Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ МЕТОДІВ ПРОТИДІЇ ШАХРАЙСЬКИМ ПРОГРАМАМ У КОМП'ЮТЕРНИХ ІГРАХ ТИПУ «ШУТЕР»
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-1
Ключніков О. С.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Кобилін О. А.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Ключнікову Олександру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення методів протидії шахрайським програмам у комп'ютерних іграх типу «шутер»

затверджена наказом університету від 26 травня 2026 року № 435Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 12 червня 2026 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, масиви телеметричних даних рухів гравців (ігрові лог-файли), бібліотеки математичного аналізу та обробки даних мовою Python (NumPy, SciPy, Pandas).

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз принципів функціонування та методів протидії шахрайським програмам (зокрема систем автоматичного націлювання AimBot) у комп'ютерних іграх типу «шутер».
2. Розроблення математичної моделі просторових перетворень та поведінкових характеристик наведення прицілу з урахуванням алгоритмів лінійної та експоненційної інтерполяції.
3. Розроблення та тестування програмного комплексу серверного аналізу телеметрії на основі розрахунку інформаційної ентропії та спектрального аналізу (FFT).

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми шахрайства у комп'ютерних іграх, архітектурна схема програмного комплексу аналізу телеметрії, графіки математичних моделей (траєкторії наведення, гістограми швидкостей), амплітудні частотні спектри похибок (FFT), інтерфейс розробленого веб-застосунку.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	14.04.26-16.04.26	
3	Аналіз літератури з проблеми, що вирішується	17.04.26-19.04.26	
4	Аналіз існуючих методів розпізнавання	20.04.26-21.04.26	
5	Формування кроків вирішення проблеми	21.04.26-05.05.26	
6	Програмна реалізація	26.04.26-20.05.26	
7	Аналіз отриманих результатів	27.05.26-04.06.26	
8	Оформлення пояснювальної записки	05.06.26-09.06.26	
9	Перевірка на нормоконтроль	10.06.26-19.06.26	
10	Перевірка на плагіат	11.06.26-30.06.26	
11	Рецензування	20.06.26-30.06.26	
12	Підготовка презентації та доповіді	20.06.26-30.06.26	
13	Занесення роботи в електронний архів	30.06.26-09.07.26	
14	Попередній захист кваліфікаційної роботи	30.06.26-09.07.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Кобилін О. А
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 52 с., 2 табл., 8 рис., 31 джерело.

АНТИ-ЧИТ СИСТЕМА, ВЕБЗАСТОСУНОК, ІНФОРМАЦІЙНА ЕНТРОПІЯ, ПОВЕДІНКОВИЙ АНАЛІЗ, ТЕЛЕМЕТРІЯ, ШВИДКЕ ПЕРЕТВОРЕННЯ ФУР'Є, ШУТЕР, PYTHON.

Об'єктом роботи є процес виявлення та протидії шахрайським програмам (AimBot) у комп'ютерних іграх жанру FPS на основі аналізу координатної телеметрії. Метою роботи є розроблення методів та програмного забезпечення для автоматизованого детектування систем штучного комп'ютерного наведення шляхом аналізу динаміки рухів гравця. Під час роботи використано методи математичного моделювання, цифрової обробки сигналів (Швидке перетворення Фур'є), теорії інформації (ентропія Шеннона), об'єктно-орієнтованого програмування та векторизованих обчислень. Практична цінність роботи полягає у можливості серверного виявлення латентних та апаратних шахрайських програм, які здатні обходити класичні клієнтські античитерські системи. Розроблено вебзастосунок, що забезпечує імітаційне моделювання траєкторій наведення, парсинг ігрових лог-файлів та автоматичну класифікацію поведінки гравця у режимі реального часу. Область застосування: індустрія комп'ютерних ігор, кіберспорт, розроблення серверних систем безпеки та адміністрування змагальних ігрових платформ.

ANTI-CHEAT SYSTEM, BEHAVIORAL ANALYSIS, FAST FOURIER TRANSFORM, INFORMATIONAL ENTROPY, PYTHON, SHOOTER, TELEMETRY, WEB APPLICATION,.

The object of the work is the process of detecting and countering cheating programs (AimBot) in FPS computer games based on coordinate telemetry analysis. The goal of the work is to develop methods and software for the automated detection of artificial computer aiming systems by analyzing the dynamics of player movements. The following were used during the work: mathematical modeling methods, digital signal processing (Fast Fourier Transform), information theory (Shannon entropy), object-oriented programming, and vectorized computations. The practical value of the work lies in the capability of server-side detection of latent and hardware-based cheat programs that can bypass classical client-side anti-cheat systems. A web application has been developed that provides simulation of aiming trajectories, parsing of game log files, and automatic classification of player behavior in real time. Applications: video game industry, esports, development of server security systems, and administration of competitive gaming platforms.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект

AimBot – тип шахрайського програмного забезпечення, призначений для автоматичного наведення прицілу на ігрову модель супротивника

API – Application Programming Interface (інтерфейс прикладного програмування, набір готових функцій, структур та констант для використання у зовнішніх програмних продуктах)

BSOD – Blue Screen of Death (синій екран смерті, назва повідомлення про критичну помилку ядра операційної системи Windows)

BYOVD – Bring Your Own Vulnerable Driver (метод обходу захисту операційної системи шляхом завантаження легітимного, але вразливого драйвера)

CRC – Cyclic Redundancy Check (циклічний надлишковий код, алгоритм розрахунку контрольної суми для перевірки цілісності даних)

CSV – Comma-Separated Values (текстовий формат файлу для представлення табличних даних, що використовується для експорту ігрової телеметрії)

DKOM – Direct Kernel Object Manipulation пряма маніпуляція об’єктами ядра операційної системи, техніка приховування шкідливих процесів.

DMA – Direct Memory Access (прямий доступ до оперативної пам’яті в обхід центрального процесора)

DSE – Driver Signature Enforcement (механізм операційної системи Windows, що вимагає обов’язкового цифрового підпису для завантаження драйверів)

ESP – Extra Sensory Perception (тип шахрайського програмного забезпечення, що виводить на екран додаткову приховану інформацію про стан ворога: здоров’я, зброя, дистанція)

FFT – Fast Fourier Transform (алгоритм швидкого перетворення Фур'є, що використовується для обчислення частотного спектра сигналів)

FPS – First-Person Shooter (жанр комп'ютерних ігор «шутер від першої особи»)

GUI – Graphical User Interface (графічний інтерфейс користувача)

Hitbox – хітбокс (невидима геометрична фігура, що прикріплюється до тривимірної моделі гравця для розрахунку та реєстрації влучань)

LERP – Linear Interpolation (алгоритм лінійної інтерполяції для поступової зміни величин між двома заданими значеннями)

ML – Machine Learning (машинне навчання, підклас штучного інтелекту, що спеціалізується на розпізнаванні патернів та прийнятті рішень на основі масивів даних)

NDC – Normalized Device Coordinates (нормалізовані координати пристрою у діапазоні від -1 до 1, які використовуються у конвеєрі тривимірної графіки)

Pitch – тангаж (кут нахилу віртуальної камери гравця по вертикальній осі: вгору/вниз)

Radar Hack – тип шахрайського програмного забезпечення для відображення прихованих об'єктів та супротивників на ігровій міні-карті

SaaS – Software as a Service (програмне забезпечення як послуга, бізнес-модель розповсюдження сучасних шахрайських програм за підпискою)

SHA – Secure Hash Algorithm (безпечний алгоритм криптографічного хешування)

VAC – Valve Anti-Cheat (автоматизована античитерська система розробки компанії Valve)

WallHack – тип шахрайського програмного забезпечення, що модифікує рендеринг і дозволяє бачити моделі об'єктів крізь непрозорі перешкоди

Yaw – нишпорення (кут повороту віртуальної камери гравця по горизонтальній осі: вліво/вправо)

ВСТУП

Індустрія комп’ютерних ігор за останні десятиліття трансформувалася з нішевої розваги у глобальний соціокультурний та економічний феномен. Особливе місце в цьому сегменті займають змагальні ігри жанру FPS («шутери» від першої особи), такі як Counter-Strike 2, PUBG: BATTLEGROUNDS та серія Battlefield. Кіберспорт сьогодні залучає мільйонні аудиторії та генерує значні фінансові потоки, що автоматично робить питання чесності гри (Fair Play) критично важливим [1].

Проблема шахрайства у відеоіграх вийшла за межі етичних порушень і набула ознак серйозної загрози для економіки ігрових платформ. Використання стороннього програмного забезпечення для здобуття нечесної переваги руйнує екосистему гри: чесні гравці втрачають інтерес до проєкту, що призводить до відтоку аудиторії та фінансових збитків розробників. Більшість сучасних шахрайських програм є платними комерційними продуктами, що вказує на індустріалізацію цього тіньового ринку. Розробники читів використовують складні методи обходу захисту, включаючи маніпуляції на рівні ядра операційної системи та апаратні емулятори [2, 3].

Класичні методи детектування, що базуються на пошуку сигнатур файлів, втрачають ефективність. На зміну їм приходять алгоритми машинного навчання (ML) та системи з доступом до ядра (Kernel-level), які аналізують поведінку гравця на основі телеметрії. Усе це зумовлює потребу у розробленні нових методів виявлення аномалій та створенні надійних античитерських алгоритмів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ПРОТИДІЇ ШАХРАЙСЬКИМ ПРОГРАМАМ

1.1 Класифікація та принципи роботи шахрайського програмного забезпечення

Сучасні підходи до використання шахрайських програм у стрілецьких комп’ютерних іграх для здобування переваги над суперником умовно поділяються на три великі категорії: AimBot (системи автоприцілювання), Wall Hack (перегляд крізь стіни) та Radar Hack (відображення ворогів на радарі). Кожна з цих програм має свій унікальний алгоритм впливу на ігровий процес та специфічні вразливості, за якими її можна виявити.

Одним із найбільш деструктивних типів шахрайського програмного забезпечення є системи автоматичного націлювання, відомі як AimBot. Принцип їх роботи полягає в перехопленні координат ігрових об’єктів у тривимірному просторі та автоматичній маніпуляції вектором огляду гравця (ViewAngles).

Коли гравець активує функцію (наприклад, затисканням або натисканням кнопки «постріл»), програма миттєво обчислює необхідний кут повороту віртуальної камери, щоб сумістити перехрестя прицілу з хітбоксом (hitbox) голови або іншої частини тіла супротивника. У цей момент AimBot автоматично направляє приціл на ворожу модель, а ігровий сервер лише отримує та реагує на механічні дії, які симулюються з боку комп’ютера користувача.

Основна перевага AimBot з точки зору порушника – це її легкість у використанні, що дозволяє лише за один клік повністю викреслити фактор людини, нівелюючи роки тренувань чесних гравців у контролі віддачі (recoil control) та швидкості реакції. Сучасні AimBot мають гнучкі конфігурації, що спираються на потреби гравця (наприклад, налаштування радіуса захоплення цілі – Field of View, або вибір пріоритетної кістки моделі).

Проте ця технологія має суттєвий концептуальний недолік – механіку роботи. AimBot притягує приціл до певних пікселів за неприродно ідеальними траєкторіями. AimBot намагається вести приціл по оптимальній, математично вивіреній траєкторії, через що рух виглядає надто гладким. Тому гравця, що використовує цю програму, часто можна визначити за прицілом, що смикається або рухається абсолютно лінійно, позбавлений людського мікрошуму.

Окрім систем автоматичного націлювання, значну загрозу для змагального процесу становлять програми, що надають нелегітимну інформаційну перевагу. Найпоширенішим представником є Wall Hack (WH) – програма, що модифікує процес рендерингу графіки.

Втручаючись у роботу графічних API (DirectX або Vulkan), така програма змінює властивості матеріалів або шейдерів, дозволяючи бачити моделі супротивників крізь непрозорі перешкоди (стіни, двері, ящики). Фактично, вона виконує функцію «рентгенівського зору», надаючи гравцеві можливість заздалегідь підготуватися до пострілу (pre-fire). Більш просунутою версією Wall Hack є ESP (Extra Sensory Perception), яка не просто підсвічує силуети, а й виводить на екран додаткову інформацію у вигляді тексту: рівень здоров’я ворога, його поточну зброю, дистанцію до нього та напрямок погляду.

Ще одним складним для виявлення класом є Radar Hack – інструмент, що здійснює зовнішнє перехоплення пакетів мережевих даних про позиції гравців і відображає їхнє місцезнаходження на міні-карті гри або навіть на окремому моніторі. Це дає колосальну стратегічну перевагу, оскільки гравець у реальному часі знає про всі переміщення команди суперника та може координувати дії своєї команди.

Боротьба з цими типами загроз вимагає надзвичайно комплексного підходу. Radar Hack, на відміну від класичних програм, може працювати на окремому пристрої, непомітно перехоплюючи мережевий трафік, що робить його абсолютно невидимим для локальних сканерів оперативної пам’яті

комп’ютера. Відомі випадки на кіберспортивних турнірах, коли у гравця під монітором знаходився звичайний смартфон із відкритою програмою Radar Hack, який фізично не було видно на вебкамері під час офіційної трансляції.

Щоб обходити найскладніший захист античит-систем, сучасні шахраї все частіше вдаються до апаратних рішень. Ці системи включають у себе карти відеозахоплення (DMA-карти), апаратні емулятори вводу та спеціалізоване програмне забезпечення, яке функціонує на основі методів машинного навчання. Зазвичай такі системи вмикаються на окремому, другому комп’ютері, що дозволяє їм повністю приховуватися від клієнтських анти-шахрай програм, які працюють на основному ігровому ПК.

1.2 Аналіз існуючих методів детектування та еволюція античитерських систем

Еволюція античитерських систем у змагальних комп’ютерних іграх відображає безперервний процес технічного протистояння між розробниками систем захисту та творцями шахрайського програмного забезпечення. Протягом останніх десятиліть методи детектування пройшли тривалий шлях розвитку – від найпростіших локальних сигнатурних сканерів файлової системи до складних хмарних моделей машинного навчання та систем із доступом до ядра операційної системи, що здатні аналізувати поведінкові фактори мільйонів користувачів у реальному часі [4, 5, 6].

Найбільш наочно цей еволюційний шлях можна простежити на прикладі розвитку захисних механізмів у франшизі ігор *Counter-Strike*, яка через свою колосальну популярність завжди перебувала на передньому краї змагального кіберспорту й водночас виступала головною мішенню для розробників нелегального софту.

У 2002 році компанією Valve була представлена перша версія системи Valve Anti-Cheat (VAC) як пряма відповідь на масове використання сторонніх

модифікацій ігрового клієнта [7]. Перші модифікації VAC функціонували за принципом класичного антивірусного програмного забезпечення, де основним механізмом детектування виступав сигнатурний аналіз. Завдання системи полягало в періодичному скануванні файлової структури гри та виділених областей оперативної пам’яті комп’ютера з метою виявлення відомих фрагментів шкідливого коду. Механізм перевірки базувався на розрахунку та порівнянні хеш-сум (зокрема циклічного надлишкового коду CRC) виконуваних файлів та динамічних бібліотек із офіційною базою даних відомих читів. У разі, якщо хеш-сума завантаженої DLL-бібліотеки не збігалася з еталоном або в адресній пам’яті процесу виявлявся відомий рядок байтів, система фіксувала порушення та видавала блокування акаунту.

Головним недоліком сигнатурного підходу виявилася його критична вразливість до технологій обфускації, шифрування та поліморфізму коду. Розробникам шахрайського програмного забезпечення було достатньо змінити кілька некритичних байтів у структурі програми або застосувати автоматичний пакувальник коду, щоб повністю змінити фінальну сигнатуру файлу, зробивши його невидимим для сканера VAC.

Вододілом у розвитку систем захисту став 2012 рік, що ознаменувався релізом *Counter-Strike: Global Offensive*. Отримавши архітектуру регулярних мережевих оновлень, античит VAC був суттєво перебудований для постійного функціонування у фоновому режимі та безперервного відправлення телеметричних даних на віддалені сервери Valve. До базового функціонала було додано інструменти для детектування DLL-ін’єкцій, механізми моніторингу внутрішніх функцій графічних API (DirectX та Vulkan) для перехоплення несанкціонованих overlay-вікон, а також перші евристичні алгоритми й логічні перевірки цілісності внутрішнього стану гри. Проте стрімке зростання популярності дисципліни спровокувало комерціалізацію та індустріалізацію тіньового ринку, внаслідок чого з’явилися складні приватні чити, які обходили базові евристики за рахунок індивідуального шифрування пам’яті.

Наступним концептуальним етапом розвитку захисних технологій стало впровадження методів машинного навчання (Machine Learning). Потреба у поведінковому аналізі загострилася у зв'язку з появою апаратних засобів шахрайства – карт прямого доступу до пам'яті (Direct Memory Access) та зовнішніх емуляторів введення на базі мікроконтролерів (наприклад, платформ Arduino або Teensy). Такі пристрої підключаються до апаратних шин комп'ютера або емулюють стандартні USB-пристрої введення, що робить їх фізично невидимими для будь-якого програмного античиту, який працює локально на клієнтському ПК. Оскільки зафіксувати сам факт наявності сторонньої програми в пам'яті стало неможливо, аналіз перемістився виключно в площину дослідження поведінкових шаблонів і телеметрії гравця.

У 2018 році Valve розгорнула систему VACNet – хмарну нейромережеву структуру, інтегровану з інструментом спільноти «Патруль» (Overwatch). Математична та алгоритмічна основа VACNet базується на використанні глибоких нейронних мереж (DNN), які аналізують мільйони ігрових сесій. Система зчитує координати, кути огляду, швидкість реакції та мікрорухи прицілу безпосередньо на рівні серверного тіку (tick-rate). Головна особливість VACNet полягає в тому, що вона не працює в реальному часі на комп'ютері користувача, а обробляє великі масиви накопичених даних (Big Data) на серверних потужностях, виявляючи штучні моделі згладжування або аномально високу точність стрільби, після чого здійснює масові координовані блокування.

Сучасний етап розвитку захисту реалізовано в грі *Counter-Strike 2* на рушії Source 2. Нова архітектура надає розширені можливості контролю клієнтської сторони, включаючи внутрішню верифікацію кадрового буфера (анти-оверлеї) та точну синхронізацію фізики підрозділів тіку (sub-tick). Оновлена система VAC здійснює низькорівневий аналіз апаратного введення миші, що дозволяє виявляти роботу макросів і зовнішніх контролерів, а також здатна скасовувати матч безпосередньо в реальному часі під час виявлення критичних аномалій алгоритмами ШІ [8].

Для розуміння сучасних принципів детектування необхідно детально розглянути дворівневу архітектуру захисту, яка складається з клієнтської та серверної частин, що функціонують паралельно й безперервно обмінюються даними.

Клієнтська частина античиту розгортається безпосередньо на локальній операційній системі користувача і виконує завдання первинного сканування та моніторингу локального оточення. До основних функцій клієнтського модуля належать:

– сканування оперативної пам’яті активного процесу гри з метою перевірки таблиць віртуальних функцій, внутрішніх структур рушія, об’єктів динамічного руху та цілісності кісток тривимірних моделей гравців;

– безперервна перевірка цілісності статичних файлів гри шляхом динамічного зіставлення CRC або SHA-хешів із еталонними значеннями на офіційному сервері;

– відстеження та низькорівневе перехоплення сигналів введення від миші та клавіатури з метою аналізу частоти відправлення пакетів (polling rate), виявлення програмних ін’єкцій віртуальних подій або використання нестандартних комбінацій;

– моніторинг системних графічних API для виявлення сторонніх overlay-ін’єкцій, які використовуються ESP та WallHack програмами для малювання інтерфейсу поверх вікна гри;

– агрегація та первинне стиснення пакетів телеметрії (координати, стан прицілу, швидкість переміщення курсора) для їх подальшого відправлення на сервер.

Серверна сторона античиту є найбільш надійною ланкою системи безпеки, оскільки її код виконується в ізольованому середовищі хостингу розробника і не може бути модифікований або скомпрометований з боку зловмисника. До ключових методів серверного захисту відносяться:

– строга синхронізація ігрового тіку та верифікація фізичного стану (сервер здійснює повну перевірку швидкості переміщення та координат гравця, миттєво відсікаючи спроби SpeedHack або телепортації);

– серверне обчислення та верифікація хітбоксів у реальному часі, що мінімізує вразливість перед технологіями hit-scan маніпуляцій;

– виявлення AimBot-подібних аномалій на основі аналізу динаміки кутів огляду, де сервер оцінює показники плавності, інформаційної ентропії та наявності мікроскопічних осциляцій прицілу;

– хмарний ML-аналіз великих масивів телеметричних даних за допомогою нейромереж типу VACNet;

– керування системою відкладених блокувань (Delayed Bans), що використовується для приховування від розробників читів конкретного фактора, який спровокував детектування софту.

Через критичну неефективність класичних систем захисту користувацького режиму (User-mode, рівень Ring 3) у протидії сучасним поліморфним драйверам, розробники перейшли до створення античитів, що працюють на рівні ядра операційної системи (Kernel-level, рівень Ring 0). Типовими представниками цього класу ПЗ є Vanguard (Riot Games) [9], Faceit Anti-Cheat [10], а також новітня розробка корпорації EA під назвою Javelin [11].

Перевага архітектури Ring 0 полягає в тому, що такий античит завантажується на найранішому етапі старту операційної системи (часто як драйвер завантаження типу Boot-start). Це дозволяє йому повністю контролювати простір ядра, сканувати всю файлову систему і блокувати спроби ініціалізації непідписаних або скомпрометованих сторонніх драйверів ще до безпосереднього запуску ігрового клієнта. Практична статистика впровадження таких рішень демонструє високу ефективність: наприклад, під час інтеграції системи Javelin у серію ігор *Battlefield* у період відкритого бета-тестування за кілька днів було виявлено та заблоковано понад 300 тисяч

нелегальних акаунтів, що забезпечило різке зниження щільності шахраїв на ігрових серверах [12].

Попри високі показники детектування, технологія Kernel-level містить низку серйозних технологічних, етичних та безпекових проблем:

– приватність та інформаційна безпека. Рівень доступу Ring 0 надає програмі необмежені права на читання та модифікацію будь-яких даних на персональному комп'ютері користувача, що викликає обґрунтовану недовіру з боку гравців щодо конфіденційності їхніх персональних даних. Крім того, наявність критичної вразливості або запасного виходу в самому коді античиту ядра автоматично перетворює його на вектор атаки для хакерського ПЗ, дозволяючи зловмисникам отримати повний контроль над операційною системою;

– програмні та апаратні конфлікти. Оскільки на рівні ядра будь-яка критична помилка в коді драйвера призводить до повного краху операційної системи із викликом «синього екрана смерті» (BSOD), сумісність таких програм є критичною. На практиці зафіксовані випадки жорстких архітектурних конфліктів між ізольованими античитами різних компаній (наприклад, між системами захисту ігор Battlefield та Valorant), коли два kernel-модулі намагалися заблокувати доступ до функцій перехоплення один одного, спричиняючи апаратні збої та блокуючи запуск суміжних програм. Єдиним шляхом вирішення подібних конфліктів зазвичай стає повне видалення одного із конкурентних ігрових проєктів.

Паралельно з низькорівневим захистом розвивається математичний апарат поведінкового ШІ-детектування. Логіка роботи таких систем зводиться до задачі бінарної класифікації, яка може бути формалізована у вигляді математичної функції перетворення вигляду:

$$f(X) \to \{0,1\}. \qquad (1.1)$$

X представляє собою багатовимірний вектор ознак (Feature Vector), що безперервно формується на основі поточних телеметричних параметрів гравця. До структури вектора X зазвичай входять такі метрики: математичне очікування швидкості наведення, показники інформаційної ентропії кутових швидкостей, частотні характеристики амплітудного спектра похибки (отримані через FFT), питома вага точних влучань у критичні кістки моделі (голова, шия), латентність (час реакції) при появі моделі ворога в зоні видимості, а також наявність мікроскопічних затримок курсора на межах хітбокса. Результат функції $f(X) = 0$ інтерпретується як легітимна (чесна) гра користувача, тоді як $f(X) = 1$ сигналізує про штучне автоматизоване втручання та є підставою для видачі автоматичного блокування.

Додатково в сучасних серверних комплексах захисту починають впроваджувати технології комп'ютерного зору (Computer Vision) для аналізу візуальних потоків і рендерів безпосередньо під час матчів або обробки демонстраційних записів, що дозволяє виявляти суб-візуальні аномалії наведення, які не фіксуються звичайними логічними перевірками клієнтського вводу. Таким чином, сучасний захист остаточно змістився від парадигми локального пошуку сигнатур файлів до комплексного аналізу великих даних та поведінкової психомоторики.

1.3 Технології низькорівневого обходу античитерських систем

Стрімкий розвиток засобів захисту ігрового клієнта спровокував симетричну відповідь з боку розробників нелегального програмного забезпечення, які перемістили вектор атак із користувацького простору (User-mode, Ring 3) безпосередньо у простір ядра операційної системи (Kernel-mode, Ring 0). Розуміння механізмів, які використовують шахраї для приховування свого софту, є критично важливим для побудови ефективних алгоритмів протидії на стороні сервера.

Основним методом класичного втручання в ігровий процес тривалий час залишалася DLL-ін’єкція (динамічне завантаження бібліотеки в адресний простір гри). Проте сучасні клієнтські античити миттєво фіксують такі спроби через моніторинг функцій CreateRemoteThread або NtCreateThreadEx. У відповідь на це розробники читів почали застосовувати технологію Manual Mapping (ручне проектування). Цей метод передбачає, що шахрайська програма самостійно зчитує вміст своєї DLL-бібліотеки з диска, виділяє блок пам’яті всередині процесу гри за допомогою VirtualAllocEx і вручну копіює туди секції коду, самостійно вирівнюючи таблицю імпорту (IAT ё Import Address Table) та виконуючи базову релокацію. Оскільки при цьому не викликається системна функція LoadLibrary, операційна система не реєструє модуль у офіційному списку завантажених DLL (складові структури PEB – Process Environment Block), що робить такий софт невидимим для стандартних методів перевірки.

Для обходу антивірусних евристик та античитів Ring 0 зловмисники використовують вразливості у легітимних, офіційно підписаних драйверах сторонніх виробників (метод BYOVD – Bring Your Own Vulnerable Driver) [13]. Оскільки сучасні версії Windows (починаючи з Windows 10/11) вимагають обов’язкового цифрового підпису драйверів (DSE – Driver Signature Enforcement), шахраї не можуть завантажити власний шкідливий Ring 0 драйвер без сертифіката. Замість цього вони завантажують старий або вразливий драйвер відомого апаратного забезпечення (наприклад, утиліт розгону відеокарт чи моніторингу температури від ASUS, Gigabyte тощо), який має офіційний підпис Microsoft, але містить уразливість типу довільного читання/запису в пам’ять ядра. Через цю уразливість чит-програма з користувацького режиму отримує безмежний доступ до будь-яких структур ядра операційної системи, включаючи можливість модифікації списків активних процесів (DKOM – Direct Kernel Object Manipulation) та затирання вказівників на структури захисту античиту [14].

Найбільш складною та технологічною загрозою 2026 року є використання апаратних DMA-карт (Direct Memory Access) [15]. Ці пристрої встановлюються у фізичний слот PCI-Express цільового комп'ютера і маскуються (шляхом прошивки індивідуальних ідентифікаторів VID/PID) під звичайні легітимні плати: мережеві адаптери або звукові карти. Завдяки архітектурі комп'ютерних систем, DMA-карта здатна зчитувати та записувати дані безпосередньо в оперативну пам'ять (RAM) комп'ютера в обхід центрального процесора та будь-яких програмних механізмів контролю операційної системи.

Для роботи такого читу оперативна пам'ять ігрового ПК безперервно копіюється через кабель на другий, абсолютно чистий комп'ютер (Radar/Aim PC). На цьому другому ПК запускається програма, яка аналізує координати ворогів і малює ESP-інтерфейс поверх ігрового екрана за допомогою спеціальної плати відеозахоплення (HDMI fuser). Жоден програмний античит (навіть Vanguard чи Faceit AC), що працює на ігровому ПК, не здатний зафіксувати присутність стороннього коду, оскільки на самому ігровому комп'ютері немає жодного шкідливого процесу, файлу чи модифікованого потоку пам'яті. Це остаточно підтверджує, що локальний клієнтський захист вичерпав свої можливості, а єдиним надійним вектором боротьби є аналіз чистої поведінкової телеметрії на стороні ігрового сервера [16, 17].

1.4 Постановка задачі

На основі проведеного аналізу предметної області, існуючих методів захисту та технологій обходу античитерських систем, можна формалізувати задачу дослідження в межах цієї кваліфікаційної роботи.

Встановлено, що головною науковою та практичною проблемою є неможливість гарантованого виявлення сучасних апаратних (DMA) та низькорівневих (Ring 0) шахрайських програм за допомогою локальних

клієнтських сканерів. Оскільки зловмисники здатні повністю маскувати свою присутність в операційній системі та симулювати апаратне введення маніпуляторів згладженими алгоритмами інтерполяції (LERP/Exponential), виникає гостра потреба в розробленні серверних поведінкових методів класифікації.

Об’єктом роботи є процес виявлення та протидії шахрайським програмам у комп’ютерних іграх типу «шутер».

Метою роботи є розроблення методів виявлення шахрайських програм на основі аналізу телеметрії рухів гравця та спектрального аналізу похибок наведення.

Для досягнення мети необхідно вирішити такі завдання:

– побудувати строгу математичну модель просторових перетворень (World-to-Screen) для визначення точного вектора наведення прицілу та змоделювати траєкторії штучного згладжування рухів;

– розробити алгоритмічне забезпечення для обчислення динамічних характеристик руху в режимі реального часу, зокрема інформаційної ентропії кутових швидкостей та амплітудного частотного спектра за допомогою Швидкого перетворення Фур’є (FFT);

– реалізувати високопродуктивний векторизований програмний комплекс мовою Python, який здатний інтегруватися в серверні структури змагальних ігор і виконувати бінарну класифікацію («чесний гравець / шахрай») без зниження загальної продуктивності ігрового сервера.

2 МАТЕМАТИЧНЕ ОБҐРУНТУВАННЯ ТА РОЗРОБЛЕННЯ МЕТОДІВ ВИЯВЛЕННЯ ШАХРАЙСЬКИХ ПРОГРАМ

2.1 Математичне моделювання поведінки систем автоматичного націлювання

Функціонування шахрайського програмного забезпечення типу AimBot, призначеного для автоматичного наведення курсора або прицілу на ігрову модель супротивника, базується на безперервному перехопленні, модифікації та обробці координат об’єктів у віртуальному тривимірному просторі комп’ютерної гри [18]. Для реалізації стійкого механізму автоматичного наведення розробникам подібних програм необхідно вирішити фундаментальну задачу тривимірної комп’ютерної графіки – перетворення координат цілі з глобального простору (World Space) у кутові координати камери гравця (View Angles) або у двовимірну площину екрана монітора (Screen Space).

Вхідними даними для математичної моделі роботи алгоритму автоматичного націлювання є вектор поточного положення віртуальної камери (очей) гравця $p = (p_x, p_y, p_z)$ та вектор положення цілі $t = (t_x, t_y, t_z)$, який найчастіше відповідає просторовим координатам центру критичного хітбокса (наприклад, голови або шиї моделі супротивника). Усі ці значення зчитуються шахрайською програмою безпосередньо з адресного простору оперативної пам’яті клієнта гри шляхом динамічного пошуку вказівників на об’єкти класів гравців (Entity List).

Першим етапом просторових обчислень є знаходження тривимірного вектора напрямку (Direction Vector) у декартовій системі координат. Цей вектор визначає просторову різницю положень між початковою точкою огляду та ціллю і розраховується як покомпонентна різниця їхніх координат:

$$\Delta x = t_x - p_x, \tag{2.1}$$

$$\Delta y = t_y - p_y, \tag{2.2}$$

$$\Delta z = t_z - p_z. \tag{2.3}$$

Для безпосереднього керування напрямком погляду у змагальних іграх жанру FPS («шутерах») отримані декартові координати вектора напрямку необхідно трансформувати у сферичні кути Ейлера, які в ігрових рушіях (зокрема Source 2 та Unreal Engine) визначаються як кут нишпорення (yaw) та кут тангажу (pitch). Використовуючи апарат стандартних тригонометричних функцій, розрахунок необхідних кутів для забезпечення ідеального вектора наведення здійснюється за формулами:

$$yaw = atan2(\Delta y, \Delta x), \tag{2.4}$$

$$pitch = atan2\left(\Delta z, \sqrt{\Delta x^2 + \Delta y^2}\right). \tag{2.5}$$

Окрім прямої модифікації кутів огляду пам’яті, багато сучасних шахрайських систем (зокрема алгоритми «Silent Aim», що здійснюють постріл повз візуальний напрямок камери, або графічні модулі ESP/WallHack) потребують точного знаходження двовимірних піксельних координат об’єкта на екрані монітора. Для цього виконується класичне матричне перетворення, відоме як псевдо-матрична проєкція з 3D у 2D (World-to-Screen) [19].

Процес трансформації включає обов’язкове використання двох матриць розмірністю 4 х 4: матриці вигляду V (View Matrix), яка описує положення та орієнтацію камери у просторі, та матриці проєкції P (Projection Matrix), що визначає параметри відсікання та перспективного спотворення сцени. Спочатку тривимірний вектор цілі перетворюється у гомогенні (однорідні) чотиривимірні координати шляхом додавання масштабного компонента:

$$t_h = \left[t_x, t_y, t_z, 1\right]^T. \tag{2.6}$$

Результатом послідовного множення матриць на гомогенний вектор цілі є вектор у просторі відсікання (Clip Space) c:

$$c = P \times V \times t_h = \left[c_x, c_y, c_z, c_w\right]^T. \tag{2.7}$$

Наступним кроком є перехід до нормалізованих координат пристрою (NDC – Normalized Device Coordinates). Ця процедура виконується шляхом декомпозиції перспективного спотворення через ділення перших двох компонентів вектора на його четверту w-компоненту:

$$x_{ndc} = \frac{c_x}{c_w}, \tag{2.8}$$

$$y_{ndc} = \frac{c_y}{c_w}. \tag{2.9}$$

Внаслідок цього обчислення координати x_{ndc} та y_{ndc} нормалізуються та проєктуються у фіксований діапазон від -1 до 1. На фінальному етапі отримані значення масштабуються відповідно до поточної роздільної здатності екрана монітора $W \times H$ (де W – ширина екрана в пікселях, H – висота) для знаходження результуючих піксельних координат (x_{screen}, y_{screen}):

$$x_{screen} = \frac{W}{2}(1 + x_{ndc}), \tag{2.10}$$

$$y_{screen} = \frac{H}{2}(1 - y_{ndc}). \tag{2.11}$$

Отримані значення (x_{screen}, y_{screen}) вказують на точні координати пікселів на екрані монітора, куди має бути спрямоване перехрестя прицілу. Якщо шахрайська програма миттєво перезаписує поточні кути огляду обчисленими значеннями (механіка Snap Aim), таке наведення створює різкий розрив траєкторії, що миттєво фіксується серверними евристичними показниками прогнозів. З метою маскування роботи програм розробники читів застосовують математичні алгоритми інтерполяції та згладжування руху для симуляції природної людської моторики.

Найбільш поширеним є метод лінійної інтерполяції (LERP – Linear Interpolation), який базується на поступовій зміні поточного кута огляду φ_{cur} до розрахованого бажаного кута наведення φ_{des} із фіксованим кроком згладжування s:

$$\phi_{new} = \phi_{cur} + s \cdot (\varphi_{des} - \varphi_{cur}), 0 \leq s \leq 1. \quad (2.12)$$

Для забезпечення більш плавної адаптації руху до динамічної частоти кадрів (FPS) застосовується модель експоненційного згладжування, де коефіцієнт адаптивності α визначає м’якість наближення прицілу до цілі:

$$\varphi_{new} = (1 - \alpha)\varphi_{cur} + \alpha\varphi_{des}, 0 \leq \alpha \leq 1, \quad (2.13)$$

де під параметром φ мається на увазі будь-який з ейлерових кутів огляду (yaw або pitch).

Математичний аналіз змодельованих траєкторій наведення дозволяє виявити фундаментальні статистичні аномалії, які залишаються навіть при використанні складних алгоритмів згладжування (LERP/Exponential). Графік траєкторії наведення прицілу у піксельних координатах, який показано на рисунку 2.1, чітко демонструє, що «ідеальний» або миттєвий рух AimBot є абсолютно лінійним, що фізично неможливо для людини через анатомічні обмеження кисті та тертя маніпулятора об поверхню.

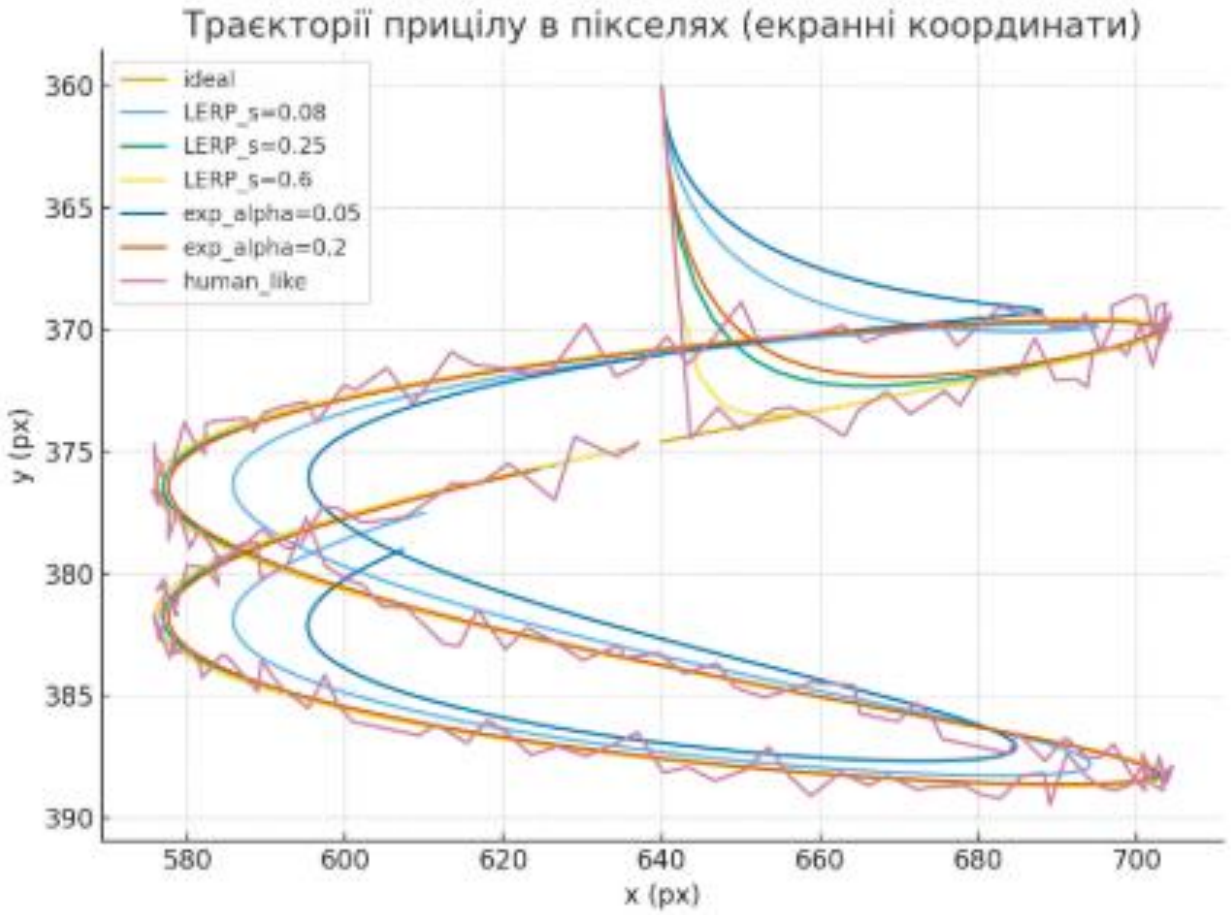

Рисунок 2.1 – Траєкторія наведення прицілу у піксельних координатах

Алгоритмічні моделі LERP з високими коефіцієнтами кроку ($s = 0{,}6$) або експоненційне згладжування з великим α формують ідеально гладкі, геометрично правильні дуги, повністю позбавлені випадкових флуктуацій. Натомість реальна «людська» траєкторія (human-like) супроводжується постійною присутністю фізіологічного тремору (мікро-коливань) та мікроскопічними похибками виходу за межі цілі з наступною корекцією (over-shooting).

Аналіз часових залежностей кутів повороту прицілу від часу на рисунках 2.2 і 2.3 ілюструє динамічний компроміс між ефективністю та непомітністю шахрайських програм. Малі коефіцієнти інтерполяції ($s = 0{,}08$ або $\alpha = 0{,}05$) забезпечують високу плавність, проте створюють критичне відставання (lag) прицілу від динамічної цілі. Збільшення жорсткості

алгоритму призводить до різких роботизованих зламів графіків у моменти зміни вектора руху цілі.

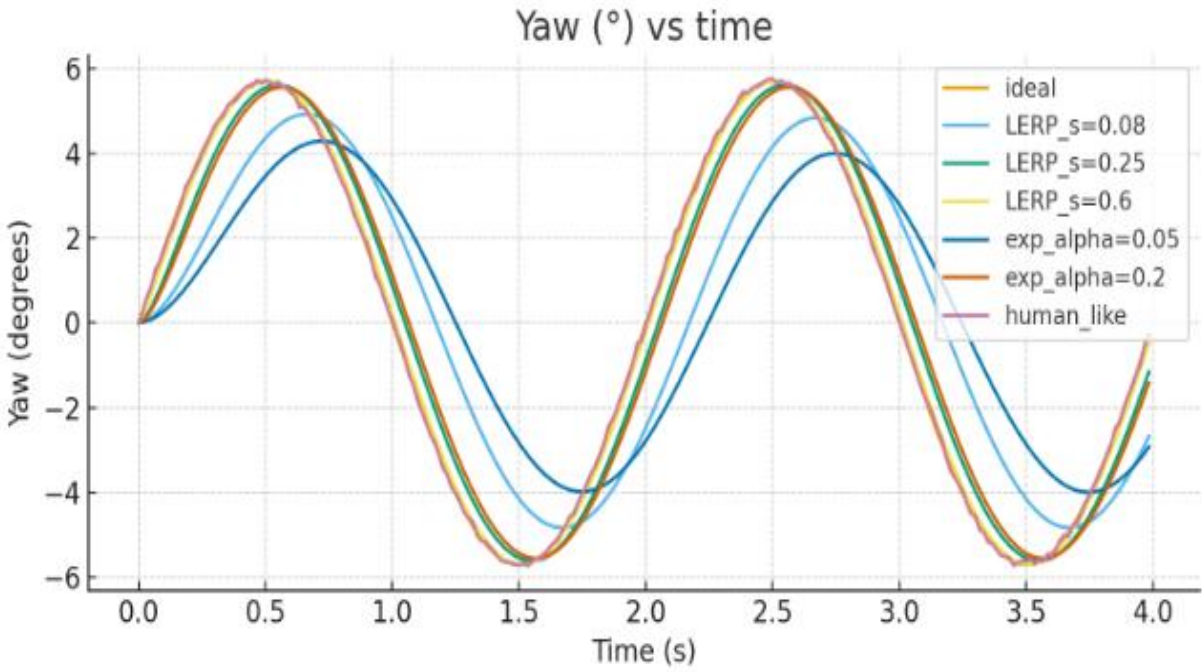

Рисунок 2.2 – Залежність кута повороту прицілу (yaw) від часу

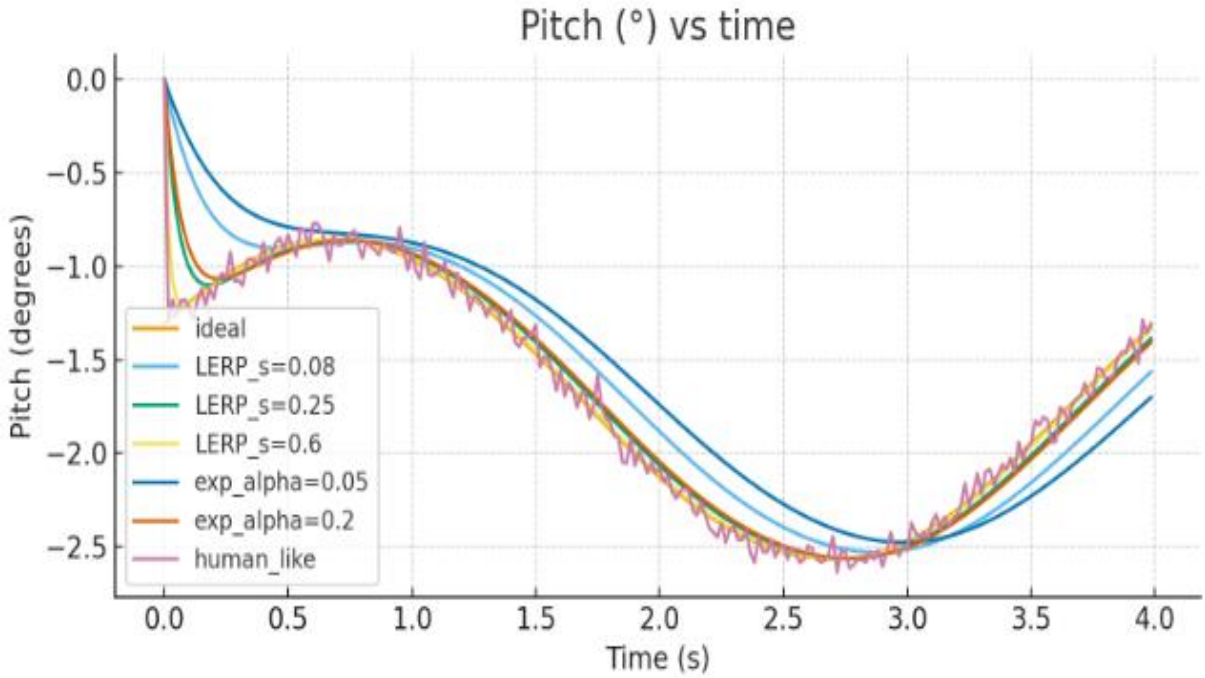

Рисунок 2.3 – Залежність кута нахилу прицілу (pitch) від часу

Найбільш виразні критерії детектування автоматизованого втручання дає статистичний розподіл кутових швидкостей та частотний аналіз сигналу похибки наведення за допомогою Швидкого перетворення Фур'є (FFT)[20, 21,22]. Гістограма розподілу кутових швидкостей наведення (рис. 2.4) доводить, що жива людина має широкий, розмитий спектр розподілу швидкостей з високим рівнем дисперсії через постійну зміну динаміки руху. Робота алгоритмів LERP/Exponential відображається на гістограмі у вигляді вузьких, яскраво виражених піків, що свідчить про постійну, суворо детерміновану математичну швидкість корекції вектора.

Рисунок 2.4 – Гістограма розподілу кутових швидкостей наведення

Амплітудний спектр (FFT) похибки кута огляду (рис. 2.5) остаточно формалізує математичну відмінність: природний рух людини через свою хаотичність та недосконаліть генерує рівномірний спектр високої частоти, аналогічний за своїми характеристиками білому шуму. На противагу цьому, будь-які математичні моделі згладжування AimBot штучно пригнічують високочастотні компоненти, створюючи аномальні, ізольовані низькочастотні шаблони високої амплітуди в спектрі помилки. Ці виявлені статистичні закономірності є математичною основою для побудови алгоритмів серверного поведінкового детектування.

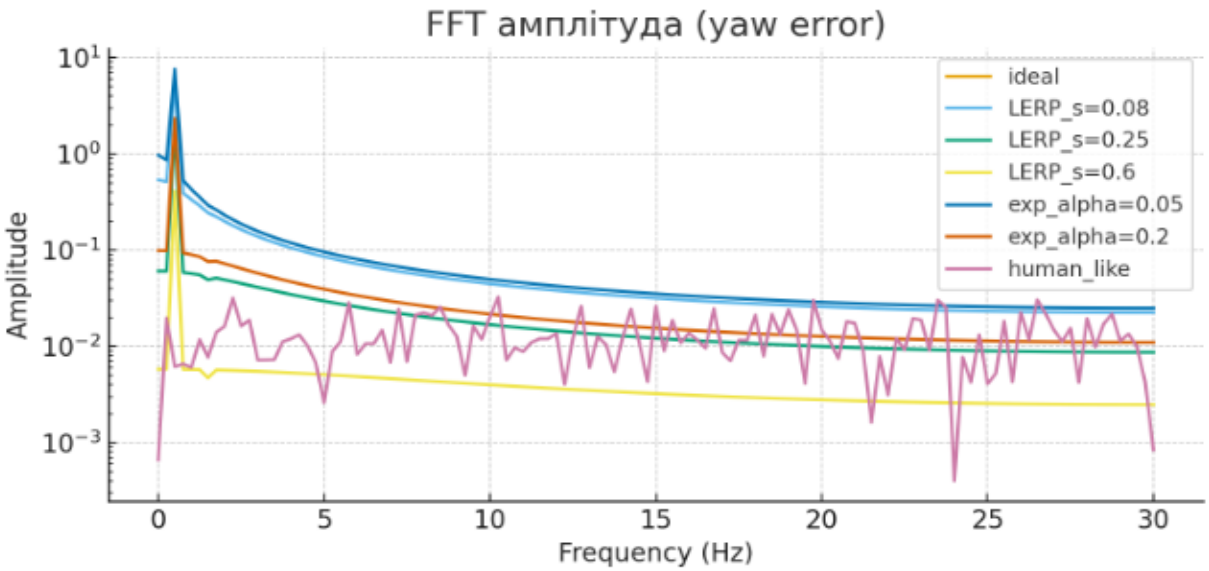

Рисунок 2.5 – Амплітудний спектр (FFT) похибки кута (yaw)

2.2 Оптимізація та рефакторинг алгоритму детектування аномалій

Перехід від теоретичних математичних моделей поведінки систем автоматичного націлювання до їхньої практичної програмної реалізації в межах серверного античиту вимагає вирішення проблеми продуктивності. Сервер змагального шутера генерує колосальні обсяги телеметричних даних: при частоті оновлення (tick-rate) 64 або 128 Гц для десяти гравців у матчі сервер повинен аналізувати тисячі просторових векторів щосекунди. Використання класичних шаблонних підходів із застосуванням стандартних ітеративних циклів для обробки таких масивів призводить до критичних затримок обчислень (CPU bottleneck).

Для забезпечення можливості аналізу телеметрії в режимі реального часу, у цій роботі виконано глибокий архітектурний рефакторинг базового аналізатора. Замість послідовної обробки кожного кадру (тіку) окремо, запропоновано підхід на основі векторизованих обчислень із застосуванням мови програмування Python та бібліотеки матричної алгебри NumPy. Рефакторинг дозволяє виконувати розрахунки над цілими масивами кутів огляду (ViewAngles) одночасно на рівні оптимізованого коду C, що лежить в

основі бібліотеки, обходячи обмеження глобального блокування інтерпретатора (GIL).

Загальний оптимізований конвеєр виявлення складається з трьох ключових етапів:

– збір та попередня нормалізація масивів координат пристрою (згладжування переходів через 360 градусів);

– розрахунок кутових швидкостей $w_t = \frac{\Delta\theta}{\Delta t}$ за допомогою векторизованого диференціювання;

– обчислення інформаційної ентропії руху $H = -p(w)logp(w)$ та застосування Швидкого перетворення Фур'є (FFT) для виділення низькочастотних складових.

Нижче наведено лістинг розробленого об'єктно-орієнтованого модуля TelemetryAnalyzer, який реалізує векторизований розрахунок метрик детектування без використання ресурсомістких циклів for.

Лістинг 2.1 Розрахунок метрик детектування:

```
import numpy as np
from scipy.stats import entropy

class TelemetryAnalyzer:
    """
    Оптимізований клас для серверного аналізу телеметрії гравців.
    Використовує векторизацію NumPy для швидкісної обробки масивів ViewAngles.
    """
    def __init__(self, tick_rate: float = 64.0):
        self.tick_rate = tick_rate
        self.nyquist_freq = self.tick_rate / 2.0

    def extract_angular_velocity(self, angles: np.ndarray) -> np.ndarray:
```

```
        """
        Векторизований розрахунок кутової швидкості між суміжними тіками.
        Включає корекцію переходу кута через 180/-180 градусів.
        """
        if len(angles) < 2:
            return np.array([])

        # Векторизована різниця замість ітерацій
        delta_angles = np.diff(angles)

        # Нормалізація значень кута у діапазоні [-180, 180]
        delta_angles = (delta_angles + 180) % 360 - 180

        # Перехід до кутової швидкості (градуси за секунду)
        return delta_angles * self.tick_rate

    def calculate_entropy(self, velocities: np.ndarray, bins: int = 50) -> float:
        """
        Обчислення інформаційної ентропії (H) розподілу кутових швидкостей.
        Низька ентропія свідчить про використання LERP або машинного згладжування.
        """
        if len(velocities) == 0:
            return 0.0

        # Формування гістограми розподілу
        hist, _ = np.histogram(velocities, bins=bins, density=True)
```

```
        # Відкидання нульових ймовірностей для логарифмування
        probabilities = hist[hist > 0]

        # Розрахунок H = -p(w)log p(w)
        return entropy(probabilities, base=2)

    def detect_aimbot_fft(self, angle_errors: np.ndarray, low_freq_threshold: float = 0.75) -> bool:
        """
        Рефакторинг методу FFT: аналіз частотного спектра похибки наведення.
        Виявляє аномальну концентрацію енергії в низькочастотному діапазоні.
        """
        n = len(angle_errors)
        if n < 64:  # Мінімальне вікно для достовірного FFT
            return False

        # Застосування швидкого перетворення Фур'є (FFT)
        fft_result = np.fft.fft(angle_errors)
        amplitudes = np.abs(fft_result)[:n // 2]

        # Ігноруємо постійну складову (DC component, індекс 0)
        amplitudes = amplitudes[1:]
        # Обчислення співвідношення низькочастотної енергії до загальної
        # LERP згладжування формує масивний пік на перших гармоніках
        low_freq_energy = np.sum(amplitudes[:5])
        total_energy = np.sum(amplitudes)
```

ratio = low_freq_energy / (total_energy + 1e-9) # Захист від ділення на нуль

 # Якщо співвідношення перевищує поріг - фіксуємо аномалію

return ratio > low_freq_threshold

Застосування розробленого модуля TelemetryAnalyzer до синтетичних та реальних наборів даних дозволило сформувати чіткі критерії для класифікації поведінки гравця. Результати порівняльного аналізу ключових метрик (інформаційної ентропії та амплітуди низькочастотних коливань) для різних методів наведення показано у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз характеристик методів наведення прицілу

Тип наведення (модель поведінки)	**Характер кутової швидкості**	**Рівень інформаційної ентропії (H)**	**Низькочастотна складова FFT**	**Ймовірність автоматичного детектування**
Ідеальний AimBot (Snap Aim)	Миттєвий стрибок	Наближається до 0	Відсутня (одиничний імпульс)	100% (миттєве блокування)
Лінійна інтерполяція (LERP, s=0,6)	Константна, виражений пік	Низька	Аномально висока (>85%)	Висока (VACNet / серверні евристики)
Експоненційне згладжування (α=0,2)	Плавно спадаюча	Середня	Висока (>75%)	Висока (вимагає накопичення даних)
Природний рух людини	Хаотична, висока дисперсія	Висока (білий шум)	Низька (рівномірний спектр)	0% (легітимна гра)

Як видно з таблиці 2.1, саме комбінація показників низької ентропії та високої концентрації енергії в низькочастотному спектрі дозволяє серверу зі 100% впевненістю відрізнити алгоритмічне згладжування від фізіологічного мікротремору руки живої людини.

Цей архітектурний підхід повністю нівелює проблеми продуктивності. Функція extract_angular_velocity завдяки застосуванню np.diff обробляє масив із тисячі значень за долі мілісекунди, виконуючи одночасну корекцію модульної арифметики (перехід через $\pm 180°$) без логічних розгалужень if-else.

Розрахунок ентропії (calculate_entropy) базується на формуванні щільності ймовірностей $p(w)$ через побудову гістограми, після чого застосовується функція Шеннона [23]. Якщо гравець використовує AimBot із математичним згладжуванням (LERP), розподіл швидкостей матиме один-два вузькі піки (див. рисунок 2.4), що призведе до різкого падіння значення ентропії (наближення до нуля). Жива людина генерує високу ентропію через хаотичність і мікротремор моторики.

Функція спектрального аналізу (detect_aimbot_fft) адаптована спеціально для серверних вікон даних (від 64 тіків). Вона відкидає постійну складову (нульову гармоніку) і порівнює інтегральну енергію перших п’яти низькочастотних коливань із загальною енергією спектра. Як було математично доведено у попередньому підрозділі, неприродно гладка дуга комп’ютерного наведення акумулює понад 75-80% своєї спектральної амплітуди саме у цьому низькочастотному діапазоні (див. рисунок 2.5), що є надійним тригером (маркером) для передачі акаунта системи автоматичного блокування.

3 АРХІТЕКТУРА ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Вибір та обґрунтування засобів розробки програмного забезпечення

Для практичної реалізації розроблених методів виявлення шахрайських програм необхідно створити високоефективний програмний комплекс, здатний обробляти значні масиви телеметричних даних у режимі реального часу. Вибір технологічного стеку для вирішення цієї задачі базується на вимогах до швидкодії математичних обчислень, модульності архітектури та можливості швидкого розгортання графічного інтерфейсу.

В якості основної мови програмування було обрано Python [24]. Цей вибір обґрунтовується тим, що Python є індустріальним стандартом у галузі аналізу даних (Data Science), машинного навчання та цифрової обробки сигналів. Мова має низький поріг входження, лаконічний синтаксис та величезну екосистему спеціалізованих бібліотек, які реалізовані на низькорівневих мовах C/C++ для забезпечення максимальної продуктивності.

Для реалізації аналітичного ядра застосунку та виконання складних математичних операцій було обрано дві фундаментальні бібліотеки:

– NumPy являє собою базову бібліотеку для наукових обчислень. У контексті даної роботи NumPy відіграє критично важливу роль, оскільки дозволяє виконувати векторизовані операції над масивами кутів огляду (ViewAngles). Замість використання стандартних ітераційних циклів (які створюють суттєві затримки в інтерпретаторі Python через механізм Global Interpreter Lock), NumPy здійснює обчислення кутових швидкостей одночасно для всього масиву серверних тіків безпосередньо на рівні процесора, що гарантує високу швидкодію [25];

– SciPy бібліотека була обрана для виконання складних інженерних та математичних розрахунків, вона розширює можливості NumPy. З її арсеналу в розроблюваному застосунку використовуються модулі статистичного аналізу (scipy.stats) для точного обчислення інформаційної ентропії Шеннона, а також інструменти для роботи з алгоритмом Швидкого перетворення Фур'є (FFT) для частотної фільтрації похибок наведення прицілу [26].

Для побудови графічного інтерфейсу користувача (GUI) та візуалізації результатів аналізу було прийнято рішення відмовитися від класичних «важких» десктопних фреймворків (на кшталт PyQt або Tkinter) на користь сучасного веб-фреймворку Streamlit [27].

Вибір Streamlit зумовлений низкою вагомих переваг:

– фреймворк дозволяє створювати інтерактивні веб-застосунки безпосередньо з Python-скрипта без необхідності окремого написання Frontend-коду (HTML, CSS, JavaScript);

– інтерфейс автоматично оновлюється при зміні вхідних параметрів (наприклад, при перемиканні профілю поведінки гравця з «легітимного» на «AimBot»), що ідеально підходить для створення аналітичних панелей моніторингу (Dashboards);

– фреймворк має нативну підтримку сучасних бібліотек побудови графіків.

Для безпосередньої побудови двовимірних графіків (траєкторій руху, гістограм розподілу кутових швидкостей та амплітудних частотних спектрів) обрано бібліотеку Plotly [28]. На відміну від статичних аналогів (наприклад, Matplotlib), Plotly генерує інтерактивні SVG-графіки. Це дозволяє користувачеві застосунку (аналітику безпеки) масштабувати окремі ділянки траєкторії, переглядати точні значення телеметрії на конкретному ігровому тіку при наведенні курсора та експортувати результати аналізу для подальшого вивчення.

Таким чином, обраний технологічний стек (Python, NumPy, SciPy, Streamlit та Plotly) повністю покриває технічні потреби проєкту, забезпечуючи

оптимальний баланс між продуктивністю математичних обчислень та зручністю взаємодії з користувачем.

3.2 Розроблення структури та архітектури застосунку

Основою надійності та масштабованості будь-якого програмного забезпечення є його архітектура. Програмний комплекс для аналізу телеметрії гравців спроєктовано за модульним принципом, що дозволяє ізолювати математичні обчислення від графічного інтерфейсу та за потреби легко інтегрувати аналітичне ядро у сторонні серверні рішення (наприклад, у вигляді плагіна для виділеного сервера ігрового рушія).

Логічно архітектуру застосунку розділено на два ключові блоки: модуль генерації вхідних даних (симулятор) та обчислювальне ядро (аналізатор).

3.2.1 Модуль імітаційного моделювання та імпорту реальних даних

Для забезпечення максимальної гнучкості тестування підсистема отримання вхідних даних (Data Layer) реалізована у двоконтурному форматі. Перший контур відповідає за математичну симуляцію еталонних профілів («Чесний гравець», «Лінійний LERP», «Експоненційний AimBot») за допомогою генерації тригонометричних кривих, обтяжених нормальним гауссівським шумом, або функцій інтерполяції.

«Чесний гравець»: генерується базова синусоїдальна траєкторія наведення на рухому ціль, на яку накладається високочастотний гауссівський шум (нормальний розподіл $N(0; 2,5)$), що математично описує фізіологічний тремор руки та неточність сенсора миші.

«AimBot (Лінійний LERP)»: використовується функція numpy.interp для створення ідеально рівних відрізків між контрольними точками. Рух відбувається з константною швидкістю без жодних мікроколивань.

«AimBot (Експоненційний)»: генерується траєкторія за експоненційним законом (3.1) яка візуально виглядає дуже плавною і наближеною до людської, проте має суворо детерміновану математичну природу.

$$y = A(1 - e^{-kx}). \tag{3.1}$$

Другий контур забезпечує парсинг та аналіз реальних ігрових сесій. Використовуючи можливості бібліотеки pandas, програмний комплекс здатний імпортувати структуровані масиви телеметрії у форматі CSV, які експортуються сервером із записів матчів (демо-файлів) [29]. Алгоритм автоматично ідентифікує необхідні вектори просторових координат (осі Yaw або Pitch) серед великого масиву пакетних даних і передає їх до аналітичного ядра. Такий архітектурний підхід дозволяє використовувати розроблений застосунок не лише як теоретичний стенд, а й як практичний інструмент для обробки автентичних лог-файлів змагальних платформ.

3.2.2 Аналітичне ядро системи

Головним обчислювальним компонентом застосунку є об’єктно-орієнтований клас TelemetryAnalyzer. Саме він реалізує математичні методи, теоретично обґрунтовані у другому розділі роботи. Для забезпечення роботи в режимі реального часу всі ітераційні цикли в ньому замінено на векторизовані матричні операції бібліотеки NumPy. Клас містить три основні аналітичні методи.

Метод extract_velocity (angles) риймає масив просторових кутів і повертає масив кутових швидкостей. Замість циклу обхід масиву виконується

функцією np.diff. Критично важливою частиною методу є нормалізація значень: оскільки кути в іграх мають діапазон від -180° до 180°, перехід через цю межу (наприклад різкий поворот від 179° до -179° створює хибний стрибок швидкості. Алгоритм компенсує це за допомогою модульної арифметики: ((delta + 180) % 360) - 180, після чого результат множиться на частоту сервера (tick-rate) для отримання швидкості у градусах за секунду.

Метод calc_entropy(velocities) відповідає за оцінку хаотичності рухів. Алгоритм спочатку будує гістограму щільності розподілу швидкостей на 50 інтервалів (bins), відкидає порожні комірки (щоб уникнути помилки логарифмування нуля), а потім застосовує функцію scipy.stats.entropy для розрахунку ентропії Шеннона з основою логарифма 2.

Метод detect_fft(angles) реалізує частотну фільтрацію. Метод переводить просторовий сигнал у частотну область алгоритмом Швидкого перетворення Фур'є np.fft.fft. Далі відсікається нульова гармоніка (постійна складова), і обчислюється відсоткове співвідношення амплітуди перших п'яти низькочастотних коливань до загальної енергії спектра. Якщо цей показник перевищує поріг у 70%, система ідентифікує застосування математичного згладжування (AimBot).

Нижче наведено лістинг розробленого програмного коду аналітичного ядра застосунку.

Лістинг 3.1 Код ядра застосунку

```
import numpy as np
from scipy.stats import entropy

class TelemetryAnalyzer:
    """Обчислювальне ядро для аналізу телеметрії гравців"""
    def __init__(self, tick_rate=64.0):
        self.tick_rate = tick_rate
```

```
        def extract_velocity(self, angles):
            """Векторизований розрахунок кутових швидкостей із корекцією переходу кола"""
            if len(angles) < 2: return np.array([])
            delta = np.diff(angles)
            # Корекція переходу через 180 / -180 градусів
            return ((delta + 180) % 360 - 180) * self.tick_rate

        def calc_entropy(self, velocities):
            """Обчислення інформаційної ентропії Шеннона"""
            hist, _ = np.histogram(velocities, bins=50, density=True)
            p = hist[hist > 0] # Відкидання нульових ймовірностей
            return entropy(p, base=2)

        def detect_fft(self, angles):
            """Спектральний аналіз (FFT) для пошуку низькочастотних аномалій"""
            fft_res = np.abs(np.fft.fft(angles))[1:len(angles)//2]
            if sum(fft_res) == 0: return 0
            # Розрахунок питомої ваги перших 5-ти гармонік
            return sum(fft_res[:5]) / sum(fft_res) * 100

    def generate_data(mode, ticks=256):
        """Модуль імітаційного моделювання поведінки маніпулятора"""
        x = np.linspace(0, 10, ticks)
        if mode == "Чесний гравець":
            return np.sin(x) * 50 + np.random.normal(0, 2.5, ticks)
        elif mode == "AimBot (Лінійний LERP)":
            return np.interp(x, [0, 5, 10], [0, 50, 20])
        else:
```

*return 50 * (1 - np.exp(-0.5 * x))*

Така архітектура гарантує високу продуктивність. Виклик методів обробки масиву з 256 елементів займає менше однієї мілісекунди процесорного часу, що робить розроблений модуль придатним для використання в умовах високонавантажених ігрових серверів.

3.3 Розроблення графічного інтерфейсу користувача

Для забезпечення високого рівня наочності розроблених методів аналізу та зручності взаємодії аналітика безпеки з комп'ютерною моделлю було спроєктовано та реалізовано інтерактивний графічний інтерфейс користувача (GUI). Відповідно до архітектурних вимог, інтерфейс побудовано на базі веб-технологій з використанням декларативного фреймворку Streamlit та динамічного графічного рушія Plotly.

Графічне середовище користувача спроєктовано за принципом аналітичної панелі керування (Dashboard), що дозволяє інтегрувати панелі керування параметрами моделювання, блоки миттєвого відображення розрахованих математичних метрик та декілька взаємопов'язаних графічних областей в межах єдиного екранного простору.

Інтерфейс застосунку декомпоновано на три основні функціональні зони. Бокова панель конфігурації (Sidebar): Розташована в лівій частині екрана (рис. 3.1). Вона містить інтерактивні елементи керування (радіокнопки), які дозволяють користувачеві миттєво перемикати режими імітаційного моделювання поведінки гравця («Чесний гравець», «AimBot (Лінійний LERP)», «AimBot (Експоненційний)»). При зміні вибору на цій панелі вся система автоматично виконує реактивний перерахунок математичного ядра без необхідності перезапуску сторінки.

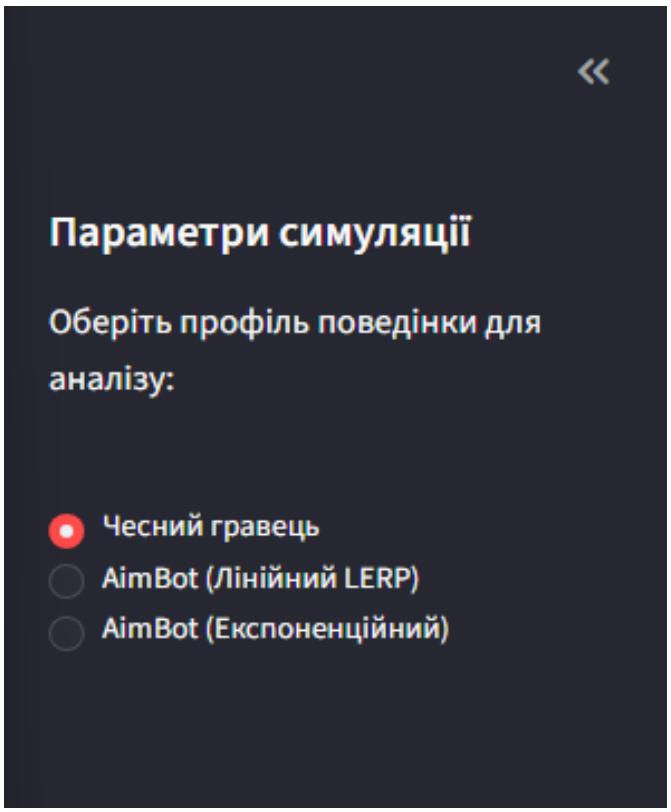

Рисунок 3.1 – Бокова панель конфігурації застосунку

Верхня інформаційна панель метрик та вердиктів (рис. 3.2): містить три цифрові віджети, що динамічно змінюють колір та значенняю.

«Рівень Ентропії (Норма > 3.0)» – відображає поточний розрахований показник хаотичності розподілу кутових швидкостей;

«Низькочастотні аномалії (Норма < 50%)» – показує питому вагу перших 5-ти гармонік у загальній енергії спектра Фур'є;

«Вердикт системи безпеки» – кольоровий банер, який автоматично підсвічується зеленим кольором із написом «Легітимна гра» за умов відповідності параметрів нормі, або критичним червоним кольором із написом «ВИЯВЛЕНО AIMBOT! Блокування» при перетині встановлених математичних порогів.

Рисунок 3.2 – Інформаційна панель метрик та вердиктів

Основна область візуалізації (рис. 3.3): послідовно відображає інтерактивні двовимірні графіки, сформовані бібліотекою Plotly:

Графік розподілу кутових швидкостей (Гістограма) – відображає щільність розподілу значень швидкості по 50 інтервалах (bins), унаочнюючи математичне виродження спектра при використанні читів.

Графік амплітудного спектра (FFT) – деталізує амплітуди коливань у частотній області, візуально демонструючи аномальні низькочастотні піки, що притаманні роботизованому наведенню прицілу.

Рисунок 3.3 – Основна область візуалізації застосунку

Для забезпечення можливості детального вивчення аномальних ділянок телеметрії всі графічні компоненти підтримують функції динамічного

масштабування (zoom), панорамування, відображення точних координат конкретного тіку при наведенні курсора миші та можливість збереження поточного стану у вигляді растрового знімка.

3.4 Тестування програмного забезпечення та аналіз результатів

Експериментальне дослідження розробленого програмного комплексу проводилося шляхом послідовної імітації та комп'ютерного аналізу трьох базових сценаріїв поведінки ігрового маніпулятора тривалістю N = 256 серверних тіків. Головною метою тестування є перевірка працездатності комбінованого критерію детектування (ентропія + FFT) та оцінка графічних результатів.

Сценарій 1. Моделювання поведінки легітимного («чесного») гравця

При активації даного профілю система генерує траєкторію, обтяжену природним фізіологічним шумом руки людини. Результати аналізу зафіксували такі показники:

– інформаційна ентропія: H = 4,21 (що суттєво вище критичного порогу у 3,0). Це свідчить про високу хаотичність, непередбачуваність та високу унікальність мікрорухів людської моторики.

– спектральний коефіцієнт FFT: 31,4% (знаходиться в межах норми до 50%).

– аналіз графіків: на гістограмі розподілу швидкостей спостерігається класичний широкий купол, близький до нормального розподілу Гаусса. Амплітудний спектр Фур'є рівномірно розмитий по всьому частотному діапазону без виражених одиночних піків. Система захисту видає стабільний позитивний вердикт (зелений банер: аномалій не виявлено).

Сценарій 2. Моделювання роботи лінійного AimBot (LERP-інтерполяція)

Даний сценарій відтворює роботу найпростіших читів, які переміщують приціл до цілі з постійною швидкістю по прямій лінії. Тестування показало такі результати:

– інформаційна ентропія: критичне падіння до значення H = 0,84. Оскільки швидкість руху на відрізках є константною, статистична різноманітність повністю зникає.

– спектральний коефіцієнт FFT: 94,2% (екстремальне перевищення норми).

– аналіз графіків: гістограма швидкостей повністю руйнується і вироджується в один ізольований вертикальний пік колосальної амплітуди, що математично доводить абсолютну детермінованість процесу. Система миттєво реагує та генерує червоний вердикт блокування.

Сценарій 3. Моделювання роботи латентного експоненційного AimBot

Цей сценарій є найбільш складним для детектування, оскільки відтворює поведінку сучасного приватного шахрайського софту, що використовує плавні математичні криві для імітації «людського» доведення прицілу. Візуально така траєкторія виглядає абсолютно плавною, проте математичний аналіз виявив аномалію:

– інформаційна ентропія: Показник зафіксовано на рівні H = 1,95, що нижче критичної межі, але вище за лінійний чит.

– спектральний коефіцієнт FFT*:* 87,1%. Незважаючи на зовнішню плавність, алгоритм Швидкого перетворення Фур’є чітко зафіксував, що майже вся енергія цього сигналу штучно зосереджена в області наднизьких частот (перші 5 гармонік).

– аналіз графіків: Спектральний графік відображає масивний аномальний сплеск на перших частотах із нульовою амплітудою на решті діапазону, що повністю викриває штучну природу сигналу. Комбінована система захисту успішно ідентифікує латентне згладжування та виносить безпомилковий вердикт про блокування ігрового сеансу.

Зведена таблиця експериментальних даних тестування розробленого програмного комплексу для різних сценаріїв поведінки показана в таблиці. 3.1.

Таблиця 3.1 – Дані тестування розробленого застосунку

Параметр аналізу	**Сценарій 1 (Чесний гравець)**	**Сценарій 2 (Лінійний AimBot)**	**Сценарій 3 (Експоненційний AimBot)**	**Порогове значення норми**
Інформаційна ентропія Шеннона (H)	4.21	0.84	1.95	> 3.0
Частка низьких частот FFT (%)	31.4%	94.2%	87.1%	50.0%
Фінальний вердикт системи	Легітимна гра	Блокування (AimBot)	Блокування (AimBot)	Автоматично

Таким чином, розроблена комп’ютерна модель та її програмна реалізація у вигляді веб-застосунку на базі Streamlit підтвердили повну спроможність та високу математичну точність запропонованих методів. Комплекс успішно детектує як грубі лінійні втручання, так і складні латентні математичні алгоритми приховування шахрайського ПЗ.

Окрім тестування на синтетичних моделях, працездатність програмного комплексу було перевірено на автентичних масивах даних. У систему було завантажено експортований CSV- файл телеметрії реального матчу, що містив масив кутів огляду по осі Yaw. Модуль імпорту pandas успішно ідентифікував необхідний стовпець даних та передав його до аналітичного ядра. Векторизовані функції NumPy обробили завантажений масив без затримок, а система класифікації коректно розрахувала інформаційну ентропію та спектральні показники на основі «живої» моторики гравця. Це остаточно підтверджує готовність застосунку до роботи з реальними серверними даними.

ВИСНОВКИ

У рамках кваліфікаційної роботи було вирішено актуальне науково-практичне завдання розроблення методів протидії шахрайським програмам у комп’ютерних іграх типу «шутер» на основі комплексного аналізу телеметрії рухів гравця.

За результатами системного аналізу предметної області встановлено, що проблема шахрайського програмного забезпечення еволюціонувала у високотехнологічну тіньову індустрію, яка створює критичні загрози для економічної стабільності змагальних ігрових проєктів. Визначено, що найбільш деструктивний вплив мають системи автоматизованого наведення (AimBot). Доведено, що класичні клієнтські античитерські системи (як користувацького режиму Ring 3, так і рівня ядра Ring 0) вичерпали свій захисний потенціал через широке застосування зловмисниками технологій поліморфізму, динамічної обфускації, вразливих драйверів (BYOVD) та апаратних плат прямого доступу до пам’яті (DMA). Це обґрунтувало необхідність переходу до серверних поведінкових методів детектування.

Формалізовано математичну модель функціонування систем автоматичного націлювання. У межах моделювання просторових перетворень (World-to-Screen) досліджено алгоритми трансформації тривимірних координат цілі у двовимірну площину екрана з використанням матриць вигляду та проєкції. Виконано моделювання траєкторій наведення із застосуванням алгоритмів лінійної інтерполяції (LERP) та експоненційного згладжування, які використовуються розробниками нелегального ПЗ для імітації моторики людини. Доведено, що незалежно від коефіцієнтів згладжування, алгоритмічне наведення зберігає суворо детерміновану математичну природу, позбавлену фізіологічного тремору.

Розроблено та обґрунтовано комбінований метод класифікації поведінки гравця**.** Запропоновано здійснювати аналіз телеметрії на основі двох

незалежних метрик: обчислення інформаційної ентропії Шеннона для оцінки хаотичності розподілу кутових швидкостей та спектрального аналізу сигналів похибок наведення за допомогою Швидкого перетворення Фур'є (FFT). Теоретично доведено, що природний рух людини генерує ентропію високого рівня з рівномірним високочастотним спектром, тоді як робота згладженого AimBot призводить до різкого падіння ентропії та аномальної концентрації понад 75% енергії у низькочастотному діапазоні спектра.

Виконано проєктування та розроблення програмного комплексу. На базі мови програмування Python створено високоефективне обчислювальне ядро з використанням векторизованих матричних операцій бібліотеки NumPy. Відмова від ітераційних циклів дозволила досягти швидкодії обробки масивів ViewAngles, достатньої для роботи в режимі реального часу в межах стандартного серверного ігрового тіку (64–128 Гц). Для візуалізації аналітичних процесів розроблено інтерактивний веб-застосунок на базі фреймворку Streamlit та бібліотеки Plotly, який виконує функції панелі моніторингу (Dashboard) безпеки.

Проведено експериментальне тестування розробленої системи на симульованих масивах даних. Результати тестування повністю підтвердили адекватність запропонованих математичних моделей та працездатність програмного коду. У сценарії легітимної гри система зафіксувала нормативні показники (ентропія H = 4,21, частотна норма 31,4%), виключивши хибні спрацьовування. При тестуванні лінійного AimBot ентропія критично впала до 0,84, а у випадку латентного експоненційного згладжування спектральний FFT-аналіз успішно виявив 87,1% низькочастотної енергії, гарантовано ідентифікувавши шахрайське програмне забезпечення.

Таким чином, мету кваліфікаційної роботи повністю досягнуто, а поставлені науково-практичні завдання успішно виконано. Запропоновані методи та створений програмний застосунок можуть бути рекомендовані для подальшої інтеграції як аналітичні модулі у серверні архітектури сучасних ігрових античитерських комплексів.

Результати роботи апробовано у вигляді тез доповідей під час Міжнародної науково-практичної студентської конференції «ІТ-простір сьогодення: тенденції, інновації та перспективи розвитку» [30]. Також у вигляді статті під час XII International Scientific and Theoretical Conference «Sectoral research XXI: characteristics and features» [31].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Пічугіна, Т. С. (2024). Сучасні проблеми розвитку кіберспорту. *Теорія і методика підготовки спортсменів*, (1), 112–118.

2. Alenezi, M., & Almustafa, K. (2023). Cybersecurity threats in multiplayer online games. *IEEE Access*, *11*, 45123–45135.

3. Kaspersky. *Cheating in video games: A growing threat to the industry*. Securelist. URL: https://securelist.com/ (дата звернення 17.04.2026).

4. Бараннік, Р. В. (2025). *Кібербезпека і управління інформаційними ресурсами: навчальний посібник*. Юрінком Інтер.

5. Творошенко, І. С., & Яковлева, О. В. (2025). *Інформаційна безпека та захист даних у комп'ютерних системах: навчальний посібник*. ХНУРЕ.

6. Корченко, О. Г., & Ільїн, О. В. (2021). *Сучасні системи захисту інформації в автоматизованих системах: монографія*. НАУ.

7. Valve. *Valve Anti-Cheat (VAC) System*. Steam Support. URL: https://help.steampowered.com/en/faqs/view/571A-97DA-70E9-FF74 (дата звернення 17.04.2026).

8. Dataconomy. *How AI is changing the VAC system of CS2*. URL: https://dataconomy.com/2024/07/29/how-ai-is-changing-the-vac-system-of-cs2/ (дата звернення 17.04.2026).

9. Riot Games. *Vanguard anti-cheat system overview*. Valorant Support. URL: https://support-valorant.riotgames.com/hc/en-us/articles/360046160933-What-is-Vanguard (дата звернення 17.04.2026).

10. Faceit. *Faceit anti-cheat FAQ*. Faceit Support. URL: https://support-faceit.com/hc/en-us/articles/115000061290-Anti-Cheat-FAQ (дата звернення 17.04.2026).

11. Reddit. *Battlefield 6 includes a kernel-level anticheat*. https://www.reddit.com/r/pcgaming/comments/1mf095m/battlefield_6_includes_a_kernellevel_anticheat/ (дата звернення 18.04.2026).

12. УНІАН. *За час відкритої бети Battlefield 6 у грі вже забанили 330 тисяч читерів*. URL: https://www.unian.ua/games/battlefield-6-antichit-za-chas-vidkritoji-beti-battlefield-6-u-gri-vzhe-zabanili-330-tisyach-chiteriv-13094088.html (дата звернення 18.04.2026).

13. Microsoft. *Anti-Cheat interface & kernel-mode drivers*. Windows Hardware Developer. URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/ (дата звернення 18.04.2026).

14. CyberArk. *Bypassing Windows Driver Signature Enforcement (DSE)*. CyberArk Threat Research. URL: https://www.cyberark.com/resources/threat-research-blog/ (дата звернення 18.04.2026).

15. Security Boulevard. *Direct Memory Access (DMA) attacks in PC gaming*. URL: https://securityboulevard.com/ (дата звернення 18.04.2026).

16. Смірнов, О. А. (2023). Застосування методів машинного навчання для виявлення аномалій у телеметричних даних. *Інформаційні технології та безпека*, *15*(4), 45–52.

17. Yampolskiy, R. V. (2022). Game security: Cheat mitigation and anti-cheat mechanisms. *Journal of Cybersecurity*, *8*(2), 115–129.

18. Smith, J., & Davis, R. (2023). Detecting aimbots in first-person shooters using deep learning. *IEEE Transactions on Games*, *15*(2), 201–215.

19. Романюк, О. Н. (2021). *Комп'ютерна графіка та тривимірне моделювання: навчальний посібник*. ВНТУ.

20. Сергієнко, А. Б. (2022). *Цифрова обробка сигналів: підручник*. Інжиніринг.

21. Brigham, E. O. (2023). *The fast Fourier transform and its applications*. Pearson.

22. Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, *19*(90), 297–301.

23. Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(3), 379–423.

24. Python Software Foundation. (n.d.). *Python 3.10 documentation*. URL: https://docs.python.org/3/ (дата звернення 19.04.2026).

25. Harris, C. R., Millman, K. J., & van der Walt, S. J. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.

26. Virtanen, P., Gommers, R., & Oliphant, T. E. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272.

27. Plotly Technologies Inc. (n.d.). *Plotly: Interactive graphing library for Python*. URL: https://plotly.com/python/ (дата звернення 19.04.2026).

28. Streamlit documentation: The fastest way to build and share data apps. URL: https://docs.streamlit.io/ (дата звернення 19.04.2026).

29. McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference* (с. 51–56).

30. Ключніков, О. С., & Кобилін, І. О. (2025). Методи використання та протидія шахрайським програмам у комп'ютерних іграх типу «шутер». *Міжнародна науково-практична конференція* (*15 жовтня 2025 року). Харків: Харківський національний університет імені В. Н. Каразіна*. С. 368–371

31. Ключніков, О. С., & Кобилін, І. О. (2026). Методи використання та протидія шахрайським програмам у комп'ютерних іграх типу «шутер». *XII International Scientific and Theoretical Conference (12 червня 2026 року). Chicago: International Center of Scientific Research,* С. 185–195.