

Qubic Auditor

AI-Powered C++ Smart Contract Analysis Tool for Qubic

Built for the RAISE Hackathon 2025 – Qubic Track

Developed by: TrustStack Auditors Qubic Track



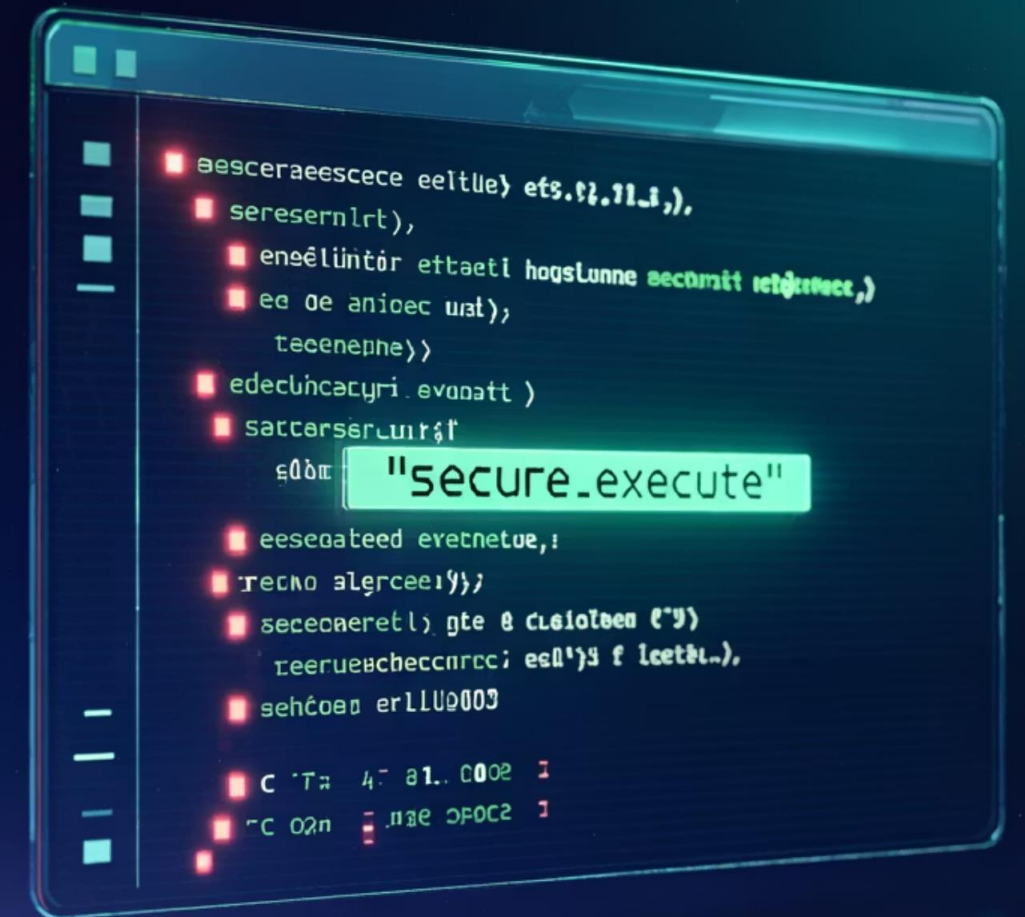
Problem & Opportunity

The Problem:

C++ smart contracts on Qubic offer flexibility and control but they are not gas-optimized, are prone to vulnerabilities, and are not aligned with industry standards.

The Opportunity:

Enable developers, investors, and security teams to audit Qubic contracts efficiently using AI and intelligent code analysis.



Solution – Qubic Auditor

An AI-powered tool that performs:



Vulnerability detection

Identifies security issues with severity levels



Gas optimization

Provides insights for improving contract efficiency



Code quality grading

Evaluates overall code quality and best practices



Exportable reports

Generate audit reports in PDF/text formats



Real-time results

View analysis with a modern user interface

Tech Stack



Frontend:

React 18 (VITE), TypeScript, Tailwind CSS



AI Engine:

Google Gemini 2.5 Flash/Pro



Deployment:

Docker



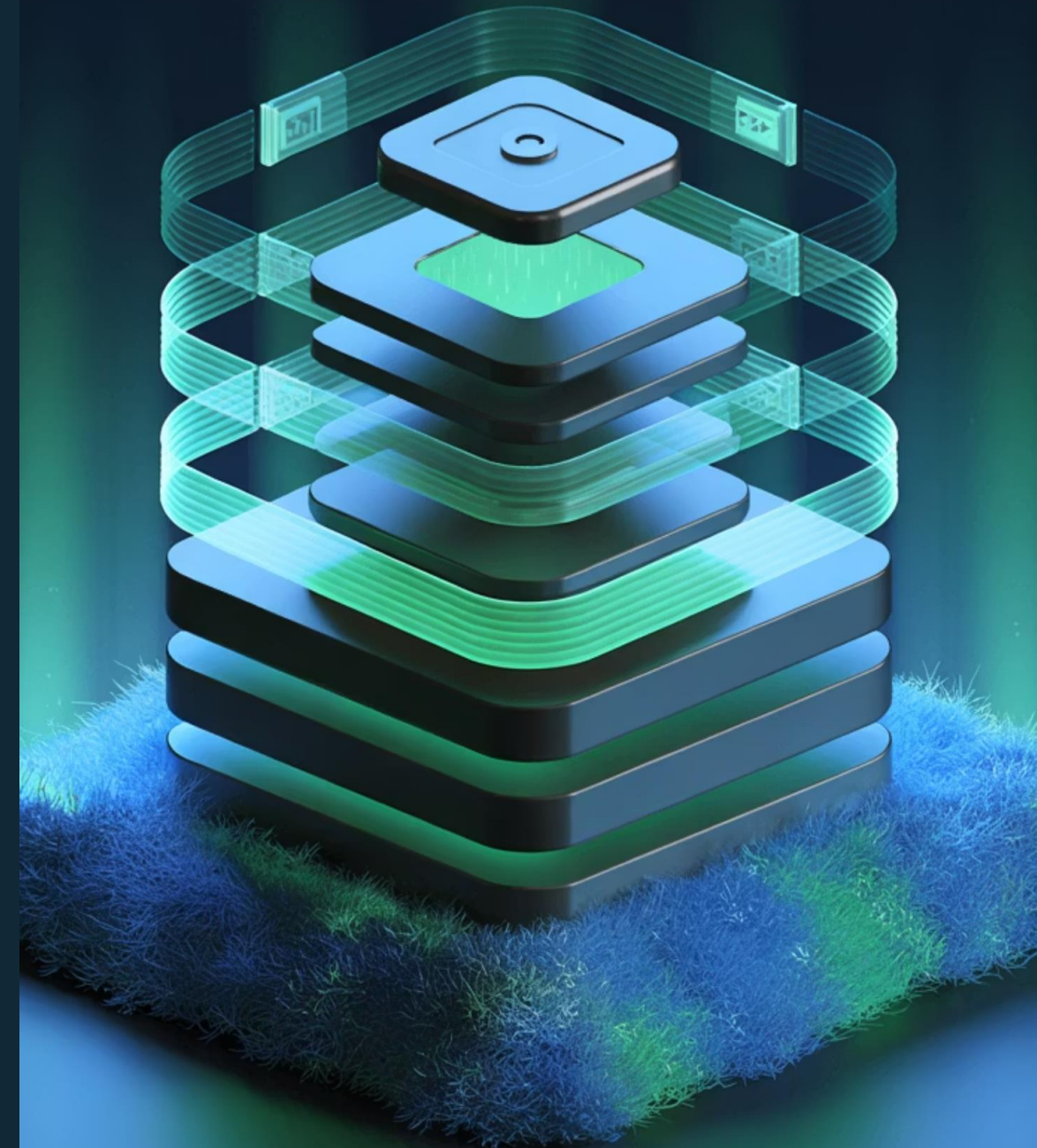
Backend:

Node.js, Express



Design System:

shadcn/ui, Radix UI



Project Flow

Upload C++ File

Start by uploading your smart contract

Gemini AI Analysis

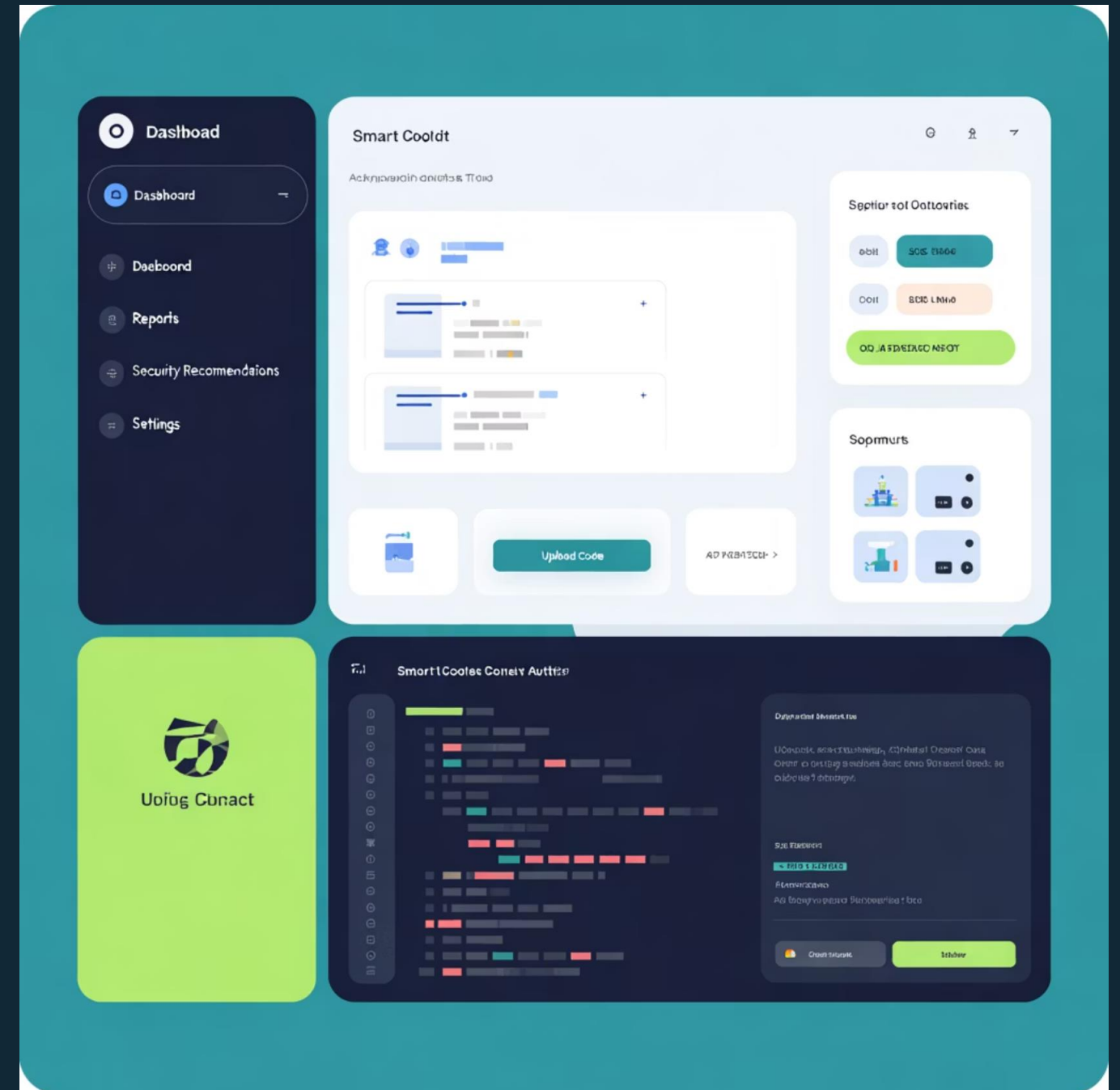
AI processes and analyzes the code

Security, Gas, and Quality Insights

View comprehensive analysis results

Export or Copy Optimized Code

Save your report or implement suggestions



How It Works

Data Flow:



Upload C++ contract

User submits smart contract code for analysis



File stored + metadata

System securely stores the contract and relevant information



AI triggers async analysis

AI performs security, gas & quality analysis in parallel



Results displayed in real-time

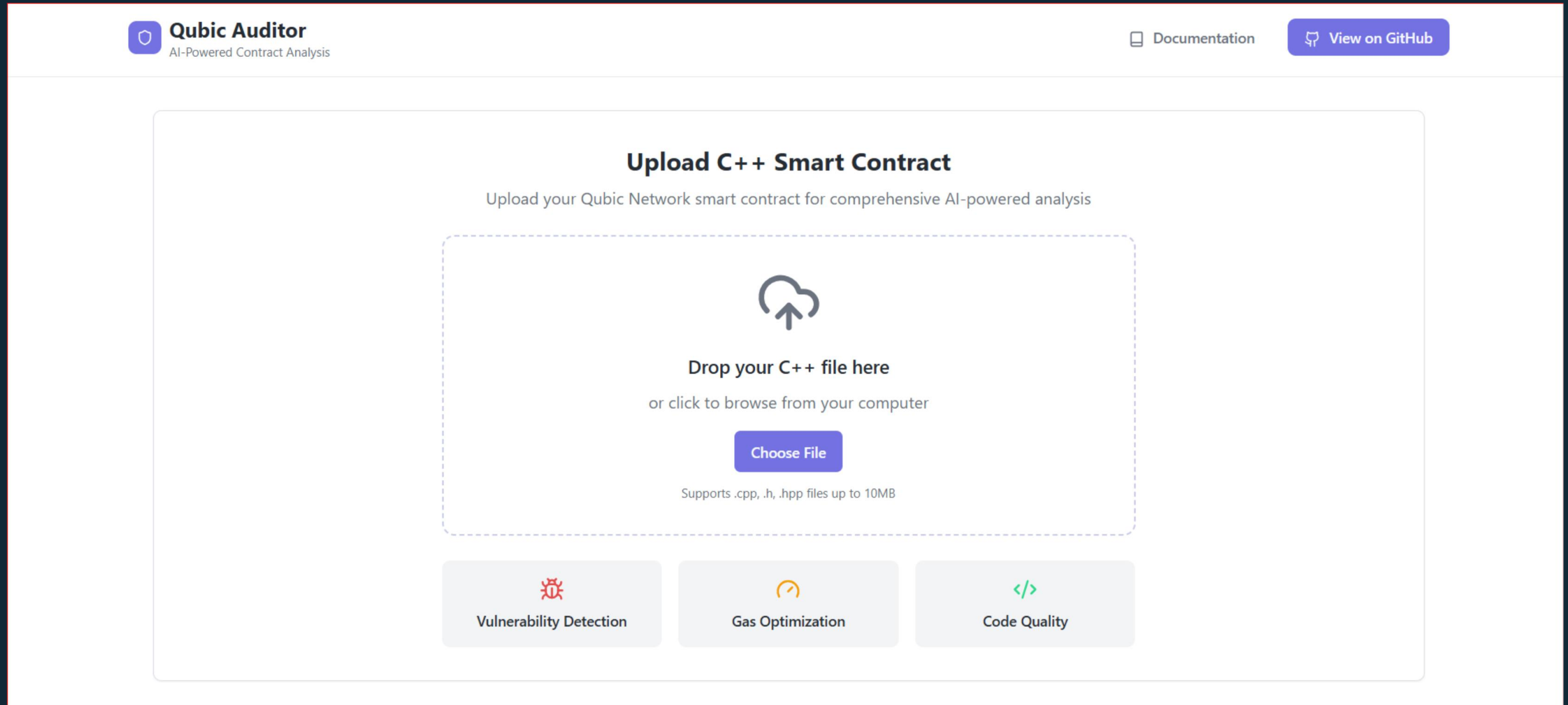
Findings appear on interactive dashboard as they're discovered




Export final report or apply suggestions

User can save findings or implement recommended changes

Application Frontend





Auditing In Progress


**Qubic Auditor**
AI-Powered Contract Analysis


[Documentation](#)[View on GitHub](#)

Analysis in Progress

 Contract parsed successfully

 Running security analysis...

 Gas optimization analysis

 Code quality review

Estimated time: 2-3 minutes

Code Quality Analysis

Code Quality Analysis

Overall grade: **C**

Documentation

55%

Test Coverage

71%

Code Complexity

Low

ⓘ Inefficient Voter Check (O(n)) Leads to High Gas Costs

The `VOTE` function iterates through all previously cast votes (`state.voters`) to check if a user has already voted on a proposal. This is a linear search with $O(n)$ complexity, where n is the total number of votes. In a smart contract context, this is highly inefficient and will result in transaction costs (gas fees) that increase with every vote cast, making the contract prohibitively expensive to use as it scales.

Line 81

ⓘ Flawed and Inefficient Voter State Management

The state design for tracking votes is flawed. The `voters` array stores a `VoterRecord` for every single vote, not for each unique voter. The constant `MAX_VOTERS` is therefore misleading; it should be `MAX_VOTES`, as it limits the total number of votes that can ever be cast across all proposals. This data structure does not scale well and is an inefficient use of storage.

Line 34

⚠ `GET_RESULTS` Method Should Be `const`

The `GET_RESULTS` function reads from the contract's state but does not modify it. Therefore, it should be declared as a `const` member function (`void GET_RESULTS(...) const`). This helps enforce correctness by allowing the compiler to catch accidental state modifications and makes the function's read-only nature clear to developers.

Line 107

Gas Optimization Report

← → ↻ ⓘ localhost:5173

☆ 6

🔖

🏠

📖 All Bookmarks

🚩 Vulnerabilities

🔍 Gas Optimization

</> Code Quality

📄 Full Report

Gas Optimization Opportunities

Potential savings: NaN% reduction

⚡

Use Smaller Integer Types for Vote Counts

Potential gas savings: 3-5%

The `Proposal` struct uses `uint32` for `yesVotes` and `noVotes`. Since `MAX_VOTERS` is defined as 1000, these counts will never exceed this value. Using the smaller `uint16` type is sufficient, reducing the storage size of each `Proposal` object by 4 bytes. This leads to overall state size reduction and cheaper storage writes when vote counts are updated.

❌ Current (Higher Gas Cost)

struct Proposal { uint32 id; uint32 yesVotes; uint32 noVotes; bool isActive; };

✅ Optimized (Lower Gas Cost)

struct Proposal { uint32 id; uint16 yesVotes; uint16 noVotes; bool isActive; };

Lines affected: 23, 24, 25, 26, 27, 28

🔄 Apply Fix

⚡

Remove Redundant Field from `VoterRecord`

Potential gas savings: 5-10%

The `VoterRecord` struct contains a `hasVoted` field that is always set to true when a record is created. The mere existence of a `VoterRecord` for a specific voter and proposal pair is enough to confirm a vote has been cast, making this field redundant. Removing it saves storage for every vote and eliminates one storage write operation from the `VOTE` function, resulting in significant gas savings.

❌ Current (Higher Gas Cost)

struct VoterRecord { m256 address; uint32 proposalId; bool hasVoted; };

✅ Optimized (Lower Gas Cost)

struct VoterRecord { m256 address; uint32 proposalId; };

Comprehensive Audit Report


←


→


↺


localhost:5173

☆

6









All Bookmarks

Comprehensive Audit Report

 Print

 Export Report

Executive Summary

Contract Details

File: smart_contract.cpp

Lines of Code: 115

Functions: 6

Analysis Date: 7/5/2025

Key Findings

Critical Issues: 0

High Severity: 1

Medium Severity: 3

Low Severity: 1

Security Assessment

The smart contract has been thoroughly analyzed for security vulnerabilities using AI-powered analysis. The overall security score is 4.5/10.

- HIGH severity: Voter Identity Spoofing via Unvalidated Input (Line 85)
- MEDIUM severity: Unrestricted Proposal Creation Leading to Denial of Service (Line 51)
- MEDIUM severity: Denial of Service via Gas Exhaustion in Voting Check (Line 76)
- And 2 more issues...

Gas Optimization Report

Current gas efficiency is rated at 97%.Several optimization opportunities have been identified:

- Use Smaller Integer Types for Vote Counts - Potential 3-5% reduction



Business Potential (SaaS Model)

Qubic Auditor as a SaaS:



Subscription plans for:

- Dev teams
- Auditing firms
- DAOs



Features:

- Bulk contract audits
- API integration
- Custom reports
- CI/CD security hooks



Monetization:

- Monthly/yearly tiers
- Enterprise audit packages
- Pay-per-audit credits

Future Enhancements

Multi-contract auditing

Support for analyzing multiple contracts simultaneously

Offline CLI version

For air-gapped environments requiring maximum security

Live vulnerability alerts

Real-time monitoring for deployed contracts



AI-assisted auto-fix

Proposed changes applied in one click

GitHub/GitLab integration

Seamless connection with version control systems

Qubic → EVM bridge audit

Specialized module for cross-chain contract auditing



Impact & Vision

"Empowering secure adoption of Qubic's smart contract ecosystem."

- 🚀 Drives adoption via **trust and transparency**

- 🔒 Makes C++ smart contract development **safer and faster**

- 🌉 Bridges gap between AI and decentralized systems