GitStake: A GitHub DApp with PoS based Open-Source Contributions, Incentivization using ERC20 tokens (GitStakeTokens – GST)

Aditi Narkar[1], Tufayl Dalvi [1], Aarya Paradkar[1], Manas Mishra[1], Harsh Poojary[1], Paras Mathpati[1] and Preeti Patil[2]

[1]UG Scholar, Terna Engineering College, Nerul Navi Mumbai, Maharashtra 400706, India
[2]Assistant Professor, Terna Engineering College, Nerul Navi Mumbai, Maharashtra 400706, India

Abstract. This paper addresses a decentralized platform designed to revolutionize open-source contributions by leveraging ERC20 tokens, Decentralized Finance (DeFi), and a Proof-of-Stake (PoS) mechanism for pull request management. The platform allows contributors to stake tokens on issues, with issue creators setting prizes for successful resolutions. The PoS mechanism prioritizes pull requests based on stake amounts, ensuring critical contributions receive the necessary focus. GitStake integrates DeFi to create a transparent, trustless incentivization system, promoting a fair and efficient reward distribution model for open-source development. GitStake also features an initial token distribution model, where contributors receive free tokens initially, followed by an ICO system for further token generation and user engagement. This approach fosters global, decentralized collaboration while ensuring the sustainability and growth of open-source projects.

Keywords: Blockchain, open-source contributions, staking mechanism, ERC20 tokens, GitHub API integration, DeFi, supply-demand dynamic.

1 Introduction

Open-source software development has enabled global collaboration and innovation, but traditional models often fail to sufficiently incentivize developers. This leads to challenges in sustaining engagement, maintaining quality, and ensuring fair compensation [1,3,4]. While platforms like GitHub provide collaboration tools, they lack decentralized reward systems that motivate contributors and ensure transparent compensation [2,5].

This research proposes GitStake, a decentralized platform designed to enhance the open-source contribution ecosystem by leveraging blockchain technologies, DeFi,

and a PoS mechanism [4,6,7]. GitStake incentivizes contributors by allowing them to stake ERC20 tokens on open issues within repositories, with issue creators setting bounties or prizes for resolving them [1,2]. Unlike traditional methods, this decentralized approach ensures that contributors are financially rewarded based on their staked amount, efforts and the importance of their contributions [3,5]. The platform's PoS system prioritizes pull requests by staking amounts, ensuring that high-value contributions are addressed more promptly [5].

GitStake also features an initial distribution of free tokens, followed by an ICO for further token generation, fostering long-term participation and growth. Integration with the GitHub API enables seamless interaction with existing repositories, allowing contributors to continue using familiar tools [7]. By decentralizing and incentivizing contributions, GitStake addresses key challenges in open-source development, providing a transparent and sustainable model for reward distribution [8].

2 System Flow

Fig. 1. ERC20 Token – GitStakeToken (GST) Contract and GithubStaking Contract Deployment

Fig. 2. GitHub User and Wallet (e.g., Meta mask) Initialization

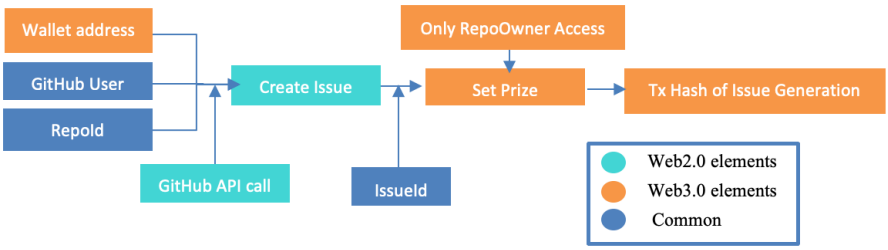

Fig. 3. Issue Creation on a Repository

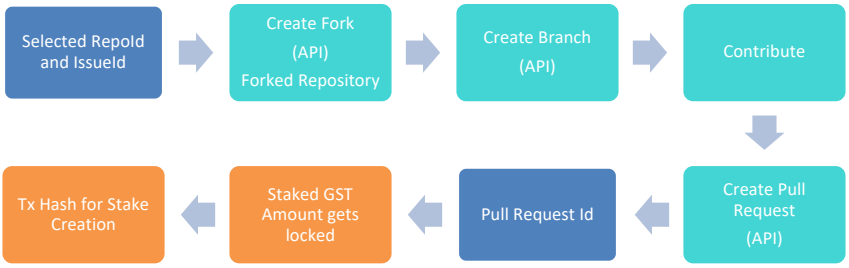

Fig. 4. Staking on Issue

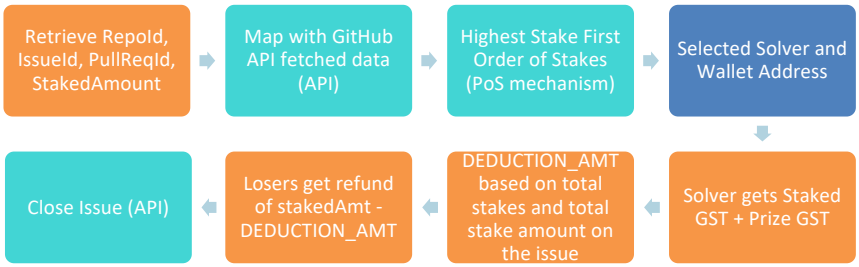

Fig. 5. Selecting a Solver on Issue

3 Contract Implementation

GitStakeToken contract is an ERC20 token which has name 'GitStakeToken' and symbol 'GST'. It is deployed by owner with initial supply as shown in Fig. 3.

The GitHubStaking contract is designed to incentivize open-source contributions by allowing users to create issues on GitHub repositories, stake tokens on these issues, and reward solvers who successfully address them as shown in Fig 3, 4 and 5. Its key functionalities include:

3.1 Core Components:

- Stake: Represents a stake made by a user (`staker`) on a GitHub pull request (`pullReqId`), with an associated amount (`amt`).
- Issue: Represents a GitHub issue, with fields for the creator's wallet address (`address`), prize amount (`amt`), status (`solved`), address (`solver`), total stakers on the issue (`stakeCount`), total staked GST on the issue (`totalStakeAmt`) and a mapping of stakes (`stakes`).
- Repo: Contains a mapping of issue IDs to `` `Issue` `` structs, representing a GitHub repository with multiple issues and `issueCount`.

- WalletStats: Stores statistics related to a user's staking performance, such as `lostStakeCount, wonStakeCount, totalAmtStaked , openAmtStaked, totalStakes, openStakes, rewardsEarned, lost_refund` and `withdraw_proceedings`.
- IssueStats: Tracks statistics for issues created by the user, including `totalPriceAmt_SetByMe, openPriceAmt_SetByMe, totalIssues_SetByMe` and `openIssues_SetByMe`.

3.2 Current Price of GST:

The price of the token dynamically increases based on the number of tokens that have already been sold.

eq.1. `CurrentPrice = RATE + k ×(SOLD_TOKENS) ^2`

- `RATE`: In eq. 1, `RATE` represents the base price of the token. It is a constant or predefined value set in the contract, which forms the initial price when no tokens have been sold.
- `k`: This is a constant multiplier that determines how quickly the price increases as more tokens are sold in eq. 1. It's part of the price adjustment mechanism.
- `SOLD_TOKENS`: This variable keeps track of the total number of tokens sold. As more tokens are sold, this value increases, and the price of future tokens goes up accordingly.
- In eq. 1, the quadratic term `SOLD_TOKENS^2` means that the price grows more rapidly as more tokens are sold. It provides a supply-demand dynamic where earlier buyers get a lower price, and later buyers pay more as shown in Fig. 6.

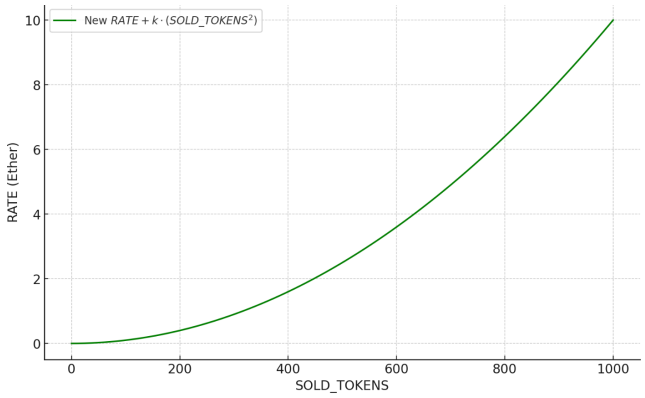

Fig. 6. Dynamic Token Rate Based on `SOLD_TOKENS` (Bonding Curve Model)

3.3 Free Token Request

During the initial token distribution phase (airdropping), users are allowed to request a set number of free tokens to encourage participation and provide an incentive to early adopters. The function ensures that the total number of free tokens distributed remains within the pre-defined limit. Once the distribution phase ends, users can no longer request free tokens and must purchase them through the ICO.

3.4 Buy / Request Token

It implements Initial Coin Offering (ICO) mechanism. The number of tokens a user receives is determined by the current token price (from `getCurrentPrice()`) and the amount of Ether they send. It dynamically adjusts the token price based on the number of tokens sold and ensures that each user receives the correct number of tokens based on the amount of ETH they send as shown in Fig 6. The quadratic price function incentivizes early participation.

eq.2. `tokenAmount = ethAmt / tokenPrice`

The eq. 2 calculates the number of tokens (`tokenAmount`) the user will receive by dividing the ETH they sent (`ethAmt`) by the current price of the token (`tokenPrice`).

`SOLD_TOKENS`: This variable is incremented by `tokenAmount` just sold.
`TOTAL_RAISED`: This variable is incremented by the amount of ETH sent by the user (`ethAmt`), tracking the total amount of ETH the contract has raised through token sales.

3.5 Selling Tokens:

The selling price of tokens is based on the current price of the tokens, which is dynamically adjusted, similar to the token purchase process as shown in Fig. 6. The logic for liquidating tokens must ensure that users can only sell tokens they have purchased and not the free tokens they initially received. To prevent abuse, free tokens (distributed during initial promotions or airdrops) are marked as non-transferable or non-sellable. This ensures users can't immediately sell free tokens to drain the contract's liquidity.

Calculate Sell Amount: The user specifies the number of tokens they want to sell. The smart contract calculates the total amount of ETH they will receive based on the current token price as in eq.3.

eq.3. `ethAmount = tokenAmount × tokenPrice`

3.6 Liquidity Risk Management:

To prevent liquidity issues (where the contract runs out of ETH to pay sellers), several mechanisms can be introduced:

1. Sell Limits: Limit the number of tokens that can be sold in a given time.
2. Dynamic Price Adjustment: If liquidity is low, adjust the token sell price to discourage mass liquidation.
3. Reserve Fund: Set aside a portion of ETH raised during the ICO for liquidity purposes to ensure there is always enough ETH for token holders to liquidate.

3.7 Issue Creation

Users can create issues by calling the `` `createIssue` `` function. They specify a repository ID, an issue ID, and the prize (in ERC20 tokens) as shown in Fig. 3.
If these checks pass, the user transfers the prize tokens to the contract. The issue is then created with the user's address as the creator, the prize amount is stored, and an event `` `IssueCreated` `` is emitted.

3.8 Staking on Issues

They specify the repository ID, issue ID, pull request ID, and the staking amount (`` `amt` ``) as shown in Fig 4. This amount gets locked till issue is solved or a certain `max_period` [7]. Optionally the contract can also give small time-dependent returns on their locked staked amount for more user engagement and participation. The function checks a major condition:

- If staker has occurred losses (`withdraw_proceedings` < 0), then value of (`balance-withdraw_proceedings`) of the staker should be greater than 30% of (`balance-amt`). Else the staker will have to buy more ERC20 tokens (GST) in order to maintain balance.

An event `` `StakePlaced` `` is emitted. The contract also updates the user's wallet statistics, including total stakes and open stakes.

3.9 Marking an Issue as Solved

The `markSolved` function allows the creator of an issue to mark it as solved when a solver's pull request is accepted by creator as shown in Fig 5. This function finalizes the issue resolution process by rewarding the solver and rejecting all other non-winning stakers.

If there are multiple stakers, the `rejectOthers` function is called to refund non-winning stakers with a deduction.

3.10 Deduction Rate

The `getDeductionRate` function calculates the rate at which non-winning stakers are penalized. When there are multiple stakers, and only one wins, others face a deduction to ensure fairness in the staking process. It is dependent on:
total: The total amount staked by all participants in the issue.
amt: The amount staked by an individual staker.
count: The total number of stakers.

The function first calculates a deduction rate (rate) using two parts:
1. A portion of the individual's staked amount, scaled by a constant `DEDUCTION_FACTOR` (set to 0.5 ether).
2. The remainder of the total staked amount, divided evenly among all stakers.
This ensures that the deduction is proportional to both the individual's stake and the total stake pool.
This function determines how much will be deducted from non-winning stakers. By applying a deduction rate that factors in both the individual's stake and the total stake, the system maintains a balance between encouraging participation and penalizing non-selection, preventing unfair losses.

3.11 Handling Multiple Stakers

The `rejectOthers` function handles the rejection of stakers who did not win the solution for an issue. After a solver is selected, this function ensures that other stakers are refunded a portion of their stakes, with a deduction applied for not winning as shown in Fig 5.
• Calculate Deduction: For each non-winning staker, the function calculates the deduction using the `getDeductionRate` function. This deduction is applied based on the total staked amount and the number of stakers.
• Calculate Refund: The refund is determined by subtracting the deduction from the original staked amount. The refund represents the amount returned to the staker after the deduction.
• Record Total Deductions: The total deductions from all non-winning stakers are added to the system's global deductions counter.

4 Discussion

4.1 Contribution to DeFi ecosystem

The GitStake platform leverages DeFi concepts to ensure trustless interactions between contributors and project maintainers. Existing research indicates that integrating DeFi into development ecosystems enhances transparency and reduces reliance on centralized entities [4, 5].

- Dynamic Token Pricing: The quadratic token pricing mechanism aligns with bonding curve models discussed by Kumar et al. [6], ensuring fair token distribution while incentivizing early adopters and maintaining liquidity.
- Free Token Distribution: Prior work on token distribution models emphasizes the importance of accessibility to encourage initial participation. GitStake's airdrop feature mirrors the strategies outlined by Zhang and Li [7], driving user engagement during the platform's early stages.

4.2 Resolving Key Open-Source Challenges

GitStake addresses several limitations of traditional contribution models:

- Prioritization of Critical Issues: By integrating staking, GitStake ensures that issues with higher stakes receive immediate attention, as highlighted by Johnson and Ray's research on resource allocation in collaborative environments [8].
- Reward Transparency: Decentralized reward systems, such as those implemented in GitStake, reduce trust deficits identified in traditional platforms [9]. This aligns with studies by Patel et al. [10], which underscore the importance of trust in sustaining open-source contributions.
- Handling Financial Risks: Token volatility, a common concern in blockchain ecosystems, is mitigated through mechanisms like deduction rates and liquidity risk management. These strategies echo the findings of Wang et al. [11], who proposed adaptive pricing to address market dynamics.

5 Sample Screen Designs

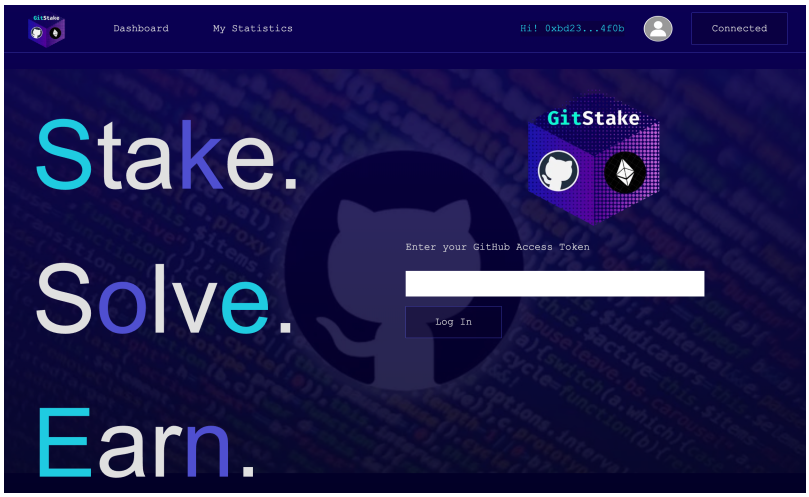

Fig. 7. Wallet (via *Connect* button) and User Login (via *GitHub Access Token*)

Fig. 8. List of Issues under a Repository including *Win Prize, Creator, Staker Count, Total Staked Amount* and *Option to Close* for each and *Highest Win Prize* among the issues. There is also a *Fork* Button to fork the repository with desired name.

Fig. 9. Details of an Issue, Linked Pull Requests indicates Stakes

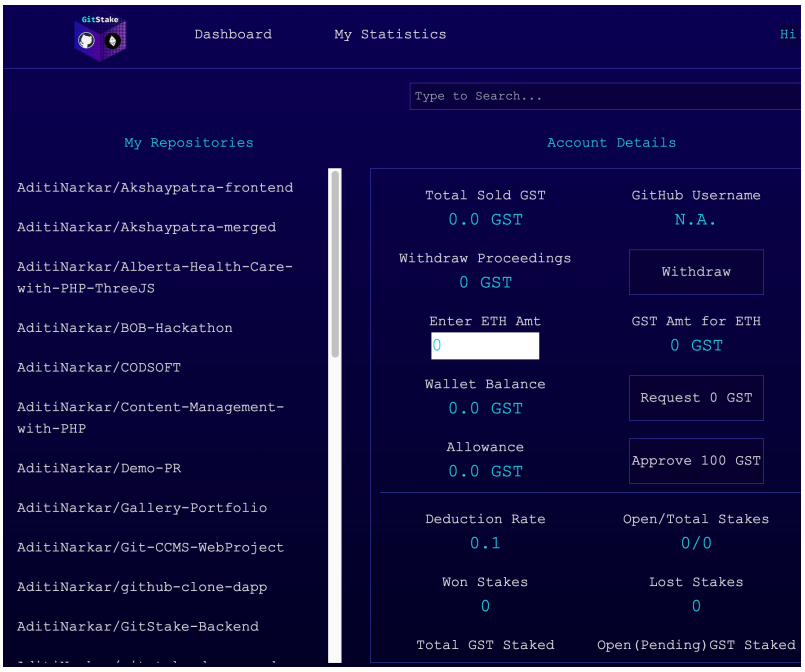

Fig. 10. Non-Deployer Wallet Stats which includes core components and user's repositories and other core functions.

6 Limitations

6.1 Token Volatility and Financial Risk

One of the core challenges in a tokenized ecosystem like GitStake is the volatility of ERC20 tokens. Fluctuations in token value may deter contributors, especially those relying on the platform for consistent rewards. The risk of financial loss due to sudden drops in token value could lead to decreased participation, particularly for contributors from regions with weaker financial safety nets.

6.2 Centralization Risk in Early Stages

During the initial token distribution and ICO phase, there is a risk of centralization, where a few participants with significant financial resources may acquire large amounts of tokens. This could allow them to dominate the staking process, skewing the reward distribution in favor of a few rather than promoting fair and equal participation.

6.3 Ethical Concerns Regarding Wealth Disparity

The platform's PoS mechanism prioritizes contributions based on the amount

staked. While this incentivizes high-value contributors, it may unintentionally create a wealth disparity, where only those who can afford to stake large amounts of tokens gain visibility and priority. This could marginalize smaller contributors or developers from less affluent backgrounds, raising ethical concerns about equity and access.

6.4 Exploitation of Free Tokens

The initial distribution of free tokens may encourage speculative behaviour, where contributors stake tokens not out of a genuine interest in the project but to exploit the rewards system. Without proper monitoring, this could lead to an environment where token holders' game the system for financial gain without contributing meaningful work.

6.5 Fraudulent Contributions

There is a risk of developers raising false issues or submitting plagiarized or low-quality code to exploit the reward system. Detecting and preventing such behaviour would require robust verification mechanisms.

7 Future Scope

7.1 Leveraged Staking with Aave or Similar Protocols

By integrating with protocols like Aave, GitStake could allow contributors to lend and borrow additional tokens against their staked assets. This would enable contributors to increase their staking power, effectively leveraging their contributions. Leveraged staking could make the platform more attractive to developers seeking higher rewards while maintaining careful risk management.

7.2 Yield Farming on Locked Stakes

GitStake could introduce a low-yield farming mechanism where staked tokens earn small but steady returns while being locked [7]. This would incentivize long-term staking by offering contributors additional rewards over extended periods, providing passive income even before resolving issues.

7.3 Dynamic Reward Adjustment Based on Market Conditions

The reward rates on staked amounts could be dynamically adjusted based on factors like token demand, liquidity, or network activity, ensuring the platform remains sustainable and competitive in the DeFi space. This dynamic system could balance the incentives between early adopters and new contributors.

References

1. Adams, H. et al.: Uniswap v2 core. Uniswap.org (2021) (https://uniswap.org/whitepaper.pdf)
2. Karp, A., Melbardis, I.: Nexus Mutual: A decentralized insurance protocol. (2022) (https://medium.com/xord/nexus-mutual-a-decentralized-insurance-mutual-6eff3665d9e1)
3. Angeris, G., Chitra, T.: Improved Price Oracles: Constant Function Market Makers. In: ACM Advances in Financial Technologies, AFT '20 (2020)
4. Aquilina, M., Frost, J., Schrimpf, A.: Decentralized Finance (DeFi): A Functional Approach. SSRN (2022) (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4325095)
5. Ankenbrand, T., et al.: Proposal for a Comprehensive (Crypto) Asset Taxonomy. In: IEEE Blockchain Conference, 2020. (2020) (https://arxiv.org/pdf/2007.11877)
6. Weingärtner, C., Freni, T., Hülsemann, A., Tumasjan, A.: Tokenomics: Decentralized Incentivization in the Context of Data Spaces. SpringerLink (2021) (https://link.springer.com/chapter/10.1007/978-3-030-93975-5_6)
7. Toderean, L., Anghel, I., et al.: A Lockable ERC20 Token for Peer-to-Peer Energy Trading. arXiv (2021) (https://arxiv.org/abs/2111.04467)
8. Schär, F.: Decentralized Finance: On Blockchain- and Smart Contract-based Financial Markets. Federal Reserve Bank of St. Louis Review, 2021. (2021) (https://doi.org/10.20955/r.103.153-74)
9. Campbell-Verduyn, M.: Bitcoin, Crypto-Coins, and the Future of Decentralized Finance. Routledge (2021) (https://www.routledge.com/Bitcoin-Crypto-Coins-and-the-Future-of-Decentralized-Finance/Campbell-Verduyn/p/book/9781138314428)
10. Buterin, V.: Ethereum Whitepaper: A Next Generation Smart Contract and Decentralized Application Platform. Ethereum Foundation (2015) (https://ethereum.org/en/whitepaper/)