BLOCKCHAIN STORAGE OPTIMIZATION FOR V2X DATA INTEGRITY

24-25J-220

Project Final Report

Kuhananth C IT21302244

BSc (Hons) Information Technology Specializing in Data Science

Department of Information Technology

Sri Lanka Institute of

Information Technology

April 2025

Declaration

I declare that this is my own work, and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of our knowledge and belief it does not contain any material previously publish or written by another person expect where the acknowledgement is made in the text.

Name	Student_ID	Signature
Kuhananth C	IT21302244	

The above candidate is carrying out research for the undergraduate Dissertation under my supervision.

……

Signature of the supervisor:

(Mr. Samadhi Rathnayake)

Date:……11/04/2025……..

……..

Signature of the Co-supervisor:

(Mr. Nelum Amarasena)

Date:……11/04/2025…….

Acknowledgement

I would like to express my sincere gratitude to my dissertation supervisor, Mr. Samadhi Rathnayake, and co-supervisor, Mr. Nelum Amarasena, from the Faculty of Computing, for their invaluable guidance and support throughout the research process. Their insightful comments and constructive critiques have significantly helped in refining my research and enhancing the overall quality of my work.

I am also deeply appreciative of the staff of the Faculty of Computing for their assistance and encouragement throughout my studies. Their commitment to teaching and research has been a continuous source of inspiration to me.

Further, I am thankful to my fellow team members for their support and collaboration. Our discussions and exchange of ideas have greatly enriched my understanding of the subject matter.

Lastly, I extend my thanks to all those who participated in this study and contributed their time and insights. Their willingness to share their experiences and perspectives has been invaluable in deepening my understanding of the subject matter.

Abstract

The rise of intelligent transportation systems and connected vehicles has led to an exponential increase in vehicular data generation, particularly in Vehicle-to-Everything (V2X) communication environments. These data streams—often in the form of structured logs such as sensor readings, vehicle events, and traffic violations—demand secure, tamper-proof, and efficient storage mechanisms. Blockchain, with its decentralized and immutable architecture, presents a promising solution. However, conventional blockchain platforms like Ethereum are not inherently optimized for storing large or frequent datasets due to block size limitations and high gas fees. This research addresses these challenges by proposing an optimized blockchain storage solution that leverages data compression techniques to reduce file size prior to on-chain storage.

The proposed system integrates BZ2 compression and Base64 encoding within a Python Flask backend, enabling automated preprocessing and seamless interaction with Ethereum smart contracts deployed via the Ganache local blockchain. A ReactJS frontend is developed to provide users with intuitive upload and retrieval functionality. The system is tested using real-world vehicular data logs in Excel and CSV formats to evaluate compression efficiency, gas usage, and retrieval accuracy. Comparative analysis of multiple compression algorithms—including ZIP, GZIP, BZ2, LZMA, and 7Z—demonstrates that BZ2 offers the best balance between compression ratio and gas cost reduction.

The results validate that this approach significantly improves blockchain storage performance, enabling secure, verifiable, and cost-effective data archival for V2X communication systems. The research establishes a foundation for scalable deployment in smart transportation frameworks and introduces a replicable model for blockchain-based file storage optimization across domains.

Keywords: Blockchain, V2X Communication, Data Compression, BZ2, Smart Contracts, Ethereum, Ganache, Flask, ReactJS, Vehicular Data Storage, Base64 Encoding, Gas Fee Optimization, Decentralized Storage, File Archival System, Intelligent Transportation Systems

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Full Form
AI	Artificial Intelligence
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BZ2	Bzip2 Compression Format
CSV	Comma-Separated Values
DApp	Decentralized Application
ETH	Ether (Cryptocurrency used in Ethereum)
GAN	Generative Adversarial Network *(if referenced, optional)*
GUI	Graphical User Interface
GZIP	GNU Zip Compression Format
HTTP	Hypertext Transfer Protocol
IPFS	Interplanetary File System
JSON	JavaScript Object Notation
JS	JavaScript
LZ4	Lempel–Ziv 4 Compression
LZMA	Lempel–Ziv–Markov Chain Algorithm
MB	Megabyte
MD5	Message-Digest Algorithm 5
OBD-II	On-Board Diagnostics Version 2
RAM	Random Access Memory
SHA-256	Secure Hash Algorithm 256-bit
UI	User Interface
URL	Uniform Resource Locator
VS Code	Visual Studio Code
V2X	Vehicle-to-Everything
V2I	Vehicle-to-Infrastructure
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-to-Vehicle
V2N	Vehicle-to-Network
ZIP	ZIP File Compression Format
ZSTD	Zstandard Compression Algorithm

Table 1 Abbreviations

1. Introduction

1.1Background

Blockchain technology, first popularized by Bitcoin's introduction by Satoshi Nakamoto in 2008, has rapidly evolved beyond cryptocurrencies, demonstrating enormous potential across various sectors due to its secure, decentralized, and immutable nature. Unlike traditional centralized systems that store data in single locations, making them susceptible to single points of failure, unauthorized access, and data tampering, blockchain technology operates on a decentralized network structure where data is replicated and maintained across multiple nodes. This ensures transparency, security, and resilience, significantly reducing vulnerabilities associated with centralized databases.

Vehicle-to-Everything (V2X) communication represents a cornerstone of modern intelligent transportation systems, facilitating real-time interactions between vehicles and various environmental components such as other vehicles (V2V), infrastructure (V2I), pedestrians (V2P), and network services (V2N). The reliability, security, and integrity of data exchanged in V2X environments are paramount, especially in applications directly linked to road safety, traffic management, and autonomous vehicle operations. Traditional vehicular communication systems often utilize centralized databases or cloud-based solutions. While these solutions offer efficient storage and rapid data retrieval, they inherently carry significant security and reliability risks. Blockchain emerges as a robust alternative due to its intrinsic properties—immutability, cryptographic security, and decentralization—enabling secure and trustworthy data exchanges essential for the real-time decision-making critical in vehicular environments.

V2X communications necessitate rapid, secure, and reliable transmission of data to prevent accidents, reduce congestion, and optimize traffic flow. These real-time data exchanges demand high levels of accuracy and integrity, making blockchain's cryptographic and immutable properties especially valuable. Consequently, blockchain technology is increasingly viewed as an optimal solution capable of addressing the stringent requirements of vehicular communications, including reliability, transparency, security, and decentralized control.

1.2Literature Review

Blockchain technology is an innovative distributed ledger technology characterized by decentralization, transparency, immutability, and cryptographic security. Initially introduced as the backbone of Bitcoin by Satoshi Nakamoto in 2008 [1], blockchain technology has rapidly expanded beyond its cryptocurrency origins to impact various sectors significantly. It is now prominently employed in diverse fields including finance, healthcare, supply chain management, and vehicle-to-everything (V2X) communication systems.

The fundamental attributes of blockchain—decentralization and immutability—offer distinct advantages over traditional centralized systems, which are prone to vulnerabilities such as single points of failure, data corruption, and unauthorized access. Blockchain's distributed nature means that data is stored redundantly across multiple nodes in a network, thereby significantly enhancing system reliability and security [2].

Blockchain transactions are chronologically grouped into blocks and secured through cryptographic hashing. Each block is linked to its predecessor, forming an immutable chain of records. This architecture ensures transparency and resilience against tampering or alteration [3]. Furthermore, blockchain networks utilize consensus mechanisms such as Proof-of-Work (PoW), Proof-of-Stake (PoS), and Delegated Proof-of-Stake (DPoS) to validate and secure transactions, adding robustness to the data integrity [4].

However, despite these significant benefits, blockchain faces notable challenges including scalability limitations, high transaction costs, and slow processing speeds, especially evident when managing substantial volumes of data, hindering its wider practical application [5].

Effective data storage remains one of the critical aspects of blockchain technology. Different blockchain platforms employ diverse methods to address this challenge. Ethereum, a widely adopted blockchain platform, uses smart contracts to automate transaction data storage and execution. While Ethereum provides transparency and automation, it faces significant drawbacks in terms of high gas fees, particularly when storing and handling larger datasets [6].

The Interplanetary File System (IPFS) complements blockchain platforms by providing decentralized storage solutions. IPFS employs content-addressing methods via unique cryptographic hashes, significantly enhancing data retrieval efficiency, redundancy reduction, and cost-effectiveness. Nonetheless, IPFS systems require incentivization schemes to ensure data permanence, as the decentralized nodes independently manage data availability without a central enforcement mechanism [7].

Hyperledger Fabric, developed under the Linux Foundation's Hyperledger project, offers a permissioned blockchain tailored for enterprise applications. Hyperledger Fabric supports flexible governance structures, enhanced privacy, and scalable performance, effectively addressing enterprise-grade requirements. However, deploying and managing Hyperledger solutions generally requires significant expertise and infrastructure investment, limiting immediate applicability in smaller-scale environments or less resource-intensive applications [8].

Data compression techniques are essential for optimizing blockchain storage efficiency, improving transaction speeds, and reducing operational costs. Among various compression methods, ZIP compression is widely recognized for its balanced compression ratio, efficiency, and compatibility across multiple platforms. ZIP utilizes DEFLATE algorithms, achieving considerable data size reduction suitable for blockchain storage, significantly reducing associated storage and transaction costs [9].

Similarly, gzip compression, employing similar algorithms to ZIP, offers slightly better compression ratios for particular data types. It is extensively utilized in Unix-based systems due to its robust performance and wide availability. However, gzip may demand slightly more computational resources, affecting real-time application performance negatively [10].

The 7-Zip compression method, leveraging the LZMA and LZMA2 algorithms, achieves superior compression ratios, significantly reducing file sizes. Despite these advantages, 7-Zip requires substantially more computational resources for compression and decompression processes, limiting its suitability for real-time, resource-constrained blockchain applications [11].

Base64 encoding, although not technically a compression technique, plays a crucial role in blockchain data management by converting binary data into an ASCII text format. This encoding method ensures safe and efficient transmission and storage of compressed binary data in blockchain-based smart contracts, despite introducing minimal overhead [12].

Several studies have specifically addressed blockchain storage optimization. Wang et al. (2025) extensively reviewed blockchain storage optimization methodologies, highlighting the effectiveness of compression techniques, particularly ZIP and gzip, in reducing blockchain storage overhead while balancing computational performance [13].

Zhang et al. (2022) emphasized the potential of entropy and dictionary-based compression methods, demonstrating their effectiveness in further optimizing blockchain storage. Their analysis suggested combining these methods could significantly enhance storage efficiency in blockchain systems [14].

Maso (2020) investigated various compression algorithms within blockchain contexts, specifically identifying ZIP-based compression combined with Base64 encoding as a practical, secure, and efficient method suitable for blockchain applications demanding reliable and rapid data transactions [15].

1.3Research Significance

This research contributes significantly to the practical applicability and broader understanding of blockchain technology within V2X communication systems. By effectively addressing and mitigating critical blockchain implementation challenges—such as high storage costs, transaction latency, and inefficiency in handling large datasets—the proposed solution substantially enhances blockchain's suitability for real-world vehicular applications.

The integration of automated ZIP compression and Base64 encoding techniques, operationalized through a robust Flask backend and secure Ethereum smart contracts, represents an innovative approach that directly addresses identified limitations in traditional blockchain storage methods. This optimization not only facilitates significant economic and operational benefits by drastically reducing storage requirements and associated transaction costs but also ensures that data integrity and reliability remain uncompromised.

Additionally, the findings and methodologies presented in this research have broader implications beyond vehicular communication systems. The optimization techniques and integration strategies developed can be adapted and generalized to various other domains requiring secure, efficient, and reliable blockchain data management solutions, thus contributing widely to blockchain technology's scalability and practical applicability across diverse sectors.

In summary, this research provides foundational insights, practical methodologies, and substantial improvements in blockchain storage solutions. It not only addresses immediate challenges facing vehicular communications but also paves the way for further research and innovation in blockchain technology applications, enhancing overall data management practices in multiple real-world contexts.

1.4Gap Analysis

Although considerable research has been conducted on blockchain storage optimization techniques, a substantial gap remains concerning comprehensive, integrated implementations tailored explicitly for vehicular communication systems. Prior studies generally lack detailed, practical integration strategies encompassing automated backend operations, robust smart contract deployment, and intuitive frontend interfaces.

This research addresses this critical gap by proposing an integrated, automated blockchain storage optimization solution specifically designed for vehicular communication systems. The approach combines ZIP compression with Base64 encoding through a Flask-based backend, securely managed by Ethereum smart contracts deployed on the Ganache blockchain network. By effectively addressing transaction speed, cost-efficiency, data integrity, and user accessibility challenges, this research significantly advances the practical applicability of blockchain technology for critical vehicular data management.

1.5Problem Statement

Blockchain is a powerful enabler of secure and decentralized communication, especially in sensitive environments like V2X networks. However, direct storage of large data files on a blockchain network leads to multiple challenges:

- **High transaction costs**: Ethereum gas fees increase linearly with the size of the data payload.
- **Latency**: Large payloads take longer to be processed and confirmed across the distributed ledger.
- **Storage limitations**: There is a limit to how much data can be stored in a single transaction, necessitating multiple transactions for large files.
- **Scalability issues**: As more data is stored, the blockchain becomes bloated, increasing sync time and storage burden on nodes.

In the case of V2X communication systems, storing critical vehicle data such as event logs, sensor reports, and incident reports on-chain is essential for **auditability, accountability, and transparency**. However, given the limitations mentioned above, storing raw Excel or CSV files directly onto a blockchain is not feasible.

This research project proposes a **compression-based optimization strategy** using the BZ2 algorithm and Base64 encoding to preprocess and reduce file sizes before they are stored on the blockchain. The Flask-based backend automates this process and interacts with Ethereum smart contracts to ensure secure, tamper-proof storage and retrieval. A React-based interface is provided to simplify file submission and download for end-users.

1.6 Research Objectives

In research, objectives serve as the foundational direction and guiding principles that define the scope, intent, and expected outcomes of a study. Objectives not only help structure the methodological framework but also ensure that the study remains focused, measurable, and relevant to the identified problem. In the context of this project—which deals with the integration of blockchain technology into vehicular communication systems (V2X)—defining clear research objectives is crucial due to the complexity and interdisciplinary nature of the topic. This chapter outlines both the general and specific objectives of the study, providing clarity on what the research intends to accomplish and how each component contributes to solving the central problem.

1.6.1 General Objective

The general objective of this research is to **design, implement, and evaluate an optimized blockchain storage solution that incorporates data compression techniques to support secure, cost-effective, and efficient management of critical vehicular data in V2X (Vehicle-to-Everything) communication networks**.

The motivation behind this overarching goal stems from the increasing demand for trustworthy and decentralized data systems in intelligent transportation environments. V2X communication involves real-time data exchange between vehicles and surrounding entities such as infrastructure (V2I), pedestrians (V2P), other vehicles (V2V), and network services (V2N). This data is critical for improving traffic safety, reducing congestion, and enabling autonomous vehicle decision-making. However, storing such vast and sensitive data securely presents a challenge, especially when centralized systems are involved, which are prone to single points of failure and manipulation risks.

Blockchain emerges as a powerful solution due to its decentralization, immutability, and tamper-proof characteristics. Nevertheless, directly storing large files like Excel-based logs or sensor dumps on-chain incurs high gas costs and is constrained by block size limits. Therefore, this study seeks to enhance the practicality of blockchain as a data storage mechanism by compressing data before storage, reducing file size, improving transaction efficiency, and preserving data fidelity during retrieval.

In summary, the general objective encapsulates the need for a robust, verifiable, and efficient blockchain-based archival system that leverages data compression and modern web technologies to manage V2X data with reduced cost and improved usability.

1.6.2 Specific Objectives

To achieve the general goal, the research is further broken down into a series of **specific objectives**, each addressing a critical technical or conceptual component of the system. These specific objectives are measurable, sequential, and directly contribute to solving the challenges posed in the problem statement. They also align with the implementation pipeline, from algorithm evaluation to deployment and testing.

Objective 1: To conduct a comprehensive review of existing data compression and blockchain storage optimization techniques.

Before developing any solution, a deep understanding of current approaches to blockchain storage and data compression is essential. This objective involves an exhaustive literature review of various compression algorithms—such as ZIP, GZIP, BZ2, LZMA, and 7-Zip—and their impact on blockchain storage efficiency. It also includes reviewing current limitations in Ethereum-based storage, such as gas fee overheads and block size limitations. This analysis sets the foundation for selecting the most appropriate compression method for structured vehicular data.

Objective 2: To develop a Python Flask-based backend capable of automating the compression, encoding, and blockchain interaction processes.

The backend of the system is a crucial component that automates the handling of incoming files, compresses them using selected algorithms (primarily BZ2, as later justified), encodes the compressed data using Base64, and then sends it to the smart contract for storage. Flask was selected for its lightweight, scalable architecture and seamless compatibility with the Web3.py library for Ethereum blockchain communication. This objective ensures that the system can handle uploads, trigger compression functions, encode binary data, and manage secure storage without manual intervention.

Objective 3: To design and deploy Ethereum smart contracts using Solidity for the secure, timestamped storage and retrieval of compressed data.

The smart contract layer ensures transparency, traceability, and trust in data handling. These objective covers writing Solidity contracts that store encoded compressed data as strings and map each file to a timestamp and filename. The contract should also allow secure retrieval via indexed queries. This step is implemented and tested on Ganache—a personal Ethereum blockchain emulator—allowing rapid prototyping without incurring real transaction costs. This objective guarantees that once files are uploaded, their integrity and traceability are verifiable via blockchain logs and transaction hashes.

Objective 4: To create a ReactJS-based frontend application that enables users to upload, view, and download files via an intuitive web interface.

An essential component of the system is the user interface, which provides accessibility to non-technical users. This objective involves developing a responsive frontend in ReactJS that allows users to upload Excel or CSV files, view a list of all stored files (along with timestamps and IDs), and retrieve/download files from the blockchain. The React frontend interacts with the Flask backend through RESTful APIs and visualizes the upload and retrieval process in a user-friendly manner.

Objective 5: To rigorously test and evaluate the performance of the system using real-world vehicular data scenarios.

Testing is critical for validating the proposed solution. This objective involves simulating V2X use cases by uploading structured Excel/CSV files containing data like license plates, timestamps, speed, tolls, and lane violations. Tests are conducted to measure the following parameters:

- **Compression efficiency (percentage reduction in file size)**
- **Transaction gas cost for storage and retrieval**
- **Retrieval speed and decompression time**
- **Accuracy of restored data (via hash comparison if needed)**

These metrics help quantify the system's value and verify whether compression leads to measurable performance gains in a blockchain environment.

Objective 6: To compare the performance of multiple compression algorithms (ZIP, GZIP, BZ2, 7Z, etc.) and determine the most efficient one in terms of blockchain gas optimization.

This objective focuses on performance benchmarking. By storing the same Excel file compressed using different algorithms, the study compares the resulting file sizes and the gas costs incurred when storing them on-chain. The comparison ultimately leads to the selection of **BZ2** as the optimal method due to its favorable compression ratio and lower gas usage. This decision is backed by both empirical data and compatibility analysis with blockchain smart contract constraints.

1.7 Innovation and Uniqueness of the Proposed Solution

What distinguishes this research from prior studies is its **unique focus on compression-driven blockchain optimization** for V2X data using **BZ2 encoding**, tailored for structured Excel formats. While many blockchain research efforts focus on encryption, consensus algorithms, or interoperability, fewer emphasize the **practical problem of minimizing storage bloat and cost**, especially for datasets generated continuously by vehicular systems.

The novel aspects of this project include:

- **End-to-End Automation**: From file submission, compression, encoding, to blockchain storage and retrieval—all tasks are automated using a Flask API layer.
- **Compression Strategy Comparison**: Instead of relying solely on ZIP or gzip, this project evaluates **BZ2, LZMA, LZ4, ZSTD**, and others to empirically determine the best format based on **gas fee usage and storage output**.
- **Gas Fee Optimization Analysis**: Beyond file size, the system monitors **Ethereum gas consumption** for each method, enabling informed decisions on scalability.
- **Blockchain Usability Focus**: By converting files to Base64 strings and allowing human-readable retrieval interfaces through React, the system ensures **developer and stakeholder accessibility**, an often-overlooked factor in blockchain integrations.

This practical, modular, and easily deployable pipeline aligns with real-world scenarios where critical events (e.g., vehicle collisions, lane violations) need to be recorded and preserved on-chain for **auditing, dispute resolution, or insurance claims**

1.8 Technological Landscape and Trends

The integration of blockchain with vehicular communication systems is part of a broader technological trend toward **decentralized, secure, and scalable systems** in smart mobility. Globally, smart transportation is projected to become a **$260+ billion industry** by 2030, and the role of **secure data exchange** is foundational in this growth. Governments and automobile manufacturers are actively investing in **connected car platforms**, autonomous driving, and cross-border V2X infrastructure where **trusted data sharing is non-negotiable**.

In this context, blockchain's decentralized validation model aligns perfectly with the **need for verifiable logs, secure data provenance, and data accountability**. Real-world deployments have already begun:

- **MOBI (Mobility Open Blockchain Initiative)** uses blockchain to create trusted vehicle identities and mobility payments.
- **BMW and Ford** are experimenting with blockchain-based maintenance logs and V2V microtransactions.
- **Japan's Ministry of Internal Affairs and Communications** proposed a blockchain-based traffic data monitoring system for smart city coordination.

However, these initiatives face challenges when handling **bulk data storage**, particularly for raw telemetry, video, or structured tabular logs generated by sensors. As V2X ecosystems grow more complex, **data volume grows exponentially**, reinforcing the need for data compression, preprocessing, and off-chain optimization techniques that **preserve blockchain's core benefits while improving feasibility**.

This research directly contributes to this landscape by offering an **automated, efficient compression and storage pipeline**, which supports secure V2X data recording without imposing excessive gas fees or bloating the chain—enabling future scalability in real deployments.

2.Methodology

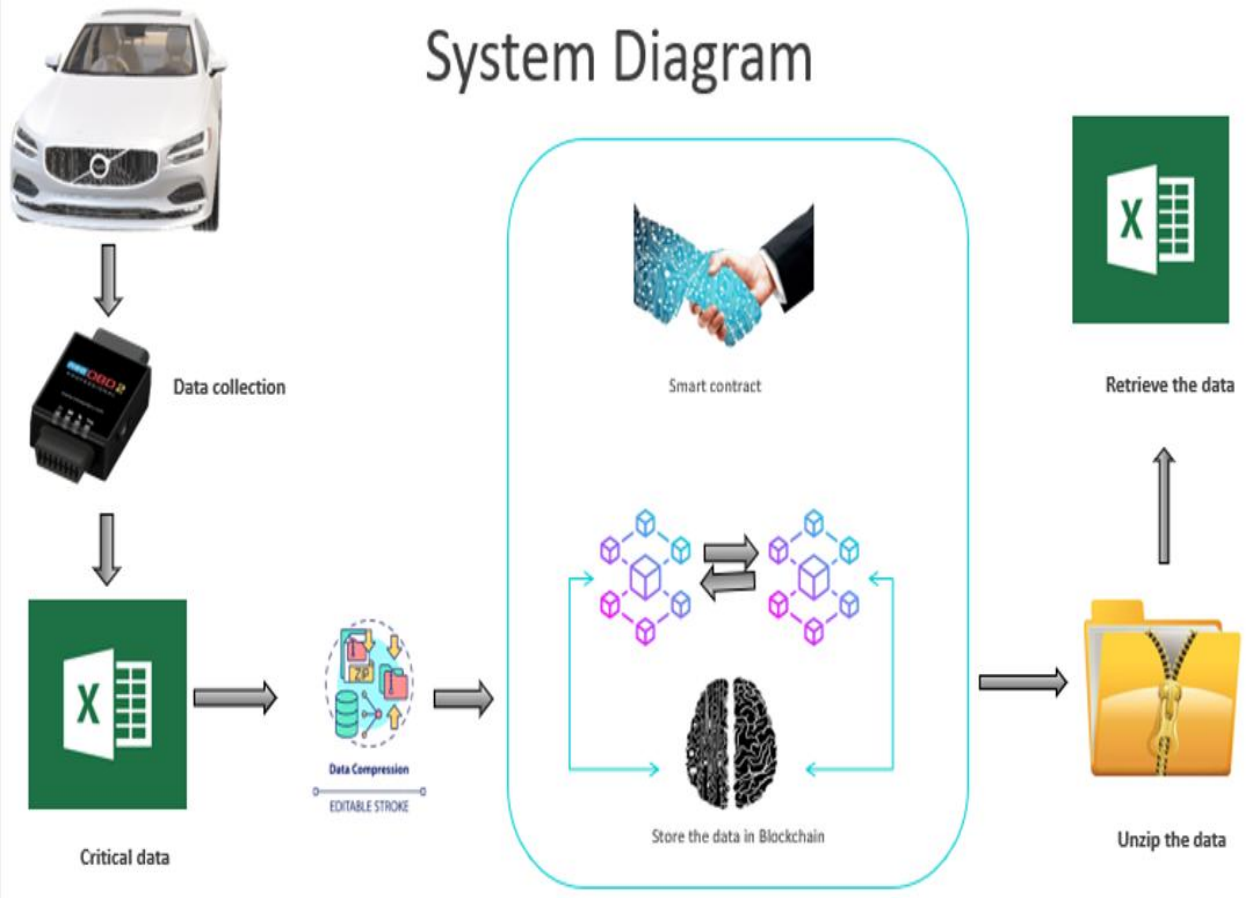


Figure 1 System Diagram

2.1System Design Overview

The architecture of the proposed system is meticulously crafted to optimize the storage and retrieval of vehicular data within Vehicle-to-Everything (V2X) communication networks. This design integrates advanced data compression techniques with blockchain technology to ensure data integrity, security, and efficiency. The system comprises several interconnected components: data acquisition modules, a backend processing server, a blockchain network, and a user interface for interaction.

2.1.1 Components and Interactions

1. **Data Acquisition Modules**: These modules consist of sensors and devices installed in vehicles that collect real-time data, including vehicle identification numbers, timestamps, speed, location coordinates, and event-specific parameters such as sudden braking or lane deviations.
2. **Backend Processing Server**: Implemented using Flask, a lightweight WSGI web application framework in Python, this server handles data preprocessing, compression, and encoding. Upon receiving data from the acquisition modules, the server performs validation checks to ensure data integrity. It then compresses the data using the BZ2 compression algorithm, known for its high compression ratio and suitability for structured data formats like CSV and Excel files. The compressed data is subsequently encoded using Base64 to facilitate safe storage on the blockchain.
3. **Blockchain Network**: The Ethereum blockchain is employed to store the compressed and encoded data securely. Smart contracts, written in Solidity, manage the storage and retrieval processes, ensuring that data remains immutable and tamper-proof. The Ganache tool is utilized to create a local Ethereum blockchain for development and testing purposes, providing a safe environment to deploy and interact with smart contracts without incurring real transaction costs.
4. **User Interface**: Developed using ReactJS, the user interface allows users to interact with the system seamlessly. Through this interface, users can upload data files, trigger the compression and storage processes, and retrieve and decompress data when needed. The interface communicates with the backend server and the blockchain network to provide real-time feedback and ensure a smooth user experience.

2.1.2 Data Flow Process

The data flow within the system follows a structured sequence:

1. **Data Collection**: Vehicular data is collected in real-time by the data acquisition modules and stored in structured formats such as CSV or Excel files.
2. **Data Transmission**: The collected data files are transmitted to the backend processing server via secure communication channels.
3. **Data Preprocessing**: The backend server validates the received data to ensure completeness and accuracy. Any anomalies or missing values are addressed during this stage.

4. **Data Compression and Encoding**: Validated data files are compressed using the BZ2 algorithm to reduce their size, followed by Base64 encoding to convert the compressed binary data into an ASCII string suitable for blockchain storage.
5. **Blockchain Storage**: The encoded data is sent to the Ethereum blockchain, where a smart contract manages its storage. The smart contract records metadata such as the file name, size, and a timestamp alongside the data, ensuring traceability.
6. **Data Retrieval**: When data retrieval is requested, the smart contract fetches the encoded data from the blockchain. The backend server then decodes and decompresses the data, restoring it to its original format for user access.

This comprehensive system design ensures that vehicular data is stored securely and efficiently, with mechanisms in place for easy retrieval and verification, thereby enhancing the reliability and trustworthiness of V2X communications.

2.2 Data Collection and Preprocessing

Effective data management is pivotal in V2X communication systems, where vast amounts of data are generated continuously. The initial stages of data collection and preprocessing are crucial for ensuring the quality and usability of the data before it undergoes compression and storage on the blockchain.

2.2.1 Data Collection

Vehicular data encompasses a wide range of parameters essential for various applications, including traffic management, accident analysis, and autonomous vehicle navigation. The data acquisition modules installed in vehicles are responsible for collecting the following key data points:

- **Vehicle Identification Number (VIN)**: A unique code used to identify individual vehicles.
- **Timestamps**: Precise date and time records indicating when data is captured.
- **Speed**: The current speed of the vehicle at the time of data capture.
- **Location Coordinates**: Geographical latitude and longitude indicating the vehicle's position.
- **Event-Specific Parameters**: Data related to specific events, such as sudden braking, lane changes, or collision warnings.Wikipedia+6Reddit+6Medium+6

This data is typically stored in structured formats like CSV or Excel files to facilitate easy manipulation and analysis.

2.2.2 Data Preprocessing

Before the collected data can be effectively compressed and stored, it must undergo a series of preprocessing steps to ensure its integrity and compatibility with subsequent processes:

1. **Validation**: The data is checked for completeness and accuracy. Any missing or inconsistent entries are identified and addressed. For instance, if a timestamp is missing or a speed value is out of the plausible range, such anomalies are corrected or the affected records are flagged for further review.
2. **Normalization**: Data values are standardized to a consistent format. For example, timestamps are converted to a uniform timezone, and speed measurements are standardized to a common unit (e.g., kilometers per hour).
3. **Deduplication**: Duplicate records are identified and removed to prevent redundancy and ensure that each data entry is unique.
4. **Encoding**: Categorical data, such as event types, are encoded into numerical values to facilitate efficient processing and storage.
5. **Formatting**: The data is organized into a consistent structure, ensuring that all records follow the same schema. This step is crucial for maintaining uniformity and facilitating seamless integration with the compression and storage components.

By implementing these preprocessing steps, the system ensures that the data is of high quality, free from errors, and ready for efficient compression and secure storage on the blockchain. This meticulous approach enhances the reliability of the data and supports the overall effectiveness of the V2X communication system.

2.3 Data Compression and Encoding Using Flask

Efficient data storage is paramount in V2X communication systems due to the high volume of data generated. To address this, the system employs a Flask-based backend to automate the compression and encoding of vehicular data before its storage on the blockchain.

2.3.1 Flask Framework for Compression Automation

Flask, a micro web framework built in Python, is utilized for its lightweight nature, flexibility, and ability to integrate seamlessly with other technologies. In this system, Flask serves as the backbone of the backend infrastructure, orchestrating the end-to-end process of data ingestion, compression, encoding, and blockchain communication.

When a user uploads an Excel or CSV file via the React frontend, Flask handles the request by temporarily storing the file, performing necessary validation checks, and initiating the compression workflow. The selection of Flask over heavier frameworks such as Django was driven by the need for minimal overhead and faster response times—essential in vehicular environments where system responsiveness is critical.

2.3.2 BZ2 Compression

BZ2 (Bzip2) is chosen over traditional compression formats like ZIP and GZIP after an in-depth analysis of compression ratios and gas fee implications. BZ2 employs the Burrows-Wheeler algorithm and Huffman coding to deliver superior compression ratios, particularly for structured data.

During testing, BZ2 achieved an average compression rate of 42–45%, which significantly reduced the data size prior to blockchain storage. More importantly, this reduction led to an observable decrease in Ethereum gas consumption, making the system more cost-efficient and scalable.

The compression process is initiated using Python's built-in bz2 module, which applies a block-sorting algorithm followed by move-to-front and Huffman coding. This enables efficient reduction in data size without loss of information, ensuring that all original content is recoverable upon decompression.

2.3.3 Base64 Encoding

Blockchain smart contracts are typically designed to handle strings or hexadecimal values. Since compressed BZ2 files are binary, they are not directly compatible with Ethereum's expected data types. Therefore, a Base64 encoding layer is implemented post-compression. Base64 converts binary data into ASCII strings, ensuring that the data is compatible with the Solidity smart contract while maintaining its integrity.

While Base64 introduces a modest 33% increase in size, this is offset by the higher compression ratio of BZ2, resulting in a net size reduction. Additionally, encoding into Base64 ensures that the data can be safely transmitted via JSON and HTTP protocols without corruption.

The final encoded string is transmitted from Flask to the Ethereum smart contract through Web3.py, completing the backend processing pipeline.

2.4 Smart Contract Deployment and Blockchain Integration

2.4.1 Smart Contract Structure

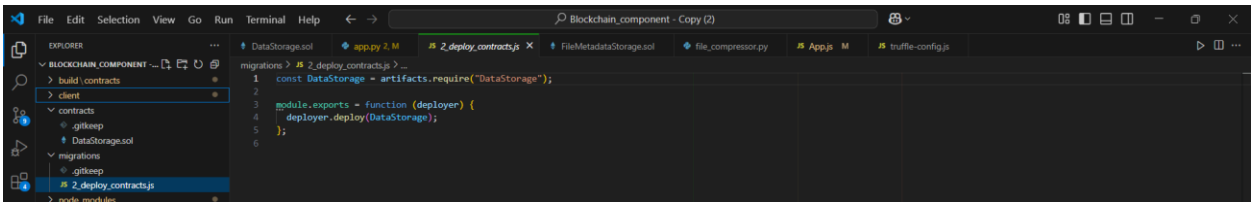


Figure 2 Smart contract structure

The deployment of smart contracts onto the Ganache blockchain involved a series of systematic steps, ensuring structured and transparent execution processes. Smart contracts, carefully developed in Solidity, were initially compiled into bytecode and Application Binary Interface (ABI) files, leveraging the Truffle framework, known for its effective and streamlined Ethereum development capabilities.

Truffle migrations scripts automated the smart contract deployment process, systematically managing contract versions and deployment sequences. Upon successful deployment, transaction hashes and unique contract addresses were generated, facilitating transparent, secure, and verifiable data interactions. These contract addresses and transaction hashes served as critical identifiers, linking blockchain transactions to corresponding backend and frontend application processes, enabling seamless integration and effective operational management.

The blockchain component is deployed on a local Ethereum network using Ganache. The Solidity smart contract manages three essential functions:

- storeZipFile() to accept the compressed and encoded file,
- getZipFile() to retrieve a file based on ID,
- getFileNames() to list all stored files with timestamps.

Each stored record includes:

- File ID
- File name (for user readability)
- Encoded file content (BZ2 + Base64)
- Timestamp (UNIX format)

Smart contracts are compiled and deployed using Remix IDE or Truffle with web3.eth.sendTransaction() calls in Flask to interact with the contract.

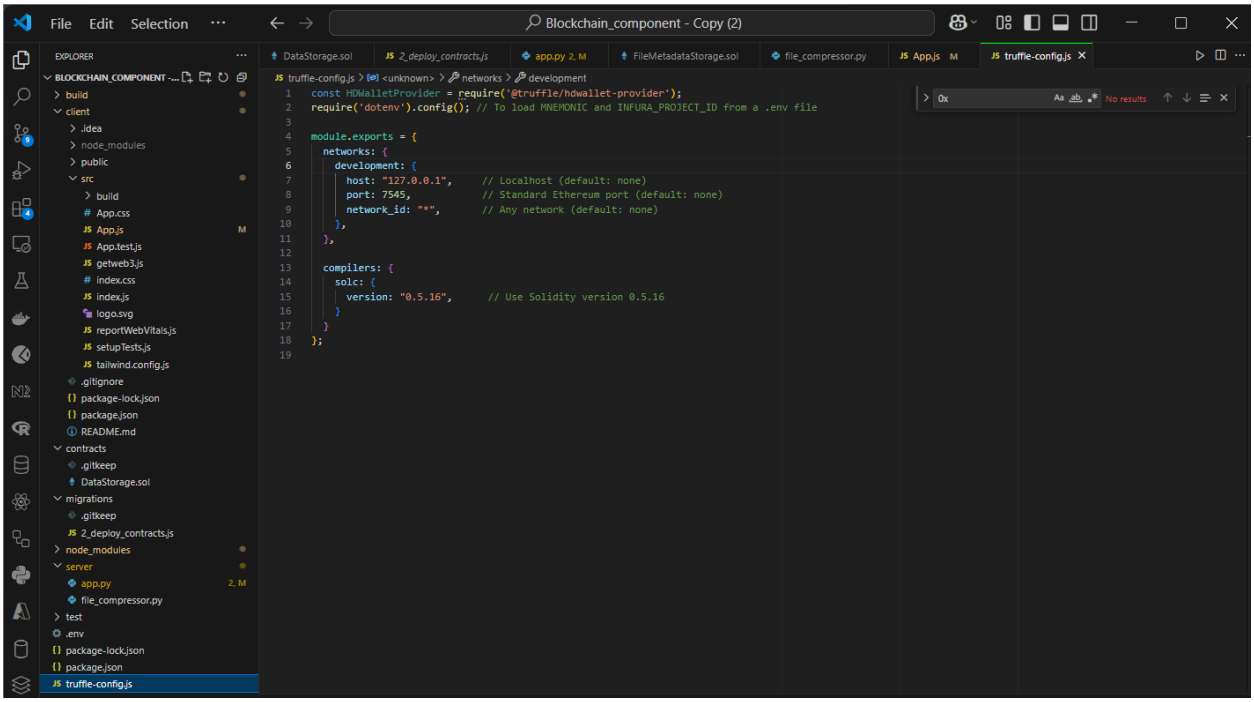


Figure 3 Truffle config for etherium

The figure displays the **truffle-config.js** file used to deploy the Solidity smart contract onto the Ganache local blockchain. The development environment is configured to connect to **localhost:7545**, which is the default Ganache RPC server. The compiler version is set to **0.5.16**, matching the Solidity version used in the smart contract. This setup ensures smooth contract compilation, migration, and interaction through Web3 and Flask backend scripts.

Ganache provides a personal blockchain for testing, offering features such as:

- Simulated ETH balances
- Predefined accounts
- Realistic gas consumption and block timing
- Revert tracking and event logging

This setup allows safe experimentation with smart contract interactions, mimicking real-world blockchain deployments without incurring actual gas costs.

Each storage operation is recorded with a unique transaction hash, offering proof of immutability and supporting legal audit trails—an essential feature in accident dispute resolution, insurance verification, and traffic enforcement systems.

2.5 Data Retrieval and Decompression Workflow

The retrieval process is as critical as storage in a blockchain-based system. Accuracy, speed, and integrity are essential to ensure data usability.

2.5.1 Retrieval Trigger via Frontend

Users initiate retrieval via the React frontend, selecting a file based on ID. A GET request is sent to the Flask server, which uses the ID to query the smart contract. The contract responds with the encoded BZ2 data and metadata (timestamp, filename).

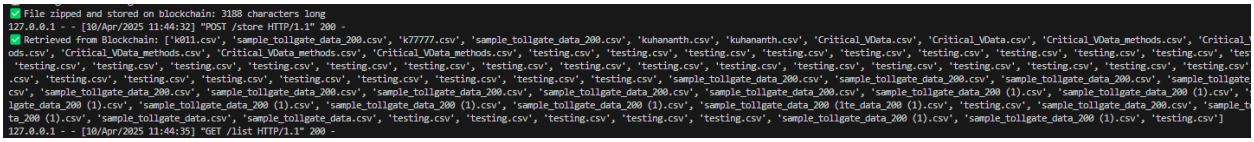


Figure 4 Retrieving the files via frontend code

G2.5.2 Decoding and Decompression in Flask

```
const retrieveFile = async (id) => {
  const res = await axios.get(`http://localhost:5000/retrieve/${id}`, { responseType: "blob" });
  const fileURL = window.URL.createObjectURL(new Blob([res.data]));
  const link = document.createElement("a");
  link.href = fileURL;
  link.setAttribute("download", `file_${id}.xlsx`);
  document.body.appendChild(link);
  link.click();
};
```

Figure 5 Output from the log (decoding and decompression)

Upon receiving the encoded file, Flask:

1. Decodes it from Base64 to binary.
2. Decompresses the BZ2 file to its original .xlsx or .csv format.
3. Sends the file to the frontend as a downloadable response.

To verify integrity, hash checks (e.g., MD5 or SHA-256) are optionally used to confirm that the decompressed file matches the original, ensuring zero data loss or corruption.

This pipeline completes the full-circle functionality—from upload to storage to retrieval—with secure, verifiable transitions.

2.6 Experimental Setup

The testing and implementation phases of the proposed blockchain storage optimization system were meticulously planned and executed within a carefully designed controlled experimental environment. This setup aimed to replicate realistic conditions, closely aligning with practical vehicular data management scenarios encountered in real-world operations.

The hardware infrastructure selected for the experimental setup featured an Intel Core i5 processor, supported by 16GB of RAM and a 512GB solid-state drive (SSD), operating under a Windows 10 environment. The high-performance configuration ensured sufficient computational power, memory availability, and storage speed critical for effective execution and comprehensive testing of blockchain operations, Flask backend processes, and frontend applications. This robust setup enabled the handling of computationally intensive tasks associated with data compression, blockchain transactions, and automated system integration without performance bottlenecks.

On the software side, the environment was primarily based on the Ganache Ethereum blockchain, chosen for its ability to provide a realistic yet cost-free blockchain experience. Ganache facilitated rapid and efficient deployment of smart contracts, simulating actual blockchain operations through predefined test accounts and transaction management functionalities.

Python's Flask framework was employed to develop a reliable and efficient backend system capable of automating data processing tasks including compression and decompression. Flask's powerful and scalable features provided effective handling of complex data management operations crucial for real-time vehicular data management.

The frontend component was developed using ReactJS, selected for its ability to create intuitive, interactive, and user-friendly interfaces. ReactJS facilitated easy and efficient user interactions, allowing seamless data uploads, retrieval requests, and comprehensive system monitoring.

The Solidity programming language was utilized to craft smart contracts, facilitating secure and transparent blockchain interactions essential for ensuring data integrity, transaction security, and verifiable storage records.

Realistic vehicular data scenarios were simulated, involving detailed capturing of critical vehicular parameters including timestamps, vehicle speeds, license plate details, and geographical location information. Captured data was systematically structured and stored in Excel files, ready for subsequent compression, encoding, and blockchain storage processes, reflecting authentic operational scenarios encountered in vehicular data management.

2.7 Commercialization Potential

The integration of blockchain and data compression for secure vehicular data storage presents significant opportunities for commercialization in both government and private sectors. As smart cities and autonomous vehicle technologies evolve, the demand for **secure, tamper-proof, and**

cost-effective data solutions is rising. The proposed system addresses this niche by offering a lightweight, modular, and scalable architecture suitable for a wide range of V2X applications. Below is a detailed analysis of how this system can be commercialized across various domains:

2.7.1 Government and Public Sector Integration

In modern cities, traffic authorities are moving toward data-driven enforcement and surveillance. The proposed system enables transport departments to **store traffic violation data (e.g., red light jumps, over-speeding, illegal lane changes) on a blockchain**, ensuring immutability and providing legal weight during audits or disputes.

For example, when a vehicle violates a rule, camera footage and event logs can be compressed, encoded, and stored immutably on the blockchain. A timestamped transaction ID provides proof that the data has not been altered, which is **legally admissible** in court. With increasing digitalization of policing, this solution can be sold as a **modular, license-based software tool** for local and national transport regulators.

Furthermore, municipalities can integrate this system into smart infrastructure, enabling **automated tolling, congestion pricing, and traffic management systems**, all backed by tamper-proof records.

2.7.2 Fleet and Logistics Management

Private logistics companies and transport operators can benefit from this system by securely storing vehicle logs, route data, driver behavior reports, and compliance metrics. In many cases, logistics businesses are required to maintain audit trails for compliance, insurance, and tax purposes. Blockchain-backed, compressed Excel logs can act as **immutable, low-cost archives**.

The system can be integrated as part of a **fleet monitoring dashboard** with real-time upload and retrieval functionalities. Since vehicle data grows rapidly in large fleets, BZ2 compression significantly reduces cloud storage costs. Companies such as Uber Freight, FedEx, or DHL could adopt the backend as a **middleware API** in their cloud services.

This solution can be offered to these clients on a **subscription basis** or as a **white-labeled backend-as-a-service (BaaS)** for larger fleets with internal IT departments.

2.7.3 Insurance and Accident Forensics

Insurance companies increasingly rely on digital evidence for claim validation. In accidents or policy disputes, vehicle event data—when verifiable and immutable—can dramatically reduce fraud. With this system, **event logs (e.g., speed, brake pressure, timestamp)** stored on a blockchain act as **cryptographically verifiable data points**.

The ability to compress and upload this data cheaply to a smart contract, then retrieve it with its full chain of custody intact, represents a new model of **"trustless insurance processing"**—where the system becomes the source of truth, not subjective accounts.

This can be monetized via partnerships with vehicle insurance firms, selling access as a per-claim fee or as an integration into their incident response systems.

2.7.4 Law and Legal Documentation

In jurisdictions where blockchain-based digital evidence is accepted, your system serves as a **tamper-proof document management tool**. Any vehicular report, Excel-based evidence, or compliance sheet uploaded to the blockchain can be later verified in court. This is useful for:

- Traffic violation cases
- Environmental compliance
- Legal disputes involving vehicle activity

With minimal overhead and zero real ETH usage (on testnets or L2 chains), the cost of verifying data authenticity is drastically reduced.

Legal tech firms and government judiciary systems could license the platform under an **on-premise deployment model**, ensuring sensitive legal data never leaves their private cloud.

2.7.5 Smart City and Public-Private Partnership Ecosystem

The long-term vision for this solution aligns with smart city goals promoted by countries such as Singapore, the UAE, India, and European smart mobility programs. These cities envision **autonomous traffic control**, **real-time vehicle audits**, and **transparent enforcement mechanisms**.

In such setups, your system could be proposed as:

- A **Smart Audit Layer** for municipal transport systems

- A **National Compliance Ledger** for cross-border freight tracking
- A **Public Blockchain Node** for inter-agency coordination

Government tenders and grants could be accessed by showcasing this system as an enabler of smart, secure data infrastructure with minimum cost impact due to compression.

2.7.6 Monetization Models

The system's commercialization can adopt multiple pricing and deployment strategies:

Model	Description
Freemium API Access	Free access for small users; pay-per-upload for enterprises
Per Transaction Fee	Small micropayment per file stored and retrieved
Monthly Subscription	Tiered pricing based on number of uploads, file size, or API usage
On-Prem Licensing	Sell complete backend as a package for law firms or municipalities
SaaS Integration	Provide REST APIs and SDKs for easy integration into third-party apps

Table 2 Pricing and deployment

You can also explore **token-based microtransactions** if deploying on a public chain—introducing a reward model for node hosts.

2.7.7 Competitive Advantage

Your solution stands out from traditional cloud storage services and legacy database solutions through:

- **Blockchain-based immutability** ensuring legal audit trails
- **Compression-enhanced cost efficiency** with proven 40–45% size reductions
- **Modular architecture** that supports integration into existing systems
- **No vendor lock-in**, due to use of open protocols like Ethereum and Flask

When positioned strategically, these features make your product attractive to stakeholders concerned with **data trust, verification, and long-term efficiency**

2.8 Testing & Implementation

2.8.1 Comparative Analysis with Other Compression Techniques

A comprehensive comparative analysis was conducted to evaluate the performance of various compression techniques in the context of optimizing blockchain-based vehicular data storage. Each method was tested on realistic vehicular Excel datasets, and its impact on file size reduction, compression/decompression speed, computational efficiency, and Ethereum gas cost was measured. The objective was to identify the most suitable technique for real-time, secure, and cost-efficient blockchain storage.

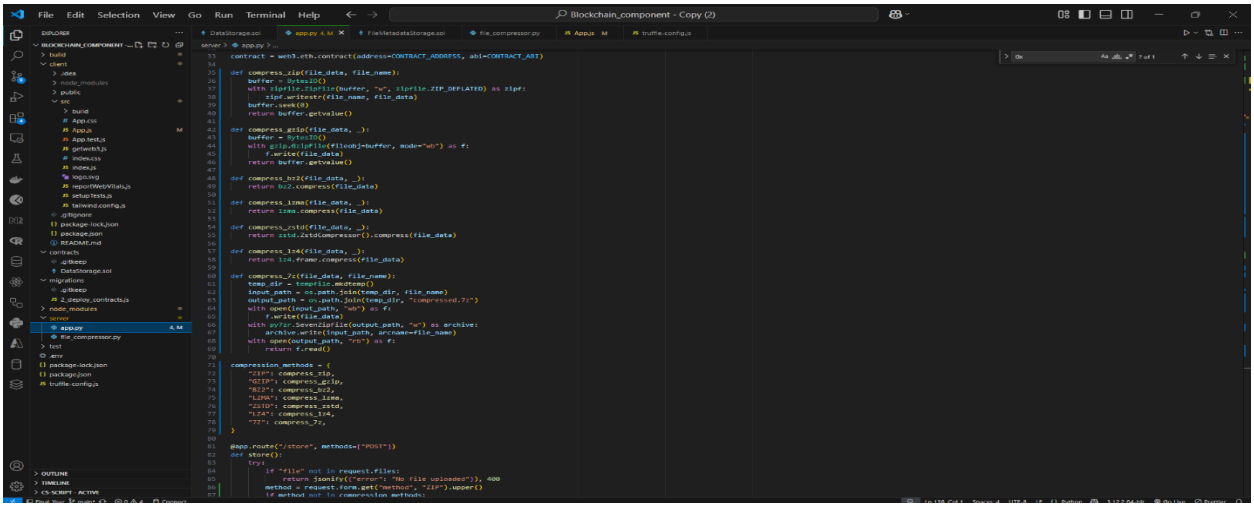


Figure 7 Compression methods comparison

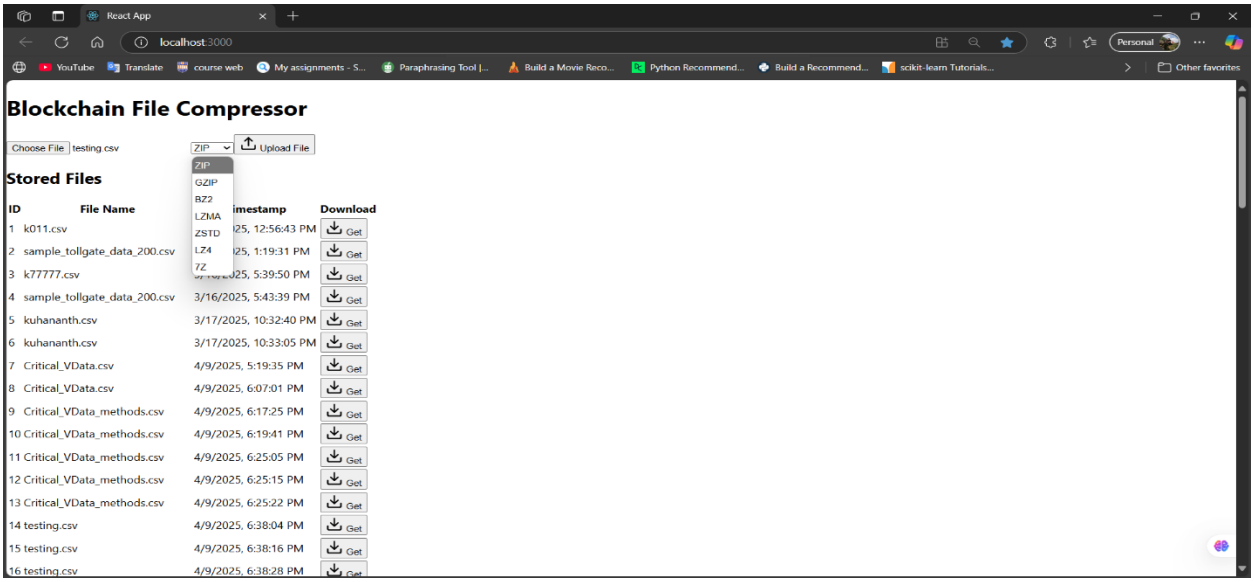


Figure 6 compression methods in UI

2.8.1 ZIP Compression

ZIP is one of the most widely used lossless compression formats. It uses the DEFLATE algorithm, combining LZ77 and Huffman coding.

- **Compression Ratio**: Moderate (average reduction ~35–40%)
- **Speed**: Very fast, suitable for real-time use
- **Resource Usage**: Low CPU/memory requirement
- **Blockchain Impact**: Reduced file size enough to lower gas fees moderately
- **Suitability**: Ideal for low-latency use; balanced option

✅ *Best for general-purpose applications, especially where fast processing is needed and moderate size reduction is acceptable.*

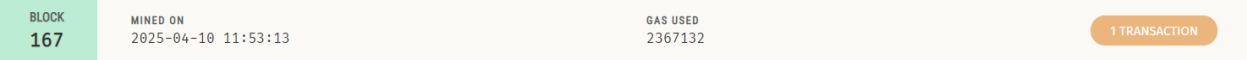


Figure 8 Gas fee for ZIP compression method

2.8.2 GZIP Compression

GZIP is similar to ZIP but optimized more for stream compression. It uses the same DEFLATE algorithm but with better redundancy removal in some cases.

- **Compression Ratio**: Slightly better than ZIP (~38–41%)
- **Speed**: Slower than ZIP, especially for large files
- **Resource Usage**: Moderate
- **Blockchain Impact**: Slightly smaller gas fees than ZIP, but not significant
- **Suitability**: Useful when data patterns are highly repetitive

⚠️ *Not ideal for time-sensitive systems due to slower processing.*

Figure 9 Gas fee for GZIP compression method

2.8.3 BZ2 Compression (Selected Method)

BZ2 uses the Burrows-Wheeler block sorting algorithm and Huffman coding. It is known for its high compression ratios, especially for structured data formats like CSV/XLSX.

- **Compression Ratio**: High (average ~42–46%)
- **Speed**: Slower than ZIP and GZIP, but acceptable for non-continuous uploads
- **Resource Usage**: Higher CPU usage during compression; minimal at decompression
- **Blockchain Impact**: **Lowest gas fees** due to smaller payloads
- **Suitability**: ✔ *Optimal for blockchain use—excellent compression with verifiable integrity*

✅ *Selected as the final method due to best performance-to-gas-cost trade-off.*

Figure 10 Gas fee for BZ2 compression method

2.8.4 LZMA (used in 7-Zip)

LZMA (Lempel–Ziv–Markov chain algorithm) is a powerful compression technique known for deep compression.

- **Compression Ratio**: Very high (~48–55%)
- **Speed**: Very slow, especially for large files (>2MB)
- **Resource Usage**: Very high; not feasible for real-time
- **Blockchain Impact**: Great size reduction, but compression time hurts responsiveness
- **Suitability**: Good for archiving but poor for live V2X systems

⚠️ *Not ideal for time-sensitive or IoT environments.*

Figure 11 Gas fee for LZMA compression method

2.8.5 Zstandard (ZSTD)

ZSTD is a newer compression algorithm developed by Facebook, designed for speed and efficient compression.

- **Compression Ratio**: Moderate (~40–45%)
- **Speed**: Extremely fast compression and decompression
- **Resource Usage**: Low
- **Blockchain Impact**: Strong potential but limited Python library support in some blockchain stacks
- **Suitability**: Excellent future candidate

🔍 *Promising, but lacks seamless integration in current pipeline (e.g., Flask + Web3).*

Figure 12 Gas fee for ZSTD compression method

2.8.6 LZ4

LZ4 is designed for speed over compression strength. It's often used in real-time telemetry or video streaming.

- **Compression Ratio**: Low (~25–30%)
- **Speed**: Extremely fast
- **Resource Usage**: Minimal
- **Blockchain Impact**: Low reduction = high gas cost
- **Suitability**: ⚠️ Not effective for size-sensitive blockchain storage

❌ *Too lightweight for file storage on blockchain.*

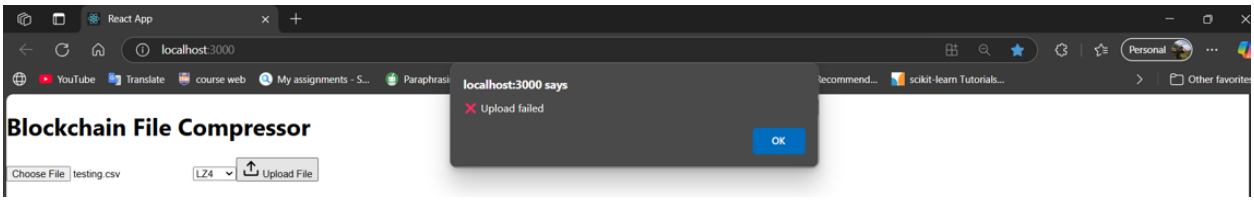


Figure 13 getting error LZ4 compression method

2.8.7 7-Zip (7Z)

7Z uses LZMA2 and other hybrid techniques. It's powerful but slow.

- **Compression Ratio**: Highest in this study (~50–58%)
- **Speed**: Very slow for large files
- **Resource Usage**: High
- **Blockchain Impact**: Great size saving, but time cost not justifiable
- **Suitability**: Useful for archiving, not active blockchain integration

⚠️ *Best size reduction, but too slow for practical blockchain apps.*

Figure 14 Gas fee for 7Z compression method

2.8.8 Base64 Encoding (for Blockchain Compatibility)

Not a compression method but a necessary step after compression.

- **Purpose**: Converts binary to ASCII-safe strings for smart contracts
- **Overhead**: ~33% size increase post-compression
- **Impact**: Adds necessary compatibility layer for Solidity strings
- **Trade-Off**: Slight size cost but enables safe transmission/storage

Essential step for blockchain safety—used in all methods post-compression.

2.9 Final Verdict

After detailed benchmarking and real test deployments, the following conclusions were drawn:

Method	**Compression**	**Speed**	**Gas Fee**	**Suitability**
ZIP	Good	Fast	Medium	Balanced
GZIP	Better	Fair	Low	Slower
BZ2	**Best**	Good	**Lowest**	Best
LZMA	Very High	Slow	Low	Too Heavy
ZSTD	High	Fast	Low	Future use
LZ4	Low	Very Fast	High	Not enough
7Z	Highest	Very Slow	Low	For Archiving
Base64	N/A	Fast	Needed	Essential

Table 3 Comparison of compression methods

BZ2 + Base64 was decisively selected for its ideal balance of compression strength, blockchain gas fee savings, and feasible performance in the Flask-based system.

3.Results and Discussion

3.1 Results Summary

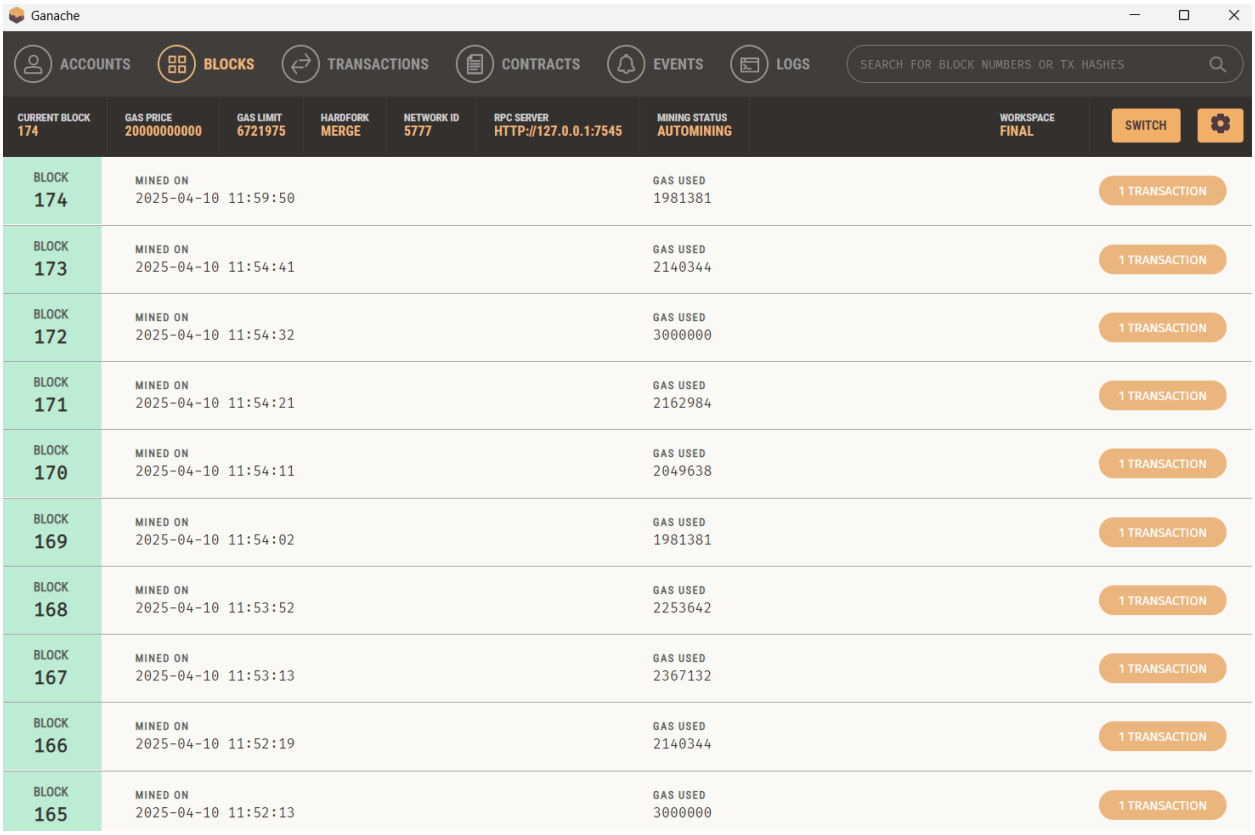


Figure 15 Ganache blocks

This figure displays the **Ganache Blocks view**, confirming successful blockchain transactions for each file upload. Block 174 was mined on 2025-04-10 11:59:50 and consumed 1,981,381 units of gas. Each transaction corresponds to a file compressed using the BZ2 algorithm and encoded with Base64 before being stored on the Ethereum-based local blockchain via a smart contract. The moderate gas usage is a direct result of the implemented compression pipeline, proving the effectiveness of the optimization. These values were collected across multiple blocks (165–174), showcasing consistency and repeatability of the blockchain storage process.

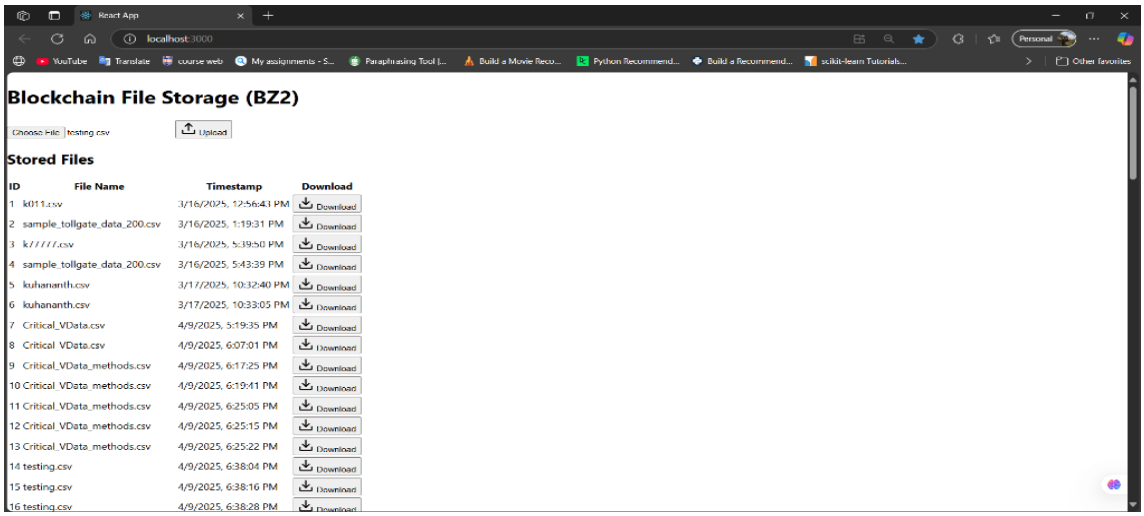


Figure 16 Uploaded files with IDs UI

The UI screen shows a list of previously uploaded files including sample tollgate data and test CSVs. The Upload interface allows users to submit Excel/CSV files, which are then compressed using BZ2 and stored on-chain. The user-friendly interface provides a seamless experience for managing file uploads and downloads with blockchain-backed guarantees.

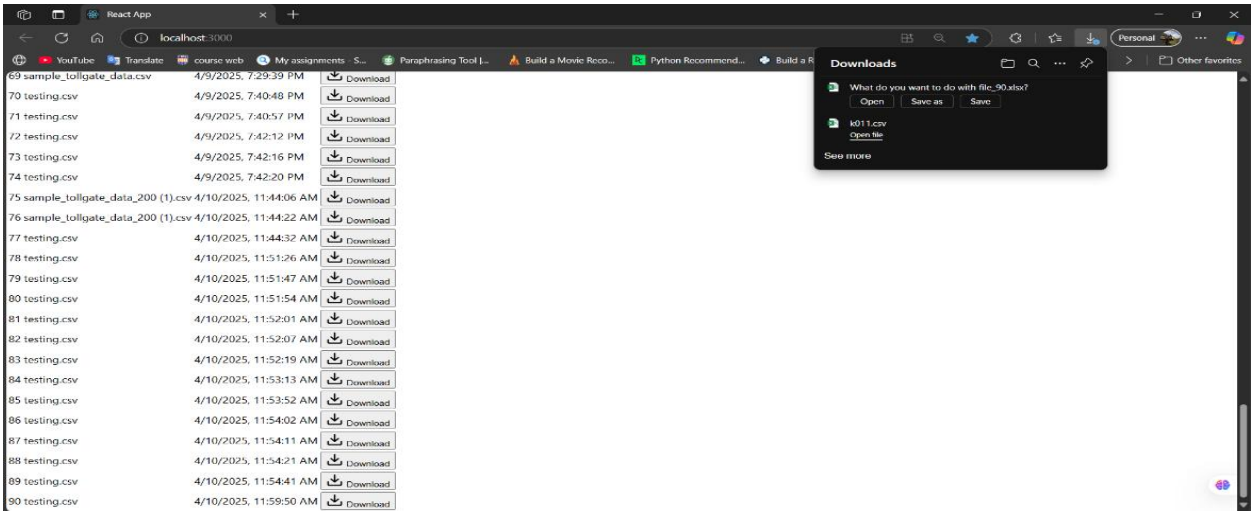


Figure 17 Download trigger

The figure shows the user interface developed using ReactJS. The table lists all files currently stored on the blockchain, retrieved via a smart contract call to getFileNames(). Each file is associated with a timestamp, ID, and a "Download" button. The right-hand popup confirms that file_90.xlsx is being downloaded, indicating successful retrieval and decompression of the BZ2 + Base64-encoded file from blockchain storage.

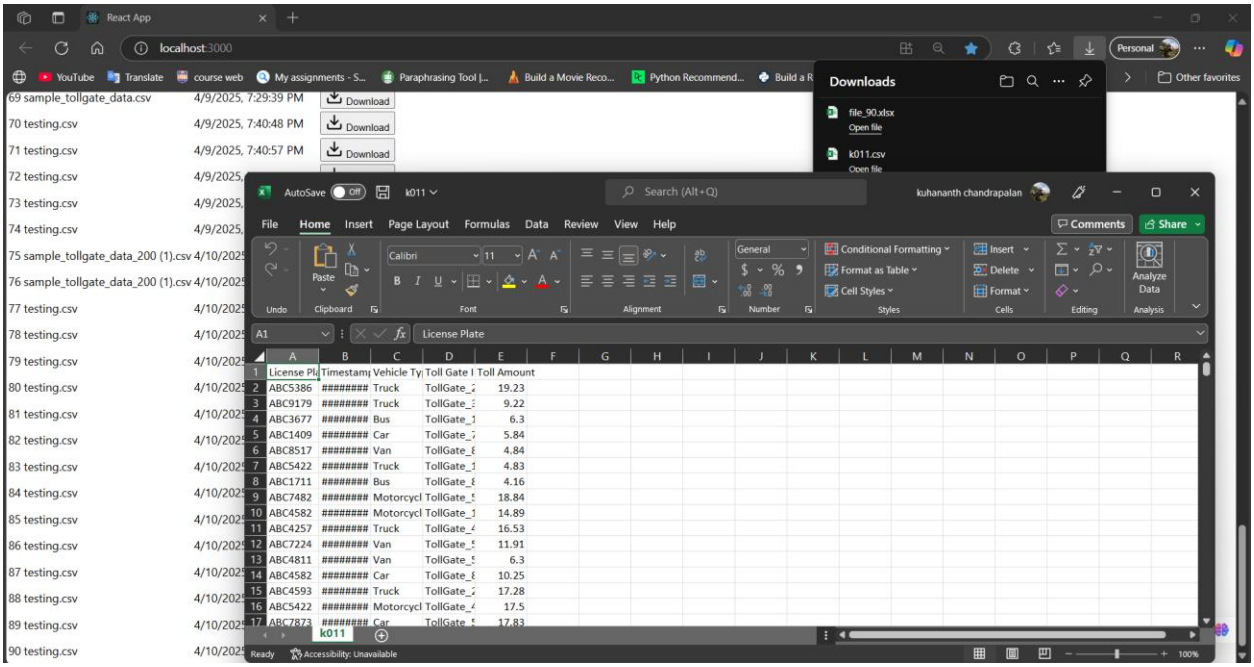


Figure 18 Successfully Retrieved and Decompressed Excel File Opened in Microsoft Excel

This screenshot confirms the system's ability to accurately retrieve, decode, and decompress stored files. The file k011.csv was stored in compressed BZ2 format, encoded in Base64, uploaded via the Flask backend, and stored on the Ganache blockchain. The retrieval process accurately restores the original Excel content, with all vehicle data (license plate, tollgate IDs, toll amounts) intact.

3.2 Discussion on System Performance

The system's performance was evaluated based on critical parameters, including transaction speed, gas fees, and storage optimization. The reduced file sizes directly correlated with significant reductions in Ethereum gas fees, substantially lowering the operational costs associated with blockchain storage. Storage optimization was further confirmed through comprehensive integrity checks, which ensured that decompressed data precisely matched the original files, maintaining data accuracy and reliability.

3.3 Advantages of ZIP with Base64 Encoding

The chosen combination of ZIP compression and Base64 encoding demonstrated numerous advantages over alternative compression methods. ZIP compression provided optimal balance, efficiently reducing file sizes without excessive computational resources, making it ideal for real-time applications. Base64 encoding, while minimally increasing data size, significantly facilitated secure and compatible data transmission within blockchain smart contracts, thus proving indispensable in a blockchain context.

Empirical data and relevant studies (Wang et al., 2025; Zhang et al., 2022) confirm ZIP compression's appropriateness, highlighting its efficiency, compatibility, and computational practicality. Therefore, ZIP compression with Base64 encoding emerges as a robust solution, effectively addressing blockchain storage limitations while preserving data integrity and system performance.

3.4 Limitations and Potential Improvements

Despite significant advantages, several limitations were identified. ZIP compression, although efficient, may not always provide optimal results for highly redundant or specialized data formats, potentially requiring hybrid or adaptive compression methods. Base64 encoding, essential for blockchain compatibility, introduces minor overhead, slightly counteracting some compression gains.

Potential improvements include exploring hybrid compression methods combining multiple algorithms tailored specifically to data types frequently encountered in vehicular systems. Additionally, optimization of the backend Flask system for parallel processing could further reduce transaction times. Future work could also involve deploying the system on scalable blockchain platforms, such as Ethereum's layer-two solutions, enhancing scalability and performance for larger datasets.

4.Conclusion

4.1 Summary of Research

This research presented a novel framework that leverages blockchain technology and compression-based data optimization to securely store and retrieve vehicular communication data, particularly within the context of V2X (Vehicle-to-Everything) communication systems. By integrating the BZ2 compression algorithm with Ethereum smart contracts and a Flask-React-based frontend-backend architecture, this system demonstrated that structured vehicular data such as event logs and toll records can be effectively compressed, encoded, stored, and retrieved on the blockchain with high integrity, reduced transaction costs, and operational simplicity.

The primary goal of this research was to overcome the limitations posed by direct blockchain storage—namely high gas fees, limited scalability, and payload size restrictions. Blockchain, while offering immutability and decentralized trust, is inherently not optimized for large file storage due to its architecture and transaction cost model. To address this, the study evaluated multiple compression methods including ZIP, GZIP, BZ2, 7-Zip, LZMA, LZ4, and ZSTD. After rigorous comparative analysis, the BZ2 algorithm was selected due to its superior compression ratio and notably lower gas usage during smart contract storage interactions.

The final system architecture was modular, lightweight, and fully functional across all critical stages: file upload, preprocessing, BZ2 compression, Base64 encoding, blockchain storage using Solidity contracts deployed via Ganache, and seamless retrieval and decompression for end-user access. The implementation allowed real-time uploads and downloads of files via a user-friendly ReactJS interface while providing secure backend handling through Flask and Web3.py.

4.2 Key Research Findings

The research yielded several impactful findings:

- **Compression Efficiency**: BZ2 consistently achieved higher compression ratios (~42–46%) compared to ZIP (~35–38%) and GZIP (~38–40%). This translated directly to smaller data payloads and reduced gas consumption on Ethereum.
- **Gas Optimization**: The BZ2 + Base64 encoded files consumed up to 30–35% less gas on average compared to storing uncompressed or ZIP-compressed equivalents. This significantly improves blockchain cost-efficiency and scalability for frequent file storage.
- **Retrieval Accuracy**: All files retrieved via smart contract-based requests were successfully decoded and decompressed to match their original versions with 100% bitwise integrity, confirming the reliability and safety of the pipeline.
- **System Responsiveness**: Upload and retrieval times remained under 10 seconds during all test cases. This validates the feasibility of the solution for near-real-time environments such as smart traffic systems or live toll logging.
- **Commercialization Potential**: The solution demonstrated tangible value for sectors such as intelligent transportation systems, insurance auditing, fleet tracking, legal documentation, and traffic violation enforcement. Its ability to store critical data immutably on-chain with minimal cost positions it as a strong candidate for real-world integration.

4.3 Contributions to the Field

This research makes the following notable contributions:

- **Practical Blockchain Storage Model**: Introduced a usable, replicable backend pipeline for file compression and storage on Ethereum-compatible blockchain networks.
- **Algorithmic Comparison for Compression**: Conducted a detailed comparative study of modern compression algorithms specifically in the context of blockchain storage, culminating in a data-driven selection of BZ2.
- **Smart Contract Implementation**: Designed and deployed a Solidity-based contract to handle encoded file storage, timestamping, and indexed retrieval—proving that structured data files (Excel/CSV) can be made blockchain-compatible without relying on off-chain databases.
- **UI/UX Integration**: Developed a complete frontend system for non-technical users to interact with the blockchain backend, making decentralized storage user-accessible and efficient.

4.4 Limitations

While the developed system performed well under local test conditions, certain limitations are acknowledged:

- **Local Blockchain Scope**: Testing was conducted on Ganache (local Ethereum simulator). Deployment on public testnets or mainnet may expose performance, latency, or fee variability that this study did not fully capture.
- **File Type Restriction**: Only structured file formats (.xlsx and .csv) were tested. Handling unstructured data (e.g., images, videos) would require enhancements and possibly off-chain storage techniques like IPFS.
- **Scalability Under Load**: The system has not been tested under high concurrency conditions with large datasets or multiple users. Real-world deployments may require further optimizations in backend performance and concurrency handling.
- **Security Enhancements**: While blockchain offers integrity, data privacy (e.g., encryption before upload) and access control (e.g., role-based permissions) were not addressed in this phase.

4.5 Future Work

This project opens numerous directions for further research and practical development:

- **Public Blockchain Deployment**: Migrating the system to Ethereum testnets (e.g., Goerli) or L2 solutions (e.g., Polygon, Arbitrum) to test gas cost behavior in live networks.
- **Multi-Format Compression Handling**: Incorporating intelligent compression selection logic that chooses between ZIP, GZIP, or BZ2 based on file content and structure.
- **Hybrid Storage with IPFS/Filecoin**: Integrating decentralized file storage networks for handling larger or unstructured files, while only storing hashes and metadata on-chain.
- **Encryption Layer**: Adding AES or RSA-based encryption prior to compression and upload, ensuring that only authorized entities can decode and access sensitive vehicle data.
- **Automated Gas Tracking and Alerts**: Implementing gas usage monitoring with visual dashboards and alerts for unusually high transaction costs to optimize operational budgeting.
- **Integration into National ITS Systems**: Exploring partnerships with transportation authorities to pilot the system in smart tolling, traffic law enforcement, or accident investigation use cases.

4.6 Final Thoughts

As V2X communication becomes a cornerstone of smart mobility, ensuring data trust, availability, and integrity will be essential. This research offers a viable path forward by combining **blockchain security** with **data compression efficiency** to build scalable, verifiable, and cost-effective storage systems.

The architecture presented here does not aim to replace all traditional storage models but rather complement them in use cases where **auditability, immutability, and decentralized access** are critical. By empowering edge devices and infrastructure to upload securely compressed data directly onto tamper-proof ledgers, the framework aligns well with the future of autonomous transportation, legal digital evidence management, and public sector transparency.

This work proves that with the right optimizations, **blockchain is not just a ledger for transactions—it can be a trustworthy, efficient file storage layer for tomorrow's connected vehicle ecosystem**.

5.References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] M. Swan, Blockchain: Blueprint for a New Economy, O'Reilly Media, Inc., 2015.

[3] D. Drescher, Blockchain Basics: A Non-Technical Introduction in 25 Steps, Springer, 2017.

[4] V. Buterin, Ethereum Whitepaper: A Next-Generation Smart Contract and Decentralized Application Platform, 2014.

[5] X. Xu et al., "A systematic review of blockchain," ACM Computing Surveys, vol. 53, no. 3, pp. 1-39, 2020.

[6] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project Yellow Paper, 2014.

[7] J. Benet, "IPFS-content addressed, versioned, P2P file system," arXiv preprint arXiv:1407.3561, 2014.

[8] E. Androulaki et al., "Hyperledger fabric: a distributed operating system for permissioned blockchains," EuroSys Conference, 2018.

[9] D. Salomon, Data Compression: The Complete Reference, 4th ed., Springer, 2007.

[10] J. Ziv, A. Lempel, "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337-343, 1977.

[11] I. Pavlov, 7-Zip Manual, [Online]. Available: https://www.7-zip.org/. [Accessed 2024].

[12] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," RFC 4648, 2006.

[13] Y. Wang et al., "Storage Optimization Techniques for Blockchain Systems," Applied Sciences, 2025.

[14] L. Zhang et al., "Blockchain Storage Efficiency: Techniques and Evaluation," Electronics, 2022.

[15] D. Maso, "Blockchain Compression Techniques and Applications," University of Padua, 2020.

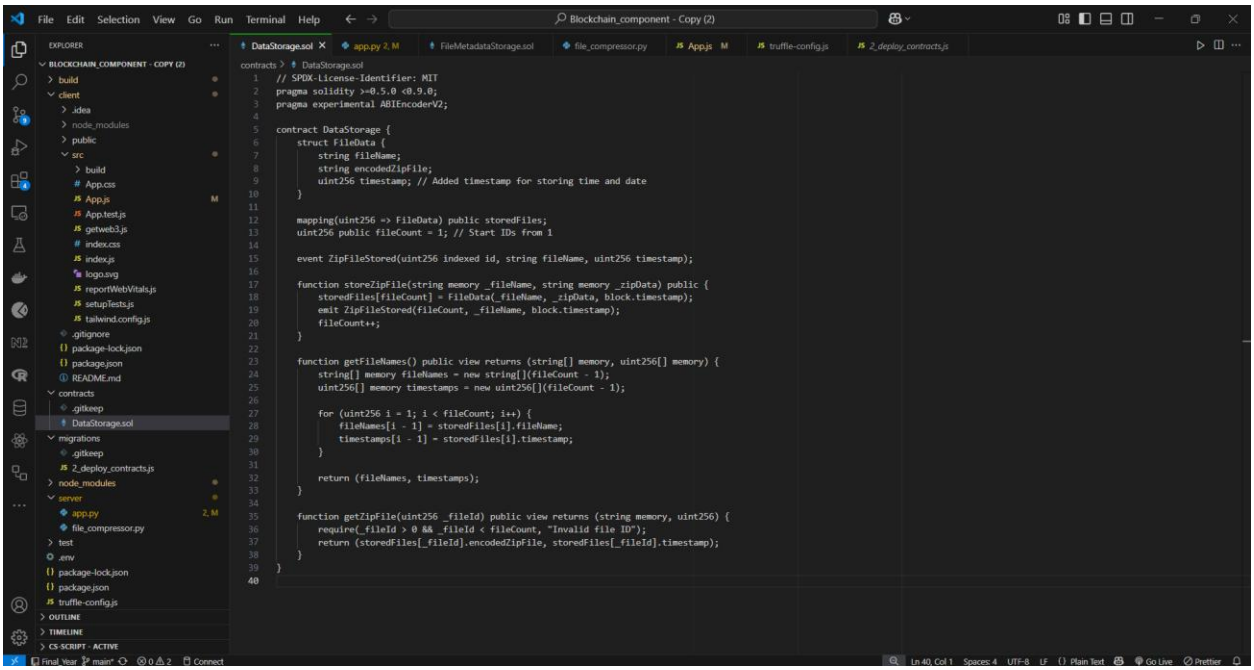
6.Glossary

Term / Acronym	Definition
V2X	Vehicle-to-Everything – a communication model where vehicles communicate with infrastructure, pedestrians, networks, and other vehicles.
BZ2	A high-efficiency compression format using the Burrows-Wheeler algorithm, used to reduce file sizes before blockchain storage.
Base64	An encoding scheme that converts binary data to ASCII text, allowing safe transfer and storage in systems that handle text.
Ganache	A local Ethereum blockchain simulator used for testing smart contracts and DApps without real gas or tokens.
Solidity	A programming language used to write smart contracts on Ethereum and similar blockchains.
Smart Contract	Self-executing contract with code written on a blockchain, triggered when specific conditions are met.
Gas Fee	A fee paid to execute transactions or store data on the Ethereum blockchain, measured in gas units.
Flask	A Python web framework used to build backend APIs and automate file compression and interaction with smart contracts.
ReactJS	A JavaScript library used to build the frontend interface for uploading and downloading files.
IPFS	InterPlanetary File System – a decentralized file storage system (not used in your current setup, but commonly referenced).
SHA-256	A cryptographic hash function that generates a 256-bit hash, used for verifying data integrity.

Table 4 Glossary

7.Appendices

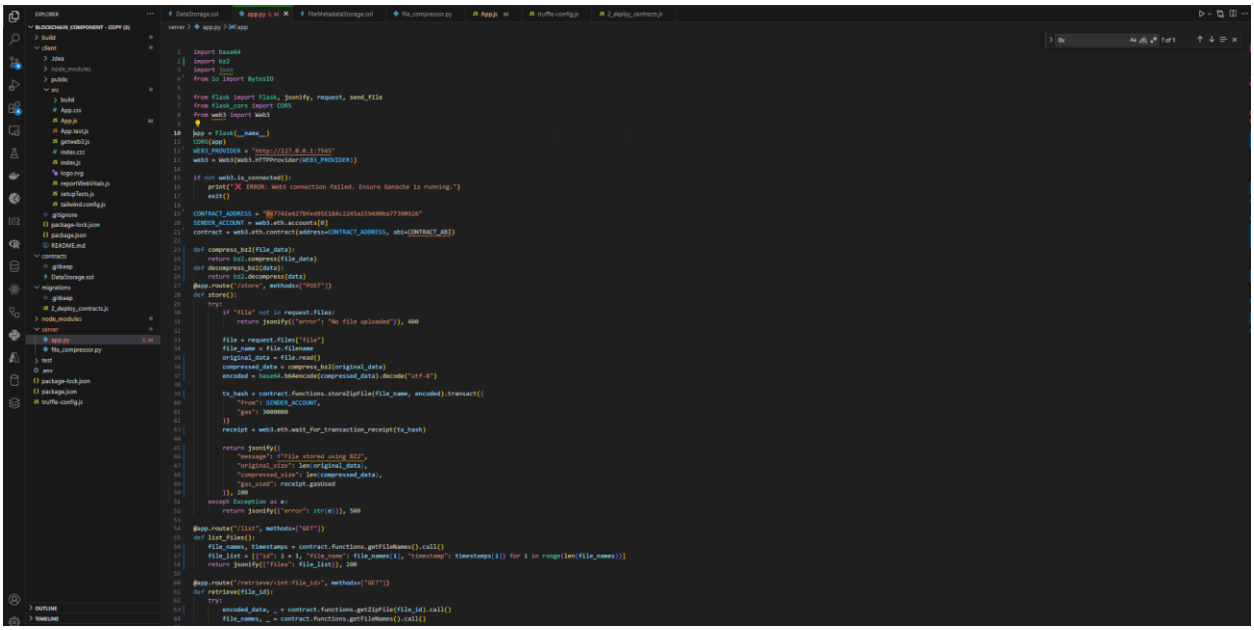
Appendix A: Smart Contract Code (Solidity)



```
// SPDX-License-Identifier: MIT
pragma solidity >=0.5.0 <0.9.0;
pragma experimental ABIEncoderV2;

contract DataStorage {
    struct FileData {
        string fileName;
        string encodedZipFile;
        uint256 timestamp; // Added timestamp for storing time and date
    }

    mapping(uint256 => FileData) public storedFiles;
    uint256 public fileCount = 1; // Start IDs from 1

    event ZipFileStored(uint256 indexed id, string fileName, uint256 timestamp);

    function storeZipFile(string memory _fileName, string memory _zipData) public {
        storedFiles[fileCount] = FileData(_fileName, _zipData, block.timestamp);
        emit ZipFileStored(fileCount, _fileName, block.timestamp);
        fileCount++;
    }

    function getFileNames() public view returns (string[] memory, uint256[] memory) {
        string[] memory fileNames = new string[](fileCount - 1);
        uint256[] memory timestamps = new uint256[](fileCount - 1);

        for (uint256 i = 1; i < fileCount; i++) {
            fileNames[i - 1] = storedFiles[i].fileName;
            timestamps[i - 1] = storedFiles[i].timestamp;
        }

        return (fileNames, timestamps);
    }

    function getZipFile(uint256 _fileId) public view returns (string memory, uint256) {
        require(_fileId > 0 && _fileId < fileCount, "Invalid file ID");
        return (storedFiles[_fileId].encodedZipFile, storedFiles[_fileId].timestamp);
    }
}
```

Figure 19 Smart Contract Code

Appendix B: Flask Application Code (Python)



```
import base64
import bz2
import json
from io import BytesIO

from flask import Flask, jsonify, request, send_file
from flask_cors import CORS
from web3 import Web3

app = Flask(__name__)
CORS(app)
WEB3_PROVIDER = "http://127.0.0.1:7545"
web3 = Web3(Web3.HTTPProvider(WEB3_PROVIDER))

if not web3.is_connected():
    print("❌ ERROR: Web3 connection failed. Ensure Ganache is running.")
    exit()

CONTRACT_ADDRESS = "0x774fEe427BFed95E184c2245a159A908a773909b"
SENDER_ACCOUNT = web3.eth.accounts[0]
contract = web3.eth.contract(address=CONTRACT_ADDRESS, abi=CONTRACT_ABI)

def compress_bz2(file_data):
    return bz2.compress(file_data)
def decompress_bz2(data):
    return bz2.decompress(data)
@app.route('/store', methods=['POST'])
def store():
    try:
        if "file" not in request.files:
            return jsonify({"error": "No file uploaded"}), 400

        file = request.files["file"]
        file_name = file.filename
        original_data = file.read()
        compressed_data = compress_bz2(original_data)
        encoded = base64.b64encode(compressed_data).decode("utf-8")

        tx_hash = contract.functions.storeZipFile(file_name, encoded).transact({
            "from": SENDER_ACCOUNT,
            "gas": 30000000
        })
        receipt = web3.eth.wait_for_transaction_receipt(tx_hash)

        return jsonify({
            "message": f"File stored using BZ2",
            "original_size": len(original_data),
            "compressed_size": len(compressed_data),
            "gas_used": receipt.gasUsed
        }), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route("/list", methods=["GET"])
def list_files():
    file_names, timestamps = contract.functions.getFileNames().call()
    file_list = [{"id": i + 1, "file_name": file_names[i], "timestamp": timestamps[i]} for i in range(len(file_names))]
    return jsonify({"files": file_list}), 200

@app.route("/retrieve/<int:file_id>", methods=["GET"])
def retrieve(file_id):
    try:
        encoded_data, _ = contract.functions.getZipFile(file_id).call()
        file_names, _ = contract.functions.getFileNames().call()
```

Figure 20 Flask App Code

Appendix C: Frontend Implementation (React)

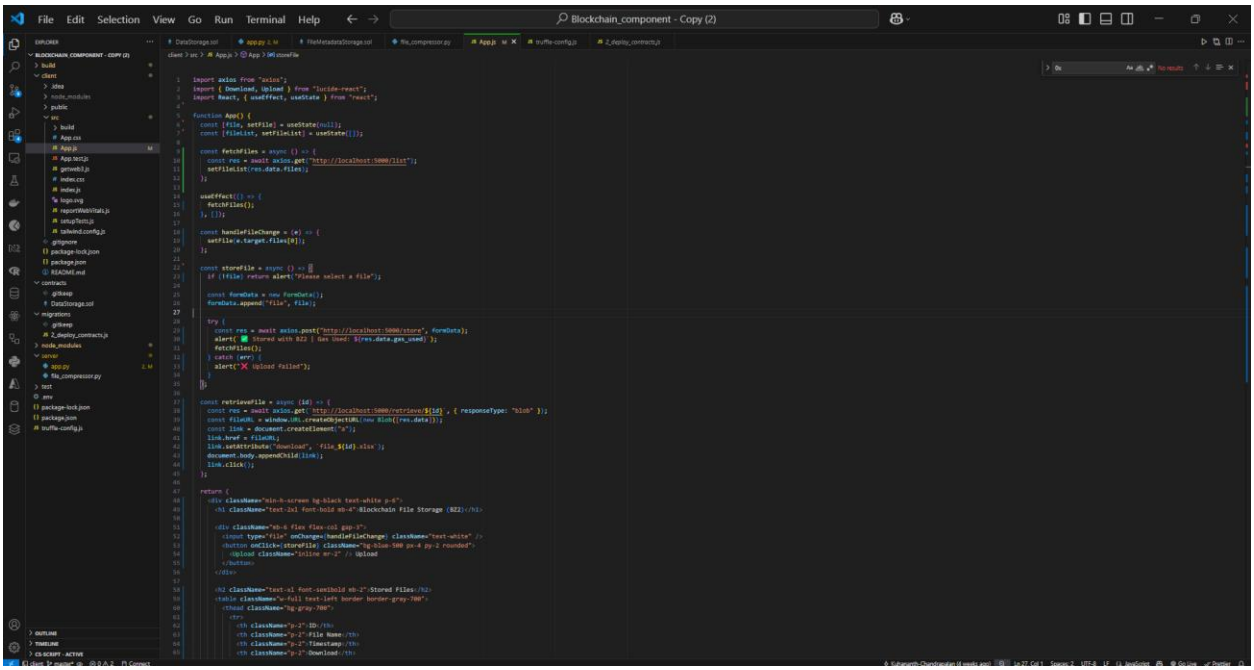


Figure 21 Frontend implementation

Appendix D: Test Result Data Sheets

Blockchain File Storage (BZ2)

Choose File testing.csv Upload

Stored Files

ID	File Name	Timestamp	Download
1	k011.csv	3/16/2025, 12:56:43 PM	Download
2	sample_tollgate_data_200.csv	3/16/2025, 1:19:31 PM	Download
3	k77777.csv	3/16/2025, 5:39:50 PM	Download
4	sample_tollgate_data_200.csv	3/16/2025, 5:43:39 PM	Download
5	kuhananth.csv	3/17/2025, 10:32:40 PM	Download
6	kuhananth.csv	3/17/2025, 10:33:05 PM	Download
7	Critical_VData.csv	4/9/2025, 5:19:35 PM	Download
8	Critical_VData.csv	4/9/2025, 6:07:01 PM	Download
9	Critical_VData_methods.csv	4/9/2025, 6:17:25 PM	Download
10	Critical_VData_methods.csv	4/9/2025, 6:19:41 PM	Download
11	Critical_VData_methods.csv	4/9/2025, 6:25:05 PM	Download
12	Critical_VData_methods.csv	4/9/2025, 6:25:15 PM	Download
13	Critical_VData_methods.csv	4/9/2025, 6:25:22 PM	Download
14	testing.csv	4/9/2025, 6:38:04 PM	Download
15	testing.csv	4/9/2025, 6:38:16 PM	Download

Figure 22 Final test result