iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.81941

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

Hybrid Blockchain E-Voting with Face-Verified Registration and On-Chain Tallying

Aakashram M[1], Karthick K[2], N. Ananda Kumar[3], Dhanush K[4], Akash B[5]
Computer Science and Engineering Arunai Engineering College Tiruvannamalai, India

Abstract: ***Electronic voting demos often split into two awk- ward halves: either a polished web form with a spreadsheet behind it, or a blockchain prototype that ignores who is eligible to vote. We built something in-between for a final-year project— a Flask web application that gates registration behind live face recognition, stores credentials in SQLite for session login, and records each accepted ballot as a Solidity smart-contract transaction on a local Ethereum node (Ganache). The contract maintains candidates, increments vote counts atomically, and maps voter string IDs so duplicates revert on- chain. Signing stays server-side—centralizing trust but matching introductory lab setups. Results cover manual regressions, timing bands, and limitations, not cryptographic proofs of electoral integrity.***
Index Terms: electronic voting, smart contracts, Ethereum, Web3, face recognition, Flask, SQLite

I. INTRODUCTION

A. Motivation

Paper and lever machines offer physical audit trails; pure web forms offer speed but weak integrity stories when a single database row can be edited in silence. Permissionless ledgers at least make aggregate transitions replayable, which is why university projects keep returning to Solidity for counting logic [1], [2]. The catch is always enrollment: a smart contract cannot, by itself, know whether voter123 is one human or many sharing a password. Blockchains replay state transitions, which aids tally trans- parency, but eligibility policies stay off-chain. We combine (i) SQLite-backed registration guarded by webcam face sim- ilarity to a labeled gallery and (ii) a Solidity tally keyed by voter string IDs with duplicate suppression, deployed on Ganache for zero-cost resets between batches. The question we answer is engineering integration— wiring biometrics, HTTP sessions, and JSON-RPC in one repo students can run on a laptop—while stating openly what we did *not* secure (liveness, coercion, key custody).

B. Problem Statement

Given a closed set of sample voters represented by frontal images on disk, the system shall (1) block web registration unless a live webcam frame matches one of those encodings within an adjustable tolerance, (2) bind a logged-in voter ID to exactly one recorded choice on-chain, and (3) expose read-only aggregates to an administrator UI. Constraints were purely practical: no paid cloud APIs, Python-first tooling, So- lidity ≥ 0.8 for student-friendly syntax, reproducible teardown between batches.

C. Objectives

We targeted four objectives: (i) Flask routes for voter registration and login backed by SQLite; (ii) OpenCV-assisted face verification against labeled images before registration; (iii) Solidity contract deployed via py-solc-x and interacted with using Web3.py; (iv) a simple administrator view of aggregated results pulled from chain state.

D. Contributions

1) A minimal end-to-end stack combining biometric gating at registration, relational credentials, and on-chain tally- ing keyed by voter string IDs.
2) A documented deployment workflow (compile → deploy→ write ABI and address into JSON consumed at run- time).
3) Open discussion of security shortcuts taken for prototyp- ing and realistic next steps toward something deployable.

E. Paper Organization

Section II reviews blockchain voting and biometric con- text. Section III lists the Python/Web3 stack [5]–[8], [11]. Sections IV–VI present architecture, implementation (routes, face gate, transactions), and smart-contract details. Section VII evaluates behavior and timing. Section VIII discusses trade- offs. Section IX concludes.

©IJRASET: All Rights are Reserved

II. RELATED WORK

A. Blockchain Tallying Versus Identity

Survey literature stresses that append-only ledgers can strengthen auditability of published totals while leaving voter authentication, ballot secrecy, and coercion resistance to sur- rounding policy and software [1], [2]. Ethereum-style virtual machines provide deterministic execution of counting rules [4]. Local emulators such as Ganache expose the same JSON-RPC methods as remote nodes, which keeps student code portable between lab PCs [7]. Our split—credentials in SQLite, counts in Solidity—matches that teaching pattern.

B. Biometrics as a Workshop Gate

Face embedding libraries built on dlib and exposed through Python [9] lower the barrier to experimenting with "something you are" checks. Spoofing surveys nonetheless document print, replay, and partial presentation attacks that defeat na¨ıve cameras [3]. We therefore treat the webcam step as a curriculum device that forces teams to discuss enrollment threat models, not as evidence-grade identity.

C. Compared to Turnkey E-Voting Products

Commercial systems bundle key management, logging, high availability, and certification paths. A minimal Flask prototype trades those features for inspectable source that graders can walk line-by-line. The limitation is intentional: we optimize for pedagogy, not feature parity with national-scale vendors.

III. TECHNOLOGY STACK

Table I summarizes responsibility boundaries. Python 3 hosts Flask for synchronous request handling and Jinja templates, avoiding SPA build pipelines for a single- semester schedule [5]. Signed cookies carry voterid after login. SQLite stores demo credentials in one file for USB-copy deployments [11]. OpenCV captures frames; face_recognition maps faces to 128-D vectors com- pared by Euclidean distance [9]. Web3.py connects to Ganache at 127.0.0.1:7545, hydrates contract objects from ABI JSON, and signs transactions with a development private key [6], [7]. The compiler interface py-solc-xpins Solidity 0.8 [8]; successful deploy writes contract_info.json con- sumed when Flask starts—changing address or ABI requires both redeploy and process restart.

TABLE I

MAJOR COMPONENTS AND ROLES

Layer	Role in prototype
Flask	HTTP UI, session, coordinates DB and Web3
SQLite	Voter credentials (demo schema)
OpenCV+FR	Webcam ingest, embeddings vs. dataset
Web3.py + Ganache	Transaction build, sign, broadcast, receipts
Solidity contract	Canonical vote counts & duplicate map
JSON ABI file	Glue between bytecode address and Python calls

IV. SYSTEM ARCHITECTURE

Fig. 1 shows browser→Flask→{SQLite, camera, Ganache→contract}. Only the Flask process signs transactions; end users never export private keys through the browser.

A. Trust Scope

We assume a cooperative lab: administrators control Ganache mnemonic accounts, the workstation is physically shared by the project team, and attackers do not remotely exploit Flask or exfiltrate voting.db. Under that narrow model, the ledger still helps *in-team* arguments about whether totals were altered between demo runs, because anyone can re-read getVoteCount after the fact. Collapse any of these assumptions and the story changes immediately—which we state in Section VII.

B. Runtime Preconditions

Ganache must be listening before launching Flask so provider calls during import succeed. The operator grants webcam permission once per OS session. dataset/ im- ages must load without Unicode path bugs on Windows. We ship vendored dlib/OpenCV wheels so first-year maintainers avoid compiling native extensions during setup week.

Table II contrasts our hybrid against a naive web poll used in many classroom demos.

TABLE II
CLASS WEB POLL VERSUS HYBRID PROTOTYPE (QUALITATIVE)

Concern	Forms + DB only	Faces + Flask + chain
Tally stealth edits via SQL	High risk	Counts anchored in contract state
Eligibility ritual	Password checkbox	Mandatory face-vs-gallery gate
Cost / infra	Minimal	Ganache VM + Solidity deploy
Key custody narrative	Not applicable	Server-signed txs (trust hotspot)

C. Data Model

SQLite. Columns: voterid, password. Duplicate IDs are rejected at insert time; Flask flash messages carry the same feedback the database error would have implied.

Chain. The contract stores candidates with vote counts and a mapping from string voter identifiers to Booleans enforcing single submission per ID.

V. IMPLEMENTATION

Environment recipe: Python venv, vendored wheels for dlib/OpenCV, run deploy_contract.py, refresh contract_info.json. Disable Flask auto-reload while debugging webcam locks.

A. HTTP Route Inventory

Table III maps URLs to behavior. All HTML lives under templates/. Flash categories (success, danger, etc.) drive Bootstrap-style alerts without a separate front-end state store.



TABLE III
FLASK ROUTES (EXCERPT)

Path	Verb	Effect
/	GET	Landing page with navigation
/user	GET	Webcam face gate → register or error flash
/register	GET/POST	Create voter row if unique ID
/login	GET/POST	Session bind on credential match
/vote	GET/POST	Read candidates; POST signs chain tx
/result	GET	Tally table (admin path in practice)
/admin	GET/POST	Demo admin password → tallies
/logout	GET	Clear session

B. Face-Verified Enrollment Path

Route /user encodes authorised gallery images (.jpg/.jpeg/.png); live frames from VideoCapture(0) compare via minimum Euclidean distance to known embeddings—accept < 0.45 (manual tuning lab-only), quit via q aborts.

Algorithm 1 (face gate, informal)

1) Load known encodings $E = \{e_1, \ldots, e_k\}$ from disk.
2) Until stop: capture frame F ; convert F to RGB; derive face encodings U for all faces in F .
3) For each u ∈ U , compute di = ‖u − ei ‖2; if mini di < 0.45, accept and exit success.
4) On key qor hardware read failure, exit failure.

Successful runs redirect into /register; failures return to index with a red flash message so the operator understands the pipeline order.

C. Authentication and Voting

Registration inserts credentials; login places voterid inside the session cookie. The vote page first issues a read-only get Candidates() .call() to populate ra- dio buttons. Before building a write, the backend checks hasVotedCheck(voterid) to short-circuit duplicates with a UI message instead of burning a failed transaction. Submission builds vote(voterId, candidateIndex), gas capped generously ($\sim 2\times10^6$), signs, broadcasts, waits for receipts; Flask flash prints revert texts. Restarting Ganache without redeploying leaves stale ABI addresses—always rerun deploy→contract_info.json→Flask reboot.
Throughput. Webcam pacing dominated wall clock; con- firmations were instantaneous on Ganache.

D. Administrator Result View

Separate admin POST checks demo credentials before rendering totals via getVoteCount—never mutating chain state.
Integration quirks: OpenCV DLL mismatches and zom- bie camera captures on Flask reload were cured by pinned wheels plus killing orphaned Python shells; mismatched nonce after chain reset taught JSON-RPC etiquette.
Configured literals. Ganache HTTP 127.0.0.1:7545; device index 0; Euclidean gate 0.45.

VI. SMART CONTRACT LOGIC

Struct array candidates[] plus mapping hasVoted keyed by voter string guards double-spend semantics. Con- structor seeds names once; external vote rejects repeats and out-of-range indices atomically.
Read helpers populate UI/admin: Table IV.

TABLE IV
CONTRACT INTERFACE EXPOSURE

Function	Type	Consumer logic
constructor(names[])	write	Deploy script
vote(id, idx)	write	Voting POST
getCandidates()	view	Populate ballot UI
getVoteCount(i)	view	Admin & result page
hasVotedCheck(id)	view	Duplicate guard

Deploy script wraps compile_source (py-solc-x), signs bytecode deploy with funded Ganache keys, persists ABI/address consumed by Flask [6], [8]. Constructor ar- guments supply the candidate slate (configured in Python before broadcast); resetting an election implies redeploying with a fresh name array. Scope stays integration-first: no zk, homomorphic aggregates, or anonymous credential layer.
Gas practice. During Ganache trials we inflated gas limits on both deploy and vote calls to absorb opcode edits without out of gas; tuning budgets would precede any test-net migration.

A. Security Posture (Honest Inventory)

- Private keys and admin passwords are embedded for lo- cal iteration; production demands environment variables, hardware wallets, or custodial patterns.
- Passwords are not hashed in the prototype path we maintained; migrate to salted slow hashes before any shared deployment.
- Biometric matching is replayable with a photo; liveness detection absent.
- Server signs all votes: voters do not control personal keys, so non-repudiation is weak.

VII. EVALUATION

A. Method and Outcomes

We paired tester/observer roles across three indepen- dent Ganache workspaces to decorrelate configuration drift.
Scripted paths included: SQLite-only enrollment sanity, suc- cessful face gate → register, bogus password rejection, first successful vote, intentional second vote on the same ID, admin tally reconciliation against raw contract getters, and camera sessions under fluorescent versus window-side glare. Euclidean threshold 0.45 arose from unstructured trials with classmates; we did not run a biometric benchmark suite.

Table V aligns stated objectives from Section I with concrete artefacts inspectors can click through in the codebase.

TABLE V
OBJECTIVES MAPPED TO ARTEFACTS

Objective (Section I)	Where it appears
Flask routes + SQLite auth	`/register`, `/login`, voting.db
Face gate before signup	Algorithm 1; route `/user`
Solidity tally + Web3.py	Voting.sol, `/vote`, deploy script
Administrator totals	Routes `/admin`, `/result`; getVoteCount

Table VI records regression checks we repeated before graded demos.

TABLE VI
MANUAL TEST MATRIX

Check	Observed	Notes
Face match → register	Pass	Lighting sensitive
Duplicate ID register	Blocked	Expected
Login / session guard	Pass	Redirect if missing
Second vote same ID	Rejected	Contract require
Post-vote counts	Consistent	Reads after mining

B. Approximate Timing

Table VII splits millisecond-grade operations from multi- second biometric work so one axis does not lie about mag- nitude. Instruments combined Chrome DevTools navigation timing, Python-side logging around Web3 calls, and handheld stopwatches during face loops on a commodity Core i5 laptop with a mechanical disk.

TABLE VII
REPRESENTATIVE LATENCY BANDS (LAB MEDIANS)

Operation	Typical	How measured
SQLite voter probe	~35 ms	Tight SELECT loop
getCandidates() RPC	~95 ms	Web3 view call
Full login navigation	~420 ms	DevTools waterfall
vote()to receipt	~380 ms	Receipt waiter
Face gate success path	~6–12 s	Wall clock
Glare / poor angle stall	$\geq$ 20 s	User abandoned

Fig. 2 visualizes only the sub-second buckets (milliseconds) shared on one axis.

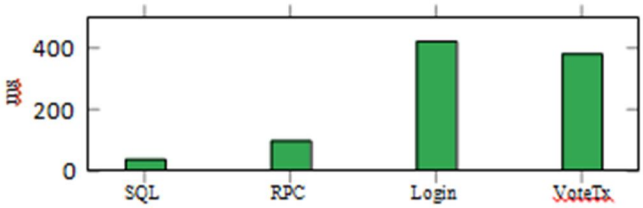


Fig. 2. Millisecond-scale operations only; webcam biometrics dominate end-to-end time separately.

Interpretation. HCI guidance suggests interactions under ~1 s feel fluid for many tasks [12]; our chain and DB layers cleared that bar while biometric capture did not.

C. Threats to Validity and Lessons

Construct validity. Passing Table VI means "student demo readiness," not "secure public election."

Internal validity. Authors built and tested the same code- base; informal peer walkthroughs mitigated tunnel vision but cannot replace independent red teams.

Operational lessons. Reviewers challenged photo spoofing immediately; admitting centralized signing undercut naïve "trustless" rhetoric but shortened debates. Migrating Ganache workloads to faucet-funded Sepolia remains the pragmatic next fidelity step for network delay awareness.

D. Laboratory Testbed

Runs used Windows 10/11 lab PCs (Intel Core i5 class, 8–16 GB RAM), Python 3.10 inside a virtual environment, Ganache exposing 7545, Solidity 0.8 via py-solc-x, and Chromium-based browsers for UI capture. Timings in Ta- ble VII are indicative of that stack, not cloud hardware.

VIII. DISCUSSION

1) Hybrid trust placement. Storing salted passwords off- chain while anchoring ballots on-chain is logically consistent when one labels each layer—identity assurance versus tally audit—rather than implying the chain solves both.

2) Teaching value. Students watched SQLite tampering dis- agree with untouched chain histories until redeploy erased state, making "what invariant lives where" concrete.

3) Feature gap list. Missing pieces for any serious pilot include bcrypt/Argon2 storage [10], per-voter wallets or blind issuance, coercion-resistant UI, observability dashboards, rate limits, cloud secrets management, auditable biometric enroll- ment with liveness.

IX. CONCLUSION

We stitched Flask biometric enrollment, SQLite sessions, Ganache-hosted Solidity tallying into one reproducible code- base. Ledger gains apply after identity questions are squared away [2]; biometric UX surfaces privacy sensitivities sooner than spreadsheets [3].

A. Future work roadmap

1) Migrate secrets to OS environment vars; rotate Flask secret_key;

2) Bcrypt or Argon2 password storage quoting Provos/Mazie`res style discipline [10];

3) Add challenge-response or blink liveness cues before honoring embeddings;

4) Optional Sepolia deployments with faucet-funded keys per student;

5) Wire continuous integration smoke tests invoking Ganache programmatically.

B. Closing remark

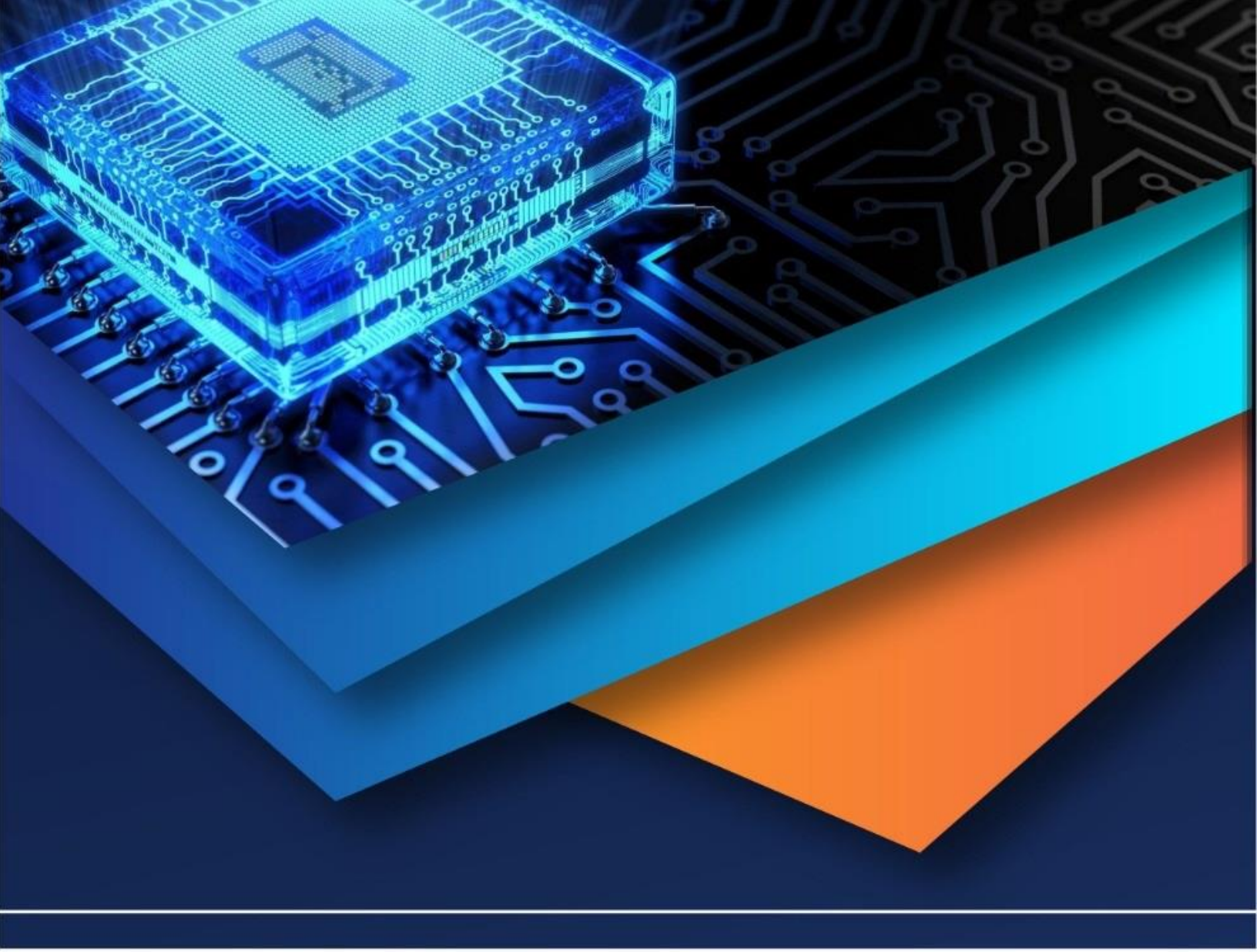
The repository is scaffolding for ex- perimentation: where replicated state helps tallies become inspectable stories, where biometrics distract from cryptog- raphy, and where months of toolchain debugging hide inside a twenty-line Solidity contract. Treating those truths explicitly was, for us, worth as much as the working demo itself.

X. ACKNOWLEDGMENT

The authors thank the Department of Computer Science and Engineering and the project guide at Arunai Engineering College for laboratory access and technical review.

REFERENCES

[1] N. Kshetri and J. Voas, "Blockchain-enabled e-voting," *IEEE Softw.*, vol. 35, no. 4, pp. 95–99, Jul./Aug. 2018.
[2] Z. Zhao et al., "Towards secure blockchain-enabled voting systems," in Proc. ACM Conf. Companion World Wide Web (WWW Companion), Lyon, France, 2018, pp. 1185–1186.
[3] A. K. Jain, K. Nandakumar, and A. Nagar, "Biometric spoof detection,"
[4] ACM Comput. Surv., vol. 51, no. 5, pp. 1–39, 2018.
[5] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Yellow Paper, 2014. [Online]. Available: https:// ethereum.github.io/yellowpaper/paper.pdf
[6] Python Software Foundation, "Flask Documentation," 2025. [Online]. Available: https://flask.palletsprojects.com/
[7] Web3.py Maintainers, "Web3.py Documentation," 2025. [Online]. Avail- able: https://web3py.readthedocs.io/
[8] Truffle Suite, "Ganache," 2025. [Online]. Available: https://trufflesuite. com/ganache/
[9] Solidity Team, "Solidity Language Documentation," 2025. [Online]. Available: https://docs.soliditylang.org/
[10] A. Rosebrock, "Face recognition with OpenCV, Python, and deep learn- ing," PyImageSearch, 2021. [Online]. Available: https://pyimagesearch. com/
[11] N. Provos and D. Mazie`res, "A future-adaptable password scheme," in
[12] Proc. USENIX Annu. Tech. Conf., 1999, pp. 81–91.
[13] SQLite Consortium, "SQLite Documentation," 2025. [Online]. Avail- able: https://sqlite.org/docs.html
[14] J. Nielsen, "Response times: The 3 important limits," Nielsen Norman Group, 1993. [Online]. Available: https://www.nngroup.com/articles/ response-times-3-important-limits/



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)