# d-FCT: Platform Architecture and Smart Contract Design

## Technical Report

*Draft version 1.0 – February 19th, 2025*

Project Catalyst Fund13
*Project ID: 1300076*

**Contributors**

Rafael Brandão (rafael@mobr.ai)
Marcio Moreno (marcio@mobr.ai)

# 1. Introduction

This technical report outlines the platform architecture and smart contract design for d-FCT (Decentralized Fact-Checking Toolkit), funded by Project Catalyst F13 under project #1300076[1]. Pronounced *de facto,* the platform aims at redefining fact-checking by structuring and supporting collaborative verification processes. The platform will leverage the Cardano ecosystem to promote transparency, scalability, and active community participation, while incentivizing contributions with its own Cardano Native Token (CNT). d-FCT's goal is to build a high-trust environment for addressing the search for structured evidence, by combining AI processing and collective validation with tokenized rewards.

Designed for decentralized collaboration, the platform enables users to upload content (such as images, videos, text, or URLs) for analysis, verification, and validate claims. The system may direct users to existing topics or offer them the option to create a new topic.

Provenance and reward mechanisms ensure contributors are compensated fairly, while a community-driven governance model enables users to influence the platform's evolution. On-chain side, the Cardano blockchain's scalable architecture ensures d-FCT can handle large-scale transactions. Off-chain side, AI tools enhance fact-checking and verification workflows, providing support for content evaluation. Fact-checking and content processing are carried out through a combination of automated tools and human reviews, to process claims and generate structured knowledge. These results are periodically registered on-chain to maintain a transparent and immutable record of verified contributions and outcomes (provenance), ensuring trust and accountability within the ecosystem.

The platform includes data storage and reputation management features. Media files, detailed claim descriptions, and supporting evidence are stored in databases, with plans to transition to decentralized storage solutions like Iagon[2], Filecoin[3], or Arweave[4]. To enhance verification, the hashes of these assets will be registered on-chain. User activity and reputation scores will be tracked on-chain, updating dynamically based on engagement.

Anti-abuse mechanisms employing heuristics are planned to detect spam, manipulation, or malicious behavior/sybil attack, ensuring platform integrity and fairness. d-FCT will implement a multi-layered mechanism, combining identity verification, economic deterrents, and reputation-based controls. For instance, requiring users to stake tokens or pay small transaction fees for submissions makes it costly for an attacker to create numerous fake accounts. A robust reputation system rewards verified users, while limiting the influence of new or unverified accounts. Integrating decentralized identity (DID) solutions could further help validate user authenticity. Rate limiting and monitoring unusual account creation patterns, along with

[1] https://milestones.projectcatalyst.io/projects/1300076
[2] https://iagon.com/
[3] https://filecoin.io/
[4] https://www.arweave.org/

community reporting and consensus-based validation, can significantly reduce the potential impact of malicious attacks.

# 2. Architecture and current implementation

d-FCT's architecture (illustrated in Figure 1) is designed to handle and verify user-submitted content through a combination of specialized components. The system features a modular structure that incorporates Web2 and Web3 components, designed to enable scalable fact-checking workflows. The proposed interaction model foresees users providing content, search input, and feedback, while consuming topics and facts through a concise UI.

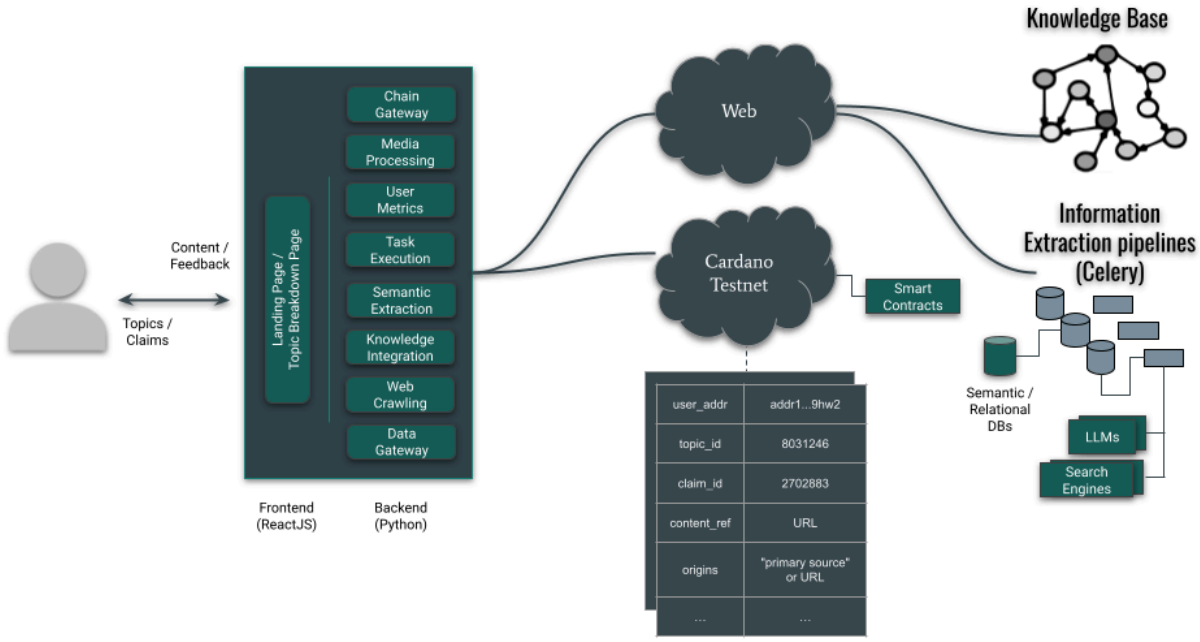



Figure 1 – d-FCT architectural overview.

## Backend

The backend is currently being implemented in Python, and handles dynamic and resource-intensive operations, ensuring communication between on-chain and off-chain systems. Off-chain operations are responsible for handling computationally intensive tasks and data management. That is, the platform's operational logic, including media processing, user metrics tracking, task execution, semantic data extraction, web crawling, etc. It also manages the data gateway to storage subsystems. Information extraction pipelines coordinate human and AI-assisted tasks in the verification processes. Table 1 summarizes the main backend components.

Table 1 – Python backend main components.

| | |
|---|---|
| Chain Gateway | Facilitates communication between off-chain processes and the blockchain. Handles smart contract interactions, monitors transaction states, and ensures data and operations remain synchronized. |
| Media Processor | Prepares and processes content (e.g., extracting frames, metadata, analyzing media integrity, transcoding, etc.). |
| Semantic Extraction | Integrates AI models and knowledge graphs to analyze submitted content and extract meaningful relationships. |
| Task Execution | Manages workflows for claim verification, including both automated and human review tasks |
| User Metrics | Tracks user activity and contributions to determine engagement and reputation scores |
| Knowledge Integration | Structures claims and content of existing topics in the knowledge graph. |
| Web Crawling | Fetches and analyzes external data sources to support enriched context and LLM RAG |
| Data Gateway | Serves as the bridge for data exchange between the frontend, backend, and on-chain systems. |

## Frontend

The frontend design features a user-centric, minimalist use case (Figure 2), that prioritizes clarity in the decentralized verification process. The UI is currently being implemented in ReactJS, allowing seamless interaction with the platform. Users will be able to search and create topics by uploading content (e.g., images, videos, text, or URLs), and engage with existing topics in the knowledge graph. It will also provide tools for reviewing claims, and submitting feedback via intuitive elements across landing and topic breakdown pages.

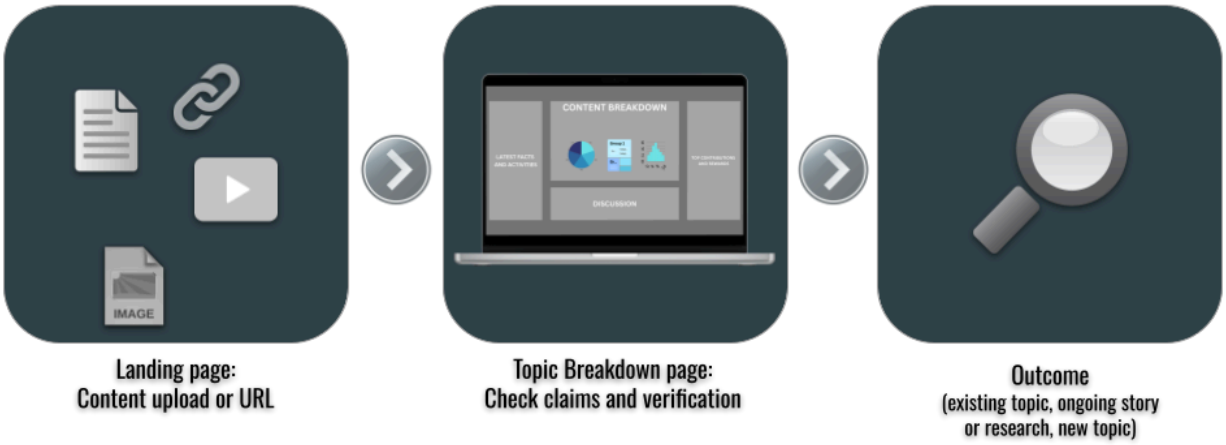
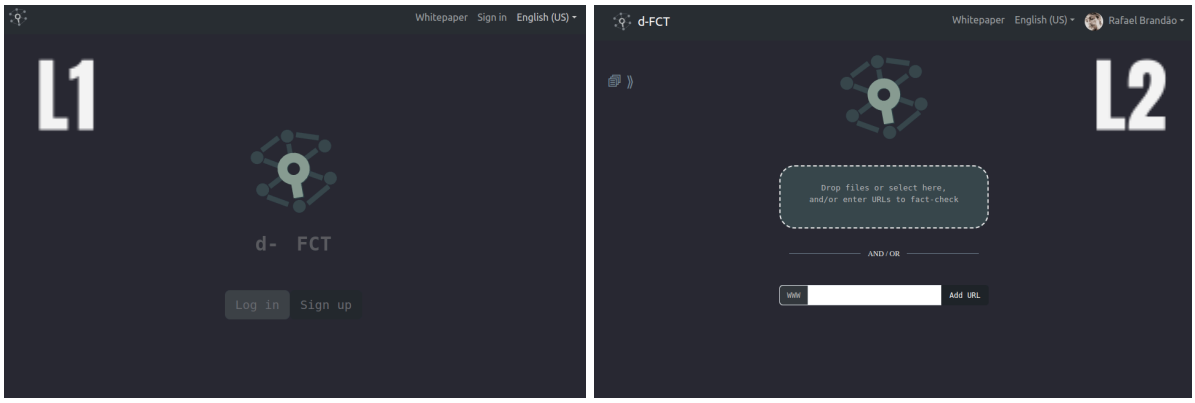



Figure 2 – minimalist use case.

A landing page will serve as the entry point for users, providing wallet connection and/or account login options. Figure 3-L1 shows the landing page view for an unlogged user, after logging in users can upload content (Figure 3-L2)—including images, videos, and text— and/or URLs for fact-checking (Figure 3-L3). This page streamlines the content submission process. Users are either directed to existing topic breakdown pages that match the content within the knowledge graph, or to a draft page that can be used on a topic proposal around the uploaded content.

The topic breakdown page (Figure 3-TB) will offer a structured exploration of individual topics, showcasing claims. Topics evolve dynamically, as users contribute with additional pieces of evidence, upvote/downvote claims, and provide review on existing information. It integrates a detailed view on contributions, rewards, and the status of claims. This two-page design balances simplicity with core functionality to produce a reasonable user experience.



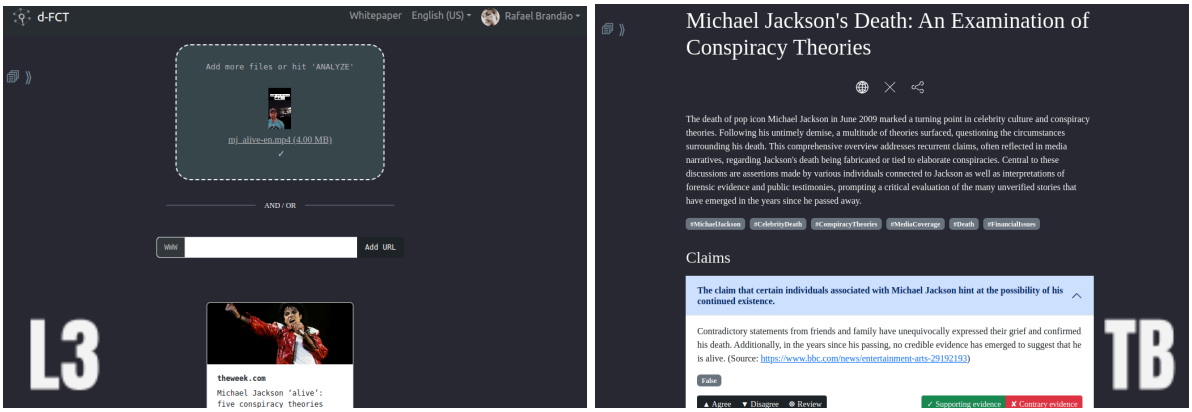



Figure 3 – Prototype front-end UI with ReactJS.

## Blockchain and smart contract

Blockchain records and smart contracts are essential to support key features on the platform, including reward distribution, governance mechanisms, and the provenance of contributions during the decentralized verification process. It is crucial to store immutable records for topics, claims, and content references, such as user addresses, identifiers, and other content metadata. For each transaction, the platform will validate the conditions required for the operation, such as ensuring the authenticity of a contribution and calculating rewards based on predefined rules.

d-FCT will leverage Cardano's EUTXO (Extended Unspent Transaction Output) model to handle transactions with transparency and determinism. The model enhances the traditional UTXO transaction processing by introducing features such as *datums* and *redeemers*, making it suitable for the d-FCT's main use cases. This section highlights relevant Cardano's concepts and features that can be applied on d-FCT. The platform's smart contract design and details, including the required data structures to manage state in the verification process, will be discussed in Section 3 (verification process), Section 4 (reward pools and distribution), and Section 5 (smart contract design).

**Cardano's Plutus script**

Scripts in the Cardano ecosystem assess the inputs provided by any party attempting to spend a UTxO, approving or rejecting the transaction. These scripts are compiled into a Plutus Core binary and deployed on-chain, ensuring secure and deterministic validation. A script context may provide information about the pending transaction, along with which input triggered the validation.

**Script address**

The hash of the Plutus script binary functions as an address that holds UTxOs, much like a typical public key address. When a transaction attempts to consume UTxOs from this address, the corresponding Plutus script is executed, evaluating the transaction's inputs—specifically the

datum, redeemer, and script context—and the transaction is deemed valid only if the script returns true.

**Datums and redeemers**

The EUTXO model introduces *datums* and *redeemers* to enhance flexibility in transaction validation. A datum is a piece of information attached to a UTxO typically used to carry state. When funds are sent to a script address, the sender includes the hash of a datum to lock those funds. Later, when attempting to spend the UTxO, the spender must provide a datum whose hash matches the attached hash, along with a redeemer that satisfies the conditions defined in the Plutus script to unlock the funds. For d-FCT, datums may contain metadata about specific operations, such as user contributions, topic or claim IDs, references to offchain content (e.g., hashes), or any other data pertinent to the verification step in question.

A redeemer is structured similarly to a datum but is provided as a separate input. It carries not only datum-like data but also additional information detailing the specific actions to be performed on the target UTxO and the computational resources reserved for the transaction. The redeemer is attached to the input transaction, serving as the key that the script evaluates to unlock funds and validate the transaction. In d-FCT, redeemers may define the operation being performed, such as distributing rewards, staking tokens for governance, or proposing a new topic for verification. For instance, a redeemer could specify the amount of tokens to be distributed for a contribution.

**Purposes**

Plutus scripts receive a script context that includes a script purpose, guiding each script to validate specific operations within a transaction. d-FCT's scripts could employ various purposes: **spending** scripts may validate the consumption of reward pools by checking the attached datum, redeemer, and transaction details. They may govern how locked assets (e.g., staked tokens, or escrowed funds) are released based on verification and dispute resolution outcomes. This can be applied in verifying claims, validating contributions, and handling disputes within d-FCT; **minting** scripts, or minting policies, could govern the creation of dFCT's CNT ($DFCT); **rewarding** scripts could control the withdrawal of staking and rewards by verifying that funds are directed to designated addresses; and **certifying** scripts can validate certificate transactions like staking registration, de-registration, or delegation by enforcing necessary signatures and approval conditions. Plutus V3 extends these capabilities with **voting** and **proposing** scripts, which facilitate transparent governance by approving votes from designated representatives and validating proposals for parameter changes or treasury withdrawals.

**Contract workflow**

A state machine can be achieved on Cardano by leveraging a multi-step smart contract workflow within the EUTXO framework. At the core, a validator script would enforce state transitions by comparing the datum stored in a UTXO (which holds state information such as topic and claim identifiers, user contributions, etc.) with the redeemer provided during a transaction. The redeemer supplies the specific input needed to transition from one state to another, such as moving from "Topic Proposed" to "Topic Under Review" or from "Contribution Under Review" to "Rewards Distributed". This design ensures that only valid state transitions

occur according to predefined rules, thereby maintaining the integrity of the process. To further enhance security and collaborative decision-making, multi-signature (multi-sig) requirements can be integrated into the smart contracts for actions that demand multiple approvals.

# 3. Decentralized verification process

The d-FCT's verification process is designed to incentivize users based on different dimensions related to their contributions. The platform enables allocating reward pools and distributing tokens proportional to each user's effort and the assessed quality of their contributions. The goal is to promote an environment where high-quality, timely, and engaged contributions are recognized and incentivized.

Figure 3 illustrates the d-FCT State Machine (**dSM**), a structured process model that governs the lifecycle of topics, claims, and contributions. From the initial topic submission to reward distribution, state transitions are triggered when specific conditions are satisfied. On-chain mechanisms must ensure that only legit users and valid contributions influence the process, preventing manipulation and promoting credibility.

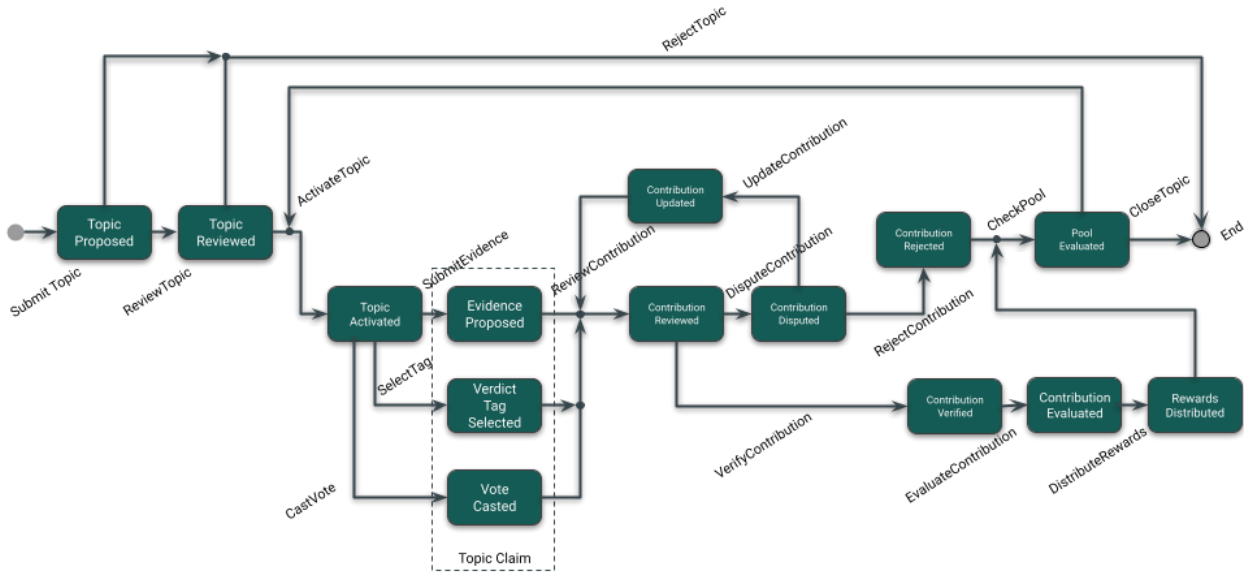



Figure 3 – d-FCT State Machine (dSM).

The dSM process starts with users submitting candidate topics for public verification. Leading to `TopicProposed` state, if the initial conditions for the topic are validated (e.g., title, description, reward token allocation, etc.) Then, the process moves to `TopicReviewed`, after selected users assess it according to the platform rules, leading to either `TopicActivated` or rejection (end state). If approved, contributors can submit supporting or contrary evidence for a specific claim (leading to `EvidenceProposed` state), select output tags (verdict) for a specific claim (leading to `VerdictTagSelected` state), and upvote/downvote claims (which leads to `VoteCasted` state).

All contributions undergo evaluation before the `ContributionReviewed` state. If challenged, contributions transition to `ContributionDisputed`, requiring an update for further scrutiny by selected users. Disputed contributions will either lead to `ContributionRejected` or `ContributionUpdated` states. The contribution can return to `ContributionReviewed` state after a new review. Verified contributions are assessed for impact and accuracy, leading to `ContributionEvaluated` state. Verified contributors receive incentives based on their contribution scores, leading to `RewardsDistributed` state. Afterwards, the pool is checked going to `PoolEvaluated`. If the reward pool still has available funds for distribution, the process can return to `TopicActivated`, if not the topic may be closed (end state) or remain in the `PoolEvaluated` state waiting for additional funding.

d-FCT proposes a contribution score system to incentivize activities and participation in the decentralized verification process. The system observes both quantitative as well as qualitative aspects. On the one hand, it quantifies verification activities, assigning specific points to participation. It also considers timeliness, reputation, and engagement in the quantitative side. On the other hand, contributions are qualitatively assessed through a rating system, where community users evaluate relevance, accuracy, and completeness of the provided content.

## Score system

Different actions within the verification process are assigned specific points to measure their contribution impact. Actions have varying weights based on their perceived importance and complexity. Contributions with higher scores receive a larger share of the reward pool.

In the current ranking proposal, submitting a topic rewards 10 points for each accepted topic. When it comes to verifying or rejecting claims, users can provide evidence to support or challenge it, earning 3 points per verified contribution for general users, and 5 points for verified high-ranked users. Up voting or down voting on a claim allows users to weigh in on the relevance of it, with each vote earning 1 point. Users can also select/vote on tags to express their opinions about the claim's verdict, rewarding 1 point as well. Currently, the following verdict tags are available:

1. **Factually correct**: if the claim is fully accurate and aligns with verified facts.
2. **Partially true**: if the claim contains a mix of accurate information and misleading or incorrect elements.
3. **Inconsistent/Wrong**: if the claim is inaccurate or contradictory but without intent to deceive.
4. **False**: if the claim is entirely untrue.
5. **Misleading**: if the claim may misrepresent context, facts, or intent, potentially leading to misunderstanding.
6. **Unverified**: if the claim could not be verified due to insufficient evidence or conflicting sources.

For reviewing contributions, users assess the submitted content and their reviews are awarded 3 points, with reviews from verified high-ranked users earning 5 points, reflecting their expertise and reliability. Finally, dispute resolution rewards users with 5 points for their involvement in resolving contested contributions, highlighting the critical role of considering dissent and disagreements to maintain the integrity of the process.

The score system aims at promoting a robust and engaged community. We plan to enable users to propose alternative ranking scores and rules through governance voting in the future. Table 2 summarizes actions and points related to the verification process.

Table 2 – Verification activities and points.

| Action | Description | Points |
|---|---|---|
| Proposing valid topics | Users propose topics to be checked | 10 points per accepted topic |
| Contribute to claim verification | Users provide evidence supporting or refuting a claim | 3 points per claim evidence<br><br>5 points per verified high-ranked user evidence |
| Voting on claims and output tags | Users upvote or downvote a claim, and/or select output tags on a claim | 1 point per valid vote |
| Reviewing contributions | Verified users review submitted contributions. High ranked users' reviews have higher weight due to their expertise | 3 points per contribution reviewed<br><br>5 points per verified high-ranked review |
| Dispute resolution | Engaging in the dispute resolution process for contested contribution and providing additional evidence | 5 points per resolved dispute |

## Timeliness

To encourage users to act promptly, the platform implements a system where early contributors can earn bonus rewards. Early contributors are those who engage with a verification task shortly after it becomes available. This approach is designed to expedite the verification process with high-quality information made available as quickly as possible. This is especially valuable for on-going events topics where the rapid availability of accurate information can significantly impact public understanding and decision-making. Prompt contributions ensure that critical

information is verified and disseminated swiftly, mitigating the risks of misleading information going viral. The reward system considers the following incentives for timely actions:

**Bonus structure**: Contributors who engage in the verification process within a configurable time frame (e.g., first 12 hours) following the topic's publication receive a bonus on top of their standard reward. This bonus is a parameterized percentage of the total reward pool allocated to the fact-checking task, encouraging users to act swiftly.

**Tiered rewards**: The bonus rewards can be structured in tiers based on the speed of contribution. For example, contributors who provide valuable input within the first time frame slice (e.g., first 2 hours) may receive a higher bonus compared to those who contribute within the next.

## Contribution assessment

In the `ReviewContribution` transition, the platform employs a rating system to structure and assess the quality of evidence provided by users. Targeting a fair reward distribution and reflective of the actual value provided, the system incorporates evaluations from selected users to determine the *relevance*, *accuracy*, and *completeness* of the contribution.

Each contribution is scored on a scale (1-10) for the three dimensions (relevance, accuracy, and completeness). Evaluations from high ranked users receive a higher weight to reflect their level of expertise. The evaluated scores from general and expert users are aggregated to determine an overall quality score. For instance, expert scores may contribute 60% to the total rating, while general reviews contribute 40%. Contributions with higher overall scores are allocated a proportional value of the reward pool, incentivizing users to provide high-quality, well-researched, and comprehensive inputs.

When establishing a board to review topic contributions, d-FCT's design considers a selection process inspired by Cardano's Proof-of-Stake validator selection. This would involve weighting users' eligibility based on factors such as their contribution history, reputation scores, and tokens staked on the platform. To expedite elections and avoid stalling verification processes, d-FCT can implement an automated, time-bound selection mechanism. This would regularly trigger a weighted random selection  (based on current contributions, reputation scores, and staked tokens), ensuring that the board is dynamically composed of active and qualified reviewers without manual intervention.

### General evaluations

Elected users can rate contributions based on their personal assessment of relevance, accuracy, and completeness. The review process involves rating on a scale of 1-10 for each of these three dimensions:

**Relevance**: Users rate whether the contribution directly addresses the claim and topic under review. That is, users evaluate whether the contribution is pertinent to the claim's context.

**Accuracy**: Users rate on the factual correctness of the information provided. This involves verifying the evidence against reliable sources or established facts.

**Completeness**: Users rate whether the contribution covers all necessary aspects of the verifying task. This includes evaluating whether the evidence is exhaustive or if there are significant gaps.

### Verified high-ranked evaluations

Verified high-ranked users, who have higher credibility and reputation within the platform, may conduct a more in-depth review. Aside from rating contributions (1-10) for each of the dimensions, they provide additional content to support their rationale. Their assessments are more rigorous and carry greater weight in the rating system. This category of users may:

**Relevance**: rate and provide content examining whether the contribution is relevant. They may also add value by introducing new insights or perspectives that enhance the overall verification process.

**Accuracy**: rate and provide content with detailed rationale for their assessment and verification of the evidence's accuracy, which could use advanced tools and methodologies to cross-check the information against multiple sources.

**Completeness**: rate and provide content evaluating whether the contribution is thorough and consider all relevant aspects of the topic. They may provide additional context or supplementary information that might be necessary for a comprehensive understanding.

Table 3 – Quality assessment of contributed evidence.

| | Relevance | Accuracy | Completeness |
|---|---|---|---|
| General evaluation | Rate (1-10) if the contribution addresses the claim or topic under review. | Rate (1-10) for the factual correctness of the information provided. | Rate (1-10) whether the contribution covers all necessary aspects of the verification task. |

| | Relevance | Accuracy | Completeness |
|---|---|---|---|
| Verified high-ranked users | Rate (1-10) if the contribution addresses the fact or topic under review.<br><br>Provide content examining contribution relevance | Rate (1-10) for the factual correctness of the information provided.<br><br>Provide content with assessment of the contribution accuracy | Rate (1-10) whether the contribution covers all necessary aspects of the verification task.<br><br>Provide content evaluating whether the contribution is thorough |

# 4. Reward pools and distribution

d-FCT's design considers reward pool allocation and distribution targeting active participation, fairness, and maintaining sustainability of the platform. When submitting topics, users allocate their tokens, creating a reward pool for verification activities.

For each proposed topic, the proposer assigns an amount of tokens to establish a reward pool . Contributors earn points through their various activities in the verification process. These points capture the volume and quality of the provided input to determine each contributor's share of the allocated reward pool. Smart contracts on the Cardano network will calculate and distribute tokens proportionally to the user scores.

**Example**
Task: Verify viral claims about a recent public event.
Reward Pool: 500 tokens

**Breakdown**

| | |
|---|---|
| User A submits the topic: | 10 points |
| High-ranked user B submits claim evidence: | 5 points |
| User C reviews User B's evidence: | 3 points |
| User D votes on the claim's validity: | 1 points |

Total points = 10 + 5 + 3 + 1 = 19 points

**User A's Reward**: (10/19) * 500 = 263.15 tokens
**User B's Reward**: (5/19) * 500 = 131.57 tokens
**User C's Reward**: (3/19) * 500 = 78.94 tokens
**User D's Reward**: (1/19) * 500 = 26.31 tokens

Other use cases related to reward pools and distribution include dispute resolution, and staking for governance.

When a claim is disputed, a portion of the reward pool can be reserved in escrow within a smart contract. The platform locks these funds until the dispute is resolved, ensuring fair handling of conflicts. Once resolved, the funds are either returned to the pool or distributed according to the outcome.

Users can allocate $DFCT tokens for staking, locking tokens to gain governance rights or boost their contribution score. For example, users who stake tokens may receive bonus rewards or enhanced voting power, ensuring that only committed contributors influence platform decisions.

In the future, other income sources may be used to establish revenue-backed and periodic reward pools. These include platform fees generated from user interactions on the platform, subscription fees from premium services, and other strategic partnerships. The platform will also tap into grants and sponsorships from ecosystem partners, and revenue-sharing agreements with content providers. In this sense, rewards will be distributed based on specific verification tasks, through periodic allocations (e.g., daily, weekly, monthly), or a hybrid approach. The combination of specific task-based rewards and periodic allocations, coupled with dynamic adjustments and revenue-backed pools, creates a robust incentive structure to support the platform's growth and sustainability.

# 5. Smart contract design

The d-FCT's smart contract design may employ either a single and comprehensive contract, or multiple specialized contracts. Each approach offers distinct trade-offs. A single contract centralizing all logic (i.e., covering provenance, governance, minting, reward distribution, etc.) may simplify development and ensure a unified state machine. However, it can become large and difficult to maintain or audit as the platform grows. Conversely, distributing functionalities across multiple contracts would enhance modularity, maintainability, and scalability. Yet, this approach can increase complexity in cross-contract communication and deployment. In practice, a hybrid strategy is often preferred, keeping core state machine logic in one contract while delegating auxiliary tasks (e.g. token minting or specialized governance features) to additional contracts.

The Testnet is the target for the d-FCT's smart contract deployment that will focus initially on managing topic lifecycle and contributions within the platform (i.e., provenance). Then, governance features will be added. The smart contract design leverages Cardano's EUTXO model, using datums, and redeemers to handle state and transaction execution. State transitions will be controlled by on-chain validators through UTXOs and associated datums to govern state transitions, ensuring only valid actions move the process forward. Table 4 lists a draft of the proposed transactions, along with their associated state, brief description, inputs, outputs, and validation logic of transactions.

Table 4 – Smart contract transactions draft.

| d-FCT use case | dSM transition / Transaction purpose | Description | Input | Logic | Output |
|---|---|---|---|---|---|
| Provenance | `SubmitTopic` (spend) | Allow users to submit new topics for fact-checking | **Datum**<br>Topic metadata (id, title, description, timestamp, proposer_addr, etc.)<br>**Redeemer**<br>Operation **`submit_topic`** | Validate initial topic metadata and ensure they meet submission criteria<br>Store the topic metadata and set the state as `TopicProposed` | UTXO with datum, e.g.:<br>`{ state:TopicProposed , int:topicId, string:proposerAddr ... }` |
| Provenance | `ReviewTopic` (spend) | Selected reviewers assess the relevance and quality of the submitted topic and related content | **Datum**<br>Topic metadata with current content refs<br>**Redeemer**<br>Operation **`review_topic`** | Verify review inputs, aggregate ratings and feedback, decide if the topic is valid<br>Transition state to `TopicReviewed` if valid or reject if criteria are not met. | UTXO with datum updated with state as `TopicReviewed` or a termination state if rejected |

| | | | | | |
|---|---|---|---|---|---|
| Provenance | `ActivateTopic` (spend) | Activates a reviewed topic, enabling users to contribute to its verification | **Datum**<br>Topic metadata with state = `TopicReviewed`<br>**Redeemer**<br>Operation **`activate_topic`** | Check for required governance approvals<br>if met, transition state to `TopicActivated` | UTXO with datum updated with state = `TopicActivated` or a termination state if rejected |
| Provenance | `CloseTopic` (spend) | Closes the topic once the fact verification process is complete | **Datum**<br>Topic metadata with state = `TopicActivated` or later states<br>**Redeemer**<br>Operation **`close_topic`** | Validate closure conditions<br>Update state to `TopicClosed` | UTXO with datum updated with state = `TopicClosed` |
| Provenance | `RejectTopic` (spend) | Rejects a proposed topic that fails to meet criteria | **Datum**<br>Topic metadata with state = `TopicProposed` or `TopicReviewed`<br>**Redeemer**<br>Operation **`reject_topic`** | Validate rejection conditions<br>Update state to `TopicRejected` | UTXO with datum updated with state = `TopicRejected` |
| Provenance | `SubmitEvidence` (spend) | Submits evidence for an active topic claim | **Datum**<br>Evidence metadata (linked topic/claim ID, content references, description)<br>Topic with state = `TopicActivated`<br>**Redeemer**<br>Operation **`submit_evidence`** | Validate evidence metadata and association with topic/claim<br>Update state to `EvidenceProposed` | UTXO with datum updated with evidence details and state = `EvidenceProposed` |
| Provenance | `CastVote` (spend) | Submits a user's vote on a claim's relevance | **Datum**<br>Voting data (claim ID, current vote tally, user id, etc.)<br>Topic with state = `TopicActivated` | Validate vote eligibility and ensure no duplicate voting<br>Update vote tallies accordingly<br>Update state to `VoteCasted` | UTXO with datum updated with evidence details and state = `VoteCasted` |

| | | | | | |
|---|---|---|---|---|---|
| | | | **Redeemer**<br><br>Operation<br>**`cast_vote`**<br><br>(specifying vote choice: upvote / downvote) | | |
| Provenance | `SelectTag`<br>(spend) | Assigns verdict tags to evidence or claims as selected by the community | **Datum**<br><br>Current tag list and associated claim/evidence identifier<br><br>Topic with state = `TopicActivated`<br><br>**Redeemer**<br><br>Operation<br>**`select_tag`**<br><br>(including selected tags) | Validate selected tags<br><br>Update the datum with the new tag list for the claim/evidence | UTXO with datum updated with evidence details and state = `VerdictTagSelected` |
| Provenance | `ReviewContribution`<br>(spend) | Reviews user contribution<br><br>Aggregate its relevance, accuracy, and completeness ratings, in case it is a claim evidence | **Datum**<br><br>Contribution data: voting and/or evidence data (claim ID, current vote tally, content refs, etc.)<br><br>Topic with state = `EvidenceProposed` or `VoteCasted` or `VerdictTagSelected`<br><br>**Redeemer**<br><br>Operation<br>**`review_contribution`** | Validate contribution evaluation<br><br>Update state to `ContributionReviewed` or flag for dispute if necessary. | UTXO with datum updated with contribution details and state = `ContributionReviewed` or `ContributionDisputed` |
| Provenance | `VerifyContribution`<br>(spend) | Finalizes a contribution as verified after thorough assessment | **Datum**<br><br>Contribution data<br><br>Topic with state = `ContributionReviewed` or `ContributionDisputed` | Confirm the contribution meets validation criteria<br><br>Update state to `ContributionVerified` | UTXO with datum updated with contribution details and state = `ContributionVerified` |

| | | | | | |
|---|---|---|---|---|---|
| | | | **Redeemer**<br><br>Operation<br>**verify_contribution** | | |
| Provenance | DisputeContribution<br>(spend) | Initiates dispute resolution for a contested contribution. | **Datum**<br><br>Contribution data (under review or disputed)<br><br>Topic with state = ContributionReeviewed<br><br>**Redeemer**<br><br>Operation<br>**dispute_contribution**<br><br>(with dispute rationale) | Validate the dispute conditions<br><br>Transition the state to ContributionDisputed and lock the contribution pending resolution | UTXO with datum updated with state = ContributionDisputed |
| Provenance | UpdateContribution<br>(spend) | Updates a disputed contribution with additional evidence or corrections | **Datum**<br><br>Contribution data<br><br>Topic with state = ContributionDisputed<br><br>**Redeemer**<br><br>Operation<br>**update_contribution**<br><br>(includes new evidence/updates) | Merge new information with existing data<br><br>Recalculate scores<br><br>Trigger new contribution assessment | UTXO with datum updated with contribution details |
| Provenance | RejectContribution<br>(spend) | Rejects a contribution that fails verification | **Datum**<br><br>Contribution data<br><br>Topic with state = ContributionDisputed<br><br>**Redeemer**<br><br>Operation<br>**reject_contribution** | Validate rejection criteria<br><br>Update state to ContributionRejected | UTXO with datum updated with state = ContributionRejected |

| | | | | | |
|---|---|---|---|---|---|
| Provenance | `EvaluateContribution` (spend) | Aggregates final scores for a verified contribution to determine its overall impact and quality | **Datum**<br>Verifying contribution data and previous score components<br>Topic with state = `ContributionDisputed`<br>**Redeemer**<br>Operation **`evaluate_contribution`**<br>(includes final scoring metrics) | Aggregate scores with appropriate weightings (e.g., expert and general reviews)<br>Finalize contribution evaluation<br>Update state to `ContributionEvaluated` | UTXO with datum updated with final score and state = `ContributionEvaluated` |
| Provenance | `CheckPool` (spend) | Aggregates final scores for a verified contribution to determine its overall impact and quality | **Datum**<br>Reward pool details (e.g. fund available, distribution so far)<br>Topic with state = `ContributionRejected` or `RewardsDistributed`<br>**Redeemer**<br>Operation **`check_pool`**<br>(includes final scoring metrics) | Validate pool consistency, verify funds availability, and trigger adjustments or alerts if discrepancies are found | UTXO with datum updated with final score and state = `PoolEvaluated` |
| Reward distribution | `DistributeRewards` (withdraw) | Distributes tokens to contributors based on evaluated scores, drawing from a reward pool | **Datum**<br>Contribution scores and reward pool details<br>**Redeemer**<br>Operation **`distribute_rewards`**<br>(specifies distribution details) | Calculate the proportional reward based on contribution scores and available funds<br>Validate conditions for reward release, then mint (or withdraw) and transfer tokens to the user's wallet address, updating the reward pool accordingly | Updated token balances<br>New UTXO reflecting state as `RewardsDistributed`<br>Remaining reward pool updated |

| | | | | | |
|---|---|---|---|---|---|
| Governance | `Staking / Unstaking`<br><br>(not a dSM transition) | Allows users to lock tokens to participate in governance and earn additional rewards | **Datum**<br><br>User stake amount and staking parameters<br><br>**Redeemer**<br><br>Operation **`stake_tokens`** | Validate available token balance, lock tokens in the staking contract, and update the user's staking status<br><br>Manage state transitions for `Staking` and `Unstaking` actions as required by the governance model | UTXO updated with staked tokens<br><br>User's stake recorded for governance participation |
| Minting | `Minting`<br><br>(not a dSM transition) | Mints new $DFCT tokens based on predefined issuance rules, such as governance approvals or reward pool funding. | **Datum**<br><br>Minting policy parameters (e.g., supply cap, distribution rules)<br><br>**Redeemer**<br><br>Operation **`mint_dfct`**<br><br>(specifies amount, purpose, and minter authorization) | Validate minting conditions: ensure minter authorization, check against total supply limits, enforce governance constraints if applicable | New UTXO with the minted $DFCT tokens assigned to the designated address<br><br>Updated datum with new supply details |

## Frameworks and programming languages

Currently, the official Cardano documentation[5] lists Plutus, Aiken, Marlowe, Opshin, and plu-ts as the languages and frameworks available for creating smart contracts. Plutus is Cardano's official smart contract language, built on Haskell and fully integrated with the Extended UTXO (EUTXO) model. It supports the development of both on-chain validators and off-chain components via the Plutus Application Backend (PAB), offering a comprehensive and mature ecosystem with extensive libraries and documentation. While it provides a reliable framework for complex operations like reward distribution, staking, and governance, its steep learning curve and functional programming paradigm may require additional training, potentially slowing initial development efforts.

Aiken is a modern language designed specifically for Cardano smart contracts, offering a simplified syntax that makes it more accessible than traditional Haskell-based languages. It is inspired by languages such as Gleam, Rust, and Elm and provides native support for on-chain validators. The project is also developing a robust toolkit for off-chain integration, which can streamline DApps development. Its modern design promises ease of use and efficient coding for complex logic, though its relative newness means that documentation and community support are still evolving compared to more established solutions.

[5] https://developers.cardano.org/docs/smart-contracts/

Marlowe is a high-level, domain-specific language tailored for financial contracts, featuring a declarative syntax that minimizes coding complexity. It is engineered for secure and safe contract creation, particularly in financial applications like payments or staking. While Marlowe excels in reducing the risk of errors through its high-level abstractions, its narrow focus on financial contracts limits its flexibility for general-purpose applications, making it less suitable for the diverse use cases required by d-FCT.

Opshin is an implementation of smart contracts for Cardano which are written in a strict subset of valid Python. The language interacts closely with the python library PyCardano. The internal data structures are defined with data types compatible with this library and allow a tight combination of off- and on-chain code, all written in Python. Despite its promise in delivering robust security features, Opshin's limited adoption, relatively sparse documentation, and community support can pose challenges for long-term stability and ease of development.

plu-ts offers a contemporary approach by combining familiar TypeScript syntax with the capability to compile to Plutus validators, making it an attractive option for teams experienced in JavaScript/TypeScript. This framework smooths the integration with d-FCT's ReactJS frontend and Python backend, reducing the learning curve compared to traditional Plutus development. However, as an experimental tool with fewer production examples and less maturity than Plutus, plu-ts might lack some advanced features and stability, necessitating careful evaluation during development.

Table 5 summarizes the language and framework options for designing Cardano smart contracts, each with its distinct strengths and trade-offs considering the d-FCT ecosystem. Plutus, being Cardano's native smart contract language, offers a robust and deterministic environment ideal for complex on-chain operations like reward distribution, contribution scoring, and governance. Its Haskell-based syntax poses a steep learning curve though. Aiken provides a more accessible alternative with simplified syntax, but its ecosystem is still maturing. Marlowe is user-friendly for financial contracts, but lacks the flexibility needed for d-FCT's broader fact-checking requirements. Opshin is promising but currently limited in documentation and support, while plu-ts offers a TypeScript-based approach that aligns well with d-FCT's ReactJS frontend, facilitating smoother integration for developers familiar with JavaScript/TypeScript.

Table 5 – Cardano smart contract languages and frameworks.

| Language / Framework | Syntax and description | Pros | Cons | On-chain / Off-chain integration |
|---|---|---|---|---|
| **Plutus** | Haskell-based smart contract language with strong static typing and functional programming paradigms | Native to Cardano; mature toolchain; robust validator development; deterministic semantics | Steep learning curve<br><br>Integration with Python backend may require additional bridging | Directly supports on-chain validator creation via the EUTXO model and offers off-chain support through the Plutus Application Backend (PAB) |
| **Aiken** | A newer, more approachable smart | Easier learning curve; faster development | Ecosystem is less mature; limited tooling | Supports on-chain development for |

| | | | | |
|---|---|---|---|---|
| | contract language designed specifically for Cardano, with a simpler and less verbose syntax than Plutus | cycle; more accessible to developers unfamiliar with Haskell | and community support compared to Plutus; evolving documentation | validators while aiming for improved integration and easier interoperability with off-chain systems |
| **Marlowe** | A domain-specific language (DSL) for financial contracts on Cardano with a highly specialized, user-friendly syntax (including a visual interface) | High-level abstraction; excellent for financial contracts; accessible for non-programmers | Limited to financial use cases; not flexible enough for general-purpose fact-checking and dynamic workflows required by d-FCT | Well-suited for on-chain financial operations, but not ideal for the broader verification, governance, and reward distribution needs of d-FCT |
| **Opshin** | Python-based language that aims to simplify smart contract development on Cardano, offering a more intuitive syntax | Potentially simpler syntax and better integration with non-Haskell ecosystems; lowers the barrier to entry | Limited documentation and community support; maturity and stability are less proven compared to established options | Support on-chain validators and basic off-chain interactions, though specifics depend on ongoing development |
| **plu-ts** | A TypeScript-based library or wrapper for Plutus, allowing developers to write smart contracts using TypeScript syntax, which is more familiar to JavaScript developers | Leverages the familiarity of TypeScript; easier integration with a ReactJS frontend; smoother developer experience for JS/TS programmers | May offer less direct control over advanced contract logic compared to native Plutus; relies on underlying Plutus functionality, which might limit access to some advanced features | Provides an abstraction for on-chain smart contract development while potentially easing off-chain integration with modern web and backend technologies (e.g., REST, GraphQL), facilitating connectivity between Python and ReactJS components |

The project will leverage Plutus for the development of smart contracts, which will be deployed on the Cardano testnet, ensuring secure execution and compliance with the platform's functional programming paradigm. On the backend, PyCardano, a Python library for Cardano transaction construction and serialization, will be utilized to interact with the smart contract and facilitate transaction processing. The frontend will use a TypeScript library such as plu-ts or Lucid for communication with the Cardano blockchain, enabling a user-friendly interface for contract interactions.

# 6. Final remarks

The d-FCT platform represents a groundbreaking fusion of decentralized technologies and conventional web infrastructure to tackle misinformation with unprecedented transparency and efficiency. Our technical report has detailed how the platform's architecture is designed to handle and verify user-submitted content through a combination of specialized components. Smart contracts will manage critical operations—ranging from provenance to governance—ensuring immutable, tamper-proof execution of the decentralized verification process. The integration of a structured scoring mechanism and contribution assessments, combined with reward pools, aims at creating an ecosystem that not only incentivizes high-quality contributions but also safeguards against abuse.

Looking forward, d-FCT's design paves the way for scalable and secure fact-checking on the Cardano testnet, with opportunities to enhance further via decentralized storage solutions and evolving smart contract frameworks. As the platform continues to mature, future iterations may incorporate additional governance features and refined off-chain analytics, reinforcing its role as a transformative tool for decentralized and transparent knowledge. The synergy between on-chain operations and off-chain processing, along with continuous advancements in AI and blockchain technologies, positions d-FCT as a pioneering model for decentralized verification and community-driven truth validation.

# Acknowledgments

This work was supported by a grant from the Fund13 of Project Catalyst. We would like to express our gratitude to the Cardano community and the Project Catalyst Team.

# References

https://milestones.projectcatalyst.io/projects/1300076

https://developers.cardano.org/docs/smart-contracts/