National College of Ireland

# Decentralized Cloud Computing: Tokenized Resource Allocation for Efficient and Scalable Cloud Services

MSc Research Project
Cloud Computing


Rahul Rajendra Shirsat
Student ID: 23296356

School of Computing
National College of Ireland

Supervisor: Sean Heeney

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Rahul Rajendra Shirsat |
| **Student ID:** | 23296356 |
| **Programme:** | Cloud Computing |
| **Year:** | 2025 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Sean Heeney |
| **Submission Due Date:** | 11/08/2025 |
| **Project Title:** | Decentralized Cloud Computing: Tokenized Resource Allocation for Efficient and Scalable Cloud Services |
| **Word Count:** | 8321 |
| **Page Count:** | 19 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

| | |
|---|---|
| **Signature:** | Rahul Rajendra Shirsat |
| **Date:** | 10th August 2025 |

## PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies). | ☐ |
| **Attach a Moodle submission receipt of the online project submission**, to each project (including multiple copies). | ☐ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | ☐ |

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Decentralized Cloud Computing: Tokenized Resource Allocation for Efficient and Scalable Cloud Services

Rahul Rajendra Shirsat
23296356

**Abstract**

The prevailing centralized model of cloud computing presents challenges related to opaque billing, potential vendor lock-in, and delayed cost visibility for users. This research project addresses these issues by designing, implementing, and evaluating a hybrid system that integrates blockchain technology with traditional cloud infrastructure for transparent and pre-paid resource allocation. The core of the solution is an ERC-20 token, "CloudToken" (CTK), deployed on an Ethereum-based network, which serves as a utility token for provisioning cloud resources. A Python-based orchestration layer acts as a bridge, monitoring on-chain token transactions and programmatically allocating Amazon Web Services (AWS) EC2 instances via the Boto3 library. The system's primary logic gates access to compute resources based on a user's token balance, ensuring that allocation is only possible after a cryptographic payment is confirmed on the blockchain. The project further demonstrates a mechanism for automated scalability by configuring an AWS Auto Scaling Group and CloudWatch alarms, enabling the infrastructure to respond dynamically to load. The evaluation, conducted through a series of scripted scenarios, successfully validates the token-gating mechanism under conditions of sufficient funds, insufficient funds, and token replenishment, proving the model's efficacy in creating a more predictable, auditable, and efficient cloud service access layer.

## 1 Introduction

The proliferation of cloud computing has fundamentally reshaped the digital landscape, offering unprecedented scalability, flexibility, and operational efficiency (Kait and Kumar, 2024). The major providers in this sector are Amazon Web Services, Microsoft Azure, and Google Cloud Platform. All three of them have a centralized, utility-based and post-paid billing model system. This system has enabled the development of numerous digital services, but there are limitations as well. Therefore, it became very complex for users to understand billing and control expenses in real-time. This causes unexpected costs and budget overruns (Darwish, 2024). The centralized format often threatens issues such as data governance and security, even vendor lock-in (Ogeawuchi et al., 2022).

At the same time, Web 3.0 and blockchain technologies are establishing fascinating paradigms of decentralization, transparency, and user-centric control (Cao et al., 2025). These derive the advantage of immutable distributed ledgers and smart contracts through trustworthy interactions and the creation of new economic baselines. Tokens representing assets and services can be accessed and value transferred more cryptographically and

auditable directly to parties without use of intermediaries (Enaya et al., 2025). It opens up possibilities to rethink how a cloud computing infrastructure should be managed and paid for when it comes to access to digital resources.

This is the primary motivation behind the study: to bring together these two powerful domains of technology. The study proposes a hybrid model that combines the mature, powerful infrastructure from a conventional cloud provider with an open, prepaid access control of a blockchain-based token economy. The central hypothesis proposes that making the user "spend" a utility token in provisioning cloud resources establishes a system that is much more efficient, predictable, and user-controlled. This approach directly confronts the problem of opaque, post-paid billing by shifting to a pre-paid, token-based consumption model. Figure 1 provides a conceptual overview of this integration.

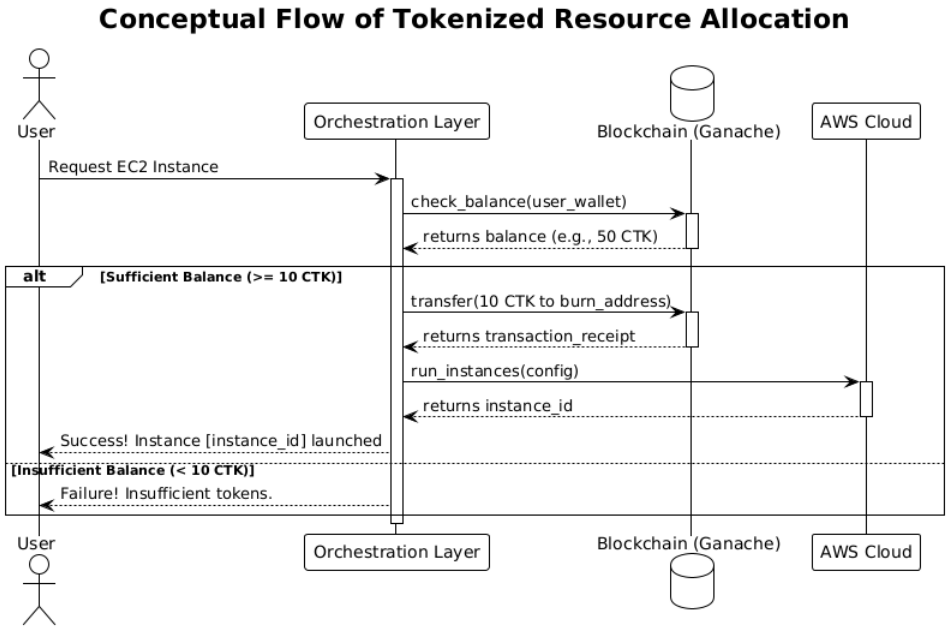


Figure 1: Conceptual flow of tokenized resource allocation where a user with tokens interacts with the orchestration layer, which then provisions AWS resources via blockchain validation.

The primary research question addressed by this work is: *How can a tokenized, blockchain-based system be integrated with a centralized cloud provider to enable efficient, transparent, and scalable allocation of compute resources?* To answer this, the following research objectives were established:

1. To design and deploy an ERC-20 compliant utility token on an Ethereum-based network to represent cloud resource credits.

2. To develop a secure orchestration layer that acts as a bridge between the blockchain and the cloud provider's API.

3. To implement a core logic that validates a user's token balance and automates the provisioning of an AWS EC2 instance upon successful token payment.

4. To demonstrate the system's capacity for scalability by configuring automated resource management based on performance metrics.

5. To evaluate the functionality and robustness of the token-gating mechanism through a series of controlled experiments.

This research contributes a practical proof-of-concept for a hybrid cloud access model. It provides a blueprint for integrating on-chain tokenomics with off-chain infrastructure, demonstrating a viable pathway toward more decentralized and user-centric cloud services. The findings show that such a system is not only feasible but also offers tangible benefits in terms of cost predictability and transparent resource governance.

The remainder of this report is structured as follows. Section 2 presents a critical review of related work in the fields of cloud computing, blockchain, and their intersection. Section 3 details the research methodology, justifying the choice of tools and the experimental approach. Section 4 provides a comprehensive design specification of the system's architecture and its components. Section 5 describes the step-by-step implementation of the solution, from smart contract deployment to cloud infrastructure configuration. Section 6 presents a thorough evaluation of the system through a series of test scenarios and discusses the findings. Finally, Section 7 concludes the report, summarizes the key contributions, and proposes directions for future research.

# 2 Related Work

This section situates the current research within the broader academic and technical landscape. It involves a critical review of literature concerning the evolution of cloud computing, the application of blockchain technology for resource management, and the security implications of integrating these distinct paradigms. The review aims to identify existing gaps and justify the novelty and relevance of the proposed tokenized resource allocation model.

## 2.1 Centralized vs. Decentralized Cloud Architectures

The dominant paradigm in cloud computing is the centralized model, where a single entity owns and manages vast data centers, offering resources as a service. This model has been extensively analyzed for its benefits, such as economies of scale, professional management, and high reliability (Kait and Kumar, 2024). However, its limitations are also well-documented. Abughazalah et al. (2024) discuss the trade-offs between centralized and decentralized models, particularly in sensitive domains like healthcare, highlighting issues of data sovereignty, single points of failure, and potential privacy breaches inherent in centralization. The post-paid, metered billing of centralized clouds is often unclear, and thus it becomes very difficult for organizations to manage their cost (Darwish, 2024).

On the other hand, decentralized computing models involve distributing the workload and the storage among many independent clients. Dai et al. (2025) also provide a survey on the emerging trends in parallel and distributed systems, which emphasize the transition toward more decentralized topologies for the purpose of achieving resilience and fault tolerance. The term "Decentralized Physical Infrastructure Network" (DePIN) has been developed to explain the situation in which an entire community-usually driven by token incentive-contributes their hardware into a common pool. This project does not provide a complete DePIN but borrows its core economic principle: the mediation of access to resources by tokens. The work of Kumar et al. (2023) goes on to discuss the challenges and opportunities in this space, pointing out the fact that the integration of mature

centralized clouds with new decentralized networks provides an interesting research area. This project intends to fill this gap by providing a hybrid model in which the stability of the centralized provider (AWS) is retained while offering the opportunity for decentralized access control through a blockchain.

## 2.2 Blockchain for Resource Allocation and Auctions

With the innate properties of immutability, transparency, and programmability through smart contracts, blockchain technology is touted as a potential avenue for resource allocation management. Extensive research delves into the use of blockchain for auction mechanisms. Liu et al. (2024) give an excellent survey on the blockchain-auctions nexus, detailing how smart contracts enforce auction rules, assure fair bidding, and settle disputes in a trustless manner. Singhal et al. (2024) extend this survey on blockchain-enabled auctions, with special emphasis in cloud resource provisioning with improved trust and economic efficiency.

While auction-based models seem to be potent, they often complicate matters for end-users, requiring them to actually engage in some dynamic bidding process. The model proposed in this research makes use of a payment-willingness system through fixed-price tokens to simplify the user experience. Instead of going through a bidding process, the user would simply pay a predetermined number of tokens for a defined resource. Nevertheless, this turns the auction less into a dynamic market and more into a decentralized vending machine, placing emphasis on simplicity and predictability rather than optimal price discovery. Savadatti et al. (2025) conducted on DLT for scalable computing indicate the need for secure yet operationally simple systems. The fixed-price token model is true to this premise and offers a visible and tractable route from payment to provisioning.

## 2.3 Security and Integration of Cloud and Blockchain

One new set of security considerations arises when integrating blockchain and cloud computing. Guha et al. (2024) review the integration of blockchain with the cloud for financial data management and highlight how blockchain can help to assure data integrity and auditability of the record stored in the cloud. They note that the bridge between the two worlds, on-chain and off-chain, is a vital security frontier. Albshaier et al. (2024) extend their analysis of the IoT, where devices interact with both cloud services and blockchain ledgers. They pinpoint the orchestration layer, which translates on-chain events to off-chain actions, as a relevant attack vector.

This project's architecture, with its Python-based orchestration layer, directly engages with this challenge. The security of the system is contingent upon the security of the AWS credentials and the private keys used by the orchestration scripts. Research by Ramirez Lopez et al. proposes hybrid architectures for securing large health records, suggesting that the channel between the blockchain and the cloud must be robustly protected. While this project uses standard API authentication and secure key management practices, it acknowledges this orchestration layer as a point of centralization that must be carefully managed. Further security enhancements, such as using AWS Key Management Service (KMS) for signing transactions or running the orchestrator in a secure enclave, are potential extensions based on the challenges identified in the literature. Tadi also explores the use of advanced cryptographic techniques like homomorphic encryption

in multi-cloud settings, which represents a more advanced future direction for securing data processing in such hybrid systems.

### 2.4 Summary and Research Justification

The literature confirms a clear trend towards exploring decentralized alternatives and enhancements to traditional cloud computing. Previous works have established the potential for blockchain to facilitate trustless resource allocation, primarily through complex auction mechanisms (Singhal et al., 2024). They have also highlighted the significant security and integration challenges of creating hybrid systems (Albshaier et al., 2024). However, there is a gap in the literature concerning a simple, direct, pre-paid model for provisioning resources from a major cloud provider using a utility token. Existing solutions either focus on fully decentralized networks, which may lack the maturity and performance of centralized providers, or they propose complex market-based allocation systems.

This research project is justified by its pragmatic approach. It does not attempt to replace the centralized cloud but to augment it with a blockchain-based access layer that addresses specific shortcomings, namely cost predictability and transparent governance. By implementing a fixed-price, token-based "vending machine" for AWS resources, this work provides a novel and practical contribution that bridges the theoretical potential of blockchain with the real-world utility of established cloud infrastructure. It directly responds to the need for more user-centric and efficient cloud service models as identified by Darwish (2024) and Kumar et al. (2023).

## 3 Methodology

This research employed an experimental and implementation-focused methodology, rooted in the principles of Design Science. The objective of Design Science is to create and evaluate IT artifacts (constructs, models, methods, or instantiations) that solve identified organizational problems. In this case, the problem is the lack of a transparent, pre-paid resource allocation mechanism in conventional cloud computing. The artifact created is the hybrid tokenized resource allocation system. The methodology involved a structured process of design, implementation, and evaluation to address the research question.

The research procedure was divided into four distinct phases, each building upon the previous one to construct and validate the complete system.

### 3.1 Phase 1: Blockchain Layer Setup and Token Creation

From this effective initiation of the entire process, the first step became laying out the on-chain architecture of the platform. The choice of technologies selected was based on industry trends and the need for a controlled development environment. The reasoning behind the adoption of Ethereum Virtual Machine (EVM) compatible blockchain is due to the long-standing maturity of its smart contract ecosystem, followed by extensive developer tooling, and, of course, adoption of the ERC-20 token standard. This means any chain of such standard has a great degree of interoperability.

Ganache was chosen as a local blockchain simulator for development and testing. This was a conscientious choice that allowed fast iteration and debugging of the smart contracts without the cost of deploying on a public testnet or the delays incurred in such

a process. Ganache offers a private Ethereum blockchain on a local machine with some pre-funded accounts and a GUI for inspecting blocks and transactions. The configured instance was set to run at default local endpoint 'HTTP://127.0.0.1:7545' with a network ID of '1337'.

The coding of the 'CloudToken' smart contract was carried out in Solidity, which happens to be the main programming language for the Ethereum platform. Being accordingly designed, the smart contract was fully up to the ERC-20 standard to make sure it can be managed from standard wallets and integrated into other platforms. The Truffle Suite was employed as the development framework for managing the contract's lifecycle. Truffle streamlines the processes of compiling Solidity code into EVM bytecode, managing deployment scripts (migrations), and interacting with the deployed contract. A standardized project structure was initialized, and the deployment to the local Ganache network was executed through a series of deterministic command-line operations.

## 3.2 Phase 2: Cloud Infrastructure Configuration

This phase involved preparing the off-chain cloud environment on Amazon Web Services. A programmatic approach was taken, using the AWS Command Line Interface (CLI) for all configurations to ensure repeatability, version control, and clear documentation of the infrastructure setup. All resources were provisioned in the 'eu-north-1' region for consistency.

A foundational step was the configuration of Identity and Access Management (IAM) to adhere to the principle of least privilege. An IAM user, 'AdminUser', was created specifically for programmatic access, equipped with an Access Key ID and a Secret Access Key. More importantly, an IAM Role named 'CloudResourceManager' was established. This role was designed with a trust policy that allows the EC2 service to assume it, a critical security best practice that obviates the need to embed credentials within the instances themselves. This role was granted a curated set of AWS-managed policies ('AmazonEC2FullAccess', 'AmazonS3FullAccess', 'CloudWatchFullAccess') to perform its designated tasks.

Essential resources were then prepared. An EC2 key pair, 'ec2keypair', was generated to enable secure shell (SSH) access to any provisioned virtual machines. An S3 bucket, named 'tokenized-storage-pool-656697807862', was created to represent a conceptual storage pool for potential future integration. To lay the groundwork for automated scalability, an EC2 Launch Template named 'TokenizedLaunchTemplate' was created. This template acts as a blueprint, defining a standard configuration for all instances, ensuring consistency across a fleet. An Auto Scaling Group, 'TokenAutoScalingGroup', was then configured to utilize this template, with policies defining the minimum, maximum, and desired number of running instances.

## 3.3 Phase 3: Orchestration Layer Development

This phase focused on creating the software bridge between the blockchain and the cloud. Python was selected as the implementation language due to its robust ecosystem of mature libraries for both Web3 and AWS interaction, making it an ideal choice for this integration task.

The Web3.py library was the primary tool for interacting with the Ganache blockchain. It handled all on-chain operations, including establishing a connection to the RPC

node, loading the deployed 'CloudToken' smart contract via its Application Binary Interface (ABI) and address, and invoking its functions, such as 'balanceOf' for queries and 'transfer' for state changes.

For off-chain operations, the Boto3 library, the official AWS SDK for Python, was used. It facilitated all API calls to AWS services. Its primary use in this project was to interact with the EC2 service to launch new instances programmatically via the 'run$_instances$'$method and to query the state of existing instances using 'describe$_instances$'.

The core business logic was encapsulated in a script named 'allocate.py'. Its workflow was designed to be strictly sequential and transactional: first, it confirms the on-chain payment, and only after that irrevocable confirmation does it proceed with the off-chain resource allocation. This was achieved by making a synchronous call that waits for the transaction receipt, ensuring atomicity from a business logic perspective.

To provide user visibility, a simple web interface was developed using the Flask microframework. This component, 'app.py', acts as a read-only dashboard, querying both the blockchain for the user's current token balance and AWS for the status of all instances provisioned by the system. This demonstrates how data from both on-chain and off-chain sources can be aggregated and presented in a unified view.

### 3.4 Phase 4: Evaluation and Testing

The final phase involved a systematic evaluation of the implemented artifact. The goal was to verify that the system behaved as expected under different conditions. To ensure consistent and repeatable tests, a dedicated Python script, 'testing_scenarios.py', was developed to automate the process.

Three specific scenarios were designed to test the core token-gating logic comprehensively. These examples are the fundamental states of the system: a user has enough funds (positive case); a user has too few funds (negative case); and a user has run out of funds and has replenished them (state-change case). The decision to use these examples makes for a strong evaluation of conditional logic in the system.

The evaluation's primary data was collected from the console outputs of the testing script, which included records of token balances before and after all transactions, unique transaction hashes for all on-chain transfers, and instance IDs for any successfully launched EC2 instances. The outputs are concrete, verifiable evidence of the system's actions during each experiment.

The next step consisted of evaluating the data to confirm the system correctly enforced allocation rules for each scenario. Each resource allocation function's success/failure was treated as the dependent variable with a user's on-chain token balance as the independent variable. The successful launch of an EC2 instance, confirmed by its unique ID in both the script's console output and the AWS Management Console, was the unequivocal test run indicator. This multi-phase methodology ensured a rigorous and well-documented approach to building and validating the tokenized resource allocation system.

## 4 Design Specification

The system is designed as a three-tier architecture, separating the concerns of the on-chain ledger, the off-chain orchestration logic, and the cloud infrastructure. This layered design enhances modularity, security, and maintainability. Figure 2 illustrates the high-level architecture.

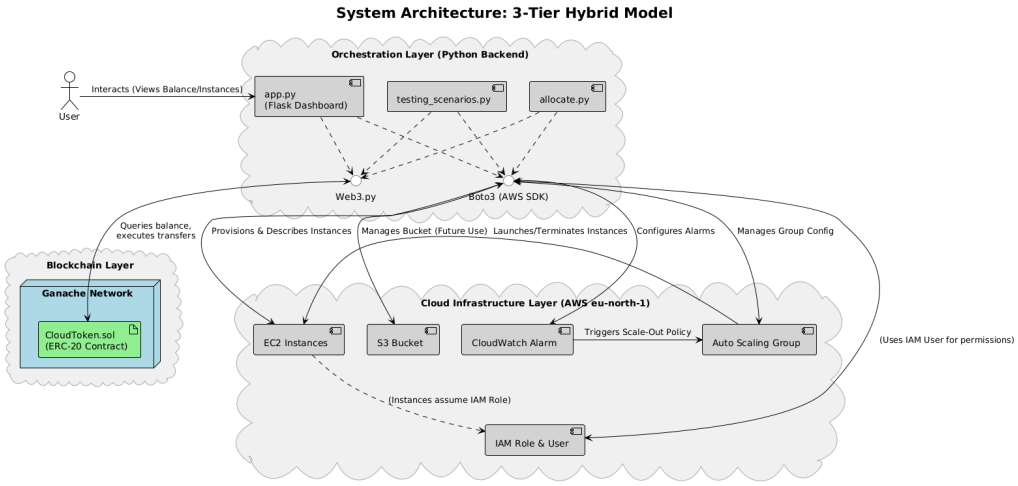


Figure 2: Three-tier architecture showing the Blockchain Layer (Ganache, Smart Contract), Orchestration Layer (Python Scripts, Web3.py, Boto3), and Cloud Layer (AWS IAM, EC2, ASG). Arrows indicate the flow of control and data between the layers.

## 4.1 Layer 1: Blockchain and Tokenomics

This layer forms the economic and governance foundation of the system. Its purpose is to provide a transparent and immutable record of resource credits (tokens) and their ownership.

### 4.1.1 CloudToken (CTK) Smart Contract

The ‘CloudToken.sol‘ contract is the central component of this layer. It is an ERC-20 compliant smart contract with the following specifications:

- **Name:** CloudToken
- **Symbol:** CTK
- **Decimals:** 18. This is a standard practice for ERC-20 tokens, allowing for high granularity in token amounts, which is essential for future pricing models that may involve fractional costs.
- **Total Supply:** 1,000,000 CTK. The entire supply is minted to the contract deployer’s address upon creation. This design choice centralizes the initial distribution, positioning the deployer account as an administrator responsible for distributing tokens to users or funding liquidity pools.

The contract implements the standard ERC-20 functions. The most critical for this system are ‘balanceOf‘, which is a read-only function to check an account’s token holdings, and ‘transfer‘, which is a state-changing function that allows a token holder to send tokens. The allowance functions (‘approve‘ and ‘transferFrom‘) are included for full ERC-20 compliance, enabling future use cases where a user might delegate spending authority to another contract or service. The address of the deployed contract on the Ganache network is ‘0xC9b76694bBE9Aef88979cB394f4DB26b2DD415C1‘.

### 4.1.2 Token Payment Mechanism

The economic model is based on a "proof-of-burn" concept for payment. When a user wishes to allocate a resource, they must transfer a specified number of tokens to a designated burn address.

- **Burn Address:** The address '0x000000000000000000000000000000000000dEaD' is used. This is a commonly used "black hole" address for which no private key is known. Tokens sent to this address are rendered permanently inaccessible, effectively removing them from the circulating supply. This mechanism serves as a clear and irreversible proof of payment.
- **Resource Cost:** For this implementation, the cost of provisioning one 't3.micro' EC2 instance is fixed at **10 CTK**. This value is hardcoded in the orchestration script but is designed to be easily configurable. This fixed-price model simplifies the user experience, providing absolute cost certainty for a given operation.

## 4.2 Layer 2: Orchestration Logic

This middleware layer serves as the trusted intermediary that connects the on-chain and off-chain worlds. It contains the business logic that translates token payments into cloud resource provisioning. The security of this layer is paramount, as it holds the credentials required to perform both on-chain and off-chain actions.

### 4.2.1 Resource Allocation Script

The core engine of the system is a Python script that orchestrates the entire process. Its logic flow is designed to be robust and sequential to prevent race conditions or resource allocation without confirmed payment. The sequence is as follows:

1. **Initialization:** The script establishes a connection to the Ganache RPC endpoint and instantiates the 'CloudToken' contract object using its ABI and address, allowing for interaction with its functions.
2. **Balance Check:** The script performs a read-only call to the 'balanceOf' function on the smart contract to query the user's current CTK balance.
3. **Conditional Logic:** It compares the user's balance against the 'required_tokens' value. If the balance is sufficient, the script proceeds. If not, it terminates with an error message, and no further actions are taken.
4. **On-Chain Payment Execution:** For a user with sufficient funds, the script constructs a 'transfer' transaction destined for the burn address. This transaction is then signed locally using the user's private key. The signed transaction is sent to the network.
5. **Synchronous Confirmation:** This is a critical design choice. The script explicitly waits for the transaction to be mined and confirmed by the blockchain. This is achieved by blocking execution until a transaction receipt is returned. This synchronous wait is essential; it guarantees that the off-chain action (cloud provisioning) only occurs after the on-chain payment is irreversible.

6. **Cloud Provisioning:** Upon receiving the transaction receipt, the script uses the Boto3 client to call the 'run_instances' method of the EC2 API. The instance is configured with a specific set of tags to identify it as having been provisioned by this system, facilitating tracking and management.

### 4.2.2 Monitoring Dashboard

The Flask framework allows development of a simplistic, read-only web application to gain transparency into system equal systems. The information from both on-chain and off-chain layers are compiled upon each page load. Each time, the application performs an on-chain query, to retrieve and present a user wallet's current token balance. Then, it identifies all of the user's running active EC2 instances by communicating with the AWS API and filtering for the system's specific named tag, whilst performing an off-chain query. After parsing the returned response, the application provides selected metadata for the user's respective instance, including instance ID, instance type, instance state, and time of launch. Thus, this offering provides a total view for the user as a system interface utilizing EC2 and token tracking.

## 4.3 Layer 3: Cloud Infrastructure

This layer consists of the actual cloud resources on AWS that are being allocated. The design emphasizes security, consistency, and scalability.

### 4.3.1 EC2 Instance Configuration

Instances provisioned by the system have a standardized configuration to ensure predictability. They use a specific Amazon Machine Image (AMI), 'ami-09278528675a8d54e', and instance type, 't3.micro'. Access is controlled via the 'ec2keypair'. A crucial design element is the use of resource tags. Each instance is programmatically tagged with 'AllocatedBy: TokenSystem' and 'TokenCost: 10'. These tags are not just metadata; they are functional, enabling the monitoring dashboard to identify system-managed resources and providing a clear audit trail for the cost of each allocated instance.

### 4.3.2 Automated Scalability Design

The system is designed to be scalable beyond single-instance provisioning. This is achieved through an AWS Auto Scaling Group (ASG), which automates the management of a fleet of instances.

- **Launch Template:** The 'TokenizedLaunchTemplate' serves as a blueprint for all instances launched by the ASG. It pre-defines the AMI, instance type, key pair, security groups, and tags, ensuring that every instance in the fleet is configured identically.
- **Auto Scaling Group:** The 'TokenAutoScalingGroup' is set up to maintain a desired count of instances. It is associated with the launch template and the particular VPC subnet, 'subnet-0dc4040155e2bedeb'. Its capacity is defined with a min size of 1, a max size of 5, and a desired capacity of 1.

- **Scaling Policy:** A simple scaling policy, 'ScaleOutPolicy', is attached to the ASG. It is configured to 'ChangeInCapacity' by an adjustment of '1'. When triggered, this policy instructs the ASG to increase its desired capacity by one, leading to the launch of a new instance.
- **CloudWatch Alarm:** The 'ScaleOutOnHighCPU' alarm triggers the scaling policy, which is built to monitor the average 'CPUUtilization' metric across ASG instances. The alarm changes to the 'ALARM' state when the average processor utilization exceeds 60% for two consecutive one-minute periods of time. This threshold-based trigger therefore allows the system to autonomously react to increased workloads, very much a step towards dynamic and self-regulating infrastructure pool.

This integrated design ensures that the system can not only allocate individual resources based on token payments but also manage a pool of resources dynamically in response to real-world performance metrics.

# 5 Implementation

In this section, we describe the concrete steps taken to implement the system described in the Design Specification. The implementation followed a phased approach, transitioning the design into operational code and configured infrastructure. The next paragraphs give an overview of the process that focused on the actions taken and tools used.

## 5.1 Blockchain layer implementation

The first critical activity was to set up the blockchain environment and deploy the smart contract. A project directory called 'cloud-token' was created to hold all artifacts for blockchain implementation. Within this directory, an instance of the Truffle framework was initiated. This kind of project scaffolding creates a default project structure with its own folders for contracts, migration scripts, and build artifacts, which is the best practice to use for managing Solidity projects.

The 'CloudToken' smart contract, a standard implementation of the ERC-20 token interface, was developed and placed in the 'contracts/' directory. To manage its deployment, a migration script named '2_deploy_cloud_token.js' was created in the 'migrations/' directory. This script contained instructions for the Truffle framework to deploy the 'CloudToken' contract to the target blockchain network.

With the contract and migration script in place, the contract was first compiled into EVM bytecode. Subsequently, it was deployed to the local Ganache instance, which was configured as the 'development' network in the Truffle settings. The output from the migration process was captured. This output confirmed the successful deployment of the contract and, most importantly, provided its unique on-chain address ('0xC9b76694bBE9Aef88979cB394f4DB26b2DD415C1') and the transaction hash of the deployment itself. This contract address is a critical piece of information, as it is the endpoint that the orchestration scripts use to interact with the contract.

## 5.2 Cloud Infrastructure Implementation

The AWS environment was configured entirely through the AWS Command Line Interface (CLI). This approach ensures that the infrastructure setup is scriptable, repeatable, and

can be version-controlled. The local environment was pre-configured with the credentials for the ‘AdminUser‘ IAM user.

A key part of the security setup was the creation of an IAM Role. A trust policy was first defined in a JSON file named ‘trust-policy.json‘. This policy specified that the AWS EC2 service principal was allowed to assume the role. Using this policy document, an IAM role named ‘CloudResourceManager‘ was created via the CLI. To grant this role the necessary permissions, several AWS-managed policies were programmatically attached to it. These policies included full access to EC2, S3, and CloudWatch, enabling any EC2 instance assuming this role to manage other resources within the account securely.

With the security foundation in place, initial resources were prepared. An EC2 key pair named ‘ec2keypair‘ was generated through the AWS Management Console to allow for secure administrative access to the instances. An S3 bucket, intended to represent a potential future storage resource pool, was also created using a CLI command.

## 5.3 Orchestration Layer Implementation

The Python scripts that form the bridge between the blockchain and the cloud were then developed. The core of this layer is the ‘allocate.py‘ script. This script leverages the ‘web3‘ library to communicate with the Ganache blockchain and the ‘boto3‘ library to interact with the AWS API.

The script’s main function, ‘check_balance_and_allocate‘, implements the primary system logic. It first connects to Ganache and instantiates a contract object representing the deployed ‘CloudToken‘. It then calls the contract’s ‘balanceOf‘ function to check the balance of the user’s wallet. If the balance is sufficient, it constructs and signs a transaction to transfer 10 CTK to the designated burn address. After sending the transaction, the script waits for the transaction to be confirmed on the blockchain. Only after receiving this confirmation does it proceed to use ‘boto3‘ to launch a ‘t3.micro‘ EC2 instance in the ‘eu-north-1‘ region, tagging it appropriately. When this script was first executed using the admin wallet, which held the entire initial token supply, the console output confirmed the successful token transfer and the subsequent launch of an EC2 instance, providing the new instance’s ID.

To provide visibility, a simple web dashboard was built using the Flask framework. The ‘app.py‘ script for this dashboard connects to both Ganache and AWS. Its main function fetches the token balance of a specific user wallet and also queries AWS for all active EC2 instances that have been tagged by the system. This data is then passed to an HTML template, which renders the balance and a table of the allocated instances. The application demonstrates the capability of aggregating and rendering real-time data from both the on-chain and off-chain systems.

## 5.4 Scalability Mechanism Implementation

Following the completion of all these phases in implementing the project, the last phase of implementation on behalf of this project dealt with the auto-scaling components in the AWS being automated to implement the idea of this system dynamic resource pool model.

The first instance task that was executed was the creation of a ‘TokenizedLaunchTemplate‘ launch template using the CLI. The launch template defined a common configuration that would include AMI, instance type, and tags for all instances in that scaling

group. This ensures that all instances are configured the same way and easier to manage.

A second step was to create an auto-scaling group named 'TokenAutoScalingGroup'. This group is tied to the new launch template just created. It was also set up with capacity specifications of only one instance at a minimum, five instances maximum, and an initial instance desired capacity of one. The group is also linked to a specific VPC subnet.

To enable the group to automatically grow, there was created and attached to it a scale policy called 'ScaleOutPolicy'. The policy is of the type 'ChangeInCapacity', with a scaling adjustment of 1, indicating that when triggered, it will add a single instance.

A CloudWatch alarm was set up as a trigger for this policy. The 'ScaleOutOnHighCPU' alarm was created to monitor the average CPU utilization metric for all instances in the 'TokenAutoScalingGroup', and set up with the condition to trigger when the resource acquired a sustained average CPU utilization of more than 60% over two consecutive one-minute intervals. Its action is directed to the ARN of the 'ScaleOutPolicy'. That would complete the implementation of a completely functional token-gated and auto-scaling cloud resource allocation system.

# 6 Evaluation

The evaluation of the implemented system was conducted to validate its core functionality against the research objectives. The primary goal was to test the hypothesis that resource allocation could be effectively and reliably gated by a user's on-chain token balance. A series of controlled experiments were designed and executed using a dedicated Python script, 'testing_scenarios.py'.

## 6.1 Experimental Setup

The test environment was carefully controlled to ensure the reliability of the results. It consisted of the local Ganache instance serving as the blockchain, two pre-configured Ganache accounts acting as the 'admin' and the 'user', the live AWS account in the 'eu-north-1' region, and the testing script itself. The cost for one EC2 instance was fixed at 10 CTK for all experiments. The testing script automated the entire process, managing the state of the user wallet by orchestrating token transfers with the admin wallet and then triggering the core allocation logic. This setup allowed for the systematic and repeatable testing of the system's primary functions.

## 6.2 Experiment 1: User with Sufficient Funds

### 6.2.1 Objective

To verify that a user with a token balance greater than or equal to the required cost can successfully allocate an EC2 instance. This represents the primary positive-path test case for the system.

### 6.2.2 Procedure

The testing script initiated this scenario by first ensuring the user wallet had an ample balance. This was achieved by having the admin wallet transfer 50 CTK to the user wallet.

With the user's account funded, the script then invoked the ‘check_balance_and_allocate‘ function, passing the user wallet's credentials to simulate a user-initiated request.

### 6.2.3 Results

The system performed as expected. The console output from the script first confirmed the successful preparatory transfer of 50 CTK. It then showed the log from the allocation function, which correctly read the 50 CTK balance and identified it as sufficient. The log subsequently displayed the transaction hash of the successful 10 CTK transfer to the burn address, followed by a message confirming the launch of a new EC2 instance, including its unique instance ID (‘i-09cd2b7a7e52ed358‘).

### 6.2.4 Analysis

The outcome of this experiment was a complete success. It demonstrated that the end-to-end workflow functions correctly when the preconditions are met. The system accurately checked the on-chain balance, executed the on-chain payment, and upon confirmation, successfully performed the off-chain action of provisioning a cloud resource. The user's balance was correctly debited by 10 CTK, leaving them with 40 CTK. This experiment validated the core functionality of the system and proved that the integration between the blockchain, the orchestration layer, and the cloud provider's API is working as designed.

## 6.3 Experiment 2: User with Insufficient Funds

### 6.3.1 Objective

To verify that a user with a token balance less than the required cost is properly denied access to resource allocation. This is a critical negative-path test case to ensure the system's integrity and prevent resource misuse.

### 6.3.2 Procedure

The above-described testing script returned the user's remaining balance (40 CTK from the previous trial) to the admin wallet in order to realize the situation. Thus, the user wallet had a balance of 0 CTK. The user wallet credentials were then given to the function 'check_balance_and_allocate'. Thus simulating an attempt at resource acquisition without funds.

### 6.3.3 Results

The system's response was indeed clear and apprised. The output from the script said that allocation function was called after the balance was cleared. The function's log said it checked the balance, found it to be 0.00 CTK, and immediately terminated the process, printing the specific error message: "Not enough tokens to allocate EC2 instance.".

### 6.3.4 Analysis

It was during that experiment that a good part of the validity logic should have proved the solidity of the system. The complete blocking of the allocation process by the token gate and the fact that no token transfer transaction was ever created or sent to an API call to

AWS must be crucial. This result is fundamental since it can determine that the system really prevents unauthorized or unfunded resources from consuming them, thus keeping the resource pool intact, remaining strict to the economic model. The on-chain balance is, thus, a hard prerequisite for any transformative off-chain activity in this regard.

## 6.4 Experiment 3: User Balance Replenished

### 6.4.1 Objective

To verify that a user who previously had insufficient funds can successfully allocate a resource after their on-chain balance is replenished. This tests the system's ability to respond to real-time changes in the blockchain's state.

### 6.4.2 Procedure

Following the failure in the previous experiment, the testing script simulated the user acquiring more tokens. It did this by having the admin wallet transfer 100 CTK to the user wallet, bringing the user's balance from 0 to 100 CTK. Immediately after this transfer was confirmed, the 'check_balance_and_allocate' function was called once more with the user wallet's credentials.

### 6.4.3 Results

The system again behaved as expected. The console output confirmed the preparatory transfer of 100 CTK. The subsequent call to the allocation function showed that it correctly read the new 100 CTK balance and deemed it sufficient. The script then logged the successful transaction hash for the 10 CTK payment and, finally, the successful launch of a new and distinct EC2 instance ('i-0dca342e22a11295c').

### 6.4.4 Analysis

This experiment successfully demonstrated the dynamic nature of the system's access control. It showed that the validation is not a one-time static check but a real-time query of the blockchain state. The system correctly transitioned from a "denied" state to an "approved" state based entirely on an on-chain event (the receipt of new tokens). This confirms that the system is responsive and that access rights are tied directly and dynamically to token ownership as recorded on the immutable ledger.

## 6.5 Discussion

The collective results of these three experiments provide a comprehensive validation of the system's core token-gating mechanism. The findings confirm that the integration of the on-chain smart contract with the off-chain orchestration script and cloud API is robust and functions exactly as designed. The system successfully enforces a pre-paid model where access to valuable cloud resources is contingent upon a verifiable, on-chain cryptographic payment.

The implications of these findings are significant. This model provides a solution to the problem of unpredictable cloud billing. Users or organizations can purchase a set amount of tokens, giving them a fixed budget for cloud expenditure. There is no possibility of accidental overspending, as the system will simply deny requests when the

token balance is depleted. This provides a level of financial predictability and control that is often difficult to achieve with traditional post-paid cloud billing.

Furthermore, the audit trail for resource allocation is perfectly transparent. Every instance provisioned is directly linked to a specific, publicly verifiable transaction on the blockchain. This creates an immutable record of who paid for what and when, which could be invaluable for accounting, compliance, and dispute resolution.

However, the evaluation also highlights the limitations of the design. The experiments were conducted on a local Ganache testnet, which has instantaneous transaction confirmation and no real-world network congestion or gas fees. In a public network environment like the Ethereum mainnet, the 'wait_for_transaction_receipt' step could introduce significant and variable latency, potentially impacting the user experience. The cost of gas for the payment transaction would also become a factor, adding another layer to the total cost of allocation.

The design's reliance on a centralized orchestration script is another critical point. While the payment and balance checks are decentralized, the action of provisioning the instance is not. The security and availability of this script are single points of failure. The evaluation did not test for failure modes of this orchestration layer, which would be an important area for further investigation. Despite these limitations, the evaluation successfully proves that the core concept is sound and that the implemented artifact meets its primary objectives.

# 7 Conclusion and Future Work

This research project set out to answer the question of how a tokenized, blockchain-based system could be integrated with a centralized cloud provider to enable efficient, transparent, and scalable resource allocation. Through a structured process of design, implementation, and rigorous evaluation, this work has successfully demonstrated a functional and robust proof-of-concept that directly addresses this question. The project has met all its stated objectives, delivering a hybrid system that bridges the gap between the decentralized world of Web3 and the established infrastructure of a major cloud provider.

The research successfully designed and deployed an ERC-20 utility token, 'CloudToken', which served as the unit of account for cloud services. A secure orchestration layer was developed in Python, effectively acting as the intermediary between the on-chain logic of the smart contract and the off-chain API of Amazon Web Services. The core logic, which gates access to EC2 instances based on a user's on-chain token balance, was implemented and proven to be effective. The system's capacity for scalability was demonstrated through the successful configuration of an AWS Auto Scaling Group and a CloudWatch alarm, which allows the resource pool to grow dynamically based on demand. The evaluation, through a series of controlled experiments, confirmed that the token-gating mechanism is reliable, preventing allocation for users with insufficient funds and permitting it for those with sufficient funds, all based on real-time on-chain data.

The key finding of this research is that a pre-paid, token-based model for cloud resource consumption is not only technically feasible but also offers significant advantages in terms of financial predictability and transparent governance. By requiring an on-chain payment before any resources are provisioned, the model eliminates the possibility of surprise bills and cost overruns, a common pain point in the traditional post-paid cloud model. Every allocation is tied to an immutable transaction on a distributed ledger,

providing an unparalleled level of auditability.

The biggest limitation of this research is its hybrid architecture— by relying on one orchestrator script you have a single point of failure and one attack vector (in terms of security). The security of the entire system depends on the security of the private keys and API credentials used by this orchestrator script. Moreover, the evaluation of the system was performed on a fully isolated local testnet environment. The experiment does not account for some very realistic factors such as: latency on a blockchain network, gas costs associated with transactions, and volatile public networks that may impact both performance and economic viability of the system in potential deployment scenarios.

Despite these limitations, the research makes a valuable contribution to the field. It provides a practical blueprint for a new class of cloud service model that leverages the strengths of both centralized and decentralized systems. It moves beyond theoretical discussions and provides a concrete implementation, demonstrating how the abstract concepts of tokenomics can be applied to solve real-world problems in cloud management.

## 7.1 Future Work

This project serves as a strong foundation upon which more advanced and robust systems can be built. Several meaningful avenues for future research and development exist:

- **Dynamic and Metered Pricing:** The current model uses a fixed price for resource allocation. A more sophisticated system could implement dynamic pricing. This could involve integrating an on-chain oracle (such as those provided by Chainlink) to feed real-time market data, such as AWS spot instance prices or current network demand, into the smart contract. This would allow the token cost of a resource to fluctuate, creating a more efficient market. Furthermore, the model could be extended from a one-time allocation fee to a continuous metering system. A sidecar process running on the provisioned EC2 instance could report usage metrics (CPU time, data transfer) to the orchestration layer, which would then make periodic token deductions from a user's escrowed balance. The resource would be automatically de-provisioned if the balance runs out. This would create a true pay-as-you-go model governed by the token economy.

- **Decentralized Orchestration and DApp Interface:** The most significant limitation to address is the centralized orchestration layer. Future work could focus on decentralizing this component. This could involve a network of oracle nodes that are responsible for monitoring on-chain events and triggering off-chain actions, with a consensus mechanism to ensure reliability. To improve user experience and security, the backend scripts could be replaced with a fully-fledged decentralized application (DApp). A user could connect their browser-based wallet (like MetaMask) to a web interface, browse available resources, and sign the payment transaction directly. This would eliminate the need for the system to handle user private keys, significantly improving the security posture.

- **Enhanced Security and Multi-Resource Support:** The security of the orchestration layer can be hardened. Instead of storing AWS credentials and private keys as environment variables, services like AWS Key Management Service (KMS) or AWS Secrets Manager could be used. AWS KMS, for instance, could be used to sign the blockchain transactions, ensuring that the private key never leaves the

secure hardware security module. The model could also be expanded horizontally to support a wider variety of cloud resources beyond EC2 instances. The logic could be adapted to allocate S3 storage space, provision database instances (RDS), or even trigger serverless functions (Lambda), with different token costs for each resource type.

- **Integration with Decentralized Infrastructure (DePIN):** The ultimate extension of this work would be to replace the centralized AWS backend with a fully decentralized physical infrastructure network (DePIN) such as Akash Network or Filecoin. In such a model, the tokens would not just be an access key to a centralized service but the native currency of a decentralized marketplace where users bid for compute or storage capacity provided by a distributed network of independent suppliers. This would realize the full vision of a truly decentralized cloud, and the architecture developed in this project could serve as an important transitional step or a model for the user-facing side of such a system.

By exploring these future directions, the principles demonstrated in this project can be evolved into a new generation of cloud computing platforms that are more transparent, equitable, and user-centric.

# References


Abughazalah, M., Alsaggaf, W., Saifuddin, S. and Sarhan, S. (2024). Centralized vs. decentralized cloud computing in healthcare. *Applied Sciences*, 14(17), p.7765.

Albshaier, L., Budokhi, A. and Aljughaiman, A. (2024). A review of security issues when integrating IoT with cloud computing and blockchain. *IEEE Access*.

Albshaier, L., Almarri, S. and Albuali, A. (2025). Federated learning for cloud and edge security: A systematic review of challenges and AI opportunities. *Electronics*, 14(5), p.1019.

Cao, B., Xiao, S., Shi, L., Wang, T., Chen, J., Wang, J., Ling, X., Xu, H., Zhang, S. and Liu, E. (2025). Web 3.0: A Survey on the Architectures, Enabling Technologies, Applications, and Challenges. *IEEE Communications Surveys & Tutorials*.

Dai, F., Hossain, M.A. and Wang, Y. (2025). State of the art in parallel and distributed systems: Emerging trends and challenges. *Electronics*, 14(4), p.677.

Darwish, D. ed. (2024). *Emerging trends in cloud computing analytics, scalability, and service models*.

Enaya, A., Fernando, X. and Kashef, R. (2025). Survey of Blockchain-Based Applications for IoT. *Applied Sciences*, 15(8), p.4562.

Guha, B., Nadanasabai, R., Gayathri, S., Goswami, M., Shankar, R.K. and Shah, R. (2024). The Integration of Cloud Computing and Blockchain for Enhanced Data Security in Financial Management: A Comprehensive Review. *Frontiers in Health Informatics*, 13(4).

Hussein, D.H., Maqdid, G. and Askar, S. (2025). Dynamic Resource Allocation in Cloud Networks Using Deep Learning: A review. *The Indonesian Journal of Computer Science*, 14(1).

Kait, R. and Kumar, T. (2024, May). Insights Into Cloud Computing: Unveiling Trends, Addressing Challenges, and Exploring Opportunities-A Systematic Review. In *2024 International Conference on Emerging Innovations and Advanced Computing (INNOCOMP)* (pp. 593-601). IEEE.

Karim, M.M., Van, D.H., Khan, S., Qu, Q. and Kholodov, Y. (2025). Ai agents meet blockchain: A survey on secure and scalable collaboration for multi-agents. *Future Internet*, 17(2), p.57.

Kumar, A., Fahad, M., Arif, H. and Hussain, H.K. (2023). Navigating the Uncharted Waters: Exploring Challenges and Opportunities in Block chain-Enabled Cloud Computing for Future Research. *BULLET: Jurnal Multidisiplin Ilmu*, 2(6), pp.1297-1305.

Liu, X., Liu, L., Yuan, Y., Long, Y.H., Li, S.X. and Wang, F.Y. (2024). When blockchain meets auction: A comprehensive survey. *IEEE Transactions on Computational Social Systems*, 11(3), pp.4242-4254.

Ogeawuchi, J.C., Akpe, O.E., Abayomi, A.A., Agboola, O.A., Ogbuefi, E.J.I.E.L.O. and Owoade, S.A.M.U.E.L. (2022). Systematic review of advanced data governance strategies for securing cloud-based data warehouses and pipelines. *Iconic Research and Engineering Journals*, 6(1), pp.784-794.

Ramirez Lopez, L.J., Martinez Poveda, L.H., Carbonell Amaya, A.F. and Millan Mayorga, D.F., Secure Channel Based on Hybrid Blockchain-Cloud Architecture for Large Health Records Files. Available at SSRN 5013912.

Savadatti, S.G., Krishnamoorthy, S. and Delhibabu, R. (2025). Survey of distributed ledger technology (dlt) for secure and scalable computing. *IEEE Access*.

Singhal, R., Sharma, V., Singhal, I. and Bansal, V. (2024). Blockchain-enabled auction for cloud resource provisioning: a survey on trust and economy. *International Journal of System Assurance Engineering and Management*, 15(7), pp.2787-2807.

Sizan, N.S., Dey, D., Layek, M.A., Uddin, M.A. and Huh, E.N. (2025). Evaluating blockchain platforms for iot applications in industry 5.0: A comprehensive review. *Blockchain: Research and Applications*, p.100276.

Syed, N., Anwar, A., Baig, Z. and Zeadally, S. (2025). Artificial Intelligence as a Service (AIaaS) for Cloud, Fog and the Edge: State-of-the-Art Practices. *ACM Computing Surveys*, 57(8), pp.1-36.

Tadi, S.R.C.C.T., EVALUATING BLOCKCHAIN AND HOMOMORPHIC ENCRYPTION FOR SECURE DATA PROCESSING IN MULTI-CLOUD HYBRID DATABASE SYSTEMS: A SYSTEMATIC LITERATURE REVIEW.