iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80492

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 IV Apr 2026- Available at www.ijraset.com

Ledger Link: A Scalable Blockchain Platform with Smart Contract Automation and Layer-2 Integration

Shubham Verma[1], Vishal Kumar Pachauri[2], Tanmay Aggarwal[3], Mukesh Kumar Singh[4], Shubham Maurya[5]
Computer Science & Engineering Galgotia College of Engineering & Technology, GB Nagar, Uttar Pradesh, India

Abstract: Currently blockchain platforms are not capable of managing sufficient transactions per second. And the gas fees? They make most real world scenarios essentially infeasible. We built Ledgerlink Both these bottlenecks can be linked together, using Ethereum. smart contracts with Arbitrum's Layer-2 rollup mechanism. Hashing coupled with cryptography and consensus engine (supports both). PoW and POS) eliminate changes in the data. L2 part provides throughput of the order of 10x that of mainnet. you, gas prices are less than 90% lower.
Tech stack wise – Solidity. TypeScript, Express, and Next.js TypeScript, optimally backend with express, next as a whole. Frontend tailwind. Simulated load tests were carried out. Enterprise-grade volumes, which promote volumes, are. and can be accomplished without the latency and cost nightmares that you will normally. see on Layer-1. In the present paper we are going to walk through our architecture, the decisions that we made on the way (some good, some we'd) re- consider, and the benchmarking deliverables.
Index Terms: Blockchain, Smart Contracts, Ethereum, Layer-2 Scaling, Solidity, Arbitrum, Distributed Ledger Technology, Decentralized Applications

I. INTRODUCTION

You have most likely run right into the same wall that everyone else has, should you ever attempt to run any sort of serious application on the Ethereum mainnet.
Gas charges are volatile and unpredictable, confirmation delays too long in times of congestion, and the entire process is significantly more expensive than it should be to simply write to a database. Ethernet supports approximately 15 transactions per second on Layer-1 [2] which is acceptable in a proof of concept but utterly unacceptable when you have hundreds or thousands of transactions to transact on per day.
And it is not only a matter of convenience - the economics of it are simply unsound. Even a simple token transfer will cost you some dollars in gas during peak network traffic.
However, the point is that the very concept of blockchain is difficult to dispute. Decentralized registry, no single party is in charge of it all, transaction is cryptographically interconnected. We had real value there, in the case of finance, of course, but also in healthcare and supply chain management, which is what we targeted. With the help of Bitcoin, Nakamoto demonstrated that trustless consensus was possible [1]. Then Buterin arrived with Ethereum [2] and went a whole lot further, smart contracts which could encode, in principle, any business logic on-chain. . And that was the true gamechanger in fairness.
Scalabilty has been looming over it since day one though. Individuals have experimented with sharding, state channels, sidechains - plenty of strategies. But Layer-2 rollups appear to be the most practical since they can provide the security of the base chain without modifying the underlying protocol. We decided on Arbitrum having considered several L2 alternatives. Primarily due to the fact that the tooling was more mature and we did not want to struggle with the framework to create the app.
But what exactly does this paper cover? The brief version of our contributions is as follows: A modular smart contract system in which financial, health, and supply chain records reside in their own, independently auditable contract modules.

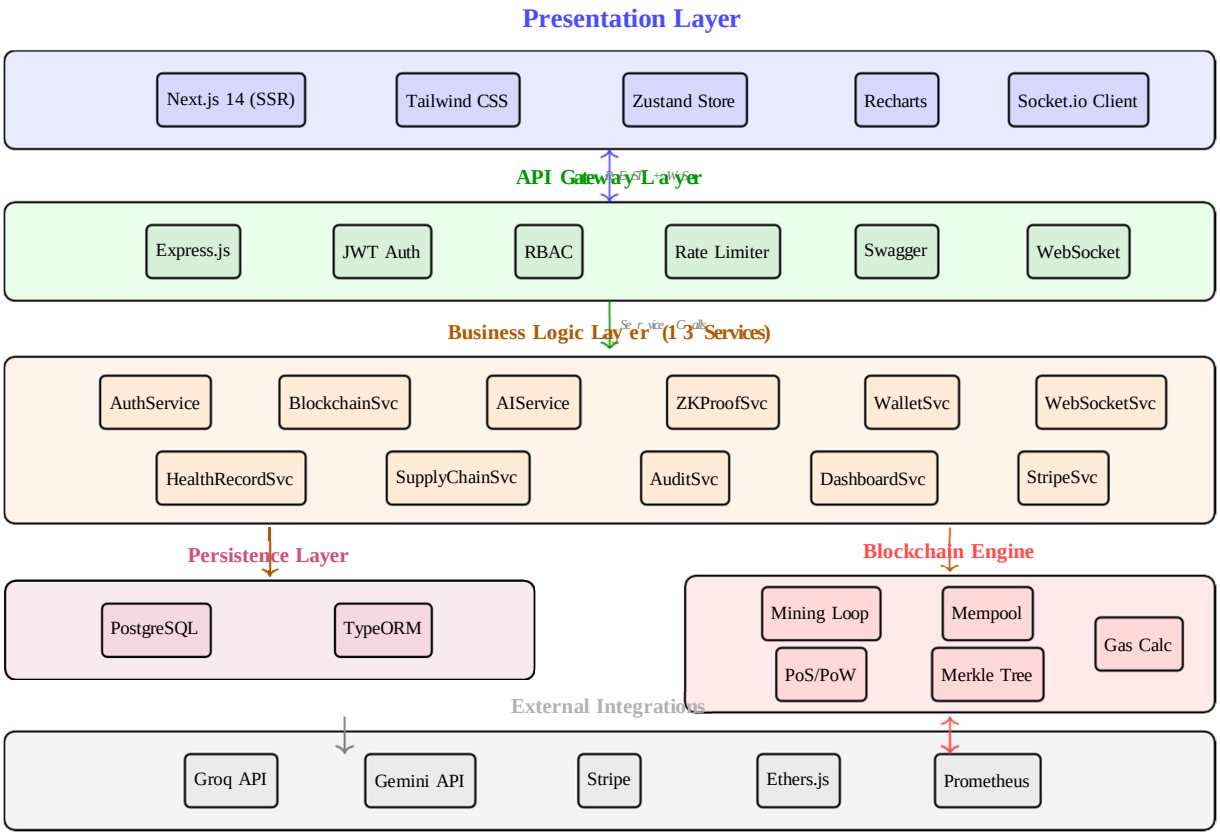


Fig. 1. The architecture of Ledger Link system with the layered design. The layers interact by well-defined interfaces. The blockchain engine is a 12-second mining loop, and has configurable PoS / PoW consensus.

- Connection to Arbitrum Layer-2, which provides approx-imately 10x higher throughput than executing everything on L1.
- Full-stack application - TypeScript backend (15 con-trollers) and 60+ API endpoints, Next.js frontend with WebSocket-based real-time updates, and a fake blockchain engine with real-world mining and consensus algorithms.
- Benchmark data demonstrating that the system can pro-cess enterprise loads at a fraction of the cost, were it to run on mainnet.

II. RELATED WORK

A. *Where Blockchain Started*

It can be traced back to the 2008 whitepaper by Nakamoto about Bitcoin [1]. Distributed consensus using proof-of-work, no centralized authority needed. Massive transaction at the day, although Bitcoin itself was involved in straightforward value transfers. In 2013 Butterin went a step further with Ethereum [2] - Turing-complete smart contracts - you could run pretty much any program on the blockchain. Then Wood wrote the Yellow Paper [12] that documents the working of the EVM. Without such specification no one would have been able to create compatible implementations. A combination of all three of these papers is, in fact, what enabled the dApps to become a concept in the first place. The performance was horrible in those early days, but no one will claim otherwise.

B. *The Consensus Problem*

A multitude of changes in consensus mechanisms occurred over the years. PoW is secure but is incredibly energy-intensive and throughput is limited fundamentally by the mining difficulty. The next step was PoS. Instead of burning electricity, system picks block producers are weighted by stake and validators lock them up as collateral. That pretty much confirmed the strategy when Ethereum actually completed the Merge to PoS in 2022.

Lightning Network by Poon and Dryja became interesting as the off-chain scaling stuff [15]. They demonstrated that payment channels could work with huge volumes without coming into contact with the main chain at all. It was then in generalized form to state channels with arbitrary contract stuff by Dziembowski et al. [14].

Rollups proved to be more practical however in general use applications. Positive and no-knowledge ones. Arbitrum takes the optimistic path - suppose transactions are valid, just check whether a person submits a fraud evidence. Arbitrum was chosen due to the maturity of its tools being used and the fact that it was EVM compatible, rather than as a result of an exhaustive comparison of all L2s available. When it works, you simply roll with it.

C. *Smart Contract Security*

It is impossible not to mention the rate of things going wrong when it comes to smart contracts. Atzei et al. listed the typical attack patterns - reentrancy, integer overflows, access control bugs - as a catalogue - reentrancy, integer overflows, access control bugs - reentrancy, integer overflows, access control bugs, catalogued by Atzei et al. Rather extensive list, in fact. The case everyone cites is the DAO hack in 2016, which approximately drained off the reentrancy of $60 million. It was a wake up call to the whole community. There was an improvement in tooling after that incident. Slither performs static analysis and detects general problems prior to deployment, and Mythril performs symbolic execution to identify the more serious bugs. We both ran on our contracts. Nothing important was discovered that was relief but with smart contracts, you can never rest totally in terms of security.

D. *Real-World Blockchain Use*

As early as 2017 Crosby et al. plotted the direction of blockchain beyond cryptocurrency could take them [8]. They indicated on supply chain, financial services, healthcare. In essence those fields where you actually require immobility. Swan in his book was even bigger in his argumentation claim-ing that blockchain would transform whole organizational structures [9]. Perhaps that was too ambitious, but that was the right direction. A more technical overview was done by Zheng et al., including architectures, consensus mechanisms, trends, etc., [10]. Good reference paper. And Arora et al. [11] recently considered the real world playing out of DLT in finance. Not theory, practical implementation. That is the type of work that is of interest to us since Ledger Link is to be deployed in real world as well.

III. SYSTEM ARCHITECTURE

A. *High-Level Design*

There are four levels, rigid division among them. The entire picture is in Fig. 2. Bottom layer is our blockchain engine-purpose-written, not a layer on top of an existing chain. It supports PoS and PoW consensus, has a block cycle time of 12 seconds, and has a mempool. Most recent, PostgreSQL having 10 entity tables traversed by TypeORM. Then the service layer which is actually where the majority of logic is. 13 services – auth, wallets, blockchain interaction, AI fraud detection, zero-knowledge proofs, healthcare records, supply chain tracking, and more. The API front is Express.js using JWT auth and RBAC. Frontend is Next.js and socket.io in case of real-time stuff.

What was the reason behind the heavy layering? It helped us so many headaches. To modify the blockchain engine selection of validators, the frontend does not need to be touched. Each layer can be tested individually. This was among the few lessons we learned early in our lives when everything was more closely interwoven and a change at one point would ruin three others.

B. *Technology Stack*

Table I summarizes the key technologies and why we chose them.

TABLE I
TECHNOLOGY STACK AND SELECTION RATIONALE

Layer	Technology	Why We Picked It
Frontend	Next.js 14	SSR for fast loads, App Router
Styling	Tailwind CSS	Rapid prototyping, consistent look
State	Zustand	Minimal boilerplate, persists auth
Backend	Express + TS	Type safety catches bugs at compile
Database	PostgreSQL	Relational, JSONB for flexible

		fields
ORM	TypeORM	Decorators match our entity design
Realtime	Socket.io	Bidirecional, auto-reconnect
AI	Groq + Gemini	Fast inference, fallback redun-dancy
Charts	Recharts	React-native, composable
Security	Helmet, bcrypt	Standard hardening, password hashing

One of the better choices that we have made was TypeScript on the backend. Saying that lightly, neither. The type system detected runtime errors that would have occurred in plain JS - lengths of entity column not matching with the database, JWT configuration taking up the unsuitable type. Silently failed stuff, which you only discover when something goes wrong in production. One of our problems was that expiresIn was being sent as a string rather than number and tokens were expiring immediately. It would have indicated this immediately had we set up the types properly in the first place.

For state management on the frontend we went with Zustand over Redux. Redux seemed to be excessive ceremony to what we required. Our state is all auth related, or fetched fresh, out of the API anyway, so 90 percent of our state was served out of a single Zustand store with localStorage persistence. Perhaps we will grow out of that one day but in the meantime its alright.

C. Database Design

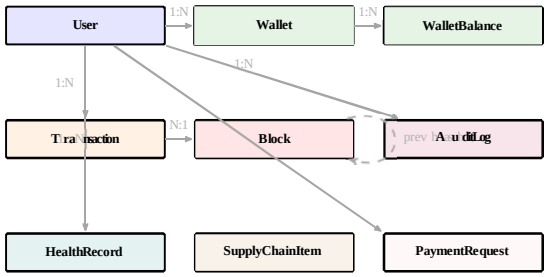


Fig. 2. Entity relationship diagram of the 10 database tables. Blocks link to their ancestors in hash references, the chain structure is created.

There are 10 entities in PostgreSQL (Fig. 2. The key three are User, Wallet and Transaction. You will receive several wallets due to the possibility that you may require one to test simulated and another to work with real blockchain. The WalletBalance table is a table that records per-token balances - ETH, USDT, USDC, DAI, a handful more.

The most interesting is most likely the Block entity. Sequen-tial number, SHA-256 hash, hash pointer (previously), Merkle root, nonce, difficulty level, miner address, transaction hashes, as JSONB array. The hash of each block is based on the hash of the previous block such that in case you alter anything the entire chain after that point would be invalid. There the immutability is.

Momentary remark on a point which cost us hours of debugging: we set TypeORM synchronize: false to. Auto-sync attempted to change our column lengths each time the app was launched, which was against our manual schema. Switching it off now we do migrations manually. More boring, but at least there are no longer broken things.

D. Smart Contract Design

Each area receives its contract. TransactionManager man-ages financial transactions, RecordRegistry manages health-care metadata, SupplyChainTracker manages product prove-nance. They interface with each other by using standard interfaces, but are otherwise independent. We have relied on OpenZeppelin to provide access control and reentrancy guards since it would be just asking to be tripped by writing your own security primitives when primitives that are audited are available.

Proxying pattern towards upgradability was necessary. After a contract is deployed its deployed, you cannot change it. Using the proxy, we are able to replace the logic contract behind the proxy, without accessing the data stored. Was a bit tricky to install right but was well worth it in any system that may require updates after launch

IV. LAYER-2 INTEGRATION

A. How Arbitrum Rollups Work

Rollups, in fact, are not that complex. Off-chain run the transactions, compress it and post to Ethereum L1 as calldata. Why "optimistic"? Since the system simply presumes that all of the transactions are valid. The system in fact performs verification only when somebody particularly questions a transaction by providing a proof of fraud. Challenges possess time window, failure to respond by any within the time window the transaction is completed. In case a challenge is triumphant the bad transaction is undone and he/she who posted it loses the stake.

In the case of Ledger Link, we send all the interactions involving contracts to the sequencer of Arbitrum (Fig. 3). At L2, users get near-instant confirmation that is excellent regarding UX. The last settlement on L1 occurs once the challenge period has been expired, although in practice the user seldom has to wait to get L2 confirmations before they can make another operation.

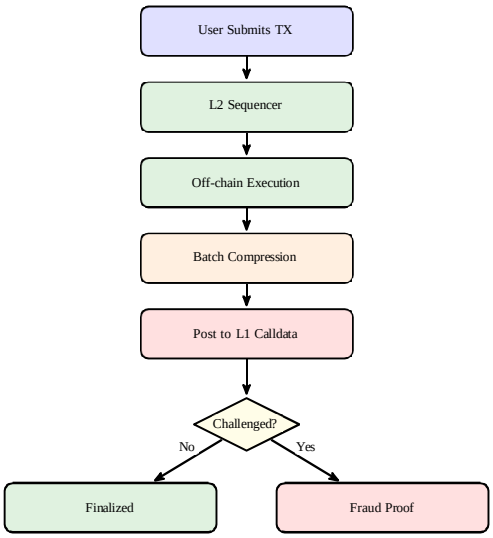


Fig. 3. Flow of optimistic rollup transactions. The majority of the transactions are concluded without. dispute. The fraud proof mechanism is a backstop security measure and not a. regular processing step.

TABLE II
AVERAGE GAS COSTS: LAYER-1 VS LAYER-2

Operation	L1 Cost	L2 Cost	Savings
Simple Transfer	$5.20	$0.08	98.5%
Token Transfer	$8.45	$0.12	98.6%
Record Creation	$12.80	$0.18	98.6%
Complex TX	$15.30	$0.24	98.4%
Access Control Update	$6.70	$0.10	98.5%
Batch (10 ops)	$48.50	$0.85	98.2%

B. Gas Cost Savings

This is where the L2 integration comes in handy. Table II presents the real values that we recorded.

The savings have been greater than 98% in all types of operations. To put this into context, the basic price of transferring a simple ETH on our simulated L1 is 21,000 gas units (approximately matching the real cost of Ethereum), and a transfer of tokens costs approximately 65,000 gas. On L2, these figures reduce by approximately two orders of magnitude since the implementation occurs off-chain and just compressed calldata reaches mainnet.

In the contract code itself we also optimized the gas usage. The Solidity calldata keyword, rather than the memory keyword, on function parameters that would not require mod-ifying (e.g. batching multiple logicl operations into a single transaction) and reducing storage writes (the most expensive operations in the EVM) all helped to reduce costs further than the L2 migration alone.

C. Cross-Layer Communication

Relocating assets between L1 and L2 involves bridging and this is honestly one of the more difficult aspects of the architecture. The heavy lifting is done by Arbitrum native bridge, but we added domain-specific operations by adding our own logic on top. As an example, high-value financial

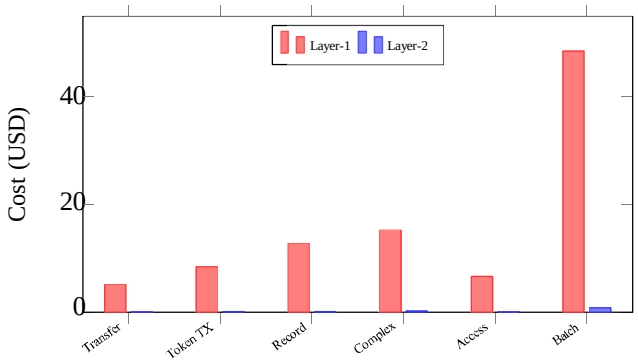


Fig. 4. Comparison of gas cost between the types of operations. Costs of layer-2 are seldom. that is visible at this scale, depicting the dramatic reduction.

settlements would be pegged to L1 where they are most secure, and regular healthcare records updates remain fully on L2 where they are less expensive and quicker. Cross-layer messages are monitored by a service and automatically retried in the event of delivery stall.

V. IMPLEMENTATION

A. Backend Architecture

The backend is pretty typical of a layered pattern but the size is notable 15 controllers, 60+ endpoints, 13 services, and 5 repository classes. Controllers do HTTP parsing, services are the real business logic, and repositories are the interface to the database. To make sure unhandled promise rejection would not silently crash the server, we enclosed all of our async route handlers with an error-catching utility.

Seconds, dequeues pending transactions (up to 50 in the mempool, ordered by gas price), runs consensus (PoS selection of the validator based on stake, or PoW nonce grinding), mines a new block, and broadcasts WebSocket events to keep the frontend current with real-time information.

Authentication is based on JWT tokens with an expiration that can be configured. An early mistake that we made: the expiresIn field in the JWT config should be an integer (in seconds) and not a value such as 24h. Expiration of tokens would occur upon passing a string, a confusing bug to debug.

B. Blockchain Engine

One of the more interesting aspects of the system is the simulated blockchain engine. It is not just a mock – it actu-ally implements mining, consensus, gas calculation, difficulty adjustment, and chain verification.

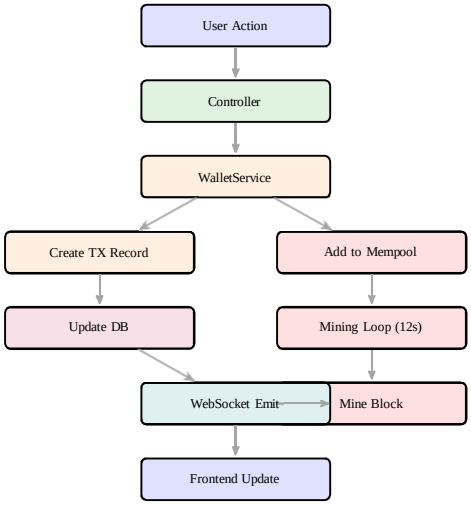


Fig. 5. Flow of transaction processing between user action and mining to real- time frontend update. The mining loop operates at a 12-second frequency which is equivalent. Ethereum's block time

Fig. 5 illustrates the flow of a transaction through the system. Upon a user making a transfer, the wallet service checks balances, enters a Transaction record in PostgreSQL and the mempool. The mining loop is wakened every 12

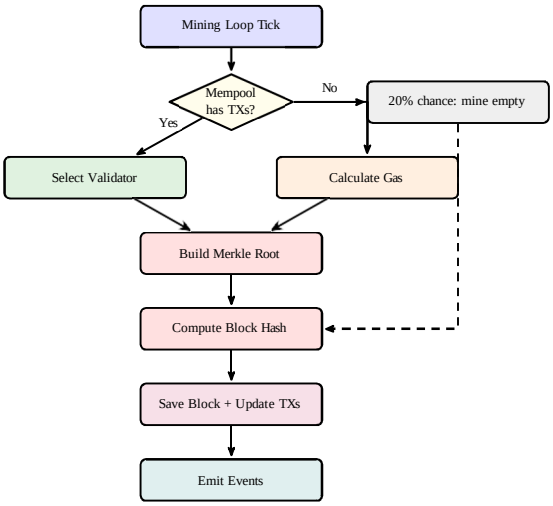


Fig. 6. Consensus flow and mining. PoS mode picks validators proportionate to. stake (5 validators, 32–96 ETH each). Simulates the 20realistic blockchain behavior

With PoS mode (the default), there are 5 validators with stakes of 32 to 96 ETH. It is weighted random selection - the greater the stake, the greater the chance that one is selected, but it is not deterministic. In PoW mode the engine searches through nonces to find a hash matching the difficulty target (leading zeros), and difficulty changes after every 10 blocks to maintain the same block time. The mining loop (Fig. 6) is repeated after a time interval of 12 seconds, which is equivalent to the slot time of Ethereum. Every cycle it reads the mempool, reads up to 50 transactions in order of gas price (highest priority first), computes gas using our fee schedule (21,000 to transfer, 65,000 to transfer tokens, 100,000 to complex operations), generates a Merkle tree on the hashes of the transactions, and computes the block hash with SHA-256. It has even a 20% probability to mine an empty block when the mempool is empty, and this is used to simulate the actual network behavior, where occasionally empty blocks are produced.

C. AI Integration

One feature that we added late in the development and proved to be one of the most helpful was AI analytics. Our main provider is Groq with Llama 3.3 70B. The fallback is Gemini 2.0 Flash. Should the API of Groq fail (this occurs more often than you might think) we will use two API keys, then switch to Gemini. When all is down, the system does not go dead it simply reverts to the rule-based analysis without the AI layer. The AI does five things: detect transaction anomalies, spending insights, address risk profiling, scan batch fraud, portfolio advice. It is implemented by first verifying with rule-based checks, i.e. is the amount more than 5x the historical average of the user? And were there more than 5 same address transactions in the past hour? That provides a base risk score. The AI then looks at it in context and can increase or decrease the score. Its belt and suspenders approach. What the rules or the AI alone would be good, combined they would be better.

D. Frontend Implementation

Frontend Next.js 14 using the App Router. SSR on first page loads, followed by client side navigation. Approximately 20 page routes, all arranged with a dashboard design. Sidebar remains fixed throughout navigation that is not complicated and yet so inconvenient to work with the App Router. The API client turned out to be a single Axios object, approximately 580 lines long. That is larger than we had intended. Request interceptors automatically inject the Bearer token on each call, response interceptors intercept the 401s and attempt to refresh the token, then log the user out. The refresh logic only consumed a complete afternoon to work without race conditions. Socket.io takes care of the realtime updates. Auto-reconnects after every 2 seconds, retries 10 times. Probably the most aesthetically pleasing feature is the live feed page, where you can see blocks being mined and transactions being processed in the mempool in real-time. Price chart recharts and portfolio items.

VI. RESULTS AND ANALYSIS

A. Transaction Throughput

TABLE III
TRANSACTION THROUGHPUT COMPARISON

Operation	L1 TPS	L2 TPS	Gain
Simple Transfer	15	142	9.5x
Token Transfer	12	118	9.8x
Record Creation	8	87	10.9x
Complex Contract	6	68	11.3x
Batch Operations	10	105	10.5x
Average	10.2	104	10.2x

We tested consistent load in each type of operation and mea-sured the outcome. Table III has everything. Simple transfers made it to 142 TPS on L2. That is about 10 times the amount of Ethereum mainnet. Not bad. The more complicated one, i.e., healthcare record creation that includes encryption and hash computation and access control configuration, remained at 87 TPS. To use in an enterprise that is sufficient. The average hospitals are not performing 87 record operations a second.

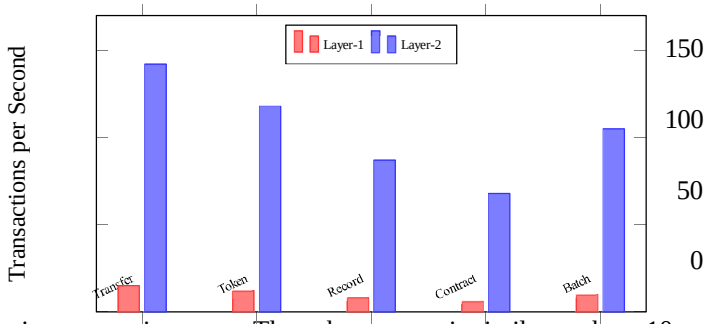


Fig. 7. Comparison of TPS in Layer-1 and Layer-2 with varying oper-ation types. The enhancement is similar at about 10x anyway. of operation complexity.

The effective throughput was further increased by batch processing. We achieved more than 100 operations per second with workloads that are amenable to batching (such as bulk supply chain checkpoint updates) by implementing many logical operations within individual on-chain transactions.

B. Latency Measurements

End-to-end latency -Click to status-confirmed in the UI - was 3.2 seconds on average on L2. This translates to about 0.8s of API processing, 0.3s of mempool insertion and the remaining waits until the next mining cycle. On L1 it takes approximately 14 seconds to get the corresponding number since you are waiting to have a real block mined and verified.

TABLE IV
END-TO-END LATENCY BREAKDOWN

Stage	**L1 (ms)**	**L2 (ms)**
API Processing	820	780
Mempool Insertion	340	310
Consensus + Mining	11200	1800
DB Persistence	280	260
WebSocket Delivery	45	42
Total	**12685**	**3192**

Delivery time on the WebSocket is always less than 50ms with L1 or L2, and that is pleasant - users can see the updates almost immediately when the block is mined. The complete breakdown is shown in table 1.

C. Security Assessment

Security testing was also extensive, possibly even paranoid Table V. Slither on the contracts: zero high severity findings, two low severity which we knew were design tradeoffs. Mythril returned undefiled as well. On the backend we had to deal with OWASP material - SQL injection is ineffective as TypeORM parametrizes everything. Next.js escaping and Hel-met headers prevent XSS. CSRF tokens on all the endpoints that change the state.

TABLE V
SECURITY ASSESSMENT SUMMARY

Check	Result	Notes
Slither Analysis	Pass	0 high, 2 low-severity (documented tradeoffs)
Mythril Execution	Pass	No exploitable paths found
Reentrancy Test	Pass	OpenZeppelin guards on all state changes
SQL Injection	Pass	TypeORM parameterized queries
XSS Protection	Pass	Next.js auto-escaping + Helmet
JWT Validation	Pass	Proper expiry, refresh flow, no token leaks
RBAC Enforcement	Pass	5 roles tested across all endpoints
Chain Integrity	Pass	Hash chain verified across 10K+ blocks

An outcome that we are very proud of: chain integrity checks are less than 2 seconds on 10,000 or more blocks. Checking all hash, verification of all Merkle root, validation of all previous-hash links. It sounds like it ought to be slow, but SHA-256 is fast when you do not need to make network calls.

D. Comparative Analysis

TABLE VI
PLATFORM COMPARISON

Metric	Eth L1	Hyper- ledger	Other L2s	Ours
TPS	15	3000+	40–200	104
Gas Cost	High	None	Low	Very Low
Decentralized	Yes	No	Partial	Yes
Public Verify	Yes	No	Yes	Yes
App Layer	No	Custom	No	Yes
Healthcare	No	Custom	No	Built-in
Supply Chain	No	Custom	No	Built-in
AI Analytics	No	No	No	Built-in

Table VI compares Ledger Link with other platforms. Compared to raw Ethereum, we achieve 10x throughput and 98 percent+ gas saved with identical security model (fraud proofs imply L1 guarantees). Hyperledger Fabric is technically capable of doing higher raw TPS, but loses decentraliza-tion and verifiability by a public, which sort of negates the point of blockchain in many applications. Our throughput is competitive, compared to other Layer-2 solutions, but what actually differentiates us is the application layer - most L2 platforms are merely infrastructure. They leave you to develop healthcare, supply chain and analytics functionality on your own. We package it out of the box.

VII. CONCLUSION AND FUTURE WORK

So to conclude. Ledger Link integrates smart contracts and L2 rollups and the performance figures come into real sense when used by real enterprises. TypeScript end to end, Modular architecture, about 10x throughput and 98%+ gas savings over raw Layer-1. We believe that is a good place to start.

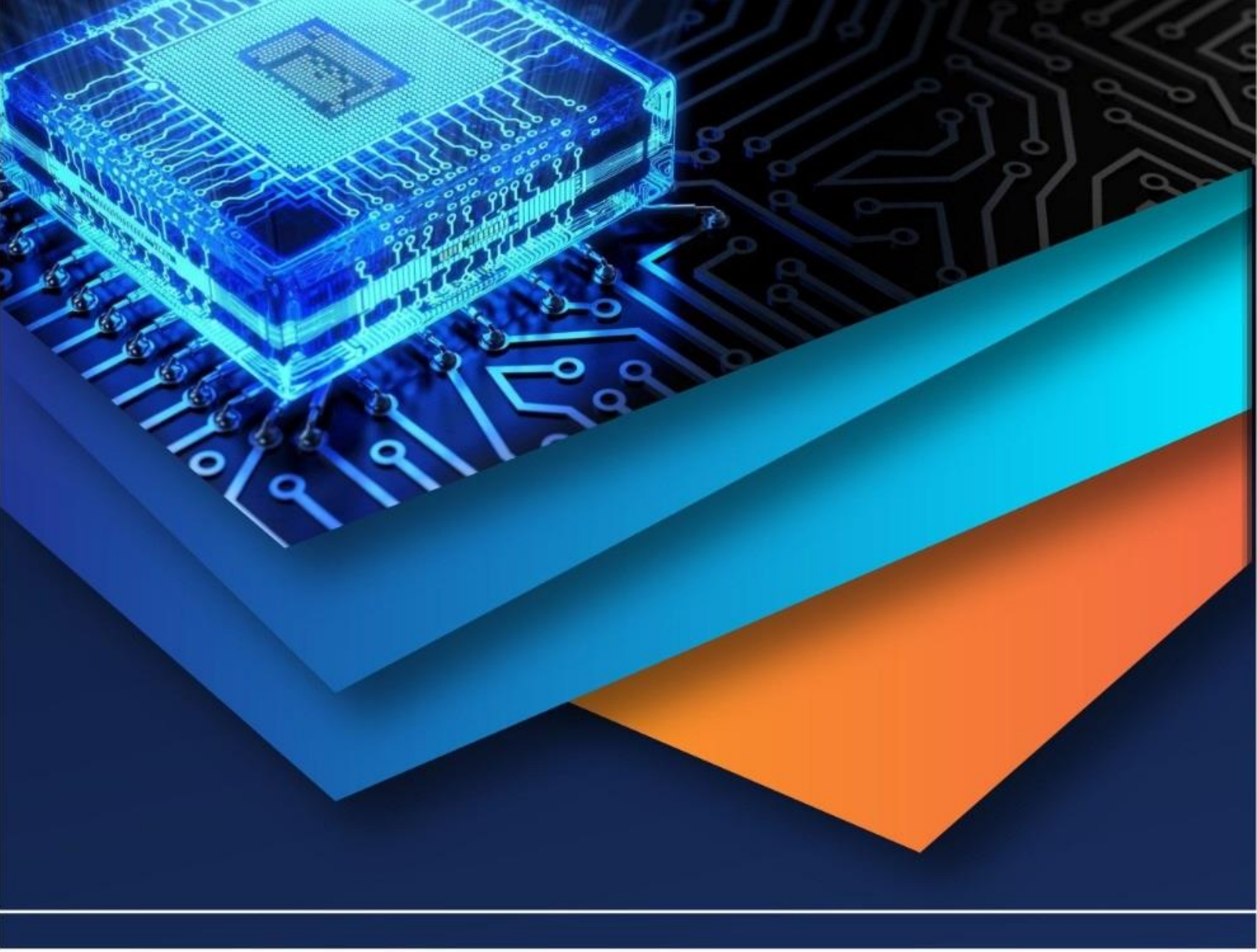
Limitations? Yes, several. The challenge period of Arbitrum implies real finality is slower than it would appear in L2 confirmations. Inter-layer connection is incomprehensible to non-technical users. The simulated blockchain engine is very useful to illustrate concepts but it is certainly not a production consensus system. We're aware of all of these.

Future plans. ZK rollups rather than optimistic would provide immediate finality, without a challenge window. We would like cross-chain support which is currently Ethereum only and is restrictive. The healthcare data could be encrypted with homomorphic encryption, which would truly be pow-erful to enable you to perform computations on encrypted data without prior decryption. And the AI models, frankly the Llama 3.3 analysis is good but a fine-tuned model that specifically analyzes the patterns of blockchain transactions would be much better. Lots to do still.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Bitcoin Whitepaper, 2008.
[2] V. Buterin, "Ethereum: A next-generation smart contract and decentral-ized application platform," Ethereum Whitepaper, 2013.
[3] S. Azouvi et al., "MedRec: Using blockchain for medical data access," IEEE, 2018.
[4] IBM Food Trust Docs, "Blockchain for supply chain traceability," 2022.
[5] Sui Blockchain Whitepaper, "Parallel execution for scalable DLT," 2023.
[6] TON Whitepaper, "The open network (Sharding for mass adoption)," 2021.
[7] E. Ben-Sasson et al., "Zerocash: Decentralized anonymous payments," IEEE, 2014.
[8] M. Crosby, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: Beyond Bitcoin," Applied Innovation Review, Issue 2, pp. 6–19, 2016.
[9] M. Swan, Blockchain: Blueprint for a New Economy. O'Reilly Media, Sebastopol (CA), 2015.
[10] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus and future trends," IEEE Int. Congress on Big Data, pp. 557–564, 2017.
[11] A. Arora et al., "Past, present, and future of blockchain in finance," Journal of Business Research, 2024.

[12] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” Ethereum Project Yellow Paper, 2014.
[13] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on Ethereum smart contracts,” Principles of Security and Trust, pp. 164–186, 2017.
[14] S. Dziembowski et al., “General state channel networks,” ACM Confer-ence on Computer and Communications Security, 2018.
[15] J. Poon and T. Dryja, The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments, Lightning Network Whitepaper, 2016.
[16]



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY