferă o vizualizare grafică clară a modulului de comunicare dintre

ECU-uri (Electronic Control Units), printre care se numără cele responsabile de controlul motorului, al frânelor sau al sistemului de iluminat. Prin intermediul platformei, utilizatorul poate analiza întregul circuit al unui pachet CAN, de la momentul generării de către ECU-ul inițial, prin central gateway și până la ECU-ul final către care este trimis.

Această animație dinamică oferă o reprezentare intuitivă a fluxului de date din rețea, evidențiind pachetele CAN prin culori diferite, în funcție de identificatorul lor (can-id).

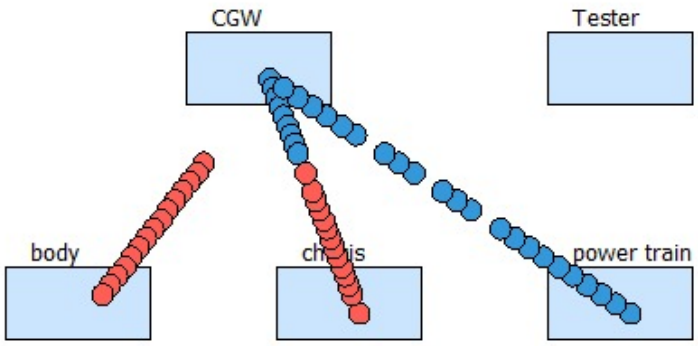


Figura 7: Comunicare intre ecu-uri

Vizualizatorul poate fi deschis cu ajutorul butonului ilustrat în figura 8.

Figura 8: Deschide schema CAN (Ctrl+N)

### 4.3.2 Interfață Grafică Interactivă

Platforma este dotată cu o interfață grafică intuitivă, bazată pe PyQt5, care permite utilizatorilor să vizualizeze și să interacționeze cu rețeaua CAN într-un mediu clar și intuitiv. Interfața integrează un set de butoane funcționale, fiecare dedicat unei acțiuni specifice, cum se poate observa în figura 9.Pentru butoane a fost folosită sintaxa nativă a emoji-urilor unicode, compatibilă cu majoritatea fonturilor moderne. Nu au fost descărcate imagini sau fonturi externe, ele au fost căutate pe emojipedia. Pentru eficiență sporită, interfața oferă scurtături de tastatură intuitive pentru fiecare acțiune importantă, reducând timpul de navigare între componentele platformei.

**Interfața grafică este împărțită în secțiuni logice:** bara de comenzi (butoanele), zona de tabele CAN, editorul JSON și vizualizatorul grafic al rețelei. Fiecare componentă este încărcată dinamic în funcție de acțiunea utilizatorului, menținând claritatea vizuală.

Figura 9: Butoanele din platforma

### 4.3.3 Tabelele de afișare pentru pachetele CAN

Butonul din figura 10 se ocupă de componenta principală a interfeței, care este reprezentată de tabele care afișează în timp real pachetele CAN detectate. Acestea sunt:

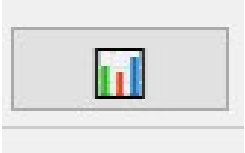

Figura 10: Comută tabelul report/command (Ctrl+T)

- **Tabel cu pachete de tip command**

  În tabelul 11 se pot observa pachetele de tip command care provin din simulator, dar și o bară de căutare pentru găsirea anumitor tipuri de pachete. Fiecare pachet are câte o culoare distinctă în funcție de ID.

Pachete CAN Procesate (SIMULATOR)

Cauta în tabel...

| | ID | Data | Source | Destination | Name | Level | Type | Period | Data Size | CARLA Var |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 47 | 1023 | chasis | power train | accelerator pedal operation ... | command | dummy | 10 | 2 | throttle |
| 2 | 88 | 15 | chasis | power train | steeering wheel operation ... | command | operation | 10 | 2 | steer |
| 3 | 47 | 1023 | chasis | power train | accelerator pedal operation ... | command | dummy | 10 | 2 | throttle |
| 4 | 88 | 15 | chasis | power train | steeering wheel operation ... | command | operation | 10 | 2 | steer |
| 5 | 47 | 1023 | chasis | power train | accelerator pedal operation ... | command | dummy | 10 | 2 | throttle |

Figura 11: Tabel pentru pachete de tip command din Simuulator

- **Tabel cu pachete de tip report**

În tabelul 23 se pot observa toate pachetele de report care transmit toate informațiile importante din mașină, de la viteză și turație, până la poziționarea GPS a mașinii. Aceste informații sunt foarte importante si nu ar trebui sa fie accesibile persoanelor neautorizate.

### 4.3.4 Structura Pachetelor CAN în Format JSON

Pachetele CAN utilizate în platformă sunt definite în fișierul `CAN-ID.json`, preluat și adaptat din proiectul PASTA [13], în care sunt descrise structurile pachetelor de tip `command`, `report` și `alert`. Fiecare pachet conține câmpuri esențiale precum: **can-id** (identificatorul unic al pachetului), **type** și **level** (tipul și nivelul pachetului, de exemplu command sau report), **source** și **execution** (modulele care comunică între ele). Există și un câmp **carlaVar** folosit pentru a lega pachetul de o variabilă din simulatorul CARLA (de exemplu, poziția GNSS, accelerația longitudinală, starea direcției etc.).

```
"36": {
    "source"     :"power train",
    "execution"  :"chasis",
    "name"       :"break output indicator",
    "level"      :"report",
    "type"       :"output",
    "period"     :10,
    "datasize"   :2,
    "min"        :0,
    "max"        :1023,
    "carlaVar"   :"brake"
},
```

Acest format oferă un nivel înalt de control asupra generării și injectării pachetelor CAN. Câmpul carlaVar leagă semnalul de o variabilă reală din simulatorul CARLA, permițând corelarea valorilor din platformă cu acțiuni reale (ex: apăsarea pedalei de accelerație). Acest format este utilizat atât pentru definirea pachetelor în fișierul JSON de bază, cât și pentru editarea manuală și injectarea directă din interfața platformei.

### 4.3.5 Vizualizare pachete în format CAN FD

Pentru ca utilizatorul să înțeleagă mai bine cum funcționează pachetele CAN, poți da dublu click pe un pachet pentru a-l deschide în format CAN-FD. În fereastra deschisă din figura 12 poți vizualiza pachetul CAN-FD în format binar într-un desen, dar și în format decimal și hexazecimal în tabel. Se poate observa în acest pachet că BRS are valoarea 1, așadar datele de la ESI până la CRC vor fi trimise de 2 ori mai rapid decât restul.

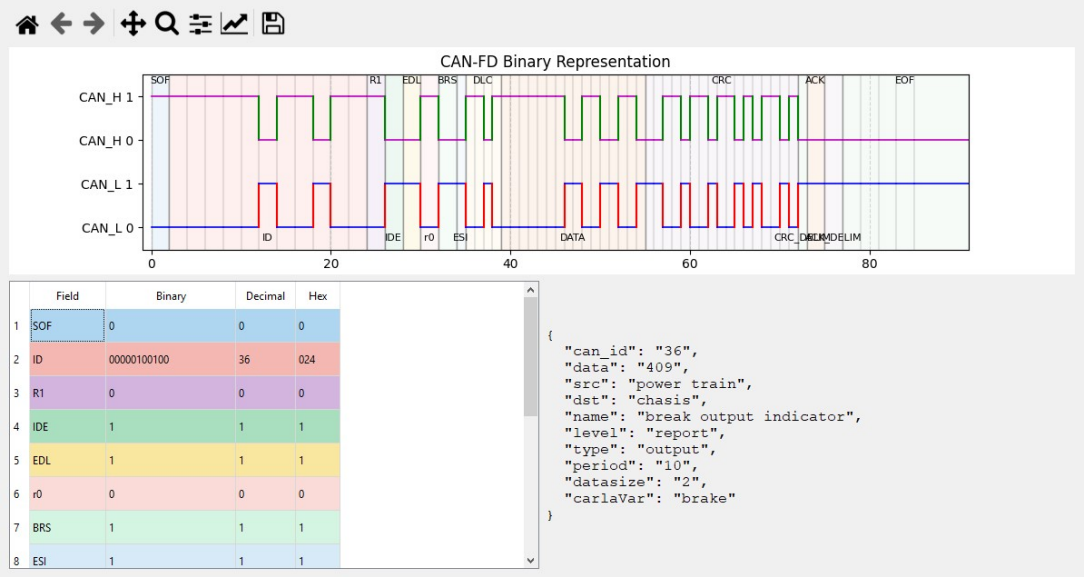


Figura 12: Vizualizare pachet în format CAN FD

În plus, butonul din figura 13 deschide în format CAN FD ultimul pachet injectat în rețea pentru a facilita verificarea imediată a conținutului transmis.

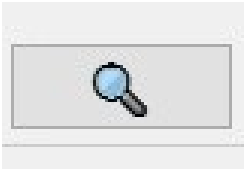

Figura 13: Vizualizare pachet

### 4.3.6 Injectarea pachetelor în platformă

În figura 14 este prezentat butonul care controlează vizibilitatea tabelului asociat pachetelor CAN injectate de utilizator. Introducerea pachetelor injectate într-un alt tabel este utilă pentru a reduce aglomerarea interfeței și pentru a permite focalizarea pe alte componente ale aplicației. Pachetele introduse sunt afișate imediat în interfață și transmise către rețeaua CAN virtuală. Dacă sunt asociate unei variabile din simulator, efectele lor se propagă automat în mediul CARLA. Platforma pune la dispoziție mai multe metode prin care aceste pachete pot fi injectate, fiecare adaptată unui scenariu diferit:

Figura 14: Comută tabelul simulatorului (Ctrl+R)

- **Importul de pachete din fișiere JSON.**
  Butonul din figura 15 este folosit pentru importul pachetelor ce se află în fișiere externe de tip .json. În plus, pachetele se pot importa și cu drag and drop. După ce pachetele sunt importate în platformă, ele sunt injectate direct în simulator.

Figura 15: Importă fișier JSON (Ctrl+O)

- **Editarea manuală a pachetelor.**
  Prin apăsarea butonului ilustrat în figura 16, se deschide editorul JSON, unde utilizatorul poate scrie de la zero un pachet sau edita unul deja injectat anterior.

Figura 16: Arată/Ascunde editorul de JSON manual (Ctrl+E)

- **Validarea și trimiterea pachetului.**
  După ce pachetul a fost scris în mod corect în editor, utilizatorul poate trimite pachetul către rețea apăsând butonul de procesare de sub editor (figura 17). La apăsarea acestui buton, platforma verifică automat dacă structura JSON respectă formatul așteptat. Dacă pachetul este valid, acesta este injectat instantaneu în rețeaua CAN virtuală și devine vizibil în tabelul de pachete active.

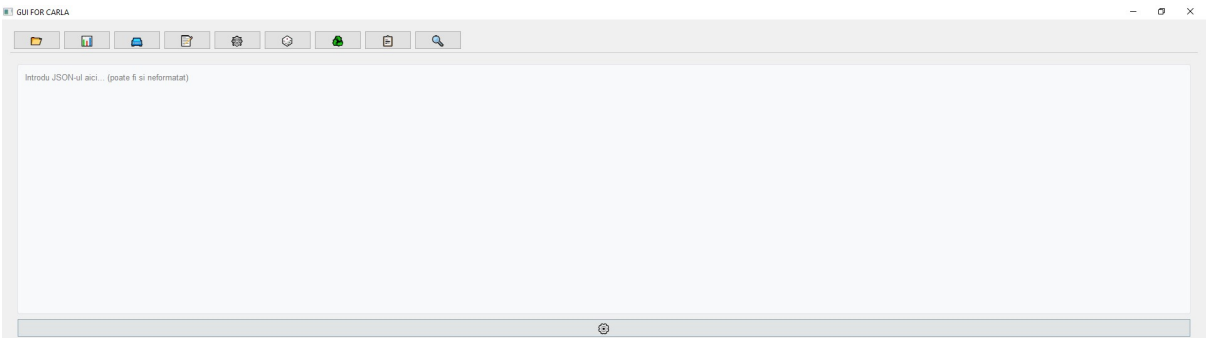


Figura 17: Editor de JSON manual

### 4.3.7 Exemplu de utilizare: modificarea vitezei vehiculului

Prin modificarea valorii unui pachet CAN de tip `command` legat de sistemul de accelerație, utilizatorul poate influența în mod direct comportamentul vehiculului simulat. Pachetul ajustat

este trimis în rețea, iar efectul se reflectă imediat în simulatorul CARLA. Interfața grafică afișează actualizarea în tabelele de procesare, după cum se poate observa în figura 18.

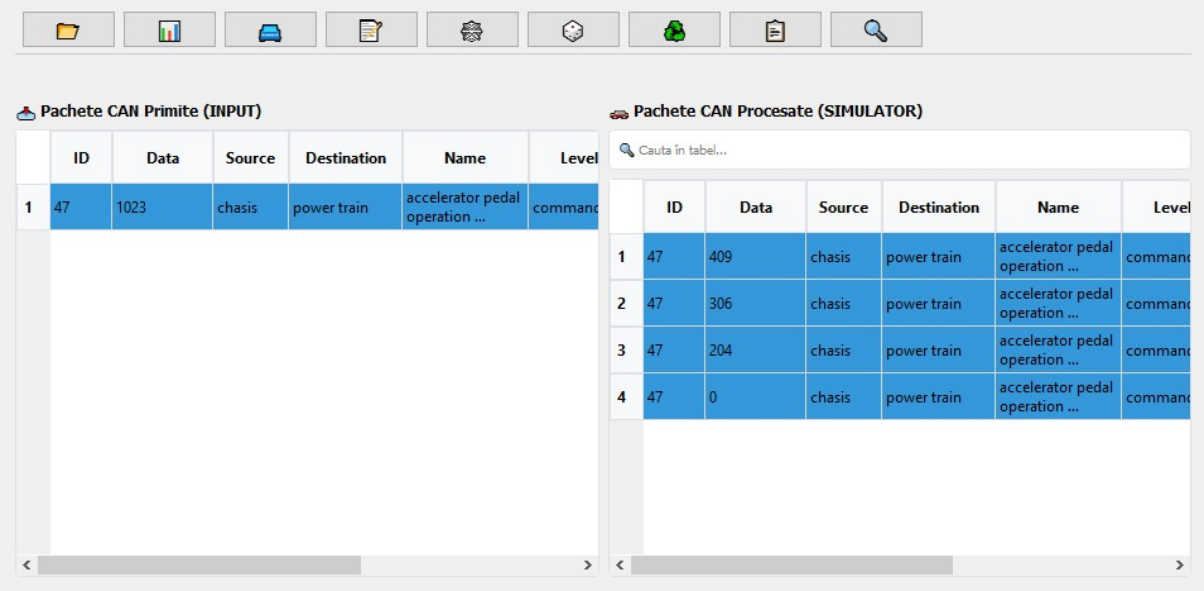


Figura 18: Exemplu scenariu de utilizare

Prin funcționalitățile sale diverse și interfața prietenoasă, platforma propusă facilitează explorarea rețelelor CAN într-un mod accesibil. Aceasta devine astfel un instrument didactic valoros, dar și o bază solidă pentru testări avansate. Capitolul următor detaliază implementarea tehnică a acestei soluții.

# 5 DETALII DE IMPLEMENTARE

Capitolul de față oferă o privire detaliată asupra modului în care a fost implementată platforma. Se prezintă atât structura generală a codului, cât și componentele esențiale dezvoltate: interfața grafică, procesarea pachetelor CAN, integrarea cu simulatorul, precum și suportul pentru senzori. Sunt incluse explicații despre procesarea pachetelor, comunicarea cu simulatorul și mecanismele din spatele interfeței grafice și ale atacurilor simulate. În partea de jos, puteți vedea structura de fișiere.

```
C:.
    CAN_ID.json
    can_json_processor.py
    can_simulator.py
    constants.py
    packet_viewer.py
    README.md
    spammer.py
    vcd_viewer.py
    visualizer.py
```

## 5.1 Structura aplicației și rolul fișierelor

Comunicarea cu simulatorul CARLA este gestionată de fișierul can-simulator.py, care extrage în timp real informații precum viteza vehiculului, poziția GPS sau evenimentele de coliziune. Aceste date sunt puse în pachete CAN folosind clasa CANPacket, utilizând structura din CAN-ID.json. Pachetele astfel generate sunt transmise către interfața grafică, printr-o conexiune cu can-json-processor.py. Ulterior, pachetele sunt afișate în tabelele din platformă, unde pot fi filtrate sau analizate. Vizualizatorul grafic (CanNetworkVisualizer) redă în mod animat traseul pachetelor între ECU-uri, oferind o imagine intuitivă a traficului CAN. În plus, utilizatorul poate deschide orice pachet într-o fereastră generată de clasa PacketViewer, unde acesta este reprezentat în detaliu, atât sub formă tabelară (binar, zecimal, hexazecimal), cât și ca semnal CAN-FD.

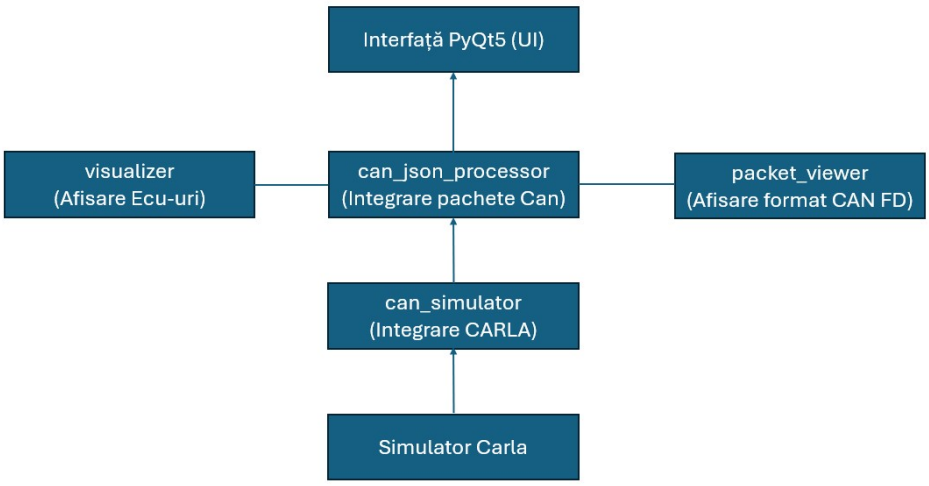


Figura 19: Modulele principale ale aplicației

## 5.2 Integrarea cu simulatorul CARLA (can-simulator.py)

Acest fișier face legătura cu simulatorul Carla. Fișierul a fost construit pe baza exemplului oficial Carla (manual-control), peste care a fost integrată clasa CanSimulator, unde se citesc toate tipurile de pachete din fișierul CAN-ID.json și se salvează în format can cu ajutorul clasei CANPacket. În secțiunile următoare este detaliată contribuția adusă acestui fișier.

### 5.2.1 KeyboardControl

Clasa se ocupă cu preluarea inputului utilizatorului de la tastatură și trimiterea către simulator. În momentul în care se trimite o comandă către simulator, se trimite și un pachet CAN către platformă.

```
elif event.key == K_RETURN:
    self.engine_on = not self.engine_on
    state = "ON" if self.engine_on else "OFF"
    world.hud.notification(f"Engine: {state}")
    self.can_simulator.add_command_packet("440")   # Trimite comanda CAN
```

În plus, în funcția parse-events, când se injectează pachete din platformă, aceasta le aplică în simulator, verifică ce tip de pachet este și ce date conține. Pentru a simplifica procesul de integrare, toate pachetele care sunt trimise către simulator sunt puse în input-packets.

```
for packet in self.window.input_packets:
    if packet["can_id"] == "440":
        if packet["data"] is None:
```

```
self.engine_on = not self.engine_on
state = "ON" if self.engine_on else "OFF"
world.hud.notification(f"Engine: {state}")
```

### 5.2.2 Logica principală de simulare CAN (CanSimulator)

Clasa a fost dezvoltată pornind de la repository-ul proiectului PASTA [13], de la care am preluat și fișierul CAN-ID.json, în care am mai adăugat câteva pachete care vor fi detaliate ulterior. Clasa reprezintă componenta esențială pentru integrarea protocolului CAN în simulatorul CARLA. Aceasta conectează datele definite în fișierul CAN-ID.json și variabilele interne ale vehiculului simulat prin CarlaVar. Ea folosește clasa CANPacket pentru încapsularea pachetului. Principalele funcționalități sunt:

- **Simularea periodică a trimiterei pachetelor**
  Funcția tick() este apelată la fiecare cadru al simulării și verifică când este timpul să trimită un anumit pachet. Ea este ajutată și de funcția tick din CANPacket.
- **Extracția valorilor din simulator**
  Valorile variabilelor precum accelerația, viteza, poziția GNSS sau starea luminilor sunt extrase din simulator și convertite în formatul așteptat de pachetul CAN.
- **Procesarea pachetelor de comandă**
  Pachetele de tip command sunt adăugate manual (ex: prin interfață sau tastatură) și sunt procesate separat, trimițând valori spre simulator (ex: pentru a comuta luminile).

Această clasă separă clar logica CAN de restul codului aplicației și permite extinderea sistemului cu ușurință, datorită organizării modulare și a legăturii directe cu variabilele interne ale simulatorului CARLA. Clasa CanSimulator folosește structura CANPacket pentru încapsularea fiecărui mesaj CAN, fie de tip „report", fie „command". CANPacket gestionează toate atributele necesare pentru descrierea unui pachet CAN, iar una dintre funcționalitățile cheie este metoda tick().

### 5.2.3 Integrarea senzorilor în simularea CAN

Pentru a extinde pachetele din CAN-ID, m-am folosit de senzorii disponibili în Carla:

- **CollisionSensor**
  Senzorul de coliziune ne transmite când mașina se lovește de un alt obiect. În plaformă, a fost legat la pachetul de tip report pentru airbag și la un pachet de tip command pentru "collision alert trigger".

```
"1200": {
    "source"     : "body",
```

```
    "execution"  : "chasis",
    "name"       : "collision alert trigger",
    "level"      : "command",
    "type"       : "operation",
    "period"     : 10,
    "datasize"   : 1,
    "min"        : 0,
    "max"        : 1,
    "carlaVar"   : "collision"
}
```

- **LaneInvasionSensor**
  Această clasă controlează senzorul de lane invasion, care trimite un semnal atunci când mașina atinge linia drumului. În cadrul platformei, senzorul a fost conectat la un pachet CAN de tip command, care, în teorie, alarmează șoferul de greșeala pe care o face.

```
"1200": {
    "source"     : "body",
    "execution"  : "chasis",
    "name"       : "collision alert trigger",
    "level"      : "command",
    "type"       : "operation",
    "period"     : 10,
    "datasize"   : 1,
    "min"        : 0,
    "max"        : 1,
    "carlaVar"   : "collision"
}
```

- **GnssSensor**
  Aceasta clasă se ocupă de senzorul de GNSS și arată unde se află vehiculul. În platformă au fost legate la 2 pachete can de tip report pentru latitudine și longitudine.

```
"2000": {
    "source"     : "GNSS",
    "execution"  : "CGW",
    "name"       : "Latitude",
    "level"      : "report",
    "type"       : "status",
    "period"     : 20,
    "datasize"   : 4,
    "min"        : -90.0,
    "max"        : 90.0,
    "carlaVar"   : "lat"
}
```

## 5.3 Interfața grafică (CanJsonProcessor)

Fișierul can-json-processor este componenta principală a interfeței grafice (GUI). Definește structura vizuală a platformei și definește toate componentele sale, de la tabele la interfața de căutare și la editor. Principalele caracteristici sunt:

- **Importul fișierelor JSON**

  Utilizatorul poate importa un fișier CAN în format JSON fie din meniu, fie prin drag-and-drop. Conținutul fișierului JSON este validat și convertit într-o structură de tip dicționar pentru a putea fi afișat în tabel și apoi să fie trimis către simulator.

```
for can_id, details in json_data.items():
    packet = {
        "can_id": can_id,
        "src": details.get("source", "N/A"),
        "dst": details.get("execution", "N/A"),
        "name": details.get("name", "N/A"),
        "level": details.get("level", "N/A"),
        "type": details.get("type", "N/A"),
        "period": details.get("period", "N/A"),
        "datasize": details.get("datasize", "N/A"),
        "carlaVar": details.get("carlaVar", "N/A"),
        "data": details.get("data", None)
    }
    self.add_packet_to_table_receive(packet)
```

- **Atacul de tip replay**
  Funcționalitatea de replay attack este realizată printr-un cod simplu, dar eficient. Pachetul este reintrodus în "input_packet" și tratat ca orice alt pachet nou, astfel se simulează un atac real:

```
def replay_last_packet(self):
    if self.replay_buffer:
        self.add_packet_to_table_receive(self.replay_buffer)
        self.input_packets.append(self.replay_buffer)
    else:
        QMessageBox.information(self, "Replay", "404")
```

- **CommandTableWindow**

  Această clasă creează un alt tabel în care au fost introduse toate pachetele de tip command, iar datele acestora pot fi modificate printr-un clic pe celula dorită și apoi introduse

în sistem prin dublu clic pe pachet. Pentru a permite editarea celulei corespunzătoare câmpului data, i-am activat opțiunea de editare prin setarea flagului corespunzător.

```
        self.table.itemClicked.connect(self.handle_item_click)
    def handle_item_click(self, item):
    if item.column() == 1:
        item.setFlags(item.flags() | Qt.ItemIsEditable)
        self.table.editItem(item)
```

## 5.4 Simularea grafică a comunicației CAN (CanNetworkVisualizer)

Această clasă se ocupă de animația transferurilor de pachete între ele. Aceasta ilustrează animația pachetelor CAN între diferite module ale vehiculului, oferind o reprezentare vizuală intuitivă a traficului de mesaje. Fiecare ECU sau entitate logică este reprezentată ca un dreptunghi colorat cu etichetă. Metoda animate_packets este apelată periodic de un QTimer și mută pachetul la fiecare cadru, până ajunge la destinație. După finalizarea animației, obiectul este eliminat din scenă pentru a elibera memoria.

```
    self.packet_timer = QTimer()
    self.packet_timer.timeout.connect(self.animate_packets)
    self.packet_timer.start(30)

def animate_packets(self):
    for packet in self.active_packets[:]:
        item = packet["item"]
        if packet["count"] < packet["steps"]:
            item.moveBy(packet["dx"], packet["dy"])
            packet["count"] += 1
        else:
            self.scene.removeItem(item)
            self.active_packets.remove(packet)
```

## 5.5 Vizualizarea detaliată a pachetelor CAN-FD (PacketViewer)

Această clasă oferă o vizualizare grafică avansată a structurii unui pachet CAN, reprezentat atât în format tabelar (binar, zecimal, hexazecimal), cât și în format grafic sub forma semnalelor CAN_L și CAN_H. Pentru afișarea datelor a fost folosit matplotlib, iar interfața este integrată cu PyQt5. Fiecare câmp este afișat cu o culoare diferită. Utilizatorul poate analiza bit cu bit structura unui pachet CAN-FD în timp real, inclusiv calculul automat al CRC-ului. Această secțiune generează structura binară a unui pachet CAN-FD pe baza valorilor extrase din câmpurile pachetului.

```
        binary_parts = [
            '0',  # SOF
            format(int(self.packet.get("can_id", 0)), '011b'),  # CAN_ID
            "0", # R1
            '1',  # IDE
            '1',  # EDL
            '0',  # r0
            '1',  # BRS
            '1',  # ESI
            format(dlc, '04b'),  # DLC
            format(data_value, f'0{data_length}b'),]
        binary_parts.append(self._compute_crc15(''.join(binary_parts)))
        binary_parts.append('1')       # CRC_DELIM
        binary_parts.append('0')       # ACK
        binary_parts.append('1')       # ACK_DELIM
        binary_parts.append('1' * 7)   # EOF
```

## 5.6 Generarea intensivă de trafic CAN (Spammer)

Pentru evaluarea performanței în condiții de trafic intens, platforma oferă posibilitatea de a genera și injecta rapid un număr mare de pachete CAN, pentru a observa cum gestionează sistemul situațiile de suprasarcină.

Implementarea se bazează pe o clasă dedicată, PacketSpammer, rulată într-un fir de execuție separat față de interfața grafică. Pentru a realiza acest paralelism, platforma utilizează mecanismul QThread din biblioteca PyQt5, care permite rularea în fundal a operațiunilor costisitoare, precum buclele repetitive. În acest mod, interfața rămâne fluentă, întrucât thread-ul secundar transmite semnale către cel principal ori de câte ori un nou pachet este pregătit pentru procesare sau afișare.

```
# În can_json_processor.py
self.spammer_thread = QThread()
self.spammer_worker = PacketSpammer()
self.spammer_worker.moveToThread(self.spammer_thread)

# În spammer.py
random_id = random.choice(keys)
details = self.packet_definitions[random_id]
data_value = random.randint(0, int(max_val))
packet = {
  "can_id": random_id,
```

```
  ...
  "data": data_value
}
self.packet_generated.emit(packet)
```

Acest mecanism permite rularea testelor de performanță într-un mod controlat și repetabil, contribuind la validarea stabilității platformei sub sarcină ridicată.

## 5.7 Personalizarea stilizarea interfeței

Un aspect important al aplicațiilor moderne este oferirea unei interfețe ușor de utilizat, cu un aspect plăcut. Așadar, platforma propusă utilizează sistemul Qt Style Sheets (QSS), un mecanism inspirat de CSS, folosit pentru a personaliza aspectul componentelor vizuale din PyQt5. Cele mai importante elemente stilizate sunt tabelele, fiindcă ele concentrează cele mai importante informații și acțiuni. Stilizarea lor a fost realizată prin aplicarea unui stil QSS dedicat, care definește culori, borduri, selecții și fonturi.

```
QTableWidget {
    border: 1px solid #dee2e6;
    border-radius: 4px;
    background: white;
}
...
```

În concluzie, fiecare modul prezentat contribuie la o platformă robustă și extensibilă, capabilă să simuleze și să analizeze în detaliu comunicația CAN, în contexte variate, inclusiv de securitate.

# 6 EVALUARE

Pentru a verifica funcționalitatea platformei și capacitatea acesteia de a simula și analiza comunicația CAN în diverse scenarii, au fost definite mai multe cazuri de testare, grupate în trei direcții principale: **testare funcțională**, **testare de performanță** și **testare în scenarii de securitate**.

## Testare funcțională

Rezultatele testării funcționale sunt sintetizate în Tabelul 1, evidențiind principalele funcționalități și comportamentul sistemului.

Tabela 1: Rezultatele testării funcționale

| **Funcționalitate** | **Descriere test și rezultat** | **Observații** |
|---|---|---|
| **Integrare cu CARLA** | A fost verificată legătura dintre `carlaVar` și variabilele reale din simulator (ex: viteză, GPS), comenzile au avut efect imediat. | Conexiunea cu CARLA s-a menținut stabilă pe tot parcursul testării. |
| **Importul fișierului JSON** | S-a testat încărcarea fișierului CAN-ID.json în interfață. | Fiecare pachet a fost identificat și clasificat corect. |
| **Injectare manuală** | S-au încărcat succesiv mai multe fișiere ce conțin pachete tip json, iar toate pachetele au fost afișate corespunzător în G.U.I. | Nu s-au semnalat erori sau pachete ignorate. |
| **Afișare pachete** | S-au testat trimiterea de pachete și din simulator, cât și din platformă. Fiecare a avut o culoare diferită în funcție de can-id. | Tabelul s-a actualizat în timp real fără întârzieri. |
| **Vizualizare flux CAN** | A fost testată afișarea grafică a fluxului de pachete între ECU-uri. Animația a indicat corect traseul și tipul fiecărui pachet. | Animația a fost fluidă |

După cum se observă din rezultate, toate componentele majore ale platformei au funcționat optim. Nu au fost identificate erori sau blocaje, iar integrarea cu simulatorul CARLA s-a realizat fără probleme. Mai mult, comenzile CAN transmise din platformă au generat reacții vizibile și corecte în simulator: trimiterea unui pachet asociat frânării a condus la oprirea vehiculului, iar injectarea unei comenzi de virare a modificat direcția de deplasare. Aceste reacții confirmă faptul că legătura dintre platformă și vehiculul simulat este nu doar stabilă, ci și funcțională din punct de vedere al controlului efectiv. Acest lucru validează stabilitatea și utilitatea aplicației în condiții normale de utilizare.

## Testarea Performanței

Testele de performanță au urmărit identificarea limitelor aplicației în scenarii cu volume mari de pachete CAN trimise succesiv, folosind `time.sleep(0)` pentru a simula un trafic intensiv.

| Număr pachete | Descriere test și rezultat | Resurse / Observații |
|---|---|---|
| **1.000** | Aplicația funcționează normal, dar cu o ușoară întârziere în animația grafică. | Timp: 0.10 sec<br>RAM: 207.81 MB |
| **10.000** | Doar animația s-a blocat în timpul testului, dar execuția platformei și a simulatorului CARLA a continuat fără întreruperi | Timp: 3.09 sec<br>RAM: 263.87 MB |
| **100.000** | Atât platforma cât și simulatorul s-au blocat temporar în timpul trimiterii pachetelor, dar și-au revenit după finalizare. | Timp: 87.12 sec<br>RAM: 778.19 MB |
| **1.000.000** | Sistemul a cedat complet. Atât simulatorul CARLA, cât și platforma grafică s-au blocat definitiv. | Test nereușit – blocaj complet |

Tabela 2: Testarea performanței la diferite volume de trafic CAN

**Mesaj de eroare pentru vizualizator:**

```
QEventDispatcherWin32::registerTimer: Failed to create a timer
(The current process has used all of its system allowance of handles
for Window Manager objects.)
```

Acesta indică faptul că s-a atins limita de handle-uri impusă de sistemul de operare Windows.

**Optimizare test:** Folosirea unei întârzieri minime `time.sleep(1e-9)` a permis rularea a 10.000 de pachete fără blocarea platformei.
**Timp:** 88.92 secunde **RAM:** 265.59 MB **CPU:** 60.0%

**Configurare sistem utilizat pentru testare:** Testele au fost efectuate pe un laptop echipat cu procesor **AMD Ryzen 5 4600H** (6 nuclee, 12 fire de execuție), 16 GB RAM la 3200 și placă video **NVIDIA GeForce GTX 1650 Ti**, rulând sistemul de operare Windows 10. În momentul testării, procesorul rula la o frecvență de 2.57 GHz.

## Testarea Securității

Au fost simulate următoarele atacuri informatice asupra rețelei CAN:

1. **Flooding.**

Atacul flooding constă în trimiterea unui număr mare de mesaje într-un interval scurt pentru a încetini sau bloca rețeaua. În platforma implementată, acest comportament este simulat cu ajutorul clasei dedicate PacketSpammer, care generează și emite automat pachete CAN. Procesul este declanșat printr-un buton special (figura 20) și rulează într-un fir de execuție separat (QThread), pentru a evita blocarea interfeței grafice.

Figura 20: Flooding Attack

2. **Replay.**
   Atacul Replay presupune redifuzarea unui mesaj CAN valid, dar într-un context temporal diferit. Acest lucru păcălește sistemul să accepte mesaje legitime în momente necorespunzătoare. Se poate declanșa fie printr-un buton dedicat (figura 21), fie prin dublu click pe un pachet din tabel.

Figura 21: Replay Attack

3. **Spoofing.**
   Spoofing constă în trimiterea de mesaje false pentru a compromite comunicarea dintre modulele ECU. Platforma permite modificarea și injectarea manuală a pachetelor prin interfața grafică dedicată (figura 22).

Figura 22: Deschide tabelul de comenzi CAN

## Rezultatele testelor de securitate

- **Flooding:** Platforma a continuat să proceseze pachetele, dar resursele de sistem s-au apropiat de limită.
- **Replay:** Pachetele au fost acceptate și procesate, fără verificări de sincronizare.

- **Spoofing:** Mesajele false au fost tratate ca valide și au fost procesate ca atare, ceea ce evidențiază lipsa unui mecanism de autentificare.

Toate testele au fost desfășurate într-un mediu controlat, fără utilizarea de echipamente hardware, iar execuția acestora a fost repetabilă. Acest lucru arată că platforma este potrivită pentru instruire, cercetare și testarea de prototipuri software. Scenariile validate demonstrează că platforma poate fi utilizată atât pentru înțelegerea comportamentului vehiculului în condiții normale, cât și în situații de risc cibernetic.

## 6.1 Justificarea reproducerii atacurilor din literatură în platforma propusă

Platforma dezvoltată simulează o rețea CAN și oferă funcționalități esențiale pentru monitorizarea, injectarea și manipularea pachetelor. Interfața grafică permite vizualizarea în timp real a traficului CAN, similar cu ceea ce s-ar putea observa prin conectarea unui dispozitiv la portul OBD-II al unei mașini reale. Deși platforma este software și nu interacționează cu hardware-ul vehiculului, comportamentul rețelei CAN este reprodus cu acuratețe.

Această abordare permite reproducerea parțială sau completă a mai multor atacuri descrise în articolul *Cyber-Security Internals of a Skoda Octavia vRS* [19]. Mai jos sunt prezentate atacurile relevante și corespondentul lor în platforma mea:

1. **Captură și redifuzare de mesaje CAN (Replay).** Articolul descrie un atac de tip replay aplicat asupra telecomenzii mașinii (key fob), în care semnalul RF este capturat și retransmis pentru a deschide vehiculul. Deși acest atac presupune semnal radio și rolling code, principiul general de replay se aplică și în rețelele CAN. Platforma permite salvarea unui pachet CAN și retransmiterea acestuia într-un moment ulterior, ceea ce reproduce fidel comportamentul unui atac de tip replay. De exemplu, un pachet CAN care comandă pornirea motorului poate fi înregistrat și redifuzat, chiar dacă proprietarul nu mai este prezent.
2. **Transmiterea de mesaje modificate (Injectare manuală).** În articol sunt evidențiate atacuri care presupun modificarea parametrilor vehiculului prin comenzi CAN, cum ar fi schimbarea pragului de tensiune pentru funcția start-stop de la 7.5V la 12.1V, ceea ce determină dezactivarea acesteia. Astfel de intervenții sunt realizate prin injectarea de mesaje CAN personalizate. Platforma permite modificarea conținutului pachetelor (ID și date) și trimiterea lor în rețea, simulând comportamentul unui atacator care trimite comenzi neautorizate.
3. **Observarea pasivă a traficului CAN.** Articolul descrie cum un atacator poate intercepta datele transmise pe magistrala CAN prin conectarea la portul OBD-II, folosind dispozitive precum USB2CAN sau CANtact. Este menționată capacitatea de a monitoriza în timp real datele transmise pe magistrală, inclusiv parametri precum viteza

motorului. Platforma oferă o funcționalitate similară, afișând toate mesajele CAN într-un tabel detaliat cu ID, tip și conținut, permițând analiza pasivă a rețelei.

4. **Transmiterea rapidă a mai multor pachete (Trafic intensiv).** Deși termenul „flooding" nu este folosit explicit în articol, autorii descriu trimiterea repetitivă de comenzi CAN pentru a modifica parametri sau a forța reacții din partea sistemului. Platforma permite generarea unui număr mare de pachete CAN într-un timp scurt, pentru a testa capacitatea de răspuns a rețelei și a identifica eventuale blocaje sau suprasolicitări.
5. **Acces la date senzoriale (ex. GNSS).** În articol sunt evidențiate riscuri privind scurgerea de date din sistemul de infotainment, inclusiv locația vehiculului. Platforma implementează pachete CAN de tip `report`, ce conțin coordonate GNSS simulate (latitudine, longitudine), demonstrând cum astfel de date pot fi interceptate sau utilizate în scopuri malițioase.

**Limitări.** Platforma nu reproduce atacuri ce implică acces fizic sau comunicații radio (precum captură SDR sau jamming), și nici interacțiuni avansate cu protocoale proprietare (ex. VW-TP 2.0). Remaparea ECU și execuția de cod într-un sistem real de infotainment nu sunt posibile.

**Concluzie.** Chiar dacă platforma funcționează în mediu software, ea reflectă cu fidelitate comportamentele critice ale unei rețele CAN reale. Acest lucru o face potrivită pentru testarea și înțelegerea riscurilor de securitate, oferind un cadru sigur, reproductibil și educațional pentru analiză.

## 6.2 Comparație cu soluții existente din industrie

Pentru a evalua poziționarea platformei dezvoltate în contextul actual al instrumentelor de testare CAN, a fost realizată o comparație cu cele mai cunoscute soluții comerciale și open-source. Tabelul 3 sintetizează caracteristicile-cheie ale acestora.

După cum se observă, platforma propusă este una dintre puținele care combină:

- **Accesibilitatea open-source**, fără licențiere restrictivă;
- **Interfață grafică intuitivă** pentru vizualizarea pachetelor și control complet asupra fluxului CAN;
- **Suport nativ pentru scenarii de atac**, fără echipamente hardware costisitoare;
- **Integrarea cu simulatorul CARLA**, oferă posibilitatea de a observa în timp real impactul comenzilor CAN asupra comportamentului vehiculului.

În timp ce soluțiile din industrie presupun adesea echipamente dedicate și sunt concepute pentru validare profesională, această platformă este gândită pentru scopuri educaționale și de cercetare, fiind accesibilă celor care doresc să exploreze securitatea rețelelor auto fără constrângeri hardware.

Tabela 3: Comparație între platforma propusă și alte soluții de testare CAN

| **Soluție** | **Open-source** | **GUI** | **Suport atacuri** | **Simulare 3D** | **Fără hardware** | **Observații cheie** |
|---|---|---|---|---|---|---|
| **Platforma propusă** | Da | Da | Da | Da | Da | Complet software, suport vizual și interactiv |
| CARLA-PASTA | Da | Limitat | Da | Da | Nu | Utilizabilă în cercetare și validare, dar dependentă de infrastructură hardware specifică |
| Vector CANoe | Nu | Da | Da (avansat) | Nu | Nu | Comercial, complex, orientat spre validare profesională |
| ETAS INCA | Nu | Da | Nu | Nu | Nu | Necesită ECU fizic; concentrat pe diagnoză și calibrare |
| CANtact + SavvyCAN | Da | Da | Da | Nu | Nu | O ustensilă educațională utilă, dar depententă de hardware |
| ICSim | Da | Da | Da | Nu | Da | Simulator simplu, orientat pe reverse engineering educațional, dar tot fără suport pentru simulare vehicul |
| UDSim | Da | Da | Da | Nu | Da | Axat pe protocoale de diagnostic, dar fără suport pentru simulare vehicul sau vizualizare în timp real |

**Resurse online pentru soluțiile prezentate:**
Pentru o explorare mai detaliată a unora dintre platformele open-source menționate în tabel, sunt disponibile următoarele link-uri oficiale:

- PASTA: `https://github.com/pasta-auto/CARLA-PASTA`
- Vector CANoe: `https://www.vector.com/int/en/products/products-a-z/software/canoe/`
- ETAS INCA: `https://www.etas.com/ww/en/products-services/software-testing-solution inca-software-products/`
- SavvyCAN: `https://github.com/collin80/SavvyCAN`
- ICSim: `https://github.com/zombieCraig/ICSim`
- UDSim: `https://github.com/zombieCraig/uds-server`

## 6.3 Limitări ale platformei

Deși platforma propusă reprezintă o soluție accesibilă și interactivă pentru simularea și testarea comunicației CAN, există câteva limitări importante care trebuie luate în considerare, mai ales în contextul utilizării în cercetare avansată sau aplicații industriale.

- **Lipsa conectivității cu rețele CAN reale (USB2CAN)**: Platforma funcționează

exclusiv în mediu virtual și nu oferă suport pentru conectarea directă la magistrale CAN fizice. Această alegere este adecvată pentru scopuri educaționale, dar limitează utilizarea în scenarii industriale, unde este necesară validarea pe echipamente reale.

- **Nu există un sistem automat de detecție a anomaliilor**: În forma actuală, platforma nu include funcționalități pentru identificarea automată a atacurilor sau comportamentelor suspecte. De exemplu, mesajele false (spoofing) sau traficul excesiv (flooding) nu declanșează alerte, ci sunt doar afișate în interfață.
- **Lipsa mecanismelor de criptare sau autentificare**: Pachetele CAN sunt procesate într-un format simplu, fără verificări de integritate sau semnături digitale. Deși acest aspect reflectă modul de funcționare al rețelelor CAN tradiționale, el limitează posibilitatea de a testa soluții moderne de securitate.
- **Integrarea limitată cu alte simulatoare**: În prezent, platforma este compatibilă doar cu simulatorul CARLA. Integrarea cu alte unelte, precum SUMO sau Matlab/Simulink, nu este încă realizată, deși este posibilă din punct de vedere tehnic.

Identificarea acestor limitări oferă un punct de plecare solid pentru direcțiile viitoare de dezvoltare. Obiectivul este extinderea platformei într-o soluție completă, capabilă să răspundă atât nevoilor educaționale, cât și celor din mediul industrial.

# 7 CONCLUZII

Această lucrare a avut ca obiectiv dezvoltarea unei platforme software capabile să simuleze comunicația CAN într-un mediu auto virtual, cu aplicabilitate în educație și cercetare. Un element definitoriu al platformei este integrarea cu simulatorul CARLA, care permite observarea în timp real a efectelor comenzilor CAN asupra comportamentului vehiculului, oferind astfel o legătură directă între datele injectate și reacțiile vehiculului simulat.

Platforma implementată oferă o interfață grafică intuitivă, animație în timp real a fluxului CAN și suport pentru acțiuni precum observarea, injectarea și redifuzarea de pachete. Pe baza acestor funcționalități, au fost simulate și testate atacuri cibernetice de tip spoofing, flooding și replay, în conformitate cu scenariile din literatura de specialitate. Rezultatele confirmă faptul că platforma poate emite și interpreta corect comenzi CAN, generând reacții specifice în simulator, precum schimbarea direcției sau modificarea vitezei vehiculului.

Testele de performanță au demonstrat că platforma poate gestiona eficient volume diferite de trafic CAN, evidențiind totodată punctele sale critice în scenarii cu încărcare ridicată. Fiind construită modular, aplicația poate fi ușor extinsă sau integrată cu alte instrumente software.

Limitările actuale – cum ar fi lipsa unei interfețe cu rețele CAN fizice, absența criptării și lipsa unui sistem de alertare automată – deschid direcții clare pentru dezvoltări viitoare. Printre pașii următori se numără integrarea cu componente hardware reale, implementarea unor mecanisme de detecție a anomaliilor și extinderea compatibilității cu alte simulatoare, precum SUMO sau Simulink.

În concluzie, platforma propusă contribuie semnificativ la înțelegerea comunicației CAN într-un mod reproductibil și interactiv, adresând cerințele actuale din educație și cercetare privind securitatea vehiculelor moderne. Lucrarea demonstrează fezabilitatea acestei soluții și oferă un punct de plecare solid pentru dezvoltări ulterioare cu aplicabilitate practică.

# BIBLIOGRAFIE


[1] Z. Qin and F. Li, "An intrusion defense approach for vehicle electronic control system," in *International Conference on Communication and Electronic Information Engineering (CEIE 2016)*. Atlantis Press, 2016, pp. 494–499, available: https://www.atlantis-press.com/article/25872603.pdf.

[2] M. Wolf, A. Weimerskirch, and C. Paar, "Security in automotive bus systems," in *Workshop on Embedded Security in Cars*. International Association for Cryptologic Research (IACR, 2004, pp. 1–13, available: https://www.weimerskirch.org/files/WolfEtAl_SecureBus.pdf.

[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE symposium on security and privacy*. IEEE, 2010, pp. 447–462, available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5504804.

[4] D. Keuper and T. Alkemade, "The connected car ways to get unauthorized access and potential implications," *Computest, Zoetermeer, The Netherlands, Tech. Rep*, 2018, available: https://www.computest.nl/documents/9/The_Connected_Car._Research_Rapport_Computest_april_2018.pdf.

[5] M. Müter, A. Groll, and F. C. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *2010 Sixth International Conference on Information Assurance and Security*. IEEE, 2010, pp. 92–98, available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5604050.

[6] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *2011 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2011, pp. 1110–1115, available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5940552.

[7] TekEye, "Can fd specification – a simple explanation of the can flexible data rate protocol," 2019. [Online]. Available: https://tekeye.uk/downloads/can_fd_spec.pdf

[8] C. Schmittner, G. Griessnig, and Z. Ma, "Status of the development of iso/sae 21434," in *European Conference on Software Process Improvement*. Springer, 2018, pp. 504–513, available: https://www.researchgate.net/profile/Christoph-Schmittner/publication/326896001_Status_of_the_Development_of_ISOSAE_21434_25th_European_Conference_EuroSPI_2018_Bilbao_Spain_September_5-7_2018_Proceedings/links/5dd280f092851c382f47068e/

Status-of-the-Development-of-ISO-SAE-21434-25th-European-Conference-EuroSPI-2018-Bilbao-Spain-pdf.

[9] C. Schmittner, Z. Ma, C. Reyes, O. Dillinger, and P. Puschner, "Using sae j3061 for automotive security requirement engineering," in *Computer Safety, Reliability, and Security: SAFECOMP 2016 Workshops, ASSURE, DECSoS, SASSUR, and TIPS, Trondheim, Norway, September 20, 2016, Proceedings 35*. Springer, 2016, pp. 157–170, available: https://www.researchgate.net/profile/Christoph-Schmittner/publication/307585960_Using_SAE_J3061_for_Automotive_Security_Requirement_Engineering/links/5bfac92ca6fdcc538819ce9c/Using-SAE-J3061-for-Automotive-Security-Requirement-Engineering.pdf.

[10] A. Yadav and C. Sandberg, "Specification of secure communication between ecus," HoliSec Project, Tech. Rep., 2018, available: https://autosec.se/wp-content/uploads/2018/04/HOLISEC_D4.1.3_v1.0.pdf.

[11] R. Thomas and T. Katja, "A state-of-the-art investigation," CyReV Project, Tech. Rep., 2019, available: https://autosec.se/wp-content/uploads/2019/11/CyReV_D1.1_A-State-of-the-Art-Investigation.pdf.

[12] S. Mahmood, H. N. Nguyen, and S. A. Shaikh, "Automotive cybersecurity testing: Survey of testbeds and methods," *Digital Transformation, Cyber Security and Resilience of Modern Societies*, pp. 219–243, 2021, available: https://pure.coventry.ac.uk/ws/portalfiles/portal/42194887/Mahmood_et_al_Automotive_Cybersecurity_Testing.pdf.

[13] T. Toyama, T. Yoshida, H. Oguma, and T. Matsumoto, "Pasta: Portable automotive security testbed with adaptability," *London, blackhat Europe*, 2018, available: https://i.blackhat.com/eu-18/Wed-Dec-5/eu-18-Toyama-PASTA-Portable-Automotive-Security-Testbed-with-Adaptability.pdf.

[14] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16, available: https://proceedings.mlr.press/v78/dosovitskiy17a/dosovitskiy17a.pdf.

[15] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility: an overview," in *Proceedings of SIMUL 2011, the third international conference on advances in system simulation*. Citeseer, 2011, available: https://www.researchgate.net/publication/225022282_SUMO_-_Simulation_of_Urban_MObility_An_Overview.

[16] T. Wu, X. Ning, W. Li, R. Huang, H. Yang, and Y. Wang, "Physical adversarial attack on vehicle detector in the carla simulator," *arXiv preprint arXiv:2007.16118*, 2020. [Online]. Available: https://arxiv.org/abs/2007.16118

[17] C. Smith, *The car hacker's handbook: a guide for the penetration tester*. no starch press, 2016, available: https://books.google.ro/books?hl=ro&lr=&id=Ao_QCwAAQBAJ&oi=fnd&pg=PR5&dq=C.+Smith,+The+Car+Hackers+Handbook:

+A+Guide+for+Penetration+Tester.++San+Francisco,+CA,+USA,+No+Starch+Press,+2016.&ots=Q-hMti95z7&sig=S2JtMAwGGtqHNX8tEdYS9EsogJU&redir_esc=y#v=onepage&q&f=false.

[18] F. Sun, L. Du, and Y. Dai, "Pyqt5-powered frontend for advanced yolov8 vehicle detection in challenging backgrounds," *IET Wireless Sensor Systems*, vol. 15, no. 1, p. e70001, 2025, available: https://ietresearch.onlinelibrary.wiley.com/doi/pdfdirect/10.1049/wss2.70001.

[19] C. Urquhart, X. Bellekens, C. Tachtatzis, R. Atkinson, H. Hindy, and A. Seeam, "Cyber-security internals of a skoda octavia vrs: A hands on approach," *IEEE Access*, vol. 7, pp. 146 057–146 069, 2019, available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8848400.

# ANEXE

**Pachete CAN Procesate (SIMULATOR)**

| | ID | Data | Source | Destination | Name | Level | Type | Period | Data Size | CARLA Var |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 36 | 0 | power train | chasis | break output indicator | report | output | 10 | 2 | brake |
| **2** | 57 | 0 | power train | chasis | throttle position | report | output | 10 | 2 | throttle |
| **3** | 67 | 0 | power train | chasis | engine rpm | report | measur... | 10 | 4 | rpm |
| **4** | 98 | 15 | power train | chasis | power steering wheel output ... | report | output | 10 | 2 | steer |
| **5** | 408 | None | power train | chasis | tire angle | report | output | 10 | 2 | tire_angle |
| **6** | 119 | 1 | power train | chasis | shift position | report | output | 10 | 1 | gear |
| **7** | 141 | 0 | body | chasis | turn signal indicator | report | output | 10 | 1 | lightTurn |
| **8** | 180 | 0 | body | chasis | airbag activation switch | report | dummy | 10 | 1 | collision |
| **9** | 1001 | None | body | chasis | closest radar point | report | output | 10 | 2 | radar |
| **10** | 1012 | None | body | chasis | closest lidar point | report | output | 10 | 4 | lidar |
| **11** | 2000 | -19 | GNSS | CGW | Latitude | report | status | 20 | 4 | lat |
| **12** | 2001 | -653 | GNSS | CGW | Longitude | report | status | 20 | 4 | lon |
| **13** | 346 | None | power train | chasis | anti-lcok brake operation | report | dummy | 50 | 1 | |
| **14** | 367 | 0 | power train | chasis | throttle adjustment | report | output | 50 | 2 | kmh |
| **15** | 387 | None | power train | chasis | engine cooling water ... | report | dummy | 50 | 1 | |
| **16** | 410 | None | power train | chasis | engine status | report | output | 50 | 1 | engine |
| **17** | 443 | 0 | body | chasis | position/head lights/high bea... | report | output | 50 | 1 | lightFront |
| **18** | 467 | False | power train | chasis | parking break status | report | output | 50 | 1 | hand_brake |

Figura 23: Tabel pentru pachete de tip report

| | ID | Data | Source | Destination | Name | Level | Type | Period | Data Size | CARLA Var |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 26 | 0 | chasis | power train | break operation indicator | command | operation | 10 | 2 | brake |
| 2 | 47 | 600 | chasis | power train | accelerator pedal operation ... | command | dummy | 10 | 2 | throttle |
| 3 | 88 | 0 | chasis | power train | steeering wheel operation ... | command | operation | 10 | 2 | steer |
| 4 | 109 | 0 | chasis | power train | shift position switch | command | operation | 10 | 1 | gear |
| 5 | 440 | 0 | chasis | power train | engine start button | command | operation | 50 | 1 | ignition |
| 6 | 131 | 0 | chasis | body | right/left turn signal, hazzard ... | command | operation | 10 | 1 | lightTurn |
| 7 | 152 | 0 | chasis | body | horn switch | command | operation | 10 | 1 | None |
| 8 | 423 | 0 | chasis | body | position/head lights/high bea... | command | operation | 50 | 1 | lightFront |
| 9 | 433 | 0 | chasis | body | headligh flashing switch | command | operation | 50 | 1 | passing |
| 10 | 604 | 0 | chasis | body | front wiper/ intermittent/lo... | command | operation | 100 | 1 | None |
| 11 | 625 | 0 | chasis | body | rear wiper/ washer switch | command | operation | 100 | 1 | None |
| 12 | 646 | 0 | body | chasis | doors lock/ unlock switch | command | operation | 100 | 1 | None |
| 13 | 457 | 0 | chasis | power train | parking brake | command | operation | 50 | 1 | hand_brake |
| 14 | 668 | 0 | chasis | body | right door/ window lifting ... | command | operation | 100 | 1 | None |
| 15 | 689 | 0 | chasis | body | left door/window lifting switch | command | operation | 100 | 1 | None |
| 16 | 1313 | 0 | body | CGW | lane departure sound alert | command | boolean | 10 | 1 | lane_depart... |
| 17 | 1200 | 0 | body | chasis | collision alert trigger | command | operation | 10 | 1 | collision |

Figura 24: Interfața pentru injectarea pachetelor de control