ORIGINAL ARTICLE
Electrica 2026; 26: 1-9

A Comparative Analysis of Monolithic and Modular Smart Contract Architectures: A Case Study of Vehicle Trading Systems

Tunahan Timuçin, Serdar Biroğul

Department of Computer Engineering, Düzce University Faculty of Engineering, Düzce, Türkiye

Cite this article as: T. Timuçin and S. Biroğul, "A comparative analysis of monolithic and modular smart contract architectures: A case study of vehicle trading systems," *Electrica*, 2026, 26, 0333, doi:10.5152/electrica.2026.25333.

WHAT IS ALREADY KNOWN ON THIS TOPIC?

- *In choosing between smart contract architectures (monolithic vs modular), it's generally known that the modular approach offers theoretical advantages such as separation of concerns, reusability, and independent upgradeability, while the monolithic approach offers practical advantages such as simplicity and potentially lower gas/deployment costs. However, it's emphasized that there are few studies that use measurable metrics in real-world scenarios.*
- *The gas optimization literature shows that call types and storage access patterns significantly impact costs, with external calls being much more expensive than internal calls. Yet, most studies focus on "low-level code optimization" (rather than*

Corresponding author:
Tunahan Timuçin

E-mail:
tunahantimucin@duzce.edu.tr

Received: October 15, 2025
Revision Requested: October 30, 2025
Last Revision Received: November 1, 2025
Accepted: December 7, 2025
Publication Date: February 20, 2026

DOI: 10.5152/electrica.2025.25333

ABSTRACT

The blockchain technology has attracted more and more interest recently as a reliable and secure platform for a variety of applications. This study presents a comprehensive comparative analysis of monolithic and modular architectural patterns in smart contracts, which have become a revolutionary technology thanks to the integration of blockchain technology. A real-world vehicle purchase and sale system was used as a case study. Two contract structures were used that perform the same function: a modular architecture with five interacting contracts, and a monolithic architecture that combines all these functions in a single contract. An empirical analysis conducted for 100 vehicle sales revealed that while the modular architecture offers advantages such as independent upgradeability, testability, and maintainability, the monolithic approach outperforms it in many metrics, including a 36.7% reduction in transaction costs and a 75% faster deployment time. The findings provide evidence-based architectural guidance for blockchain and smart contract developers in selecting appropriate design patterns, particularly in real-world applications where gas costs are critical.

Index Terms—Blockchain, DApps, monolithic and modular architectures, smart contract

I. INTRODUCTION

A. Background and Motivation

Decentralized applications (DApps) have come to the fore with the proliferation of smart contracts and blockchain integration, enabling immutable and transparent transactions [1]. Smart contracts, known as self-executing programs, have become the cornerstone of advanced DApp applications by being deployed on blockchain platforms like Ethereum [2]. However, like every system, these systems become more complex over time and as technology advances, and they strive to choose an architecture that maximizes performance to reduce costs and maintain their sustainability [3]. Metrics such as gas consumption, execution speed, and deployment complexity play a role in smart contract architecture selection. By evaluating these metrics, the most critical architectural choice for smart contracts is between monolithic and modular architecture patterns [4, 5]. A review of empirical research reveals that there is a paucity of studies investigating and demonstrating these architectural approaches with measurable metrics in real-world scenarios.

B. Problem Statement

When examining existing applications, anecdotal evidence or theoretical principles are often used in selecting architectural patterns. This does not resolve the dilemma faced by developers: whether to choose a modular architecture, which offers advantages such as reusability, better separation of concerns, and independent upgradeability, or a monolithic architecture, which offers simplicity, potentially lower gas costs, and deployment costs [6]. Metrics such as transaction speed and gas consumption have not been precisely measured in the literature. Furthermore, the vehicle trading system used as a case study in this study presents unique and positive challenges for a good architectural comparison of these metrics:

- Regulatory compliance requirements
- Complex verification workflows requiring cross-domain data validation

directly measuring architectural-level impacts).

- *In terms of security and upgradeability, it's known that architectural decisions affect the attack surface, and modular architectures are more adaptable to practices like component-based updates. Furthermore, most vehicle/car-focused blockchain studies either don't perform performance benchmarks or compare alternative architectures (mostly presenting a single architecture at a "demonstration" level).*

WHAT THIS STUDY ADDS ON THIS TOPIC?

- *Based on a realistic case study (a vehicle trading system), two different architectures performing the same function (modular with 5 contracts vs. monolithic with a single contract) are designed and experimentally/empirically measured on the Ethereum testnet; the metric set is also expanded to include dimensions such as execution time, gas (tx+deploy), TCO (10-10,000 transactions), and bytecode size.*
- *The study concretizes the "reason" for the architectural differences: the external call overhead (~2600 gas) in the modular design is significantly higher than the internal call overhead (~100 gas) in the monolithic design; this widens the cost/latency difference, especially in multi-step transactions.*
- *It provides numerical and decision-making results + a decision framework: it reports that the monolithic architecture is superior in many metrics such as deployment gas (-14.4%), deployment time (-75%), transaction cost (e.g., -36.7% in 100 transactions), purchase gas (95k vs 185k), and transaction speed (-66% latency), and based on this, it provides a rule set for the question of "which architecture should be chosen under which conditions" with a decision matrix/decision framework; it also positions an open-source/near-production reference implementation and cost analysis framework as a contribution.*

- Multiple stakeholders with different responsibilities (notary, general directorate of security (GDS), tax office, etc.)
- High transaction frequency with cost sensitivity, etc.

C. Research Objectives

This study aims to address the following research questions:

1. RQ1: How do transaction execution speed and deployment time compare between the two architectural approaches?
2. RQ2: What is the quantitative difference in gas consumption between monolithic and modular smart contract architectures for vehicle trading operations?
3. RQ3: What are the implications of each architectural model on sustainability, scalability, and security?
4. RQ4: How insignificant is the initial deployment cost difference compared to operational costs?

D. Contributions

This work makes the following contributions to the field:

1. Open-Source Implementation: Production-ready, complete, smart contract implementations are provided in both architectural styles, and reference implementations are provided for future research and development.
2. Decision Framework: A decision matrix is proposed to guide developers in selecting appropriate architectural patterns based on project characteristics.
3. Cost Analysis Framework: A detailed cost analysis framework is developed that includes transaction, deployment, and total cost of ownership (TCO) calculations across various use cases.
4. Empirical Comparison: The first comprehensive empirical comparison of monolithic and modular smart contract architectures with real-world applications and quantitative metrics is presented.
5. Performance Comparison: Performance comparisons are created for both architectures across multiple dimensions, such as execution time, gas consumption, and deployment complexity.

The study concludes with a literature review in Section 2, a detailed explanation of the methodology in Section 3, an introduction to contract architectures and implementations in Section 4, experimental results in Section 5, and a conclusion in Section 6.

II. RELATED WORKS

A. Smart Contract Architecture Patterns

There are studies in the literature that examine architectural patterns in detail. Wohrer and Zdun [7], who cataloged 18 different design patterns, primarily patterns such as registry and factory patterns, focused on design patterns rather than high-level decision mechanisms. Xu et al. [8], who proposed a classification of smart contract patterns, used three different categories: architectural patterns, design patterns, and idiom patterns. This study, which lacks an empirical comparison of different architectural approaches, emphasized the importance of separating concerns in large-scale smart contract systems.

Chen et al. [9], who conducted a study on security patterns in smart contracts, revealed that architectural decisions significantly impact the attack surface of smart contracts. This study, which highlights modular architecture in the security field and suggests that it can reduce risks, has not been empirically validated.

B. Gas Optimization Techniques

Gas optimization is the most important optimization problem to solve in smart contracts. Such an important topic has also found its place in research. Pérez and Livshits [10], who stated that function call types and storage access patterns significantly affect gas consumption, presented a comprehensive study on efficient coding practices.

When their results were examined, they found that external function calls generate 2600 times the gas load compared to internal calls. This problem constitutes one of the key points of this study.

Chen et al. [11] developed a static optimizer for gas fee reduction, while Porkodi and Kesavaraja [12] investigated escalating gas costs. Liu et al. [13] proposed function dispatch reordering to reduce invocation fees, and Wang et al. [14] focused on detecting gas-expensive patterns in smart contracts.

Albert et al. [15], who developed a tool called GASOL, used genetic algorithms to optimize solidity code and attempted to optimize gas cost. While this tool focused on low-level optimization, this study focuses on higher-level architectural decisions.

Chen et al. [16], who aimed to estimate gas consumption before deployment and proposed a paradigm for this purpose, did not address the architectural impacts of monolithic and modular designs.

C. Smart Contract Upgradeability

The immutable nature of blockchain deployments has enabled researchers to investigate the upgradability patterns of smart contracts. A variety of upgradability patterns, including strategy patterns, infinite storage patterns, and proxy patterns, have been identified by Palladino [17].

Upgradability approaches are compatible with modular architectures by enabling independent component updates. Rodler et al. [18], who emphasized the importance of clear upgrade paths and proper separation of concerns, achieved this through their analysis of the Parity Wallet hack. Their findings demonstrate that the correct implementation of modular architecture can significantly increase system resilience.

D. Blockchain-Based Vehicle Systems

Vehicle trading, the real-world application used in this study, has also attracted the attention of blockchain researchers. Sharma et al. [19], who proposed a blockchain-based vehicle lifecycle management system, did not provide a performance comparison or address architectural trade-offs.

Banerjee et al. proposed a framework to increase safety, reliability, and sustainability in smart transportation systems. Researchers also used blockchain and smart contract technologies for emergency situations and security, aiming to increase energy efficiency by reducing communication costs [20].

Yuan and Wang [21], who developed a vehicle registration and transfer system on Ethereum with a monolithic architecture, only demonstrated their work but did not include alternative architectural approaches in their focus and analysis.

E. Research Gap

While studies addressing individual aspects of smart contracts, such as power optimization, design patterns, upgradeability, and domain-specific applications, exist in the literature, there are no studies that utilize quantitative comparisons of cost, security, speed, and sustainability across monolithic and modular architectures and realistic use cases. This study fills the gap in the literature by providing measurable metrics and empirical evidence to support architectural decisions in smart contract development.

III. METHODOLOGY AND SYSTEM ARCHITECTURE

A. Experimental Design

Two vehicle trading systems, functionally identical, were designed using different architectural patterns: (1) modular architecture (five different contracts working in collaboration) and (2) monolithic architecture (an architecture that combines all functions into a single contract). Both systems were validated on the Ethereum testnet, the Remix IDE, using the Solidity language version 0.8.0 and above, and deployed to the Ethereum testnet.

Ethereum is the most widely supported blockchain platform for smart contracts. The Ethereum Virtual Machine, thanks to its active developer support, comprehensive documentation, and support for the Solidity programming language, provides significant flexibility in the development, deployment, and testing of smart contracts. Furthermore, performance measurement metrics such as transaction costs, gas consumption, and deployment time are standardized and generally accepted on Ethereum. Furthermore, the transaction costs of transactions on Ethereum are higher than on other platforms, and new approaches need to be developed to reduce them.

Alternative platforms such as Binance Smart Chain, Solana, Polygon, and Avalanche offer lower transaction fees, but longer block confirmation times and different gas calculations limit the generalizability of comparison results. For these reasons, the Ethereum platform was chosen for this study to enable an objective comparison of smart contract architectures.

Evaluation metrics: Execution time, gas consumption (for transactions and deployments), TCO for exchange volume transactions (10-10 000 Transactions), and Bytecode size.

Economic assumptions: Cost calculations used an ETH price of $2500 and a gas price of 30 Gwei (moderate network congestion).

Test scenarios: For the testing process, 5 basic transactions were evaluated: (1) Vehicle registration, (2) Vehicle mortgage lien or tax debt status, (3) Notary approval, (4) Full vehicle purchase transaction, and (5) Full system deployment.

B. System Architecture

In the test case developed in this study, a vehicle trading system was developed with a system where four different stakeholders collaborate and conduct transactions through sequential verification, deployed on a blockchain-based network. Four different stakeholders—Vehicle Owners, the GDS, the President of Revenue Management, and Notary Services—securely conduct vehicle trading by verifying each other. The workflow is as follows: Vehicle registration—Vehicle inspections (GDS + PRM)—Notary approval—Sale—Ownership transfer.

1) Modular Architecture:

The modular architecture is built through the collaboration of five different contracts with five different interfaces. Algorithm 1 shows the pseudocode of the developed modular smart contract architecture.

Algorithm 1: ModularVehiclePurchase(plate, buyer, payment)

1: **BEGIN**

2: gas ← 21000

3: approved ← **EXTERNAL_CALL**(Notary.check(plate)) // +2600 gas

4: IF NOT approved THEN RETURN FAIL

5: vehicle ← **EXTERNAL_CALL**(Reg.getVehicle(plate)) // +2600 gas

6: **EXTERNAL_CALL**(GDS.isClean(plate)) // +2600 gas

7: **EXTERNAL_CALL**(PRM.hasDebt(plate)) // +2600 gas

8: **TRANSFER**(vehicle.owner, payment) // +21000 gas

9: **EXTERNAL_CALL**(Reg.updateOwner(plate, buyer)) // +5000 gas

10: RETURN SUCCESS, gas_total = 185000

11: **END**

Reg.sol: This contract maintains vehicle registration, ownership records, and metadata. (450 000 gas)

GDS.sol: This contract controls the vehicle's mortgage or lien status. (320 000 gas)

PRM.sol: This contract controls the vehicle's tax liability. (290 000 gas)

Notary.sol: This contract handles inter-contract verification and final sales approval. (580 000 gas)

CarSell.sol: This contract handles sales and payment transactions. (520 000 gas)

Total Distribution: 2 160 000 gas

Fig. 1 shows the structure developed with modular architecture for the Vehicle trading system.

Communication Model: Vehicles are initially registered in the Reg contract. Data for these vehicles is retrieved by the GDS and PRM contracts, and the verification phase is carried out. The Notary contract obtains the latest vehicle information from the GDS and PRM contracts and then approves it. CarSell sells vehicles that qualify for contracts. Each external call incurs approximately 2600 in gas overhead and state access costs.

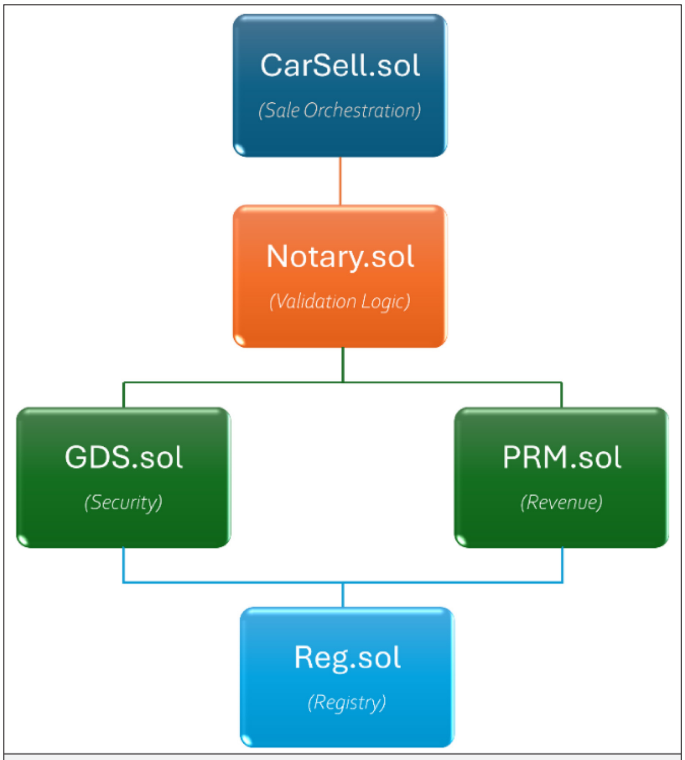


Fig. 1. Modular architecture design.

2) Monolithic Architecture:
A monolithic architecture is a contract that combines all stakeholders and functions into a single contract (CarSellMonolithic.sol), where internal function calls and interactions are made. Algorithm 2 shows the pseudocode of the developed monolithic smart contract architecture.

Algorithm 2: MonolithicVehiclePurchase(plate, buyer, payment)

1: **BEGIN**

2: gas ← 21000

3: approved ← **INTERNAL_CALL**(checkApproval(plate)) // +100 gas

4: IF NOT approved THEN RETURN FAIL

5: vehicle ← **READ_STATE**(vehicles[plate]) // +6300 gas

6: **INTERNAL_CALL**(isClean(plate)) // +100 gas

7: **INTERNAL_CALL**(hasDebt(plate)) // +100 gas

8: **TRANSFER**(vehicle.owner, payment) // +21000 gas

9: **WRITE_STATE**(vehicles[plate].owner ← buyer) // +5000 gas

10: RETURN SUCCESS, gas_total = 95 000

11: **END**

All state variables and other domains such as registration, revenue, approval, and security are contained within the same contract.

Total Distribution: 1 850 000 gas (14.4% lower)

Fig. 2 shows the structure developed with monolithic architecture for the Vehicle trading system.

Communication Model: Because contract boundaries are removed, all functions directly access the shared state, and all functions are executed as internal calls with an overhead of approximately 100 gas.

3) Key Architectural Differences:
Table I summarizes the main differences of the monolithic and modular architectures developed for the vehicle trading system with 5 features.

IV. RESULTS AND ANALYSIS

A. Gas Consumption Analysis

On the Ethereum platform, each transaction consumes a certain amount of gas. The total gas consumption of a function or contract call can be calculated as in (1):

$$Gas_{total} = \sum_{i=1}^{n} G_i \quad (1)$$

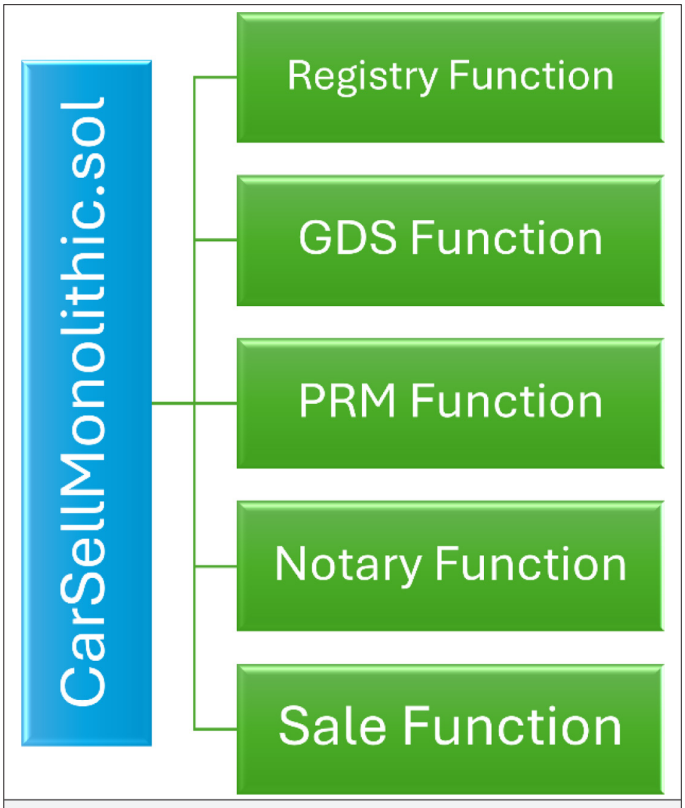


Fig. 2. Monolithic architecture design.

In this equation, *n* is used to represent the number of transactions executed in the contract. Each transaction type has a fixed gas cost on Ethereum (an example would be an ADD transaction = 3 gas).

The total transaction cost is calculated by multiplying the unit gas price by the total amount of gas used. This calculation is given in (2).

$$Cost_{transaction} = Gas_{total} \times Gas\ Price \tag{2}$$

In this equation, GasPrice is typically expressed in Gwei and represents the price the transaction owner will pay per unit of gas.

Gas_{total} is the total amount of gas spent during the contract execution.

If you want to calculate the total transaction cost in Ether, the unit conversion factor (10^9 Gwei = 1 Ether) is used. This calculation is given in (3).

$$Cost_{ETH} = \frac{Cost_{transaction}}{10^9} \tag{3}$$

The deployment cost is calculated similarly, but in this case, it represents the total amount of gas used when the contract is initially loaded onto the network. This calculation is given in (4).

$$Cost_{deploy} = Gas_{deploy} \times Gas\ Price \tag{4}$$

These equations form the basis of the "deployment time" and "transaction cost" measurements in the article and ensure that the comparison between modular and monolithic architectures is supported by objective metrics.

Table II shows the gas consumption costs for different processes. Depending on the process complexity, modular architecture has higher costs, ranging from 2.9% to 48.6%.

Key Findings: While the overall cost difference for simple transactions is between 3% and 5%, as the complexity of transactions increases, the cost increase can exceed 80% when moving from monolithic to modular. When a complete transaction is completed from start to finish, this increase reaches 94.7%. This is largely due to the five different external calls in the modular architecture. Fig. 3 shows the variation of gas consumption according to different transaction types.

B. Total Cost of Ownership

The TCO includes operational and distribution costs across different transaction volumes. For a better analysis, TCO was also calculated in this study.

Table III includes the total cost (distribution and operational) calculations for monolithic and architectural applications, based on the number of transactions after the application with the vehicle trading system.

A visual representation of total costs according to the number of transactions calculated in Table III is also given in Fig. 4.

Break-Even Analysis: The break-even point indicates the point at which the additional processing cost (e.g., from external calls) in the modular architecture is offset (i.e., the total costs are equal) compared to the monolithic architecture. This calculation is given in (5).

$$N_{break-even} = \frac{Cost_{deploy}^{monolithic} - Cost_{deploy}^{modular}}{Cost_{transaction}^{modular} - Cost_{transaction}^{monolithic}} \tag{5}$$

TABLE I. KEY ARCHITECTURAL DIFFERENCES

Aspect	Monolithic	Modular
Contracts	One unified	Five independents
State access	Direct	Cross-contract
Deployment	Single transaction	Sequential, five transactions
Call type	Internal (~100 gas)	External (~2600 gas)
Upgradeability	Full redeployment	Component level

TABLE II. GAS CONSUMPTION BY OPERATION TYPE

Operation	Monolithic	Modular	Overhead	Higher %
Vehicle registration	82 000	85 000	3000	3.7
Add seizure (GDS)	46 000	48 000	2000	4.3
Add tax debt (PRM)	44 000	46 000	2000	4.5
Notary approval	68 000	125 000	57 000	83.8
Complete trade	95 000	185 000	90 000	94.7

Fig. 3. Gas consumption.

The difference in deployment costs, equal to 310 000 gas, is offset after three transactions. After that, the disadvantage of the modular architecture increases linearly. At an enterprise scale of 10 000 transactions, the monolithic approach saves $135 623.

C. Performance Metrics

One metric that reveals significant performance differences is execution time analysis. It is a key determinant of performance, particularly when multiple contracts interact in a coordinated manner. Table IV includes a performance metric comparison of Monolithic and Modular Smart contract architectures, and these two types of architectures are compared in terms of speed for four different operations.

TABLE III. TCO ANALYSIS ACROSS TRANSACTION VOLUMES (USD)

Transactions	Monolithic	Modular	Savings	Cheaper %
10	$323	$458	$136	29.6
100	$1976	$3125	$1148	36.7
500	$9326	$14 975	$5648	37.7
1,000	$18 514	$29 787	$11 273	37.8
10,000	$160 489	$296 112	$135 623	45.8

A monolithic architecture executes a single transaction, while a modular architecture involves a complex transaction structure. Waiting for block confirmation in each contract requiring multiple consecutive transactions increases latency. Therefore, latency is 66%–75% lower in a monolithic architecture.

D. Qualitative Comparison

This study evaluated several architectural quality attributes simultaneously. Development complexity, testability, maintainability, and security metrics were evaluated, and an overall assessment was made.

Development Complexity: Monolithic architecture offers easy debugging and simple deployment processes, while modular architecture requires interface management, deployment organization, and address coordination.

Testability: Monolithic contracts require integrated testing, while modular contracts support separate unit tests for individual components.

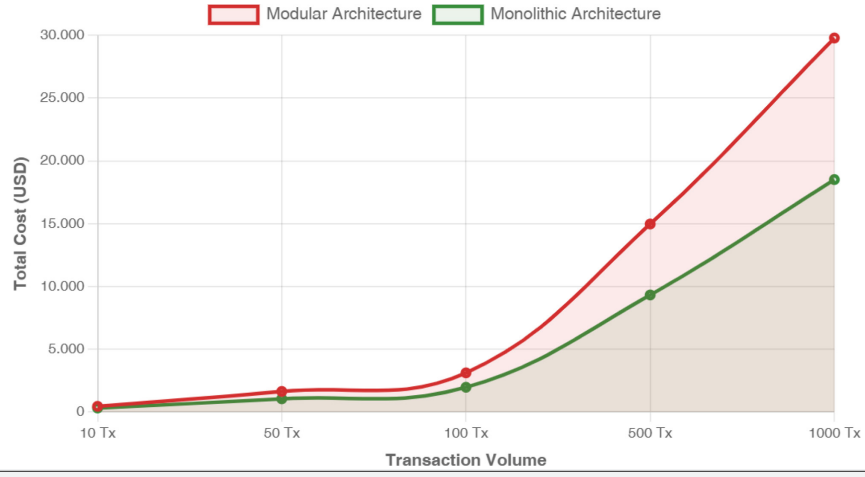


Fig. 4. Total deployment costs comparison for Monolithic and Modular architectures.

TABLE IV. PERFORMANCE METRIC COMPARISON OF MONOLITHIC AND MODULAR SMART CONTRACT ARCHITECTURES

Metric	Monolithic	Modular	Speedup
Deployment	13.7 ± 1.1	58.3 ± 4.2	4.26x
Vehicle Reg.	13.1 ± 0.9	14.2 ± 1.3	1.08x
Notary	14.3 ± 1.2	42.5 ± 3.7	2.97x
Complete trading	14.6 ± 1.3	43.1 ± 3.9	2.95x

Maintainability: Because modular architectures contain independent contracts, contract updates are easier. However, the need to maintain compatibility between interfaces also introduces complexity.

Security: By eliminating the need for transitions between contracts and reducing the attack surface with a single contract, monolithic architecture offers a more secure architecture.

Overall Assessment: All metrics were evaluated. While modular architecture offers advantages such as ease of maintenance and component-level testing, monolithic architecture stands out in metrics such as speed, security, complexity, bytecode efficiency, network load, deployment cost, and transaction cost.

E. Decision Framework

A smart contract for a real-world vehicle trading system was developed, and empirical analyses were conducted for two different smart contract architectures: monolithic and modular, with a total of 10 different metrics. The findings are presented in a tabular format in Table V.

According to the empirical findings, the following situations warrant the use of a modular architecture:

- When multiple independent organizations have their own components,
- When frequent updates are required,
- When component reuse is required across projects,
- When large teams require parallel development,
- When working in layer-two and low-cost networks.

When a monolithic architecture should be used:

- When security is important,
- When execution speed is important,
- When power optimization is required,
- When transaction volume exceeds 10 transactions,
- When system complexity is low to medium,
- When team size is small to medium.

V. CONCLUSION

This study presents a comprehensive and detailed analysis and empirical comparison of monolithic and modular smart contract architectures using a real-world application of a vehicle trading system. The findings definitively demonstrate that monolithic architectures outperform most metrics. Offering 66%–75% faster execution speeds, 37%–46% lower operational costs, and 14% lower deployment costs, monolithic architectures also stand out with their superior security.

While modular architecture offers theoretical advantages in testability and maintainability metrics, these features, coupled with a costly structure, also pose a deterrent for Ethereum networks.

One of the main contributions of this study is the measurement of the financial obligations and advantages of modular architecture in blockchain systems. Monolithic architectures are gaining prominence for applications such as vehicle trading.

As blockchain technologies advance, such as second-layer scaling solutions and reduced gas costs, the advantages of monolithic and modular architectures may shift. However, the results provide evidence of the power of monolithic architecture in smart contract development within the current Ethereum blockchain ecosystem.

This study is expected to contribute to developers conducting architectural research for blockchain systems by providing guidance in making practical decisions and serving as a methodological template.

TABLE V. COMPREHENSIVE PERFORMANCE COMPARISON

Metric	Monolithic	Modular	Winner	Difference
Deployment gas	1 850 000	2 160 000	Monolithic	–14.4%
Deployment time	12–15 sec	45–60 sec	Monolithic	–75%
Bytecode size	8500 bytes	10 800 bytes	Monolithic	–21.3%
Test isolation	Good	Excellent	Modular	Independent
Security	High	Medium	Monolithic	Atomic ops
100 Tx cost (USD)	1976.25$	3124.50$	Monolithic	–36.7%
1000 Tx cost (USD)	18 513$	29 787$	Monolithic	–37.8%
Maintainability	Medium	High	Modular	Flexible
Purchase gas	95 000	185 000	Monolithic	–48.6%
Transaction speed	12–15 seconds	36-45 seconds	Monolithic	–66%

Future studies could expand this analysis to different blockchain infrastructures (e.g., BNB Chain, Polygon, or Avalanche) to compare architectural differences across platforms. Furthermore, approaches such as dynamic gas optimization, off-chain execution (Layer-2 solutions), and automatic contract partitioning could be examined to further assess the potential of modular architecture to reduce gas costs.

Data Availability Statement: The data that support the findings of this study are available on request from the corresponding author.

Peer-review: Externally peer-reviewed.

Author Contributions: Concept – T.T.; Design – T.T.; Supervision – S.B.; Resources – T.T., S.B.; Materials – T.T.; Data Collection and/or Processing – T.T.; Analysis and/or Interpretation – T.T., S.B.; Literature Search – T.T., S.B.; Writing – T.T.; Critical Review – T.T., S.B.

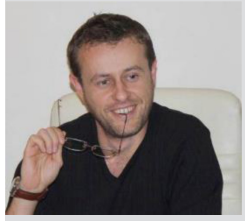
Declaration of Interests: The authors have no conflicts of interest to declare.

Funding: The authors declared that this study has received no financial support.

REFERENCES

1. V. Buterin et al., "A Next-Generation Smart Contract and Decentralized Application Platform", *white paper*, Vol. 3, no. 37, pp. 2–1, 2014.
2. N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997. [CrossRef]
3. T. Górski, "Smart contract design pattern for processing logically coherent transaction types," *Appl. Sci.*, vol. 14, no. 6, p. 2224, Mar. 2024. [CrossRef]
4. A. M. Antonopoulos, and G. Wood, *Mastering Ethereum: Building Smart Contracts and Dapps*. O'reilly Media, 2018.
5. Z. Liu, W. Feng, Y. Zhang, and C. Zhu, "Research on the architecture of transactional smart contracts based on blockchains," *Electronics*, vol. 12, no. 18, p. 3923, Sep. 2023. [CrossRef]
6. Z. Zheng, S. Xie, H. N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *Int. J. Web Grid Serv.*, vol. 14, no. 4, pp. 352–375, 2018. [CrossRef]
7. M. Wohrer, and U. Zdun, "Smart contracts: Security patterns in the Ethereum ecosystem and solidity," in 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2018, pp. 2–8. [CrossRef]
8. X. Xu, C. Pautasso, L. Zhu, Q. Lu, and I. Weber, "A pattern collection for blockchain-based applications," in Proceedings of the 23rd European Conference on Pattern Languages of Programs, 2018, pp. 1–20. [CrossRef]
9. J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "Defining smart contract defects on Ethereum," *IEEE Trans. Softw. Eng*, vol. 48, no. 1, pp. 327–345, 2022. [CrossRef]
10. D. Perez, and B. Livshits, "Smart contract vulnerabilities: Vulnerable does not imply exploited," in 30th USENIX Security Symposium (USENIX Security 21), 2021, pp. 1325–1341.
11. A. Kuhlman, and A. Wicaksana, "Smart contract optimization for gas fee reduction with static solidity optimizer," *Discov. Appl. Sci*, vol. 7, No. 8, Aug. 2025. [CrossRef]
12. S. Porkodi, and D. Kesavaraja, "Escalating gas cost optimization in smart contract," *Wireless Pers. Commun.*, vol. 136, no. 1, pp. 35–59, Jun. 2024. [CrossRef]
13. Y. Liu, W. Song, M. Christakis, and M. Pradel, "FunRedisp: Reordering function dispatch in smart contracts to reduce invocation gas fees," in Proc. 33rd ACM SIGSOFT Int. Symp. Software Testing and Analysis (ISSTA), Vienna, Austria. New York, NY, USA: ACM, 2024, pp. 516–527. [CrossRef]
14. S. Kumar, and K. Panda, "Gas-expensive patterns detection to optimize smart contracts," *Appl. Soft Comput.*, vol. 146, p. 110676, 2023. [CrossRef]
15. E. M. A. Albiol, J. Correas, G. Román-Díez, P. Gordillo Alguacil, and A. Rubio, '*GASOL: Gas Analysis and Optimization for Ethereum Smart Contracts*', 2020.
16. T. Chen, X. Li, X. Luo, and X. Zhang, "Under-optimized smart contracts devour your money," in 2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER). New York: IEEE, 2017, pp. 442–446. [CrossRef]
17. S. Palladino, 'The parity wallet hack explained, OpenZeppelin Blog Jul.19, 2017 .
18. M. Rodler, W. Li, G. O. Karame, and L. Davi, 'Sereum: Protecting existing smart contracts against re-entrancy attacks', *arXiv Preprint ArXiv:1812*, Vol. 05934, 2018.
19. P. K. Sharma, S. Y. Moon, and J. H. Park, "Block-VN: A distributed Blockchain based vehicular network architecture in smart city," *J. Inf. Process. Syst.*, vol. 13, no. 1, 2017.
20. S. Banerjee, D. Das, P. Chatterjee, B. Blakely, and U. Ghosh, "A blockchain-enabled sustainable safety management framework for connected vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 25, no. 6, pp. 5271–5281, 2024. [CrossRef]
21. Y. Yuan, and F.-Y. Wang, "Towards blockchain-based intelligent transportation systems," in 2016 IEEE 19th international conference on intelligent transportation systems. New York: IEEE, 2016, pp. 2663–2668. [CrossRef]

Tunahan Timuçin was born in 1991. He received his BS in Computer Engineering from Ankara University, Faculty of Engineering in 2015, his MS in Computer Engineering from Düzce University, Faculty of Engineering in 2018, and his PhD in Computer Engineering from Düzce University, Faculty of Engineering in 2024. In his master's thesis, he worked on operating room scheduling using genetic algorithms. In his doctoral thesis, he worked on blockchain, smart contracts, and their integration with artificial intelligence. He worked as a Research Assistant at Düzce University between 2017 and 2025 and currently serves as an Assistant Professor.



Serdar Biroğul was born in 1980. He received the B.Sc. degree from the Department of Electricity, Faculty of Technical Education, Gazi University, the M.Sc. degree from the Department of Electricity, Gazi University, and the Ph.D. degree from the Department of Electricity, Gazi University. He studied genetic algorithms in his master's thesis and studied data fusion in base station broadcast frequency planning in his Ph.D. thesis. He worked as a Research Assistant with Gazi University from 2002 to 2008, and as a Research Assistant (Dr.) with Mugla University from 2008 to 2010. From 2010 to 2012, he spent two years with Caretta Software Consultancy as a Dr. Research and Development Expert and Quality Manager, and in this company, he mainly focused on the European Union Euroka Celtic project that was named Context Based Digital Personality (CBDP) and Tubitak (The Scientific and Technological Research Council of Turkey) projects. He has been working as an Associate Professor Dr. with Duzce University since 2021. Starting in 2013, he took on several administrative tasks, such as Vice Dean, Director of the Distance Education Research Center, a member of the Faculty Administrative Board, and Head of the Software Engineering Department.