Building Blockchain-Based Options Markets

Student Name: William Hipperson
Supervisor Name: Dr Ioannis Ivrissimtzis
Submitted as Part of the Degree BSc Natural Sciences to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—Retail binary options markets have historically struggled to uphold the basic principle that trades should settle as agreed, with brokers frequently denying payouts on winning positions or arbitrarily closing investor accounts. This paper presents a decentralised binary options platform that eliminates broker-driven fraud by using blockchain and smart contracts to enforce trustless, immutable execution and automated payouts. Results show that the decentralised system provides strong security guarantees—including role immutability, access restrictions, and oracle integrity—and remains cost-competitive for buyers across trades worth $100 to $2,000. While sellers incur fees approximately two to three times higher than those charged by traditional exchanges, the system offers significant value through the elimination of counterparty risk. Performance analysis further highlights the sensitivity of costs to Ethereum network conditions and the potential for stablecoin integration to enhance price predictability. A proposed automated margining system outlines how future iterations could support dynamic risk management and credit-based collateral requirements. These findings demonstrate that blockchain can provide a robust, trust-minimised foundation for retail derivatives trading. The platform offers a viable alternative to traditional exchanges and highlights the broader potential of decentralised infrastructure in reshaping financial systems that depend on trust, fair and transparent execution.

Index Terms—Financial Derivatives, Smart Contracts, Decentralisation.

1 Introduction

Retail binary options markets have long been plagued by structural vulnerabilities, where traders often face significant counterparty risk due to opaque broker practices and weak regulatory oversight [1]. Cases of denied payouts, arbitrary account closures, and other forms of broker-driven fraud have eroded trust and highlighted systemic deficiencies in the existing market infrastructure. These problems are especially acute in retail derivatives trading, where individual investors have limited recourse against misconduct. Addressing these challenges is critical not only to protect participants but also to establish a foundation for fair, transparent, and accessible financial products. This paper investigates how decentralised technology can be leveraged to solve these persistent issues at their root, eliminating the need for trusted intermediaries by enforcing fair execution directly through code.

The platform was developed and tested on Ethereum using Ether (ETH) as the native currency. However, recognising the volatility of ETH relative to fiat currencies, the envisioned production deployment targets a custom-built network that uses a dollar-pegged stablecoin [2] as the settlement medium. Stablecoins offer a means to mitigate currency risk and improve cost predictability, which is critical for appealing to retail investors who require reliability, affordability, and transparency [2].

Through empirical evaluation, we find that buyer-side transaction fees remain competitive with traditional exchanges across realistic retail trade sizes ($100–$2,000), while seller-side fees are higher. Furthermore, the system satisfies strong security properties—including access control, role immutability, and trusted oracle integration—validating its suitability for adversarial environments.

This research aims to investigate whether blockchain-based financial infrastructure can serve as a secure and commercially viable alternative to traditional market structures and regulatory mechanisms. The results presented herein demonstrate that decentralised trading platforms, when properly engineered, offer a compelling path toward fairer, more resilient financial systems.

1.1 Objectives and Deliverables

A primary goal was to develop Solidity-based smart contracts capable of replicating and improving upon the role of traditional financial contracts typically handled by centralised exchanges. The proposed system was intended to guarantee payouts to the winning party through pre-funded contracts and to automate clearing and settlement entirely on-chain. Another key objective was to give users the flexibility to define custom contract parameters—such as strike price, expiration date, and position at the point of creation. A significant part of the design focused on enabling a secure and transparent flow of Ether (ETH) throughout the contract lifecycle. This begins with the initial deposit made by the creator. It continues with the transfer of the option premium by the buyer. Finally, it ends with the distribution of the payout at contract expiration. This ETH flow was intended to be managed exclusively through smart contract logic, promoting trustless execution and minimising reliance on intermediaries.

These objectives follow, formalised and sorted by complexity as basic, intermediate and advanced.

The basic objectives encompassed everything needed to create a system with the core functionalities.

1) Develop a prototype Binary Option smart contract with hardcoded parameters.
2) Build a basic Python Command Line Interface to interact with the blockchain network and deployed contracts.
3) Implement automated ETH transfers for payouts.

The intermediate objectives were focused on creating a functional exchange where multiple contracts could be created and interacted with. To match the added functionality in the smart contracts, the `Web3Py` app needed to be extended to allow for simple interactions with contracts.

1) Enable customisable contract parameters supported by time and asset price Oracles.
2) Design and deploy a Factory Contract to automate contract creation and management.
3) Extend the application to include market and portfolio visualisations.

The Advanced Objectives aimed to extend the system to broader use cases. The addition of Margining with automated adjustment was thought to make the exchange more attractive to investors creating contracts with longer terms and bigger payouts.

1) Establish the foundational components for an automated margining system, including communication mechanisms for assessing exposure.

From these Objectives, the project deliverables were decided:

1) A Python-based application interfacing with deployed smart contracts.
2) A suite of fully functional Solidity smart contracts enabling Binary Option trading.
3) Communication tools and architecture designed to support future implementation of automated margining systems.

1.2 Blockchain

Blockchain technology, as described by Nakamoto [3], is a distributed and decentralised ledger system that operates without the need for central authorities. At its core, blockchain allows users to transact directly with one another, bypassing traditional financial institutions, which typically act as intermediaries to prevent issues like double-spending. Instead, blockchain relies on cryptographic proof through a peer-to-peer network that timestamps transactions using proof-of-work or proof-of-stake [4] mechanisms.

In this system, transactions are grouped into blocks, each containing the hash of the current transaction and the hash of the previous one, linking them together to form an unbreakable chain. This structure makes blockchain inherently secure because altering a block would require redoing the proof-of-work for that block and all subsequent blocks, a task computationally impractical for any attacker unless they control the majority of the network's computational power.

Blockchain's decentralised nature is upheld by nodes (computers in the network), which verify and validate transactions independently, ensuring their accuracy. This decentralised verification ensures that the system remains secure as long as honest nodes control the majority of the computational power. This feature of consensus, where nodes agree on the longest chain of valid transactions, removes the need for central authorities and underpins the security and trustworthiness of the blockchain.

1.3 Smart Contracts

Smart contracts are digital agreements stored on a blockchain that automatically execute when predetermined terms and conditions are met. This automation provides all parties with certainty about the contract's outcome, ensuring fairness and transparency. By replacing a trusted third party with a blockchain network, smart contracts reduce costs and the chance of administrative errors or human intervention voiding the agreement [5]. Smart contracts build on decentralised marketplaces by facilitating execution of interdependent transactions.

1.4 Decentralisation

A decentralised approach to transactions implies that no single authority has unilateral control over the operation, validation, or governance of a system. Instead, decision-making and operational power are distributed across multiple participants or components, reducing the risk of systemic failure, corruption, or manipulation. However, decentralisation is not a binary it is multifaceted and can be evaluated along several distinct dimensions: architectural, political, and logical [6].

Architectural decentralisation concerns the infrastructure itself. It asks whether the system relies on a single or small number of machines to function. In a highly decentralised architecture, a large portion of the network could fail without disrupting the system as a whole. This promotes fault tolerance, as multiple independent components would need to fail simultaneously to bring down the network. However, decentralised systems can still be vulnerable to common mode failures—scenarios where superficially independent components fail due to shared dependencies [6].

Political decentralisation refers to who controls the components in the network. Even if a system is architecturally decentralised, it may be politically centralised if the majority of nodes or validators are operated by a single entity or closely aligned group. Political decentralisation reduces the risk of control or censorship being concentrated in one set of hands, thereby promoting more equitable and trustless systems [6].

Logical decentralisation pertains to how the system behaves and evolves over time. A logically decentralised system behaves like a swarm of independent actors, rather than a single monolithic machine. Interestingly, most public blockchains are logically centralised in the sense that they behave as one unified state machine with consensus on a single history of events. At the same time, they are politically and architecturally decentralised, with independent participants maintaining and operating the network [6].

There are several key reasons to pursue decentralisation in the design of financial infrastructure and transaction systems. Firstly, decentralisation increases fault tolerance.

In decentralised systems, the failure of individual components is less likely to bring down the entire network. Secondly, decentralisation enhances attack resistance. Public blockchains, for instance, require significant computational resources and coordination to compromise, making attacks prohibitively expensive. Finally, decentralised systems exhibit greater collusion resistance. Because power is distributed, it becomes harder for coordinated actors to act in ways that disadvantage less organised participants or the broader user base [6].

These qualities make decentralised systems especially attractive for applications where trust, resilience, and transparency are paramount—such as in decentralised finance (DeFi) and peer-to-peer derivatives markets.

1.5 Financial Derivatives

Derivative securities are financial instruments whose value is derived from an underlying asset [7]. These underlying assets can include Equities, Commodities, Foreign Currencies, Interest Rates and Cryptocurrencies. Derivatives play a critical role in modern finance, enabling participants to hedge risk, speculate on price movements, or gain exposure to assets without direct ownership.

Most derivatives are contractual agreements to buy or sell an asset under specified terms at a future date, or upon certain conditions. A common example is the Call Option. In exchange for a premium, a Call Option grants the buyer the right, but not the obligation, to purchase a specified quantity of an asset at a predetermined price (the strike price) before or at expiry. The buyer benefits if the asset's market price exceeds the strike price, while the seller profits if the market price remains below it. The seller of the option assumes a short position, collecting the premium as compensation for this risk [7].

It is important to understand that derivatives do not create or eliminate market risk; rather, they transfer it. For every party with long exposure, there is an equal short exposure somewhere else in the market. This zero-sum nature defines derivative markets, and underpins their use for risk management and speculative positioning [8].

Counterparty risk refers to the risk of the losing side of a contract defaulting on their obligation to provide the defined payout. This risk is much harder to mitigate than market risk, it's for this reason that central authorities have been established to ensure contractual obligations are performed.

Clearing houses manage counterparty risk through margining, requiring participants to deposit additional funds as market movements increase the likelihood of losses on their positions. There are two primary types of margin. Variation margin (VM) is exchanged daily and reflects changes in the market value of an open derivatives position—essentially marking trades to market and settling gains or losses incrementally. Initial margin (IM), on the other hand, is posted at the start of a trade to protect against the risk that the defaulting party's position cannot be closed out or replaced immediately. IM covers potential exposure during this close-out period. Together, these mechanisms serve to reduce the systemic risk posed by derivatives markets and are central to regulatory frameworks in both developed and emerging financial systems [9].

1.5.1 Market Structures

Derivatives are conventionally traded through two main market structures [10]:

1) Exchange-Traded Derivatives (ETDs) Standardised contracts traded on organised exchanges such as the Chicago Mercantile Exchange (CME). These contracts benefit from centralised clearing, transparency, and lower counterparty risk due to the involvement of a clearing house.

2) Over-the-Counter (OTC) Derivatives Customised contracts traded bilaterally between parties, usually without the intermediation of a clearing house. OTC markets offer greater flexibility in contract design but come with increased counterparty and credit risks, as clearing is typically not centralised.

Historically, the derivatives space was dominated by OTC contracts tailored to institutional needs. However, in the aftermath of the 2008 financial crisis, there has been a growing push toward standardisation and central clearing [10].

1.5.2 Derivatives Lifecycles

The lifecycle of a derivative contract typically involves the following stages [10]:

1) **Creation:** The terms of the contract are defined, either through standard exchange templates or bespoke OTC negotiation.
2) **Execution:** The contract is agreed upon by the involved parties.
3) **Clearing:** A clearing house or intermediary validates the trade and ensures margin requirements are met throughout the contract's life, reducing counterparty risk.
4) **Settlement:** At maturity or upon trigger conditions the contract is settled according to its terms, either through asset delivery or cash payment.

1.6 Fraud in Binary Options Markets

A Binary Option is a contract where the buyer speculates on whether an asset's price will move above or below a specific strike price at a set date, receiving a predetermined payout if correct, or nothing if incorrect [7]. Binary options, banned for retail consumers in the UK, were prone to manipulation by fraudulent brokers who altered software to fake prices and closed trading accounts without warning. The UK's Financial Conduct Authority (FCA) implemented a permanent ban on these instruments in 2019, citing widespread consumer harm and market abuse [11].

In its ruling, the FCA highlighted that binary options resembled gambling products disguised as financial instruments. UK retail investors experienced severe losses, with fraudulent brokers manipulating outcomes, generating false trades, and refusing withdrawals. The FCA's permanent ban covered all binary options, including those marketed from abroad, with estimated savings from fraud for consumers reaching £17 million per year [11].

1.7 Project Outcomes

The project successfully delivered a secure, decentralised, and cost-effective binary options platform tailored for retail users. Security was validated through static and adversarial testing, ensuring strict access controls and automated, tamper-proof payouts.

Buyer-side fees remained highly competitive, consistently below 0.1% for contracts between $100 and $2,000, outperforming traditional brokerage models. Although creator-side fees were higher due to contract deployment costs, they remained manageable and could be further optimised.

Political decentralisation was achieved, removing reliance on trusted intermediaries while maintaining transparent, predictable execution. The results demonstrate that blockchain-based derivatives trading can offer a viable, trustless alternative to traditional markets, significantly reducing counterparty risk for retail participants.

2 Related Work

To situate this project within the broader context of decentralised finance and smart contract development, this section reviews contemporary efforts aimed at implementing blockchain-based financial systems. Particular focus is given to approaches that address over-the-counter (OTC) derivatives, standardisation through the ISDA Common Domain Model, and user-focused improvements in DeFi usability. These works collectively highlight ongoing challenges and innovations in automating complex financial instruments, ensuring regulatory compliance, and improving decentralised infrastructure—issues that this project seeks to address through a simplified, trustless binary options platform.

2.1 Decentralised Finance

Decentralised finance (DeFi) is emerging as a popular alternative to traditional financial institutions, offering enhanced security against hacking and lower transaction costs by eliminating the need for intermediaries. It also facilitates faster 24/7 transaction settlements, extending market hours, and increases efficiency through distributed record-keeping without relying on large data centers. Additionally, DeFi simplifies cross-border payments, enabling financial products to be traded anytime on any exchange [12].

Smart contracts play a crucial role in decentralised finance (DeFi) by automating transactions and interactions such as token issuance, decentralised exchanges (DEXs), and protocols for loanable funds (PLFs). By facilitating automated, trustless exchanges, smart contracts enhance the efficiency of financial transactions, reduce costs, and increase transparency. However, challenges remain, including reliance on external data oracles and difficulties in enforcing contracts for off-chain assets, which highlight the need for ongoing integration with traditional legal frameworks [13]. With regards to legal requirements, [14] proposes a system of *"imbedded regulation"* whereby traditional regulatory frameworks are encoded in the architecture of all Decentralised Financial applications.

2.2 Contemporary Methodologies

To address the implementation of smart contracts for over the counter (OTC) derivatives, Fries et al. [15] propose the development of a decentralised prototype using Ethereum/Quorum infrastructure. The authors first contextualise the challenges facing the OTC derivatives market, emphasising the risks associated with counterparty credit and procedural inefficiencies. To mitigate these issues, they propose smart contracts with deterministic terms that encompass the entire lifecycle of derivative transactions, thereby enhancing clarity and reducing disputes. A key aspect of their methodology is the incorporation of reliable timing mechanisms and event triggers, to facilitate settlement by a passive approach. The use of decentralised ledger technology (DLT) is highlighted for its potential to improve process transparency and efficiency, although challenges such as slower performance and reliance on external valuation oracles are acknowledged. A prototype is developed that enables banks to interact with smart contracts while leveraging a centralised valuation service for off-chain data delivery.

The methodology outlined by Casey-Fierro [16] focuses on the development of a Smart Confirmation Contract system, which is designed to align with the International Swaps and Derivatives Association's (ISDA) standards for over the counter (OTC) derivatives. The approach employs modular smart contracts that operate on Ethereum-compatible blockchains, integrating the ISDA's Common Domain Model (CDM) [17]. Key steps in this methodology include the translation of the CDM into Solidity and the modularisation of contract components. The Smart Confirmation Contract acts as a central hub coordinating various logical modules that handle distinct functions within the derivatives lifecycle, allowing for flexibility and modular updates. Additionally, the integration of CDM modules ensures standardisation of contract terms, thereby enhancing interoperability within derivatives infrastructure. The methodology also introduces logic modules that govern the contract lifecycle, supported by a user interface and real-time data feeds via Chainlink oracles.

In addressing the challenges associated with the user-friendliness of token transfers on blockchain exchanges, ViswasReddy et al. [18] introduce a decentralised web application that leverages ERC-20 standards to enhance efficiency and accessibility within DeFi. While ERC-20 tokens have facilitated the growth of tokenised ecosystems, the user experience has been hindered by issues such as network congestion and high gas fees. To combat these problems, they propose a decentralised application designed to simplify token transfers through smart contracts. The methodology involves the development of smart contracts using Solidity, which manage core token functionalities, and a user interface built with HTML and JavaScript for intuitive user interaction. From their testing, `Hardhat` was revealed to be the best development environment, scoring highly on efficiency and debugging capabilities. Tools such as MetaMask, `Etherscan` are identified as essential for allowing users to connect with the market and access their data. Future improvements, such as gas fee optimisation and cross-chain compatibility, are also highlighted as essential steps for ongoing enhancement.

2.3 ERC20 Tokens

ERC-20 is a widely adopted Ethereum token standard that defines a set of necessary methods and events supported by fungible tokens, enabling interoperability between decentralised exchanges and applications. It remains foundational in DeFi for standardising token behaviours within smart contracts [4].

2.4 Legal Framework

The ISDA [17] outlines a framework for creating legally enforceable smart contracts in the derivatives market. It emphasises combining legal requirements with operational efficiency. The approach includes identifying contract components that can be automated and converting legal terms into structured, machine-readable formats. The framework addresses complexities in automating contract terms, such as distinguishing between operational clauses and those needing human judgment. Similarly to [15] this paper emphasises the need for margin buffers to ensure transactions complete, and deterministic early termination clauses in smart contract logic. Here, the ISDA advocates for collaboration between legal experts and programmers to align legal drafting with smart contract logic. This ensures automated terms retain their intended legal meaning. The ISDA Common Domain Model (CDM) plays a central role by providing standardised templates and functional components, ultimately enhancing market efficiency and legal compliance across jurisdictions.

The Common Domain Model [19] aims to standardise the lifecycle of derivatives, enhancing market efficiency by establishing a single central standard that promotes innovation within financial markets. By providing both machine-readable and machine-executable formats, this framework seeks to facilitate automation and interoperability across various platforms. The implementation of a standardised (CDM) to decentralised derivatives is essential, as it allows all events and processes to utilise shared data, significantly reducing time and risk associated with derivatives transactions. This consistency in data and sources not only fosters transparency between regulators and market participants but also supports the integration of distributed and shared technologies, ultimately transforming the current market structure into a more cohesive and efficient ecosystem.

3 Methodology

This section outlines the technical approach taken to design, implement, and evaluate the decentralised binary options trading platform. The methodology spans the entire development lifecycle—from environment setup and smart contract design to execution flow, testing, and deployment. Each component was engineered with a focus on security, modularity, and cost-efficiency. Development leveraged Ethereum-compatible tools, including Solidity, `Hardhat`, and `Web3Py`. Particular emphasis was placed on creating an execution model enforced entirely on-chain, supported by oracles for external data and an automated Python-based interface for user interaction. The methodology was guided by real-world constraints and practical considerations, ensuring the system's architecture remained grounded in both usability and robustness.

3.1 Development Environment

This project utilised Solidity, the most widely adopted language for Smart Contract development. Designed with an Object-Oriented approach, Solidity replaces traditional classes with contracts and libraries. Its compatibility with Ethereum networks stems from its ability to compile directly into Ethereum Virtual Machine (EVM) bytecode [20]. Solidity version 0.8.23 was selected for its stability and access to the latest language features. All contracts were compiled into JSON files, which include both the Application Binary Interface (ABI) and the compiled bytecode—crucial components for deploying and interacting with contracts programmatically.

Solidity was favoured over other Smart Contract languages due to its expressive access modifiers, enabling developers to define fine-grained security controls.

To compile, deploy, and interact with these contracts, the `Web3Py` library was used in Python. `Web3Py` provides a comprehensive suite of tools for blockchain interaction, including transaction signing APIs, gas price estimation, and conversion between Python-native and EVM-compatible data types. `Web3Py` also allows developers to predict gas usage prior to execution and retrieve gas data from mined transactions—features that are instrumental for optimising smart contract performance [21]. While `Web3Py` offers robust functionality, it presents certain limitations—most notably, the difficulty of contract verification, as tools like `Etherscan` are primarily designed for JavaScript-based environments such as `Web3.js`, making bytecode-source matching more cumbersome in our system.

Python was chosen over JavaScript-based alternatives like `Web3.js` primarily due to time constraints. While `Web3.js` offers more robust tools for frontend development, `Web3Py` in Python enabled quicker development of back-end logic, which was prioritised during early stages of the project where functionality outweighed interface requirements. In the later stages of development the choice to use Python for App development incurred a cost on the quality of the UI produced due to Python's inferior Graphical User Interface tooling.

To interface with the blockchain, the `HTTP` provider in `Web3Py` was configured to connect with a locally hosted `Hardhat` network [21]. `Hardhat` serves as a powerful Ethereum development environment offering Solidity compilation, script automation, and integrated testing via Mocha and Chai. `Hardhat` launches a local Ethereum network pre-configured with accounts funded with 10,000 virtual ETH, facilitating fast and cost-free prototyping. `Hardhat`'s detailed stack traces and interactive console significantly enhance the debugging process [22].

`Hardhat` was chosen over the `Sepolia` test-net [23] for its superior flexibility and developer-focused tooling. One of the key advantages of using `Hardhat` is its ability to reset the local blockchain to its initial state almost instantaneously. This is invaluable during rapid development and testing cycles, where consistent baseline conditions are necessary. Additionally, `Hardhat` enables full control over a large number of accounts, allowing the simulation of complex multi-party interactions without external dependencies. Unlike public test-nets such as `Sepolia`, which require wallet integration and faucet management [23], `Hardhat` simplifies authentication and account management by removing the need for wallet-based logins entirely. This streamlined setup facilitates faster iteration, especially when switching between accounts. Furthermore, `Hardhat` provides detailed error reporting directly in the console [22], eliminating the reliance on third-party services like `Etherscan` [24] for transaction debugging.

However, a key limitation of using `Hardhat` is its inability to simulate real-world blockchain conditions—particularly network congestion, fluctuating gas prices, and unpredictable transaction ordering. These factors are critical when evaluating the practical performance and user experience of decentralised financial applications. In this respect, a public test-net like `Sepolia` [23] may offer a more accurate representation of production environments, allowing developers to assess how the system behaves under realistic network conditions and to better anticipate user-facing costs and delays.

3.2 Contract Design

Initially, a prototype Binary Option contract was developed with hardcoded parameters. This prototype was used to implement and test the automated ETH transfer logic: depositing the payout into the contract, transferring the contract price to the creator, and paying out the final amount to either the creator or buyer upon expiration.

To introduce access control and improve security, a second contract called `CreateBO` (Create Binary Option) was created. This contract serves as a wrapper around the core `BinaryOption` contract, masking its address and enforcing a two-layered access model. Only verified users, either the contract's creator or a registered buyer, can call `CreateBO` functions. Any direct access to `BinaryOption` is blocked; calls to it are only allowed if they originate from its registered `CreateBO` contract. Additionally, the address of the Binary Option contract is stored using solidity's `private` modifier [20], limiting access to each Binary Option contract instance. This design ensures robust encapsulation and limits attack vectors that would otherwise exist if the core contract were public. Most of these controls are enforced on-chain using custom Solidity modifiers such as `onlyFactory`, `onlyCreator`, and `onlyThis`, ensuring that role-based access is strictly maintained at the contract level. This delegation of security enforcement to the blockchain significantly reduces reliance on the Python back-end, which acts merely as a relay for data and time updates.

To enable contract settlement based on expiration timing, a decentralised time mechanism was required. While Solidity offers `block.timestamp`, relying solely on this can be insecure, as miners can influence it slightly [25]. This issue is exacerbated in time-sensitive financial instruments. To overcome this, a `TimeOracle` contract was developed. This contract securely maintains the current time by accepting timestamp updates pushed from the Python back-end. A scheduler component in the back-end defines five evenly spaced checkpoints from deployment to expiry. At each point, it relays the current time and asset price—retrieved via the `YahooFinance` Python module to `CreateBO`, which in turn updates the oracle. This mechanism ensures accurate, tamper-resistant tracking of both time and price data, supporting fair contract settlement.

To enforce clean code separation and modularity, the contracts utilise Solidity interfaces [20] to facilitate inter-contract function calls. These interfaces define public APIs that allow contract components to interact without tightly coupling their implementations. However, Solidity interfaces only support `external` functions, which exposes them to potential misuse if not carefully managed. To mitigate this, function wrapping was used extensively: core functionality was implemented in `internal` or `private` functions and then invoked by `external` wrapper functions that apply strict access control via modifiers [20]. This pattern ensured that sensitive logic was never directly accessible through interface calls, reducing the risk of vulnerabilities stemming from external invocation and enforcing proper usage flows across the contract system.

Solidity's function visibility specifiers—`external`, `public`, `internal`, and `private`—are applied consistently across the codebase. `External` functions reduce gas cost for external calls; `private` variables protect internal logic; `internal` functions support inheritance and logic reuse. Pure and view functions are used for non-state-changing operations, enabling read-only access at no gas cost.

Several immutable variables are initialised during contract construction, including creator and factory addresses. Using `immutable` enhances gas efficiency and reinforces the integrity of critical configuration parameters by inhibiting all methods of changing specified variable values.

Key state variables, such as expiration timestamps, asset tickers, and ownership records, are carefully scoped and access-controlled. These variables define the essential state of each contract instance, and their visibility—usually `private` or `internal`—ensures they cannot be modified or misused externally. For example, buyer and creator roles are stored in the `CreateBO` contract to gate function access, and the `expiryDate` and `underlyingPrice` in `TimeOracle` regulate contract maturity and settlement fairness.

3.3 Contract Lifecycle

A central component of the system's architecture is the `Factory` contract, which plays a foundational role in coordinating the creation of new binary options contracts. Factory contracts are a widely adopted design pattern in smart contract development, particularly for applications requiring the efficient deployment of multiple contract instances with shared logic but unique parameters. In this implementation, the `Factory` acts as the orchestrator of contract creation, standardising the deployment process while ensuring traceability and enforceable constraints on new market listings. Its inclusion not only enhances scalability and modularity but also simplifies user interaction by serving as the single point of entry for contract deployment.

3.3.1 Creation

The `Factory` contract's Deploy function is responsible for creating a `CreateBO` and `Binary Option` contract pair.

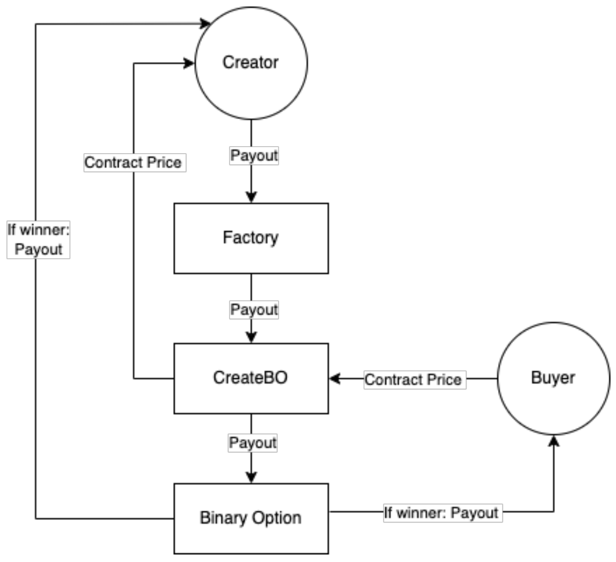

Fig. 1. Flow of ETH through the contract architecture.

During deployment, the contract creator specifies key parameters, outlined in Table 1., except for the `strikePrice`. Each Contract's strike price is defined by querying the current price of the underlying asset that the creator has chosen. This restriction prevents creators from deploying contracts that are unlikely to sell. Without such a restriction a creator could deploy a contract betting that an asset's price will be lower than an unfeasibly high future level, guaranteeing a win if their contract is bought. If deployed, such a contract is very unlikely to sell due to the almost-zero probability of the buy side winning. However deploying such contracts would cause unnecessary congestion, leading to higher costs for users without providing any liquidity. To minimise congestion due to uncompetitive contracts, creators are restricted to betting on whether or not the underlying will have risen above, or dropped below its price at contract creation.

To successfully create the contract pair, the creator must ensure they have sufficient ETH available to cover the required payout, which is automatically deposited into the Binary Option contract. This process follows the structured flow outlined in Fig 1.: the creator's account transfers the necessary funds to the factory contract, which then facilitates the creation of the `CreateBO` contract, ultimately leading to the deployment of the `Binary Option` contract.

3.3.2 Execution

Once validated, contracts are listed on an exchange interface, allowing users to view their parameters and select contracts they wish to purchase. To buy a contract, a prospective buyer inputs the address of the corresponding `CreateBO` contract into the `buy()` function on the front end. This process automatically transfers the contract price to the creator via the `CreateBO` contract and updates the `buyer` state variable in the corresponding `CreateBO`, `Binary Option` contract pair. Once the buyer state is set, the contract is considered sold and can no longer be purchased by another user.

TABLE 1
Parameters for Binary Options Contract.

Parameter	Type	Description
_ticker	`bytes32`	Symbol or identifier of the underlying asset (e.g., BTC, ETH). Used to reference price data from an oracle.
_strikePrice	`uint256`	The price level that determines the outcome of the binary option. If the market price crosses this level at expiration, the contract pays out.
_expiryDate	`uint256`	The UNIX timestamp representing the expiration date/time of the contract. Determines when the condition is evaluated.
_payout	`uint256`	The amount paid to the holder if the contract resolves in their favour (i.e., their prediction is correct).
_position	`bool`	Indicates whether the **creator** is assuming a Long (`true`) or Short (`false`) exposure to the underlying asset.
_contractPrice	`uint256`	The cost to purchase the option. This is typically paid upfront and reflects the premium for taking on the position.

3.3.3 Clearing

Since contract creators were required to deposit the full payout at creation time, the clearing logic did not involve any dynamic margin adjustments.

To support contract exercise and payout at expiry, an off-chain scheduler was used to periodically push updates to the on-chain `TimeOracle` contract. Updates were performed at five evenly spaced intervals, including the expiration date. Each update recorded the current UNIX time and the underlying asset's price, retrieved from a financial API. Upon receiving these updates, the `TimeOracle` forwarded the data to the associated `CreateBO` contract via the `receiveUpdate()` function. The choice to refresh the underlying's price stemmed from guidance presented in the ISDA's Common Domain Model framework, where they highlight the importance of being sure of derivatives' values throughout their life [17]. The choice to use five time stamps is arbitrary, however in future iterations the frequency of could be decided based on the volatility of the underlying or user preferences.

The `CreateBO` contract then compared the received timestamp against its predefined `strikeDate`. If the contract had reached or passed its expiration, it automatically called the `terminate()` function in the underlying `BinaryOption` contract, passing along the final asset price. This function executed the logic for determining the recipient of payout.

This architecture ensured that the clearing and settlement process was decentralised and minimally reliant on manual intervention. The inclusion of asset price and time tracking through the oracle system also enabled accurate state visualisations, such as whether a contract was *in the money* or *out of the money* [7], mirroring the payoff structure of European-style options.

3.3.4 Settlement

Settlement is handled automatically as soon as the expiration date is reached. If the contract has not been bought, the creator receives the payout back. However, if the contract has been purchased, the payout is distributed to the winning party based on the contract's outcome as outlined in Algorithm 2. It is important to note here that the position variable in stored in the contract pair refers to the **creator's** exposure. It follows that the buyer's exposure is the opposite, maintaining the conservation of risk [7].

Algorithm 1 BinaryOption.terminate()

```
1:  procedure SETTLE
2:      if not isBought then
3:          recipient ← contractCreator
4:      else
5:          if position is long then
6:              if expiryPrice ≥ strikePrice then
7:                  recipient ← contractBuyer
8:              else
9:                  recipient ← contractCreator
10:             end if
11:         else
12:             if expiryPrice ≤ strikePrice then
13:                 recipient ← contractBuyer
14:             else
15:                 recipient ← contractCreator
16:             end if
17:         end if
18:     end if
19:     Transfer balance to recipient
20:     isExpired ← true
21: end procedure
```

3.4 Market Structure

The implemented trading platform functioned as an automated, 24/7 derivatives exchange with anonymous trading. It reflects the always-on, permissionless characteristics of decentralised finance, while also incorporating several features traditionally associated with conventional financial markets. This hybrid nature creates a unique market structure, balancing decentralisation with practical centralised coordination mechanisms.

To support its intended functionality, the derivatives marketplace had to satisfy a number of operational and technical requirements. These included the ability to list available contracts, displaying relevant parameters such as expiry dates, strike prices, and underlying assets. In addition, the system had to maintain a record of investor portfolios, automatically exercise contracts upon expiry, and perform contract clearing and settlement without manual intervention.

A further critical requirement was to provide investors with maximum flexibility to define bespoke contracts, supporting a wide range of parameter customisations. All transactions needed to also preserve user anonymity, ensuring that individual identities are not linked to specific trading activities.

A novel element of this architecture lies in how contracts are handled. Unlike conventional exchanges where the central body manages, holds, and executes contracts, this platform allows contracts to exist solely between the creator and the buyer, similar to OTC arrangements. Each contract is executed, cleared, and settled autonomously by its own logic, with minimal reliance on the application backend to support data retrieval and status updates.

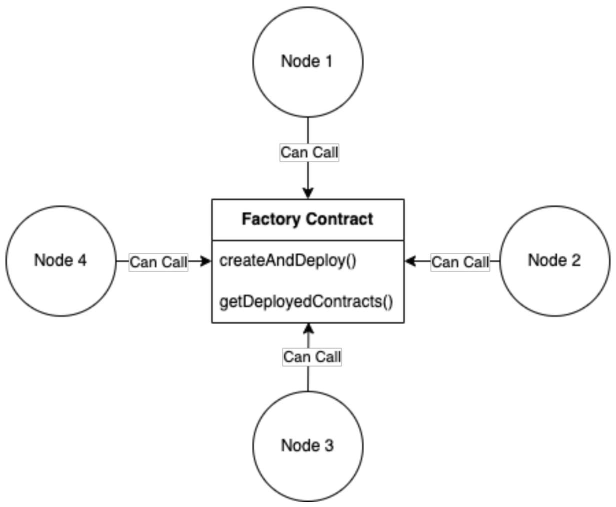

Fig. 2. User access to `Factory` contract.

3.4.1 Central Factory Contract

The system's coordination is facilitated by a central `Factory` contract deployed during network initialisation. This contract plays a crucial role in tracking the lifecycle of all deployed, purchased, and expired contracts. It also provides a single access point through which buyers can browse live contracts and view their portfolios.

An essential design consideration was the need for real-time access to live contract data. Parsing blockchain data to retrieve such information is computationally expensive and time-consuming, making it impractical at scale. To address this, the `Factory` contract creates and deploys individual contracts (referred to as `createBO` contracts), storing their addresses in an on-chain array. This array can be accessed by the off-chain Python application, which retrieves contract addresses and calls each contract's `getStatus` function to obtain their current state. This mechanism enables efficient and reliable data collection for frontend display and analysis.

The choice to centralise the `Factory` contract stems from considerations of efficiency, scalability, and simplicity. By storing all contract addresses in a single location, the system avoids the high costs and complexity of dispersed storage. The resulting architecture forms a dpeth-1 tree rather than a fully connected graph (Fig 2.), significantly reducing gas fees for buyers and minimising network congestion. `Factory` contracts are highly extensible; multiple factories can be deployed on the same network, each dedicated to handling a specific asset class or contract type. This modular approach enables support for a wide range of derivatives—including futures, forwards, and swaps—across diverse underlying assets such as equities, interest rates, commodities, and foreign exchange instruments. The aim of the `Factory` contract was therefore, to act as a repository for specified contract logic and to track the contracts it deploys.

Despite its advantages, the central `Factory` contract presents certain security risks. If compromised, it could

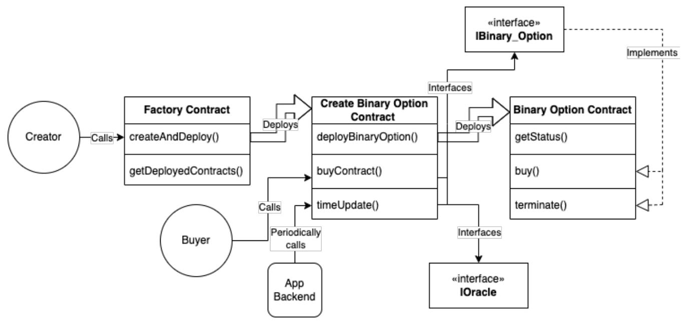

Fig. 3. Web3Py App backend and smart contract UML diagram.

enable malicious modifications to contract variables or allow invalid contracts to be injected into the system. To mitigate such vulnerabilities, the architecture could adopt practices from traditional financial systems. This may involve segmenting the market into distinct time periods [7] or asset classes and replacing the `Factory` contract periodically to minimise systemic risk.

3.5 Python Decentralised App

The decentralised application (dApp) is implemented as a Python command-line interface (CLI) using the `Streamlit` plugin to generate interactive windows that display user portfolios and available contracts. Upon initialisation, the back-end establishes a connection to the Ethereum network running locally. Users can log in as either a Contract Creator or a Contract Buyer; this distinction is primarily for security purposes, as it helps regulate access to specific functionalities. While each role comes with different permissions, users are free to switch between roles at any time. Once logged in, users are granted access to the network's Factory contract. Contract Creators have the ability to deploy new contracts, while Buyers can browse the marketplace for available contracts and purchase them as desired. Both types of users are also able to view their account balances and monitor their current portfolios. After a contract is created, the back-end manages the termination process automatically, ensuring seamless execution of contract lifecycles. Fig 3. illustrates the interoperation between the smart contracts and the App's back-end.

3.6 Testing

After the exchange was fully operational, the `Slither` [26] static analysis tool was used to evaluate the security of the smart contract system, revealing several critical and informational findings. Most notably, multiple **reentrancy vulnerabilities** were detected in functions such as `BinaryOption._terminate()` and `CreateBO.buyContract()`, where external calls were made before updating internal state, violating the checks-effects-interactions pattern [27] and potentially allowing malicious re-entry. Additionally, several functions were found to **transfer Ether to potentially arbitrary addresses** without sufficient access validation, posing a risk of unauthorised fund extraction. Several constructors lacked **zero address validation**, increasing the chance of deploying contracts with invalid or insecure configurations. The analysis also highlighted the absence of **event emissions** for key state changes, which reduced the system's transparency and traceability. Use of **low-level calls** like `.call{value: ...}()` without proper error handling was flagged as dangerous due to their susceptibility to failure and potential misuse. Furthermore, the contracts relied on a Solidity version (`^0.8.20`) that contains known compiler bugs, introducing latent risks. Other minor issues included redundant boolean checks, non-standard naming conventions, and a lack of `immutable` declarations for variables that didn't change over the contract's life.

To mitigate these vulnerabilities, the affected functions were refactored to follow the checks-effects-interactions pattern, ensuring internal state is always updated before any external calls. Access controls were strengthened with additional `require` statements and role-based modifiers to validate address permissions before executing sensitive logic. Constructors were updated to reject zero-address inputs, and critical state changes were made observable through event emissions. These modifications were implemented prior to formal testing, ensuring the system was secure before exposure to simulated adversarial conditions.

3.7 Automated Margining Foundations

Although full implementation was beyond the scope of this project, the advanced objective of laying the foundations for an automated margining system can be meaningfully addressed through the design of a proposed framework. This proposal envisions a robust, decentralised architecture capable of managing dynamic risk exposure and collateral requirements, thereby extending the system's applicability to longer-term and higher-value derivatives.

At the heart of this design is the introduction of individual investor account contracts. These smart contracts function as secure, on-chain user profiles. They continuously track open positions, monitor account balances, and query oracle-provided price data to assess exposure in real time. They would also log the outcomes of past contracts, forming a performance history that contributes to an investor-specific credit score. This score would be used to dynamically adjust the required initial margin when deploying new contracts—rewarding reliable behaviour with reduced collateral requirements and incentivising responsible participation.

In parallel, a central clearing contract would oversee margin compliance across the platform. It would aggregate risk data from all investor accounts, assess exposure on an ongoing basis, and identify contracts most in need of margin top-ups. Where shortfalls are detected, the clearing contract would automatically reallocate funds from the investor's account to the relevant contract. If insufficient funds are available, the system would issue margin call alerts and, if necessary, trigger liquidations or enforce penalties to maintain system integrity. Upon each contract's settlement, credit scores would be updated, reinforcing the feedback loop that ties user behaviour to platform privileges.

This proposed system meets the project's advanced objective by establishing the conceptual and architectural groundwork for future integration of automated margining. It aligns with real-world financial practices while remaining faithful to the decentralised principles underpinning the platform. By clearly defining how such a mechanism could function in practice, this proposal ensures the objective is addressed substantively, even in the absence of full implementation.

4 Results

This section presents the outcomes of testing our trading platform, with a focus on both security and operational efficiency. Given the high-stakes nature of financial smart contracts, the system was subjected to rigorous analysis to verify its robustness against adversarial behaviours, unauthorised access, and improper state manipulation. In parallel, gas usage was carefully measured across key functions to assess the platform's cost efficiency. These results provide a critical evaluation of the platform's readiness for real-world deployment, highlighting its strengths in trustless execution and economic viability, while also identifying key areas for future optimisation.

4.1 Security

The most significant application-level vulnerabilities identified in the system pertain to contract manipulation, unauthorised access to payouts, and the integrity of time-based data. If exploited, these vulnerabilities could lead to serious consequences such as financial loss for users, disruption of service functionality.

To be considered secure against these threats, the system had to meet several key criteria. First, ensuring that the `creator` of a binary option is permanently set upon contract creation and cannot be changed. Similarly, the `buyer` variable in both the `CreateBO` and `BinaryOption` contracts can only be assigned through the `buy` function, and once set, it remains immutable. To protect user funds, neither the creator nor the buyer can withdraw funds before the option has matured, effectively preventing early withdrawals. Additionally, the system is designed so that no third party can drain or access funds from any contract, including the `Factory`, `CreateBO`, or `BinaryOption` contracts. Finally, critical variables such as `currentTime`, `currentAssetPrice`, and `timeDelta` are strictly controlled and can only be updated through the trusted `TimeOracle` contract. These safeguards collectively ensure that only valid participants can interact with Binary Options contract instances, that all actions conform strictly to the encoded logic, and that all external data remains accurate, unbiased, and tamper-proof.

4.1.1 Adversarial Simulation

To rigorously test the system's resilience against malicious behaviour, we developed and deployed a targeted adversarial test suite comprising two key components: a smart contract, `MaliciousValidator.sol`, and its corresponding Python harness, `MaliciousValidator.py`. These components simulated a highly capable adversary with full access to the deployed contracts' addresses, including `CreateBO`, `BinaryOption`, and `TimeOracle`. While such comprehensive knowledge would be difficult for an attacker to acquire in a real-world deployment, this testing framework represents a worst-case scenario, ensuring robust analysis.

The `MaliciousValidator` contract was deployed by an account produced by the locally run `Hardhat` network. It includes five dedicated functions that each attempt to exploit a specific vulnerability: early withdrawal by either party, overwriting the `buyer` variable post-assignment, mutating the `creator` identity, unauthorised draining of funds from any of the contracts, and modifying critical oracle-controlled parameters like `currentTime`, `currentAssetPrice`, or `timeDelta`. Each function emits a `TestResult` event, reporting the test name and whether it passed or failed.

The Python script automated this process by deploying `MaliciousValidator` with references to the live contract addresses and executing each test sequentially. It listens for emitted events to determine outcomes, confirming that all safeguards are working as intended. This testing process affirms that the buyer variable can only be set once via the `buy` function, the creator cannot be changed after initialisation, premature withdrawals are blocked, contract funds are inaccessible to third parties, and oracle-controlled variables cannot be tampered with externally. By demonstrating that none of these attacks succeed, the tests validate the system's security architecture under adversarial conditions.

4.2 Gas Costs and Efficiency

All interactions with the smart contract incur gas fees, which are the native costs associated with computation and storage on the Ethereum blockchain. Gas fees act as rewards to miners for their verification of transactions, compensating for the computational resources they use. These fees are not fixed; they fluctuate based on network congestion and the complexity of the operations being executed. As such, understanding and managing gas fees is essential for both the usability and economic viability of the system.

The main operational costs of the binary options platform are derived from several key actions. These include contract deployment, contract buying (i.e., entering a position), contract clearing and contract settlement. Among these, deployment represents a one-time cost, while the remaining activities—particularly buying and settling contracts—incur recurring expenses each time they are executed. To quantify these costs, gas usage was measured through test-net deployments and a range of simulated user scenarios to reflect real usage patterns.

Maintaining low operational costs is especially important for ensuring user adoption. High gas fees can discourage users from engaging with the platform, particularly those wishing to place small-value trades. If fees become too high, the platform risks excluding average users during periods of heavy network activity, thereby undermining its accessibility and appeal.

Responsibility for covering these gas costs is divided among different participants in the system. The option creator is responsible for paying cost of deploying the contract and setting any custom parameters. The creator also pays for gas fees associated with clearing and the periodic asset price updates. Contract buyers are only liable to pay the gas fees associated with buying a contract.

Table 2 shows the gas usage for the essential exchange functions. Although the gas numbers appear large, this is

TABLE 2
Gas usage and execution time for key contract functions.

Function	Contract	Caller	Gas Used	Time (s)
Deploy Factory	`Factory`	Admin	3,798,349	0.029
Create and Deploy	`Factory`	Creator	3,834,100	0.0564
Buy	`CreateBO`	Buyer	92,869	0.0256
Time Update	`Factory`	Creator	395,194	0.116

expected given that gas measures extremely fine-grained computational steps [28]. In this setup, the contract creator is responsible for paying a total of approximately 4.2 million gas units over the complete lifetime of a contract (3.8 million for creating and deploying an individual `CreateBO` contract and 395,194 for the five time and asset price updates). Contract buyers, on the other hand, incur roughly 93,000 gas units when buying a contract.

It is important to note that the gas usage values presented are from a single example of contract deployment and interaction, not an average over many trials. This is justified because the contracts are highly standardised — each `CreateBO` and `BinaryOption` pair is deployed with identical code and structure, leading to predictable and consistent gas costs across instances. As a result, the measurements shown reliably represent the gas usage a typical user or creator would experience over the full lifetime of a contract.

5 Evaluation

This section critically evaluates the platform across four key dimensions: security, performance, decentralisation, and commercial viability. Each dimension is analysed through the lens of both technical implementation and real-world applicability, with the aim of identifying the system's strengths, limitations, and areas for future improvement. The evaluation draws on quantitative results, such as gas fee benchmarking against traditional exchanges, as well as qualitative assessments rooted in best practices for smart contract development and decentralised system design. Additionally, a reflection on the software engineering methodology employed throughout the project is provided to contextualise the technical outcomes within the broader development process.

5.1 Security

The security testing methodology involved a combination of static analysis and adversarial simulation, providing a thorough evaluation of the system's robustness. The use of the Slither static analysis tool successfully identified a range of vulnerabilities early in the development cycle, including reentrancy risks, access control weaknesses, and insufficient validation checks [29]. These findings prompted significant refactoring, leading to the adoption of best practices such as the checks-effects-interactions pattern [27], strengthened access control, and the addition of event emissions for key state changes. Following these improvements, a dedicated adversarial test suite was deployed, simulating malicious behaviours under a worst-case assumption of full contract address knowledge. The adversarial tests confirmed that critical security properties—such as the immutability of creator and buyer assignments, the protection against premature or unauthorised withdrawals, and the integrity of time and price data via the time and asset price oracle were properly enforced.

However, the methodology is not without its flaws. First, testing was conducted exclusively on a locally hosted `Hardhat` network, which cannot fully replicate real-world conditions such as fluctuating gas prices, network congestion, or miner manipulation of timestamps [30]. Additionally, while the adversarial tests covered key logical attacks, they did not simulate denial-of-service (DoS) attacks, frontend manipulation, or subtle race conditions that could emerge under production-scale load.

Several assumptions underpin the security evaluation. Primarily, it is assumed that the TimeOracle will only be updated by trusted parties and that the data it provides is accurate and timely. It is also assumed that the `Hardhat` network's simulation of Ethereum behavior is a sufficiently accurate stand-in for production conditions. Moreover, the testing assumes that contract addresses remain public knowledge and that malicious users will have full visibility into deployed contracts—representing a strong adversarial model.

These assumptions are generally realistic within the context of decentralised finance. Trust in oracles is a common industry-wide challenge [31], and mitigation techniques such as multi-source aggregation or trusted update permissions were partially implemented. Relying on `Hardhat` for functional correctness is acceptable at the development stage, although future evaluations would benefit from public test-net deployments to better capture environmental risks. While the absence of formal verification reduces the level of absolute certainty, the combination of static analysis and adversarial testing offers a high degree of practical confidence in the system's security under its stated assumptions.

Overall, the security methodology is robust relative to the project's scale and scope, effectively mitigating major vulnerabilities while identifying areas for future hardening—particularly around oracle trust assumptions and resilience to environmental risks.

5.2 Performance

5.2.1 Fee Structures on Traditional Derivatives Exchanges

To establish a baseline for evaluating the cost efficiency of the proposed blockchain-based exchange, it is necessary to examine the transaction fees charged by traditional financial derivatives exchanges. These fees are typically assessed on a per-contract, per-side basis and may vary depending on the product type, user classification (retail or institutional), and trade volume.

The IG Group, one of the largest derivatives exchanges in the UK, charges $1 per contract on Options on US stocks and Exchange Traded Funds. These fees cover all brokerage and clearing costs [32]. Eurex, the primary European derivatives exchange, offers slightly lower fees for retail accounts, often between €0.20 and €0.50 per contract, with volume-based discounts available to institutional participants [33]. The

Chicago Board Options Exchange (CBOE) presents a more complex structure, where equity and index options typically incur \$0.40 to \$0.65 per contract per side for retail customers, depending on the order type, routing, and whether the user is classified as a professional trader [34].

While the precise structure and magnitude of these fees vary, a typical retail user can expect to pay around \$1.00 per complete trade (round-trip), excluding broker commissions. These fees form a practical benchmark for comparing the operational costs of our decentralised platforms.

5.2.2 Retail Trade size Assumptions

The intended users of this platform are retail investors, individuals trading relatively small contract sizes compared to institutional participants. To benchmark the cost-effectiveness of the system for this demographic, it is necessary to establish realistic expectations for trade size and behaviour.

Empirical data from the U.S. options market reveals that retail investors frequently engage in micro-trading. According to [35], approximately 41.6% of all retail options trades during their study period had a notional value below \$250, and nearly 60% were under \$500. These trades predominantly involved short-dated contracts, with nearly half of retail volume concentrated in weekly options. This pattern suggests that retail investors favour low-cost, high-leverage positions and are less concerned with holding positions over extended periods.

Given these findings, we assumed a typical notional trade size for a retail user falls within the range of \$100 to \$2,000. This range captures the vast majority of observed behaviour in traditional markets and aligns with the types of speculative, short-term trades that binary options contracts naturally accommodate. It also reflects the financial limits and risk appetite of retail traders.

These assumptions are critical for evaluating the practical viability of the blockchain-based system. If gas costs per trade exceed a few dollars, the system risks becoming uneconomical for this user segment especially when traditional platforms can internalise or subsidise such costs through payment for order flow. Thus, the \$100–\$2,000 notional range provides a grounded basis for cost comparisons and helps define the thresholds at which blockchain-based options trading becomes competitive.

5.2.3 Defining Cost Benchmarks

To ensure consistency, comparisons were carried out in terms of fee rates on contracts denominated in US Dollars. The rates charged to investors on traditional exchanges were calculated by (1), where f represents our assumed traditional exchange fee (\$1) and c represents the contract cost.

$$r = 100f/c \tag{1}$$

Rates charged to investors on the proposed exchange were calculated by (2), where g represents the gas used and k represents the gas fee.

$$r = 100(g * k)/c \tag{2}$$

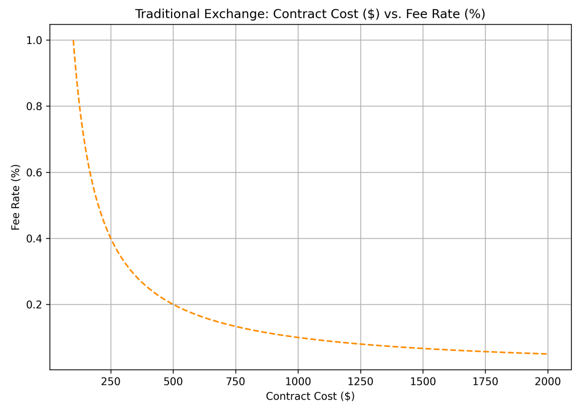

Fig. 4. Fee scheme for a traditional Derivatives exchange constructed using Equation (1) by substituting \$1 for f and computing r for values of $c \in$ [\$100, \$2,000].

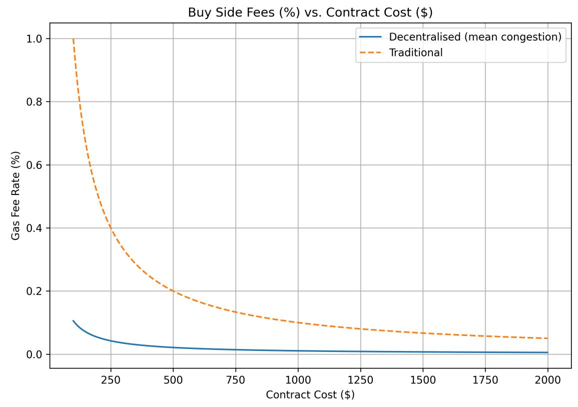

Fig. 5. Percentage fee incurred by contract buyers on our exchange compared to traditional exchanges for contracts priced between \$100 and \$2,000.

Fig 4. uses Equation (1) to plot our approximation of typical fees charged to investors on either side of a contract by traditional derivatives exchanges.

5.2.4 Breakeven Analysis

To determine the viability of the blockchain-based platform under real market conditions, a breakeven analysis was performed by comparing the percentage fee rate incurred by users against traditional exchange benchmarks. Using six months of historical Ethereum gas price data, transaction costs were computed for both the buyer and the creator under varying contract sizes. These were plotted alongside fee estimates from traditional exchanges, such as IG Group.

The analysis revealed that for buyers, the decentralised platform consistently outperformed traditional exchanges across the full range of contract costs tested (\$100–\$2000), with gas fees typically remaining below 0.1% of the notional value. For creators (sellers), however, fees charged only fall into the benchmark range for contracts costing \$500 or more.

To compute the fee curves that the buy side (Fig. 5) and sell side (Fig. 6) face respectively, Equation (2) was used. In both cases, k was substituted with the 180-day

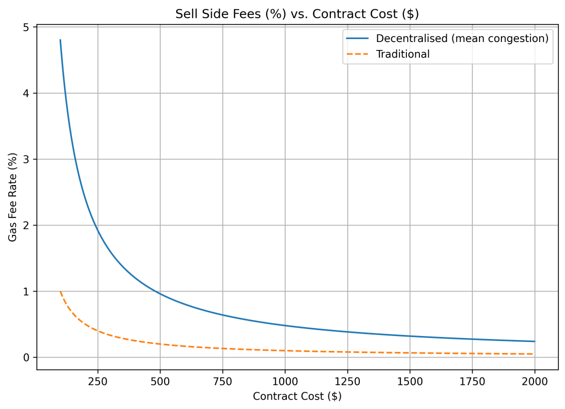

Fig. 6. Percentage fee incurred by contract sellers on our exchange compared to traditional exchanges for contracts priced between $100 and $2,000.

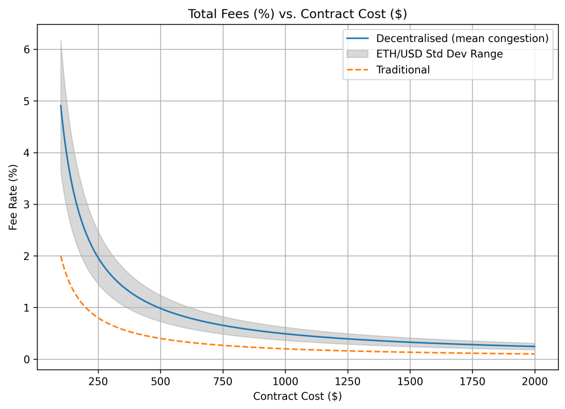

Fig. 7. Total percentage fees charged to buyers and sellers on our exchange compared to traditional exchanges for contracts priced between $100 and $2,000.

mean gas price on the Ethereum main-net [36]. The curves labelled *"Traditional"* in Figs. 5 and 6 are constructed in the same way as the curve shown in Fig. 4, enabling a direct comparison to traditional exchange fees. To facilitate this comparison, the mean gas price, originally measured in wei, was first converted to Ether and then to USD, assuming a fixed exchange rate of $2,780 per Ether [37], consistent with the historical average during the observation window. For the buy-side curve, g was replaced by the buyer's total gas usage, as outlined in Table 2. Similarly, the sell-side curve was computed using the creator's total gas usage from Table 2. This approach ensures that fee rates for decentralised transactions are expressed in units (percentage of contract notional) directly comparable to those charged by traditional financial intermediaries. It is important to note that the exchange rate between USD and ETH plays a significant role in this calculation; when ETH is cheap, the system, as currently implemented, becomes less expensive to use. The fee uncertainty due to volatile exchange rates further motivates the use of a dollar-pegged stablecoin as the exchange currency in any future production versions of this technology.

While the breakeven analysis reveals a significant difference in the percentage fee burden between buyers and creators, this disparity is largely structural rather than inefficiency-driven. The elevated gas cost incurred by creators arises for two main reasons. First, the platform's architecture enforces automated clearing logic at the time of contract deployment. Since the buyer is not yet defined during deployment, the creator must assume responsibility for initiating the contract's lifecycle, including registering it with oracles and setting up the on-chain state necessary for future settlement. Second, creators bear the additional burden of paying gas fees associated with deploying the `CreateBO` and `BinaryOption`.

Despite this disparity, the imbalance can be mitigated through dynamic pricing strategies. In particular, the contract price (paid by the buyer) can be structured to reflect both the risk premium associated with the creator's position and a proportional share of operational costs. One approach would be for the contract price to include sufficient stablecoin liquidity such that buyers indirectly contribute to 50% of the total gas fees. This solution relies on contract creators to be sufficiently liquid in case their contract is bought late in its lifecycle.

Fig. 7 indicates that, in aggregate, users of the proposed exchange can expect to incur fees approximately two to three times higher than those faced by participants on traditional financial exchanges. It should be noted, however, that this comparison would benefit from additional data regarding fee structures on existing Binary Options platforms. At present, there are no active Binary Options exchanges, limiting the precision of the comparative analysis. Furthermore, it is plausible that users with lower risk tolerances may be willing to accept higher transaction costs in exchange for the elimination of counterparty risk. A more detailed economic analysis is required to determine whether the risk mitigation benefits provided by the decentralised framework sufficiently compensate for the higher fee burden observed.

5.3 Decentralisation

The platform exhibits a mixed decentralisation profile when evaluated against the framework proposed by Buterin [6], which distinguishes between architectural, political, and logical decentralisation.

Architecturally, the system is centralised in its deployment phase—an administrator is required to deploy and maintain the Factory contract, which serves as the coordination point for all contract creation. While this introduces a single point of dependency, it also allows for greater legal oversight, and fault tolerance can be improved by rotating administrative responsibilities among a set of trusted participants.

Politically, the platform is designed to be decentralised. It is intended for use by individuals rather than institutions, and each participant interacts with the system from an independent position without privileged control. This promotes openness and reduces the risk of collusion or censorship. However, political decentralisation is inherently empirical—it depends not just on the architecture of the system, but on how it is adopted and used in practice. True decentralisation in this dimension will be better assessed once the platform is live and its user base begins to grow. Only then can it be determined whether control and influence remain widely distributed or become concentrated among a small subset of users.

Logically, however, the system is centralised: all contracts conform to a fixed protocol and operate through a unified structure, meaning that while users can freely enter and exit the platform, the underlying logic remains singular and non-distributed in nature. This centralisation of application logic enhances usability and ensures coherent contract execution, but may limit flexibility or composability in more heterogeneous ecosystems.

5.4 Commercial Viability

The platform demonstrates strong potential for commercial viability, particularly within retail-focused markets. However, our findings indicate that users can expect to incur fees approximately two to three times higher than those faced by participants on traditional derivatives exchanges under average conditions (Fig. 7). Although this represents a premium over centralised platforms, it must be weighed against the platform's unique value propositions, including trustless settlement, openness, and elimination of counterparty risk.

The trustless execution model ensures that once a contract is created, the seller cannot retract or dispute the payout. This provides buyers with a high degree of assurance that they will receive their winnings if the contract settles in their favor—an assurance that was not always available in retail Binary Options markets prior to 2019 regulatory reforms [11]. Furthermore, the platform's open architecture ensures that no contract creator can exclude others from participation, enhancing market fairness and accessibility.

Another notable strength lies in the platform's decision to forgo margining. While this restricts the platform to fully-collateralised instruments and limits exposure to complex products such as leveraged options, it simultaneously eliminates the risk of defaults, margin calls, or cascading liquidations. Users can never lose more than they initially commit, making the system particularly attractive to risk-averse or inexperienced traders. However, this benefit imposes a significant opportunity cost on sellers, who must deposit the entire potential payout at the time of contract creation, thereby reducing capital efficiency compared to margin-based systems.

A key challenge is the sensitivity of transaction costs to Ethereum network conditions and ETH/USD exchange rate fluctuations. As shown in Fig. 8, during periods of elevated network congestion, gas fees may spike substantially, making small trades uneconomical. Additionally, because fees are denominated in ETH, users are exposed to exchange rate volatility, leading to unpredictable transaction costs in fiat terms. This volatility strengthens the case for employing dollar-pegged stablecoins in future iterations of the platform to stabilize operational costs.

Despite these limitations, the platform remains viable for targeted retail use cases, particularly for contract sizes above $500 where fee rates are more competitive relative to traditional exchanges. Strategic improvements—such as stablecoin integration, deployment on Layer 2 networks, or dynamic fee-sharing models between buyers and creators—could further enhance its commercial appeal and competitiveness.

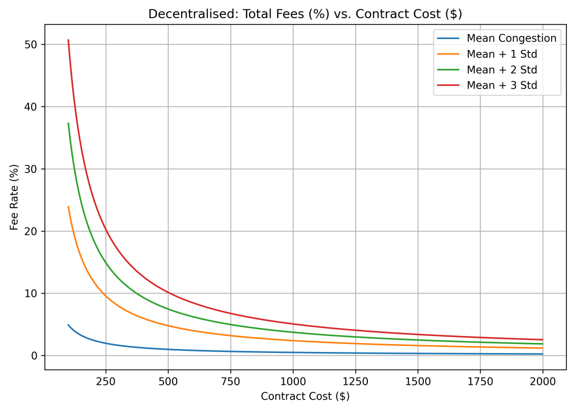

Fig. 8. Total fee rate charged at mean gas price to mean plus three standard deviations.

5.5 Project Management

The development of the platform followed an agile methodology [38], with an iterative approach that allowed for continuous refinement of both functionality and design. The project was structured around clearly defined sprints, each focused on implementing a specific subset of features, such as contract creation logic, oracle integration, or gas cost optimisation. Regular reassessment of priorities ensured that the implementation remained flexible in response to design discoveries and testing outcomes. For example, an early plan to use Solidity's native `block.timestamp` for expiry logic was later replaced with a custom `TimeOracle` contract after recognising the security risks of miner-controlled timestamps. This pivot reflects the project's commitment to secure and maintainable design, even when it required reworking core components.

Version control was used extensively throughout development via Git, enabling a clear record of changes and safe experimentation across branches. This allowed for parallel development and testing of different architectural approaches without risking the stability of the main codebase. Changes were tracked and documented, facilitating traceability and collaboration, even in a solo-development context.

A key design decision early in the project was to use `Web3Py` instead of the more commonly adopted `Web3.js` library for interacting with the blockchain. This choice aligned with the developer's Python background, allowing faster progress on back-end functionality, even at the cost of a more sophisticated user interface. While this trade-off led to a minimalistic front end—initially developed as a command-line interface—the reduced overhead enabled earlier testing by inspection and a sharper focus on smart contract functionality. A basic `Streamlit` interface was later introduced to display user portfolios and contract listings, though the full migration of the front end to `Streamlit` could not be completed due to time constraints.

Smart contract development began with a purely functional approach, prioritising logic correctness and operational flow. As the platform matured, additional security features were introduced in response to testing outcomes and recognised best practices. Built-in and custom access modifiers were progressively added to restrict critical functions to authorised users. 'require' statements were placed at potential failure points to provide explicit, contextualised error messages—mitigating the common issue of opaque Solidity revert errors. On the front end, 'try–except' blocks were added to capture and handle transaction failures without crashing the application, improving user feedback and fault tolerance. Over time, contracts were hardened against malicious behaviour with Solidity's standard security mechanisms [39], including reentrancy guards and third-party access restrictions.

This gradual layering of robustness reflects the project's evolutionary design philosophy. Initial functionality was delivered early and refined incrementally, allowing for an agile response to both functional and security-related challenges. Collectively, the software engineering process prioritised stability, transparency, and practical trade-offs between implementation depth and user experience.

6 Conclusion

This project successfully demonstrated the feasibility of using blockchain and smart contracts to create a trustless, decentralised binary options trading platform. Through careful smart contract design, access control, and oracle integration, the system achieved its primary objective of providing guaranteed, immutable settlement without reliance on intermediaries. Security was rigorously validated through static analysis and adversarial testing, ensuring robust protection against manipulation, unauthorised access, and early withdrawals. From an architectural perspective, the platform balances decentralised settlement with a centralised coordination layer via the Factory contract, enabling operational efficiency while preserving political decentralisation among participants.

Performance evaluation revealed that for typical retail trade sizes (\$100–\$2,000), buyer-side transaction fees remained highly competitive, consistently under 0.1% of notional value—outperforming traditional brokers in most cases. However, seller-side costs were higher, particularly for smaller contracts, due to the architectural burden of contract deployment and oracle registration. In aggregate, users are expected to pay approximately two to three times the transaction fees incurred on traditional centralised platforms, reflecting both gas costs and Ethereum's network constraints. This fee premium, although non-trivial, must be weighed against the platform's elimination of counterparty risk—a benefit that may justify higher costs for risk-averse or security-conscious users.

Cost efficiency is strongest in mid-sized contract ranges (above \$500 notional), where gas costs become proportionally lower, and is highly sensitive to Ethereum network conditions and ETH/USD price volatility. These findings reinforce the importance of stablecoin adoption and Layer 2 scaling solutions in future iterations, to further reduce cost unpredictability and enhance user accessibility.

In comparison to contemporary efforts reviewed in the literature, this project adopts a simpler, more targeted approach. While Fries et al. [15] and Casey-Fierro [16] propose comprehensive smart contract systems aligned with institutional frameworks such as ISDA standards, this work focuses instead on retail users and short-dated, fully collateralised binary options. Whereas previous prototypes often rely on partially centralised off-chain services for valuation or event triggers, this project prioritises fully on-chain trust minimisation wherever feasible, albeit at the cost of high exchange fees. This trade-off reflects a deliberate design choice to maximise user protection in contexts where traditional broker trust has historically been lacking.

Overall, this project provides strong evidence that decentralised smart contract-based infrastructure can serve as a secure and economically viable alternative to traditional binary options markets. While challenges remain in cost optimisation and user experience refinement, the results affirm the broader potential for decentralised finance to reshape retail trading ecosystems, delivering transparency, fairness, and resilience through technology rather than trust.

While this project demonstrates a functional and secure decentralised binary options trading platform, there are several key areas for future development. A major improvement would be the implementation of a more user-friendly, web-based interface, ideally with wallet integration via tools such as MetaMask [40] to streamline user interaction and improve accessibility. Expanding the range of supported underlying assets is another important step; future versions could accommodate derivatives such as options, swaps, and futures based on cryptocurrencies, interest rates, exchange rates, and even tokenised government and corporate bonds.

There is scope for deeper legal integration. While the current system encodes basic enforceability, future work could focus on embedding regulatory frameworks directly into smart contract logic, improving transparency and auditability for regulators. Architecturally, decentralisation could be further enhanced by rotating administrative control or introducing governance mechanisms, such as voting among participants, to distribute trust and minimise reliance on a single operator.

Building on the core system, the platform also provides a strong foundation for extending support to a wider variety of derivative instruments. One natural progression would be the implementation of Ethereum-based vanilla options, futures, and forwards. These instruments would follow similar lifecycle mechanics but would require more sophisticated pricing models and margining structures, particularly for forwards and futures that settle against future spot prices. Beyond standard contracts, the platform could also support exotic options such as barrier, compound, or lookback options. Implementing such instruments would require expanded oracle functionality and more flexible contract templates but would significantly increase the platform's utility, particularly for sophisticated retail users.

A further extension involves developing modularised smart derivatives that emulate over-the-counter (OTC) structures, allowing counterparties to define all contract terms—including strike logic, expiry mechanics, and custom settlement rules—directly through the user interface. This would make the system highly flexible and composable, broadening its applicability across diverse market needs. Finally, as blockchain-based equity exchanges continue to emerge, the platform could evolve to support equity-linked derivatives, enabling users to trade options and structured products on tokenised stocks. This would open the door to decentralised equity risk management and create new speculative opportunities within the broader DeFi ecosystem.

References

[1] F. C. Authority, *Binary options scams*, FCA, Mar. 2023. [Online]. Available: https://www.fca.org.uk/consumers/binary-options-scams (visited on 10/17/2024).

[2] K. Ito, M. Mita, S. Ohsawa, and H. Tanaka, "What is stablecoin?: A survey on its mechanism and potential as decentralized payment systems," *International Journal of Service and Knowledge Management*, vol. 4, pp. 71–86, 2020. DOI: 10.52731/ijskm.v4.i2.574. (visited on 10/17/2024).

[3] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system satoshi nakamoto*, cryptovest, Oct. 2008. [Online]. Available: www.crytovest.co.uk (visited on 10/17/2024).

[4] C. Smith, *Erc-20 token standard*, ethereum.org, Jun. 2024. [Online]. Available: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/ (visited on 10/28/2024).

[5] IBM, *What are smart contracts on blockchain?* IBM, 2024. [Online]. Available: https://www.ibm.com/topics/smart-contracts (visited on 10/20/2024).

[6] V. Buterin, *The meaning of decentralization*, Medium, Feb. 2017. [Online]. Available: https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274.

[7] J. C. Hull, *Options, futures, and other derivatives*. Pearson, Prentice Hall, 1993.

[8] Clearmatics, *Understanding derivatives trading (post 1): The basics*, Medium, Jul. 2021. [Online]. Available: https://medium.com/clearmatics/understanding-derivatives-trading-post-1-the-basics-bd28f235219d (visited on 03/31/2025).

[9] I. Swaps and D. Association, *Derivatives, margining and risk in emerging market and developing economies*, isda.org, Oct. 2024. [Online]. Available: https://www.isda.org/2024/10/08/derivatives-margining-and-risk-in-emerging-market-and-developing-economies/#:~:text=One%20of%20the%20most%20important,or%20margin%2C%20for%20derivatives%20transactions. (visited on 04/23/2025).

[10] Clearmatics, *Understanding derivatives trading (post 2): A glimpse at derivative market structures*, Medium, Aug. 2021. [Online]. Available: https://medium.com/clearmatics/understanding-derivatives-trading-post-2-a-glimpse-at-derivative-market-structures-45498d639299 (visited on 03/31/2025).

[11] F. C. A., *Fca confirms permanent ban on the sale of binary options to retail consumers*, FCA, Mar. 2019. [Online]. Available: https://www.fca.org.uk/news/statements/fca-confirms-permanent-ban-sale-binary-options-retail-consumers (visited on 10/17/2024).

[12] V. V. BHANDARKAR, A. A. BHANDARKAR, and A. SHIVA, "Digital stocks using blockchain technology the possible future of stocks?" *INTERNATIONAL JOURNAL OF MANAGEMENT*, vol. 10, Jun. 2019. DOI: 10.34218/ijm.10.3.2019/005.

[13] K. John, L. Kogan, and F. Saleh, "Smart contracts and decentralized finance," *Annual Review of Financial Economics*, vol. 15, 2023. DOI: 10.1146/annurev-financial-110921-. (visited on 10/27/2024).

[14] D. A. Zetzsche, D. W. Arner, and R. P. Buckley, "Decentralized finance," *Journal of Financial Regulation*, vol. 6, pp. 172–203, Sep. 2020. DOI: 10.1093/jfr/fjaa010. (visited on 10/27/2024).

[15] C. P. Fries, P. Kohl-Landgraf, B. Paffen, *et al.*, "Implementing a financial derivative as smart contract," *SSRN Electronic Journal*, 2019. DOI: 10.2139/ssrn.3342785.

[16] F. Casey-Fierro, *Smart confirmation contracts: An architecture for isda smart contracts author*, Sep. 2023. (visited on 10/27/2024).

[17] I. Swaps and D. A. (ISDA), *Isda legal guidelines for smart derivatives contracts: Equity derivatives 2*, Oct. 2018. (visited on 10/27/2024).

[18] M. C. V. ViswasReddy, V. Sai, G. J. Rao, and N. Devi, "Efficient token transfer using erc-20 decentralized exchange," *International Conference on Intelligent Systems for Cybersecurity (ISCS)*, May 2024. DOI: 10.1109/iscs61804.2024.10581224. (visited on 10/27/2024).

[19] I. Swaps and D. A. (ISDA), *What is the isda cdm?* 2018. [Online]. Available: https://www.isda.org/a/z8AEE/ISDA-CDM-Factsheet.pdf (visited on 10/27/2024).

[20] T. S. Authors, *Solidity — solidity 0.8.29 documentation*, Solidity-lang.org, 2016. [Online]. Available: https://docs.soliditylang.org/en/v0.8.29/ (visited on 04/24/2025).

[21] T. E. Foundation, *Overview — web3.py 7.2.0 documentation*, web3py.io, 2016. [Online]. Available: https://web3py.readthedocs.io/en/stable/overview.html (visited on 04/23/2025).

[22] N. Foundation, *Ethereum development environment for professionals by nomic labs*, hardhat.org, 2024. [Online]. Available: https://hardhat.org/ (visited on 10/28/2024).

[23] G. C. Web3, *Google cloud web3 portal*, Google.com, 2024. [Online]. Available: https://cloud.google.com/application/web3/faucet/ethereum/sepolia (visited on 04/24/2025).

[24] Ethereum, *Ethereum (eth) blockchain explorer*, Ethereum (ETH) Blockchain Explorer, 2019. [Online]. Available: https://etherscan.io/ (visited on 04/24/2025).

[25] P. Tantikul and S. Ngamsuriyaroj, "Exploring vulnerabilities in solidity smart contract," *Proceedings of the 6th International Conference on Information Systems Security and Privacy*, pp. 317–324, 2020. DOI: 10.5220/0008909803170324. [Online]. Available: https://pdfs.semanticscholar.org/1f61/143148ac4c0bc62b59ab3410333702d407f6.pdf (visited on 04/23/2025).

[26] B. Samuels, *Slither code repository*, GitHub, Jun. 2021. [Online]. Available: https://github.com/crytic/slither (visited on 04/24/2025).

[27] Fravoll, *Solidity patterns: Checks-effects-interactions pattern*, Fravoll.github.io, 2020. [Online]. Available: https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html (visited on 04/26/2025).

[28] G. Wood, *Ethereum: A secure decentralised generalised transaction ledger*, Ethereum Project, 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf (visited on 04/26/2025).

[29] J. Feist, G. Grieco, and T. Grobert, "Slither: A static analysis framework for smart contracts," in *Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, ACM, 2019, pp. 8–15. DOI: 10.1145/3329989.3330051.

[30] C. Diligence, *Timestamp dependence: Ethereum smart contract security best practices*, ConsenSys, 2020. [Online]. Available: https://consensys.github.io/smart-contract-best-practices/known_attacks/#timestamp-dependence (visited on 04/26/2025).

[31] S. Nazarov and S. Ellis, *Chainlink: A decentralized oracle network*, SmartContract.com, 2017. [Online]. Available: https://link.smartcontract.com/whitepaper (visited on 04/26/2025).

[32] IG Group, *Costs and charges*, Accessed April 2025, 2025. [Online]. Available: https://www.ig.com/uk/charges#spreads.

[33] Eurex Exchange, *Fee schedule*, Accessed April 2025, 2024. [Online]. Available: https://www.eurex.com/ec-en/rules-regs/rules-and-regulations/3.-Price-List-32640.

[34] CBOE Global Markets, *Us options fee schedule*, Accessed April 2025, 2024. [Online]. Available: https://www.cboe.com/us/options/membership/fee_schedule/cboe/.

[35] S. Bryzgalova, A. Pavlova, and N. Sikorskaya, "Retail trading in options and the rise of the big three wholesalers," *The Journal of Finance*, vol. 78, no. 5, pp. 2301–2344, 2023. DOI: 10.1111/jofi.13253. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1111/jofi.13285.

[36] Etherscan, *Ethereum average gas price chart — etherscan*, Ethereum (ETH) Blockchain Explorer, 2025. [Online]. Available: https://etherscan.io/chart/gasprice (visited on 04/17/2025).

[37] G. Finance, *Ether (eth) price, real-time quote & news - google finance*, www.google.com, 2025. [Online]. Available: https://www.google.com/finance/quote/ETH-USD (visited on 04/26/2025).

[38] S. Laoyan, *What is agile methodology? (a beginner's guide)*, Asana, Feb. 2025. [Online]. Available: https://asana.com/resources/agile-methodology.

[39] F. Kochan, *Solidity and smart contract vulnerabilities on ethereum — quicknode*, Quicknode.com, Mar. 2025. [Online]. Available: https://www.quicknode.com/guides/ethereum-development/smart-contracts/common-solidity-vulnerabilities-on-ethereum (visited on 04/22/2025).

[40] MetaMask, metamask.io, 2024. [Online]. Available: https://metamask.io/sdk/ (visited on 10/28/2024).