Smart Ways to Reduce Gas Fees for NFT Platforms on the Polygon Blockchain

Rahul Raghavan Tharammal a, \*, Prashant Nitnaware b

a *Department of Information Technology, Pillai College of Engineering, Maharashtra, India*

*E-mail:* *[trahul22me@student.mes.ac.in](mailto:trahul22me@student.mes.ac.in) ; ORCID:* [*https://orcid.org/0009-0002-7262-8975*](https://orcid.org/0009-0002-7262-8975)

b *Department of Information Technology, Pillai College of Engineering, Maharashtra, India*

*E-mail:* [*pnitnaware@mes.ac.in*](mailto:pnitnaware@mes.ac.in)

**Abstract.** NFT markets using blockchain technology are becoming increasingly popular in the digital asset market. They offer unique benefits such as owner history tracking, ownership segmentation, and ensuring safe operations. But using trading apps on the blockchain can be expensive because of something called gas fees. This paper explores ways to reduce these costs in the implementation and execution of contracts in the NFT market. We tested our ideas with Polygon blockchain and Solidity smart contracts, using VS code editor for testing. Our solutions include developing efficient NFT marketplace contracts to list property, execute transactions, and ensure ownership. Our analyzes show an average reduction of 35.14% in transaction costs and a 5.10% reduction in the cost of a single transaction, especially with multiple transactions role, these changes are expected to significantly reduce private sector gas costs, making NFT markets more efficient and effective.

Keywords: marketplace, blockchain technology, Gas Cost, Decentralize application, smart contract.

**1. Introduction**

NFT markets are online platforms where individuals can participate in trading, buying and using specific digital assets, each represented by a NFT token. These tokens serve as proof of ownership and authenticity of the digital asset. The NFT marketplace offers many benefits, such as facilitating efficiencies and ensuring the uniqueness and legitimacy of digital collections through cryptographic verification Compared to traditional centralized databases It is a choice they want to store data in today’s NFT market The NFT market plays an important role in the digital asset ecosystem by protecting derivative assets, protecting intellectual property rights and providing transparency on industry This forum addresses the individual collectors and institutional players involved in the buying and selling of digital assets Preferred.

When the polygon blockchain receives information about a transaction from the NFT marketplace, that transaction must be verified and then added to the blockchain network. A smart contract is like a draft contract that defines the terms, conditions, and in some cases, the performance of services. For example, it shows how to verify ownership and authenticity of NFTs, before they are added to the blockchain. Once set up, the smart contract automatically processes transactions and stores data. This allows NFT trading systems to operate in a decentralized and transparent manner, without relying on a central authority.

# 

# \*Corresponding author. E-mail: [trahul22me@student.mes.ac.in](mailto:trahul22me@student.mes.ac.in)

Blockchain technology faces challenges when it comes to handling business efficiencies, especially in areas such as better management of transactions, better utilization of resources, and faster processing of transactions The speed with which blockchain processes transactions can affect various aspects of the non-fungible token (NFT) market can , how quickly transactions are completed Transaction processing in blockchain requires transactions to be confirmed and recorded in the blockchain network, which requires electronic things are used. Smart contracts, which are simply pre-defined contracts that perform the various tasks, are critical to handling these tasks more efficiently. Additionally, there are gas costs associated with blockchain transactions. This cost is a form of compensation for the resources used in the transaction [1,2].

When using smart contracts in the NFT market, it is important to avoid unnecessary costs due to high gas costs. This money can compensate developers and users. To address this issue, manufacturers should focus on applying regulations to inefficient gas. But without clear guidelines it can be difficult to identify parts of the code that consume a lot of gas and replace them with more efficient methods to understand how Ethereum Virtual Machine works, where how much gas various applications reside understanding that the use of its data components is important in order to handle gas efficiently. There is a demand in the market for compilers that can process bytecodes efficiently to reduce gas consumption [3].

This research introduces an advanced smart contract designed for the NFT market. The main objective is to reduce gas consumption, thus reducing utility costs. Furthermore, the paper investigates various optimization methods, using variable weighting methods, reducing performance, fixing variables, and saving methods etc. This study is organized so that Chapters 2 and 3 we provide review related works and background. Chapter 4 explains the method used in this study. Chapter 5 shows the findings and discussion, and Chapter 6 complements the paper with conclusions

**2. Related Work**

Blockchain technology has gained significant popularity within the NFT (Non-Fungible Token) market owing to its robust support for decentralized applications (DApps). Ethereum stands out as one of the most widely used blockchains in this domain. This section delves into research that examines the enhancement of smart contracts to boost efficiency while mitigating rising costs.

The research conducted by N.M, J.V, J.G, Shaw, Ghosh aims to verify the issue of high gas costs associated with smart contracts in Ethereum blockchain system for quality certificate validation This is maximized by the use of three basic efficiencies: the use of bytes32 data type,[9]. Combining variables, combining revert () functions for error handling and scheduling of those variables, significantly reduces gas prices, resulting in savings of up to 0.18 % Also, researchers show that they can further reducing gas costs by increasing production volumes. In the study, D.S. Pramulia and B.A[14] This system had two main components: a ballot manager and a voting module. The role of an election manager includes tasks such as managing voter registration, managing election considerations, and smart contracts. To participate, voters had to register for their Ethereum accounts and vote. The researchers implemented, tested, and thoroughly evaluated the system. Their findings highlighted the importance of choosing the right gas price. The adoption of lower gas prices achieved a balanced approach between uptime and gas costs, ensuring efficiency while maintaining overall system performance As a result, service levels of the Ethereum network influenced the notes, which naturally affected the price of gas.

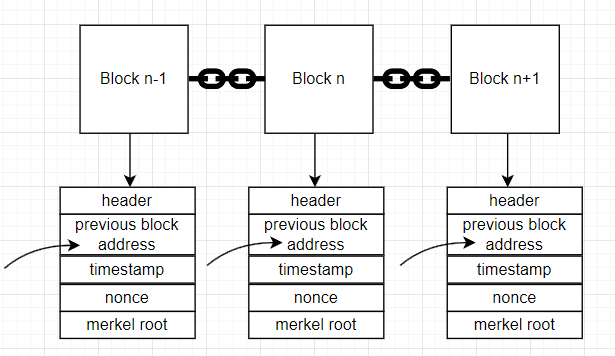
C. Li [15] has introduced the concept of overcoming major obstacles in the Ethereum blockchain ecosystem in a recent study. The course dives into the challenges of gas estimation for loop performance using a trace-based approach to accurately measure gas consumption in Lee loop behaviour and optimizing gas consumption associated with storage utilization, arrays, and machine learning algorithms -following recommendations, studied Recommend the contract is requalified with the goal of increasing gas consumption, reducing inventory and eliminating unnecessary set tests. Through detailed analysis, the study demonstrates the effectiveness of these proposed methods in accurately calculating gas costs and simplifying gas consumption. Ultimately, these developments are paving the way for affordable smart contracts.

L. M, M. M, and G. De performed the experiments. S, Barabino, D. T [16] Smart on the Ethereum blockchain provides a comprehensive analysis of strategies aimed at improving gas performance for contract purposes Different types of proposals divided into study areas which is important however. These categories involve steps for handling outside transactions, such as contracts and delivering transactions to an external address. Furthermore, in terms of saving memory storage space [17], optimizing storage performance for consistent data storage, reducing gas consumption to perform smart contract activities, overall, their performance in smart contracts on Ethereum provides benefits.

**3. Background**

*3.1. Polygon Technology*

Polygon, formerly known as Matic Network, serves as a solution for Ethereum's scalability problems, expensive gas fees, and sluggish transaction speeds. Acting as a supportive layer alongside Ethereum, Polygon employs various techniques like Plasma chains and optimistic rollups to significantly boost throughput while upholding Ethereum's security and decentralization principles. It offers developers a platform to create and launch decentralized applications (DApps) with reduced expenses and quicker transaction processing. Polygon's design allows smooth interaction between Ethereum and itself, facilitating easy asset transfers and smart contract operations across both networks. Moreover, it simplifies DApp development and integration through its range of toolkits and APIs. With a focus on scalability, user-friendliness, and compatibility, Polygon has become a leading choice for enhancing Ethereum's capabilities, attracting both developers and users to its active network. Blockchain technology brings unique advantages like privacy, anonymity, and trust, which find applications across various sectors such as food, healthcare, finance, insurance, and entertainment [4,5].



1. Polygon node structure.

*3.2. Smart Contract*

A smart contract is like a digital contract that automatically activates when it is set up. Instead of the terms of the contract being written on paper, they have been enshrined in an electronic program. This program runs on decentralized computer networks, which means they are controlled by a single authority. Smart contracts allow people to enter into contracts directly with each other without the need for intermediaries or lawyers like banks to oversee everything. The program automatically enforces the terms of the contract, provided certain conditions are met. This eliminates the need for intermediaries and saves time and money. Smart contracts are more secure because once established, they cannot be changed. They’re also transparent, which means everyone involved can see what’s going on. These features make smart contracts valuable in industries such as banking, real estate, and supply chain management. They are changing the way businesses are run, making them faster, safer and more reliable. Once this contract is executed, this contract is converted to bytecode and stored in the Polygon blockchain through a special transaction called the contract creation transaction Upon successful completion of this transaction, a unique address is generated for the smart contract [6,7,8].

*3.3. Distributed Ledger flow pattern*.

Understanding how a Distributed Ledger works helps us know when gas is used in a transaction [5]. Let's break it down:

1. Initiating a Transaction: When a user wants to do something like buying or transferring an NFT, they start by signing the transaction with their private key.
2. Sending to the NFT Marketplace: The signed transaction goes to the NFT marketplace platform.
3. Propagation Across Nodes: The transaction moves through different nodes in the NFT marketplace network.
4. Selection and Validation: Validators or miners, who process transactions, pick transactions from a pool based on rules and regulations. They assess the transaction's complexity to calculate the gas needed.
5. Gas Limit Check: The validator checks if the gas limit set by the user, which indicates the maximum allowed gas, is enough to cover the expected gas expenses. If the gas limit is lowered, the transaction won't go through. Gas prices are calculated by multiplying the gas price chosen by the user with the gas limit.
6. Transaction Aggregation and Mining: Validators group chosen transactions into a block and try to solve a cryptographic challenge. When a validator finds a solution for their block, they share it with other nodes.
7. Validation and Consensus: Nodes check the solution's authenticity and, if it's legit, they agree to add the block to the NFT marketplace blockchain. Once most nodes agree, the block is added, and the transaction is finalized.
8. Transaction Fee Calculation: The total fee for the transaction is calculated by multiplying the gas price by the gas consumed:

Fee = Gas Used × Price per Gas Unit

*3.4. Gas Fees*

Gas fees in blockchain refer to transaction fees required to perform transactions or smart contracts on the blockchain network, specifically for platforms like Ethereum [3] These fees are "gas" to pay users electronically damages used to implement and perform the services and to decide the complexity and mathematical complexity of the transaction or smart contract. Simple transactions require less gas compared to complex operations with more calculated smart contracts. Gas costs fluctuate based on grid saturation; During periods of high demand, such as when the number of services or increased usage in decentralized applications (DApps), the cost of gas increases Miners can prioritize services with higher gas costs, forcing users to use them are encouraged to pay more to speed up their transactions [5],[10-12]. Gas billing plays an important role in network security and efficiency and also serves as a distribution tool on the blockchain. But rising gas costs could challenge users, especially for small businesses, and spur ongoing efforts in the blockchain community to optimize and scale the network to reduce costs and provide experience their use has improved. Transaction fees serve as incentives for miners to incorporate transactions into the blockchain [9].

**4. Methodology**

In this research project, various gas fee optimization techniques previously explored in academic studies were implemented on a non-fungible token (NFT) marketplace built upon a Solidity smart contract. The smart contract was designed to handle essential functions such as validating the identities of buyers and sellers, logging transactions securely onto the blockchain, and maintaining an accurate record of NFT asset ownership details. Development and testing of the smart contract were conducted using the VS Integrated Development Environment (IDE), while Hardhat served as the private polygon blockchain for experimental purposes. By applying these optimization methods within the context of NFT transactions, the study aimed to enhance efficiency and cost-effectiveness in gas usage, thereby improving the overall performance of the NFT marketplace on the Ethereum network.

*4.1. Method for Gas cost reduction*

*4.1.1. Merging more than one variable to single word*

Effective use of storage facilities is essential to reduce gas costs and increase productivity in the NFT Marketplace Smart Contract. Each storage slot in the Ethereum Virtual Machine (EVM) can hold up to 32 bytes of data, as detailed in Algorithm 1. However, if the data stored in the slot is less than 32 bytes, it is abandoned where not used to minimize waste and maximize efficiency The connection between - is intelligent. When using the SSTORE opcode to store data, it is important to understand that the cost of gas depends on the existing price and the new stored price Compared to zero, the cost is expensive. Furthermore, replacing a slot is more expensive than leaving it unchanged, and replacing a slot that already holds data (called a "dirty" slot) is more expensive than replacing a clean one Such considerations is central to NFT marketplace smart contract where efficiency is paramount. For example, suppose we use a struct to store information about NFT tokens, including their unique ID and ownership. By organizing variables such as token IDs and ownership data in the struct, the compiler can efficiently insert them into storage slots, as shown in Algorithm 2. This not only saves space but reduces gas charges during contract execution thus carefully arranging data to storage match slots for insertion.

**Algorithm 1**

1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.0;
3. {
4. uint8 a = 10;
5. uint8 b = 20;
6. uint8 c = 30;
7. bytes32 combined;
8. combined = bytes32(uint(a) | (uint(b) << 8) | (uint(c) << 16));
9. }

**Algorithm 2**

1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.0;
3. }
4. uint8 a = 10;
5. uint8 b = 20;
6. uint8 c = 30;
7. bytes32 combined = combineVariables (a, b, c);
8. function combineVariables (uint8 a, uint8 b, uint8 c) internal pure returns (bytes32) {
9. return bytes32(uint(a) | (uint(b) << 8) | (uint(c) << 16));
10. }

*4.1.2. Using uint265 in place of int,uint*

In computer programming, especially in languages like Solidity used for creating smart contracts on blockchain platforms like Ethereum, the decision between using *int* and *uint* data types is very important for the efficiency and security of the code. Here's the difference: *int* can represent both positive and negative values, while *uint* is for non-negative values only. As computational needs become more complex and applications require larger numerical ranges, there's a need for more extensive data types explained in algorithm 3. That's where *uint256* comes in. It's a 256-bit unsigned integer, offering a much larger range of non-negative values compared to the traditional 128-bit *uint*. However, modern applications, especially in areas like blockchain and decentralized finance (DeFi), demand even larger numerical ranges. This is where *uint265* steps in. It's an even more powerful data type, expanding the numerical space to 265 bits as showed in algorithm 4, which provides an enormous range of non-negative integers. By using *uint265*, developers can ensure their applications are compatible with the evolving demands, accommodating larger numerical values while still maintaining the integrity and efficiency of their code. Moreover, transitioning from *uintN* to *uint256*, removes the need for extra gas costs related to type conversion and padding.

**Algorithm 3** – Before Combining

1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.0;
3. contract BeforeCombining {
4. uint8 var1;
5. uint16 var2;
6. uint32 var3;
7. uint40 var4;
8. function combineVars() public view returns (uint256) {
9. uint256 combined = uint256(var1) | (uint256(var2) << 8) | (uint256(var3) << 24) | (uint256(var4) << 56);
10. return combined;
11. }
12. }

**Algorithm 4** – After Combining

1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.0;
3. contract AfterCombining {
4. uint256 combined;
5. uint256 var1;
6. uint256 var2;
7. uint256 var3;
8. uint256 var4;
9. function combineVars () public view returns (uint256) {
10. combined = var1 | (var2 << 8) | (var3 << 24) | (var4 << 56);
11. return combined;
12. }
13. }

*4.1.3. Using byte in place of string*

Developers with Solidity often choose to use `bytes32` instead of `string` in some cases due to its efficiency and cost. Both types of data represent sequences of bytes, but they have specific characteristics that are suitable for different purposes. `bytes32` is basically a set of bytes with a fixed size, always with a length of 32 bytes. This fixed size ensures that data is properly stored and used on the Ethereum blockchain, according to the amount of gas there are predictable values ​​for storage and retrieval etc. Furthermore, being a primitive data type, `bytes32` allows for direct use in arithmetic bitwise operations, whereby cryptographic computation and similar low-level operations are facilitated In contrast, `string` represents a variable-length UTF-8 encoded string, which provides greater flexibility in handling textual data. String processing, however, consumes a lot of gas compared to `bytes32`, especially for large strings or complex operations such as concatenation or substring extraction. When deciding between `bytes32` and `string`, developers often consider factors such as gas costs, data size, and the specific needs of their application. If fixed-size data is sufficient and gas efficiency is important, `bytes32` is the preferred option. Conversely, if you are dealing with variable-length text data or need it to be compatible with off-chain systems, despite its expensive gas, `string` may be more appropriate. Algorithm 5 shows how to store values ​​to convert a string variable to bytes, as shown in Algorithm 6.

**Algorithm 5** – Before Converting String To Byte

1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.0;
3. contract beforeConvertingStringToByte {
4. string variable1 = "hello";
5. string variable2 = "world";
6. string combinedVariable = variable1 + variable2;
7. }

**Algorithm 6** – After Converting String To Byte

1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.0;
3. contract afterConvertingStringToByte {
4. bytes32 variable1 = "hello";
5. bytes32 variable2 = "world";
6. bytes32 combinedVariable = bytes4(uint(variable1) + uint(variable2));
7. }

*4.1.4. Removing unnecessary storing data variable*

Storing additional information on the Polygon blockchain incurs expenses known as gas fees. These fees rise as more storage space is utilized. To save on these fees, it's important to avoid storing unnecessary or duplicate information. This not only reduces the overall storage required but also lowers the associated gas expenses. Smart contracts that contain unnecessary data require more computational work, which translates to higher gas costs. By removing this excess data, the computational burden is lightened, resulting in lower gas expenses. Additionally, cutting out redundant variables and functions helps to further reduce gas consumption. For instance, in algorithm 7, there are multiple variables used across different functions, which can increase gas costs. However, in algorithm 8, these variables are streamlined, achieving the same functionality with fewer lines of code.

**Algorithm 7** – More Than One Variables

1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.0;
3. contract Example {
4. uint256 public variable1;
5. uint256 public variable2;
6. uint256 public variable3;
7. uint256 public variable4;
8. constructor (uint256i \_var1, uint256i \_var2, uint256i i\_var3, uint256i \_var4) {
9. var1 = i\_var1;
10. var2 = i\_var2;
11. var3 = i\_var3;
12. var4 = i\_var4;
13. }
14. }

**Algorithm 8** – Less Function

1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.0;
3. contract Example {
4. uint256 public combinedVariables;
5. constructor (uint256 \_var1, uint256 \_var2, uint256 \_var3, uint256 \_var4) {
6. combinedVariables = (\_var1 << 96) | (\_var2 << 64) | (\_var3 << 32) | \_var4;
7. }
8. }

*4.2. Smart contract*

We've created a smart contract for an NFT marketplace using a technique described in Section 4A to lower gas costs. This contract powers a basic NFT marketplace where users can buy, sell, and manage their NFTs. It's built to efficiently handle NFT metadata and transactions. The contract includes a simple task of connecting NFTs and organizing their information. It also has features to confirm ownership of an NFT, verify its information, and facilitate smooth transactions between buyers and sellers. These processes are optimized to keep gas costs low, ensuring that operations within the NFT marketplace are affordable for users.

The source code for the methodology using solidity is share on GitHub[[1]](#footnote-1).

**5. Result and Discussion**

In this part, we assess how much the gas fees decrease as a result of each optimization technique. We test the smart contract using the Remix IDE to gauge how these methods affect the gas cost in the NFT marketplace.

*5.1. Situation 1*

Initially, when the basic version of the NFT marketplace smart contract was deployed, it required a gas cost of 10,14,875. Later, by consolidating variables into a 32-byte single word within a structured format, there was a reduction of 9.45% in gas costs during the deployment of the contract. However, this optimization also caused a 2.34% increase in gas costs for individual transactions involving NFTs.

*5.2. Situation 2 - Using uint265 in place of int, uint*

Converting the data types from integers and unsigned integers of various sizes to a fixed size of 256 bits led to a further reduce to 8.364% in the gas for deploying the program. This optimization also contributed to reducing the gas cost for each transaction by 5.71% during the entire process of handling NFTs.

*5.3. Situation 3 - Using string in place of byte*

Converting the text into bytes led to a significant decrease in the amount of resources needed for deployment, cutting down the gas cost by 25.26% compared to the original setup. However, this improvement didn't impact the gas usage related to the NFT process.

*5.4. Situation 4 - Removing unnecessary storing data variable*

We managed to reduce deployment gas usage by about 10.97% by getting rid of unnecessary data storage. This approach resulted in a noticeable 4.36% drop in the gas cost for each NFT transaction.

1. Comparison Of Before and After Enhancement

Figure 2 presents a comparison of the quantity of gas utilized during contract deployment before and after optimization. By implementing these methods, we managed to significantly decrease gas usage by 451,598 units, which is equivalent to a reduction of 30.14%.

In one transaction, the gas fees have dropped by 2.56%. This shows how important these improvements are, especially when applied to many transactions. As more transactions happen, these enhancements are expected to significantly lower gas fees even more.

**5. Conclusions**

In summary, this research project aimed to cut down on the fees associated with transactions on a smart contract operating on the Polygon blockchain, specifically within an NFT marketplace. By adopting the proposed optimization methods, we managed to significantly reduce transaction fees by 2.36 percent per transaction. This decrease has the potential to make NFT marketplaces more cost-efficient and streamlined. Our findings underscore the importance of optimizing gas fees in boosting the affordability and scalability of NFT platforms. Reducing gas expenses enhances the scalability and inclusivity of the system, enabling a wider range of users to engage in more transactions without exceeding their budget. Subsequent research could explore further optimization methods and evaluate how they affect the efficiency and safety of different blockchain networks.

**References**

1. K. Toyoda, K. Machi, Y. Ohtake, and A. N. Zhang, “Function-Level bottleneck analysis of Private Proof-of-Authority Ethereum Blockchain,” IEEE Access, vol. 8, pp. 141611–141621, Jan. 2020, doi: 10.1109/access.2020.3011876.
2. K. M. Khan, J. Arshad, and M. M. Khan, “Investigating performance constraints for blockchain based secure e-voting system,” Future Generation Computer Systems, vol. 105, pp. 13–26, Apr. 2020, doi: 10.1016/j.future.2019.11.005.
3. T. Chen, X. Li, X. Luo and X. Zhang, "Under-optimized smart contracts devour your money," 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), Klagenfurt, Austria, 2017, pp. 442-446, doi: 10.1109/SANER.2017.7884650.
4. R. A. Canessane, N. Srinivasan, A. Beuria, A. K. Singh, and B. Kumar, “Decentralised applications using Ethereum Blockchain”. 2019 Fifth International Conference on Science Technology Engineering and Mathematics (ICONSTEM), 2019. doi: 10.1109/iconstem.2019.8918887.
5. G. A. Pierro and H. Rocha, The influence factors on Ethereum transaction fees. 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2019.
6. S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur and H. -N. Lee, "Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract," in IEEE Access, vol. 10, pp. 6605-6621, 2022, doi: 10.1109/ACCESS.2021.3140091.
7. W. Zou et al., "Smart Contract Development: Challenges and Opportunities," in IEEE Transactions on Software Engineering, vol. 47, no. 10, pp. 2084-2106, 1 Oct. 2021, doi: 10.1109/TSE.2019.2942301.
8. Aldweesh, M. Alharby, M. Mehrnezhad and A. Van Moorsel, "OpBench: A CPU Performance Benchmark for Ethereum Smart Contract Operation Code," 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 2019, pp. 274-281, doi: 10.1109/Blockchain.2019.00043.
9. N. Masla, V. Vyas, J. Gautam, R. N. Shaw and A. Ghosh, "Reduction in Gas Cost for Blockchain Enabled Smart Contract," 2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON), Kuala Lumpur, Malaysia, 2021, pp. 1-6, doi: 10.1109/GUCON50781.2021.9573701.
10. D. S. Andrea, S. Laudanna, A. Vacca, C. A. Visaggio, and G. Canfora, “Profiling gas consumption in solidity smart contracts,” arXiv.org, Aug. 12, 2020. https://arxiv.org/abs/2008.05449
11. S. Suriyakarn, “Solidity 102 #1: Keeping Gas Cost under Control - Band Protocol - Medium,” Medium, Dec. 10, 2021. [Online]. Available: https://medium.com/bandprotocol/solidity-102-1-keeping-gas-cost-under-control-ae95b835807f (accessed on march 10,2024)
12. R. Yu, A. M. Oguti, M. S. Obaidat, S.-C. Li, P. Wang, and K. Hsiao, “Blockchain-based solutions for mobile crowdsensing: A comprehensive survey,” Computer Science Review, vol. 50, p. 100589, Nov. 2023, doi: 10.1016/j.cosrev.2023.100589.
13. “Types — Solidity 0.8.20 documentation.” https://docs.soliditylang.org/en/v0.8.20/types.html (accessed on march 10,2024)
14. D. Pramulia and B. Anggorojati, "Implementation and evaluation of blockchain based e-voting system with Ethereum and Metamask," 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), Jakarta, Indonesia, 2020, pp. 18-23, doi: 10.1109/ICIMCIS51567.2020.9354310.
15. C. Li, “Gas estimation and optimization for smart contracts on ethereum,” 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Nov. 2021, doi: 10.1109/ase51524.2021.9678932.
16. G. Destefanis, "Design Patterns for Smart Contract in Ethereum," in 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C), Stuttgart, Germany, 2021 pp. 121-122.
17. doi: 10.1109/ICSA-C52384.2021.0003
18. “evm-opcodes/gas.md at main · wolflo/evm-opcodes,” GitHub. https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a7-sstore (accessed on march 10, 2024)

1. <https://github.com/RahulTharammal/Efficient-Gas-Fee-Strategies-for-Polygon-NFT-Platforms> [↑](#footnote-ref-1)