CrowdChain: A Decentralized Crowdfunding dApp on Ethereum with Community-Governed Creator Verification

Vedanti Nagane[1], Prataprao Kale[2], Dhiraj Shinde[3], Prof. Shreya Nehe[4]
[1,2,3,4]*Computer Engineering, Genba Sopanrao Moze College of Engineering, Balewadi, Pune, India*

Abstract – Most crowdfunding platforms today are built on centralized infrastructure, which brings with it a familiar set of problems: users have to trust that the platform won't mishandle funds, fees eat into what creators actually receive, and there's rarely any real transparency around how money flows. This paper introduces CrowdChain, a crowdfunding dApp we built on top of Ethereum that tries to tackle these issues at the architecture level rather than papering over them. The standout piece of the system is a creator verification mechanism where community members - not a platform team - decide who gets to launch campaigns. Applicants upload proof documents to IPFS and need to clear a threshold of community upvotes before they can create a campaign. Everything else - collecting contributions, checking whether goals were met, releasing funds to creators, and issuing refunds if a campaign falls short - is handled automatically by a Solidity smart contract on Ethereum. The frontend is kept deliberately lightweight: plain JavaScript, Ethers.js v6 for talking to the chain, and Pinata for IPFS pinning. There's no admin, no platform cut, and the contract logic is fully open to inspect. We tested the system locally on Hardhat and deployed it to the Sepolia testnet; both environments confirmed the core functionality works as intended. CrowdChain shows that you don't need a centralized middleman to run a trustworthy fundraising platform.

Index Terms-Blockchain, Crowdfunding, Smart Contracts, Decentralized Autonomous Organizations, Ethereum, IPFS, DAOs, Tokenomics.

I. INTRODUCTION

Crowdfunding has become a go-to funding mechanism for all kinds of people, independent creators, early-stage startups, NGOs, research groups who need capital but can't or don't want to go through traditional financial channels. Platforms like Kickstarter and Indiegogo have shown just how much appetite there is for this model. At the same time, those platforms come with real structural baggage. They hold onto contributor funds as custodians, they take a notable cut [1], and their processes for vetting creators happen behind closed doors. From a contributor's perspective, there's no way to independently verify whether their money went where it was supposed to go.

Ethereum and its smart contract ecosystem offer a different path. A smart contract is essentially code that lives on a blockchain and runs exactly as written, no one can modify it after deployment, and it executes automatically when conditions are met [2]. That removes the need for a central custodian entirely, since the contract itself enforces all the rules [3]. We wanted to see how far you could take that idea in the context of crowdfunding.

CrowdChain is our answer to that question. It's a fully decentralized crowdfunding platform with three core design choices baked in. First, creator verification is handled by the community: to get campaign creation rights, an applicant has to submit identity documents to IPFS and collect enough upvotes from other verified participants. Second, every financial operation, contributions, withdrawals, refunds, is enforced by a smart contract with no administrative override. Third, and perhaps most simply: there's no owner, no admin key, and no fee mechanism anywhere in the system.

The rest of the paper breaks down as follows:

- Section 2 covers the methodology and the tech stack we used.
- Section 3 goes through the system architecture in detail.
- Section 4 defines the project's scope and how it differs from existing platforms.

- Section 5 lays out the specific objectives we set out to achieve.
- Section 6 discusses results and findings from testing.
- Section 7 wraps up and points to future work.

II. METHODOLOGY

A. Implementation Approach

We structured CrowdChain around a clear separation of concerns across three layers: the smart contract, the storage backend, and the frontend UI. The contract is the source of truth for everything, all business logic lives there. The frontend is purely presentational; it reads from and writes to the contract, but no trust-critical operations happen on the client side.

Hardhat manages the development environment. It gives us a local Ethereum node to work against, handles Solidity compilation, and lets us write deployment scripts that automate the full compile-and-deploy cycle. One nice touch: the deployment script writes the newly deployed contract address directly into the frontend config file, so there's no manual step where someone has to copy-paste an address and risk getting it wrong.

B. Technology Stack

Here's what the system is built on:

- Solidity 0.8.20 - our contract language of choice. The 0.8.x line has built-in overflow protection which removes a whole class of potential bugs [2].
- Hardhat 2.22 - used for local blockchain simulation, compilation, and deployment scripting.
- Ethers.js v6 - the JavaScript library that lets the frontend talk to the Ethereum node through MetaMask. It handles wallet connections, contract calls, and transaction submissions.
- IPFS via Pinata - creator documents (PDFs) get pinned to IPFS through Pinata's API. The resulting CID is what gets stored on-chain [4], so the document reference is permanent and content-addressed.
- MetaMask - the browser wallet extension that manages user accounts and signs transactions [2].
- Node.js HTTP Server - a minimal static server to serve the frontend locally, hence avoiding the security restrictions browsers place on directly opened HTML files.
- HTML5, CSS3, and Vanilla JavaScript - no frontend framework. Keeping the dependency footprint small makes the project easier to audit and run.

C. Development Workflow

Getting the project running takes three commands. npm run node spins up a local Hardhat blockchain with twenty pre-funded accounts. npm run deploy compiles the contract, deploys it, and writes the address into the frontend automatically. npm start launches the static server at http://localhost:3000. For testnet deployment, environment variables supply the private key and RPC endpoint - the deployment script itself doesn't need any changes.

III. SYSTEM ARCHITECTURE

A. Overview

CrowdChain is a three-tier system. The smart contract layer owns all state and enforces all rules. The storage layer (IPFS via Pinata) handles the off-chain documents that can't or shouldn't live on-chain. The browser-based frontend sits on top, talking to the contract through the MetaMask-injected provider using Ethers.js as the abstraction layer.

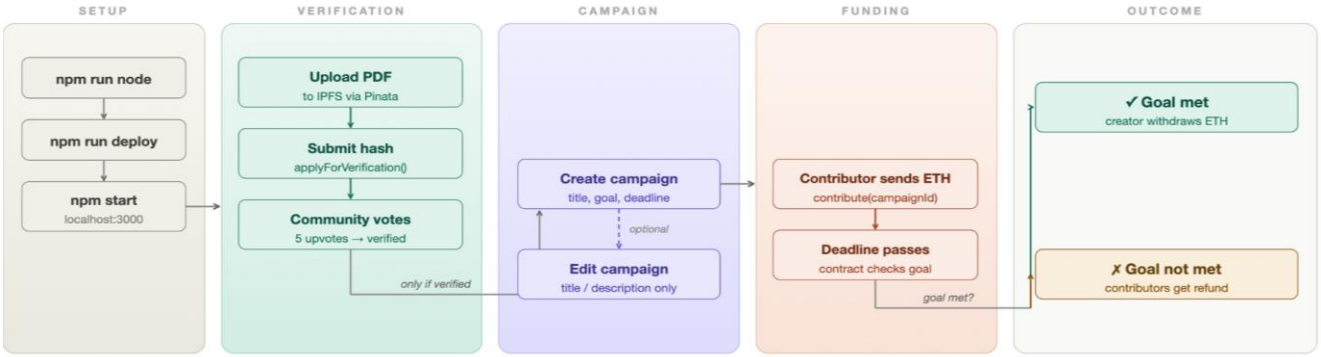

Fig. 1. System Flow

B. Smart Contract Architecture

The core of the system is Crowdfunding.sol. It defines two main structs. CreatorApplication holds an applicant's wallet address, the IPFS CID of their proof document, running upvote and downvote totals, a verified flag, an existence guard, and a nested mapping to track who's already voted on them. Campaign holds the campaign ID, the creator's address, title, description, funding goal in wei, deadline as a Unix timestamp, total raised so far, a withdrawal flag, and an existence guard.

State lives in three mappings: creator address → CreatorApplication, campaign ID → Campaign, and a nested campaign ID → contributor address → amount contributed. That last one is exclusively for processing refunds when a campaign misses its goal.

The contract exposes seven external functions: applyForVerification, voteOnCreator, createCampaign, contribute, withdrawFunds, refund, and editCampaign. Each one opens with require checks that have to pass before any state changes happen. The onlyVerified modifier gates createCampaign so only addresses with a verified flag can call it. The campaignExists modifier prevents anyone from operating on a campaign ID that doesn't exist.

C. Creator Verification Flow

Verification happens in two stages. In the first, a prospective creator uploads a PDF, like a government ID or company registration to IPFS through the Pinata API. The CID that comes back gets submitted on-chain via applyForVerification [4]. The contract stores it and marks the address as having an active application, which blocks re-application from the same address.

In the second stage, other community members can look up the applicant by wallet address in the Vote on Creators tab, pull up their document from IPFS, and cast an upvote or downvote by calling voteOnCreator [3].

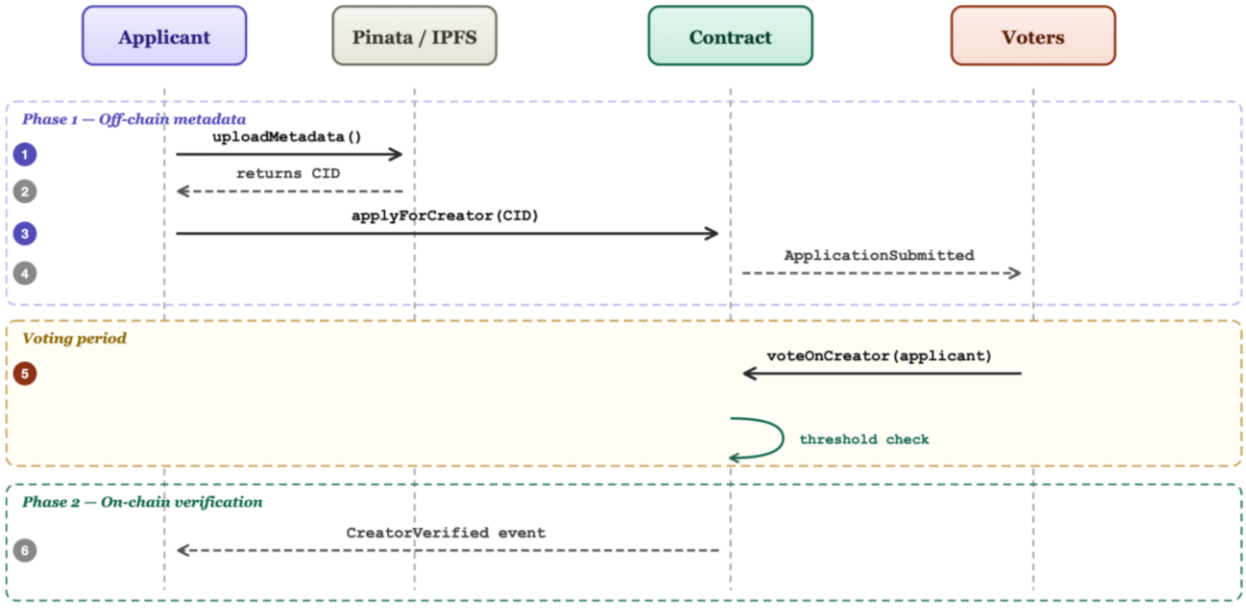

Fig. 2. Creator Verification - Sequence Diagram

The contract enforces a few rules around voting: you can't vote on yourself, you can't vote on the same applicant twice, and you can't vote on someone who's already been verified. Once an applicant's upvotes hit the VOTE_THRESHOLD (currently 5), the contract automatically flips their verified flag to true and fires a CreatorVerified event. No admin needs to do anything.

D. Campaign Lifecycle

A verified creator can call createCampaign with a title, description, goal in wei, and a deadline as a Unix

timestamp. The contract checks that the title isn't empty, the goal is above zero, and the deadline is actually in the future relative to block.timestamp.

Contributors call the paid contribution function and attach the ETH they wish to send. The contract checks that the campaign deadline has not passed and. The amount awarded is not zero. These contributions will be added to the campaign total and recorded in the per contributor mapping.

Once the deadline passes, one of two things will happen. If the target is met or exceeded, the creator can contact Fund withdrawal, which will transfer the entire balance to their address. We follow the Checks-Effects-Interactions (CEI) pattern here: the withdrawn flag gets set to true before the external call, which prevents reentrancy [5]. If the goal wasn't met, contributors can call refund to get their specific contribution back. Same pattern applies, the per-contributor balance is zeroed out before the transfer.

E. Campaign Editing

There's an editCampaign function that lets creators update a campaign's title and description. The funding goal and deadline are locked after creation - changing those after people have already contributed would undermine the whole point of the contract. The function also checks that the withdrawal flag hasn't been set, so you can't edit a campaign that's already been closed out.

F. Frontend Architecture

The frontend is a single-page application in vanilla HTML (Hyper Text Markup Language), CSS (Cascading Style Sheets), and Java Script, served by a simple Node.js HTTP server. A status bar at the top of the page shows what's happening at each stage of a transaction - submitted, confirmed, succeeded, or failed with color through MetaMask, the frontend sets up an Ethers.js BrowserProvider and Signer, then creates a contract instance using the ABI and address from contract.js.

There are five tabs: Home (overview and live stats), Get Verified (upload to IPFS and submit the application), Vote on Creators (look up an address and vote), Create Campaign (form for verified users), and All Campaigns (grid of all campaigns with controls for contributing, withdrawing, and claiming refunds). A status bar at the top of the page shows what's happening at each stage of a transaction - submitted, confirmed, succeeded, or failed with color coding and auto-dismiss for successful states.

IV. SCOPE OF THE PROJECT

A. Application Areas

CrowdChain is suited to any fundraising situation where contributor trust and transparency around fund usage actually matter. That covers a wide permissionless, anyone with an Ethereum Community-driven initiatives, charitable campaigns, early-stage startup fundraising, academic and research financing. Because it's permissionless, anyone with an Ethereum wallet can participate - there's no geographic restriction or identity gate controlled by a central authority.

B. Differentiation from Existing Approaches

Platforms like Kickstarter, Go FundMe — and Indiegogo are custodians - Kickstarter hold funds and enforce terms through legal agreements and internal reviews. The only centralized dependency is Pinata between 3-5% of funds raised, before payment processing charges [1]. Creator vetting, where it exists, is done internally and isn't visible to contributors [3].

CrowdChain is different in a few fundamental ways. Fund custody and release logic are in a publicly verifiable smart contract - anyone can read the exact conditions under which money moves. Creator verification happens on-chain and is publicly queryable, not locked inside a platform's admin panel. There are no fees. No registration. No email. The only centralized dependency is Pinata for IPFS document pinning.

C. Advantages

The main advantages over centralized platforms:

1) Trustless fund management: the contract enforces fund release conditions and no party - including us as developers - can override it [6].
2) Transparent verification: votes and verification status are recorded on-chain — so contributors can independently verify whether the creator is legitimate.
3) No platform fees: 100% ETH contribution goes to the creator if the campaign is successful. Nothing is extracted.

4) Censorship resistance: no central party can block a verified creator from launching a campaign or stop a contributor from participating.
5) Automatic refunds: if a campaign fails, contributors can reclaim their funds directly through the contract. There's no platform goodwill required [1].

V. OBJECTIVES

CrowdChain was built to address specific, concrete problems in existing crowdfunding infrastructure:

1) Eliminating custodial risk: no single entity holds unilateral control over funds, which removes the risk of platform insolvency, exit fraud, or arbitrary freezing.
2) Replacing opaque verification: on-chain community voting can be audited by anyone, which is a big improvement over closed platform review processes [3].
3) Automatic rule enforcement: the terms that are agreed upon are encoded in the contract and enforced without exception. No human intermediary can choose not to follow through.
4) Reduced costs for content creators: no platform fees means more of the capital raised will be used for campaigns.
5) Contributor protection guarantee: contracts enforce refunds technically, not just legally - which is a stronger guarantee than what a centralized platform can offer.
6) Persistent document storage: keeping verification documents on IPFS means they stay accessible and can't be taken down by any central server going offline [4], [7].

VI. RESULTS AND DISCUSSION

A. Functional Outcomes

We deployed and tested CrowdChain on a local Hardhat node with twenty pre-funded accounts (10,000 ETH each), which let us simulate realistic multi-user scenarios. All seven contract functions - applyForVerification, voteOnCreator, createCampaign, contribute, withdrawFunds, refund, and editCampaign - worked correctly in both happy-path and error scenarios. Revert messages were accurate across every tested failure case: self-voting, double-voting, unauthorized withdrawals, contributions to expired campaigns. The IPFS pipeline through Pinata's v1 API uploaded documents successfully and returned valid CIDs within normal latency [7]. We confirmed those CIDs were resolvable through the public IPFS gateway at https://ipfs.io/ipfs/, which validates the full end-to-end persistence story.

B. Transparency & Security Analysis

Using a public blockchain as the state layer means every action - application submission, vote, campaign creation, contribution, withdrawal, refund - is recorded as an on-chain transaction with an associated event. Anyone can reconstruct the full history of platform activity from the event logs without asking anyone's permission.

The security model leans on the EVM's deterministic execution and the correctness of the contract code. We applied the Checks-Effects-Interactions pattern in both withdrawFunds and refund to guard against reentrancy [5]. In refund, the per-contributor balance is zeroed before the ETH transfer. In withdraw Funds, withdrawal alerts are set before transfer. Either way, a recursive call can't extract additional funds because the state has already been updated. The editCampaign function only touches title and description - goal and deadline stay immutable post-creation, protecting contributors from manipulation after they've already committed. It also blocks edits on campaigns where withdrawal has already happened.

C. Limitations

A few limitations are worth acknowledging honestly:

1) The five-upvote VOTE_THRESHOLD works fine in a controlled environment, but it's probably not robust enough against Sybil attacks in a large open network. Someone with enough wallets could upvote their own application through to verification [3]. We looked at mitigations like minimum account age or on-chain identity binding but didn't implement them in this version.
2) Pinata API credentials are currently embedded in client-side JavaScript, meaning anyone who views the page source can see them. In a production deployment, uploads would go through a backend relay that keeps credentials server-side.
3) The VOTE_THRESHOLD, along with other parameters are hardcoded. Changing them requires redeploying the contract. A formal on-

chain governance mechanism would be the right fix for this.

4) Gas fees apply to all contract interactions, as with any EVM-based system. On Ethereum mainnet this could be a real barrier for small campaigns, though deploying on a Layer 2 would bring costs down substantially [1].

VII. CONCLUSION AND FUTURE WORKS

A. Summary of Contributions

This paper presented CrowdChain, a decentralized crowdfunding platform on Ethereum. The work makes three main contributions. First, we built and tested a working implementation of community-governed creator verification using IPFS-backed documents and on-chain voting - a transparent alternative to platform-controlled review. Second, the entire financial lifecycle of a campaign (creation, contributions, withdrawal, refunds) is encoded in a single auditable Solidity contract with no admin escape hatch. Third, the frontend makes all of this accessible to non-technical users through a standard browser interface with MetaMask. Testing on both a local Hardhat node and the Sepolia testnet confirmed that everything functions correctly across all the core use cases. We think CrowdChain is a solid proof of concept and a usable reference architecture for anyone building community-governed DeFi applications.

B. Future Work

There are several directions we'd like to take this further:

- Sybil resistance: plugging in something like Ethereum Attestation Service (EAS) or a proof-of-humanity registry would make it significantly harder to game the verification vote [4].
- On-chain governance: replacing the hardcoded VOTE_THRESHOLD with a DAO-style governance mechanism would let stakeholders vote on parameter changes without requiring a contract redeploy.
- Layer 2 deployment: moving to Optimism, Arbitrum, or Polygon would drop transaction costs by one to two orders of magnitude and make the platform viable for smaller campaigns [1].
- Milestone-based fund release: breaking campaigns into funding tranches that unlock based on milestone proofs would give contributors a finer-grained accountability mechanism, similar to how traditional grants work [6].
- Campaign media support: letting creators attach images and supplementary documents via IPFS would improve how campaigns present themselves and make them easier to discover.
- Backend IPFS relay: routing Pinata uploads through a server-side relay would resolve the credential exposure issue in the current frontend-only implementation.

In short, CrowdChain demonstrates that a fee-free, trust-minimized crowdfunding platform is genuinely achievable on existing blockchain infrastructure. There's no theoretical blocker - only engineering work left to do.

REFERENCES

[1] Y. Aprilinda, E. Princes, H. L. H. S. Warnars, and D. Fitrianah, "Systematic review of technology regulation and trust for crowdfunding sustainability," in Proc. IEEE, pp. 1–7, 2025.

[2] R. Banoth and M. B. Dave, "A survey on decentralized application based on blockchain platform," in Proc. Int. Conf. Sustainable Computing and Data Communication Systems (ICSCDS), pp. 1171–1174, 2022.

[3] M. Zichichi, M. Contu, S. Ferretti, and G. D'Angelo, "Likestarter: a smart-contract based social dao for crowdfunding," in Proc. 2nd Workshop on Cryptocurrencies and Blockchains for Distributed Systems (CryBlock), (Paris, France), pp. 1–6, 2019.

[4] S. Majumder, R. Zaha, M. M. H. Manik, and K. M. R. Alam, "A blockchain based scalable framework for academic document verification," in Proc. 26th Int. Conf. Computer and Information Technology (ICCIT), (Cox's Bazar, Bangladesh), pp. 1–6, Dec. 2023.

[5] I. J. Villanueva, M. Srinivasan, and F. U. Rehman, "Metamorphic testing for smart contract validation: A case study of ethereum-based crowdfunding contracts." arXiv:2501.09955, Jan. 2025.

[6] H. Liu, J. Li, S. Yuan, W. Cao, and B. Li, "A smart contract based crowdfunding mechanism for

hierarchical federated learning." arXiv:2205.06101, May 2022.

[7] S. Badlani, S. Maniar, T. Aditya, and K. Devadkar, "Educrypto: Transforming education using blockchain," in Proc. 6th Int. Conf. Intelligent Computing and Control Systems (ICICCS), pp. 829–836, 2022.