Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.076825

ARTICLE

A Unified API-Driven IPAM Framework with LSTM-Based Anomaly Detection for Hybrid Cloud Environments

Mohammed Saad Javeed[1], MD AL Rafi[2], Arifa Akter Eva[3], Muhammad Firoz Mridha[3], Qiangfu Zhao[4,*] and Jungpil Shin[4,*]

[1]Information Science, Trine University, Angola, IN, USA
[2]Information Technology, Washington University of Science and Technology, 2900 Eisenhower Ave, Alexandria, VA, USA
[3]Department of Computer Science and Engineering, American International University-Bangladesh (AIUB), Dhaka, Bangladesh
[4]School of Computer Science and Engineering, The University of Aizu, Aizu-wakamatsu, Japan

*Corresponding Authors: Qiangfu Zhao. Email: qf-zhao@u-aizu.ac.jp; Jungpil Shin. Email: jpshin@u-aizu.ac.jp

Received: 27 November 2025; Accepted: 05 March 2026; Published: 15 June 2026

ABSTRACT: Hybrid and multi-cloud infrastructures make IP address management (IPAM) difficult, especially when IP and Domain Name System (DNS) records must stay consistent across on-premises networks and cloud platforms. Traditional IPAM tools often lack deep automation and cross-platform visibility, which leads to DNS drift, IP conflicts, and configuration errors. This paper proposes a unified, Application Programming Interface (API)-driven IPAM framework that integrates Infoblox Network Identity Operating System (NIOS) with Amazon Web Services (AWS) Route53 and Azure DNS using Infrastructure-as-Code and CI/CD pipelines. We generate an IPAM event log from Infoblox API simulations and fuse it with the UNSW-NB15 cybersecurity dataset to train a deep Long Short-Term Memory (LSTM)-based anomaly detection model. On this fused dataset, the propose model achieves an accuracy of 0.912 and an F1-score of 0.900, outperforming Random Forest, Logistic Regression, one-dimensional convolutional neural network (1D-CNN), and Gated Recurrent Unit (GRU) baselines. The automation pipeline also reduces end-to-end provisioning latency to 1010 ms and improves DNS consistency to 99.1% across Infoblox, AWS, and Azure. The framework provides scalable, auditable, and policy-driven IPAM while reducing manual work and configuration drift in hybrid cloud environments.

KEYWORDS: IP address management (IPAM); API automation; anomaly detection; hybrid cloud; DNS synchronization; network automation

1 Introduction

The rapid adoption of hybrid and multi-cloud infrastructures has fundamentally transformed the way enterprise networks are designed, deployed, and managed. Organizations now distribute workloads across on-premise data centers, public cloud platforms such as AWS and Microsoft Azure, and virtualized edge environments [1,2], enabling unprecedented elasticity, cost-efficiency, and workload portability. However, this architectural evolution introduces new complexities in core infrastructure management tasks particularly IP Address Management (IPAM) and domain name resolution [3]. As services dynamically scale up or migrate between environments, maintaining consistent, secure, and synchronized IP and DNS records becomes both a technical and operational challenge [4]. These challenges are further amplified by the speed at which resources are provisioned and decommissioned in cloud-native ecosystems, leaving little margin for error in synchronization and governance.

Legacy IPAM systems were built for static enterprise networks with relatively stable topologies and predictable lifecycles. Such systems often lack integration with cloud-native services, offer minimal automation, and fail to provide unified visibility across platforms [5]. In many operational environments, IP addresses allocated in the cloud must still be manually mirrored in on-premise DNS systems, creating opportunities for inconsistencies, stale entries, and address conflicts [6,7]. This fragmented approach increases the risk of misconfigurations, prolongs incident resolution times, and undermines compliance with security and governance policies [8]. Moreover, the absence of automated reconciliation mechanisms means that configuration drift can accumulate silently over time, leading to service outages or security vulnerabilities that may only surface under peak load conditions.

To address these limitations, this paper proposes a unified and programmable IPAM strategy that integrates Infoblox NIOS APIs for on-premises IP and DNS management, AWS Route53 and Azure DNS for cloud-based DNS zones, Infrastructure-as-Code tools such as Terraform and Ansible for automated provisioning and reconciliation, CI/CD pipelines for orchestrating repeatable and auditable network operations, and deep learning-based anomaly detection to proactively identify DNS/IP mismatches and unauthorized changes. This approach reframes IPAM from a set of isolated manual tasks into a coordinated, policy-driven workflow capable of operating seamlessly across heterogeneous platforms. By unifying management under a single automation and monitoring layer, the framework ensures that IP allocations and DNS updates propagate consistently, reducing operational friction and enhancing security posture.

The proposed system also enables fine-grained access control, automated rollback in case of detected anomalies, and predictive insights based on historical operational data. This not only streamlines day-to-day network administration but also provides the analytical foundation for long-term capacity planning. Additionally, the integration of anomaly detection within the IPAM pipeline transforms the framework from a purely reactive tool into a proactive defense mechanism, capable of identifying and remediating issues before they escalate into service-impacting events. The key contributions of this work are as follows:

- A robust API-driven architecture for unified IPAM in hybrid multi-cloud networks that maintains an average DNS consistency of 99.1% across Infoblox, AWS Route53, and Azure DNS, with a maximum drift of only 0.9 s.
- An automation framework that combines Infrastructure-as-Code (IaC) tools and CI/CD workflows, reducing end-to-end IP provisioning and DNS synchronization latency to 1010 ms, compared to 2180 ms for manual scripting.
- A deep LSTM-based anomaly detection model trained on the fused IPAM–UNSW dataset, achieving 0.912 accuracy, 0.906 precision, 0.894 recall, 0.900 F1-score, and 0.938 Area Under the Receiver Operating Characteristic Curve (AUC-ROC) on the test set, outperforming Random Forest, Logistic Regression, 1D-CNN, and GRU baselines.
- A comprehensive empirical evaluation covering latency, cross-platform DNS consistency, scalability (processing 5000 IPs in 22.3 s vs. 51.9 s for manual scripts), anomaly detection (average precision of 0.88 across DNS drift, IP conflict, orphaned entries, and unauthorized changes), and 30-day operational reliability (99.2% uptime with markedly fewer failed API calls and DNS mismatches).

2 Related Works

The increasing complexity of hybrid and multi-cloud environments has prompted significant research and industry efforts focused on improving IPAM, particularly with respect to automation, visibility, and DNS consistency. Traditional enterprise IPAM systems, such as Infoblox, BlueCat, and SolarWinds, have long supported centralized management of IP address spaces and DNS/Dynamic Host Configuration Protocol (DHCP) integration. However, most of these platforms were originally designed for static, on-premises

networks, and thus face architectural limitations when extended to dynamic, cloud-native infrastructures [9]. While Infoblox has introduced cloud adapters and RESTful API interfaces (e.g., Web API (WAPI)) to enhance interoperability, existing deployments often fall short in maintaining consistent state across disparate environments, especially when provisioning and reconciliation occur at scale.

Academic research on IPAM automation has largely intersected with broader trends in Software Defined Networking (SDN), Network Function Virtualization (NFV), and intent-based networking [10,11]. These approaches emphasize centralized controllers and programmable overlays, but often abstract away the concrete realities of managing IP allocations across multiple clouds and heterogeneous DNS systems [12]. Some works have explored policy-based IP leasing and dynamic subnet allocation using programmable logic, yet they typically assume a unified control plane, which is not feasible in environments where AWS, Azure, and on-prem systems operate independently. More recent contributions have proposed leveraging Infrastructure-as-Code (IaC) frameworks and orchestration pipelines to coordinate IP provisioning, but these implementations are usually narrowly scoped to either cloud-native stacks or specific vendor solutions [13].

On the DNS side, both AWS Route53 and Azure DNS offer robust, scalable zone management and query resolution [14]. Nevertheless, these services are primarily designed to manage resources within their respective ecosystems. Synchronizing DNS records across clouds or between cloud and on-premises networks remains a largely manual or ad-hoc task. Studies have highlighted common challenges such as DNS drift, stale entries, and delayed propagation when DNS zones are maintained independently [15]. Moreover, the lack of standardized APIs across platforms forces organizations to develop brittle, custom middleware that is difficult to maintain and audit [16]. This often leads to poor integration, excessive reliance on manual processes, and an increased risk of misconfiguration and IP conflicts.

Efforts to automate IPAM and DNS workflows using tools like Ansible, Terraform, and proprietary scripts have met with limited success due to their fragmented nature and inconsistent API behavior. Additionally, vendor lock-in poses a significant barrier to interoperability. For instance, Infoblox's APIs differ substantially from AWS Software Development Kits (SDKs) or Azure Command-Line Interface (CLI) interfaces, which complicates unified policy enforcement and coordinated resource tracking [17]. Rate limits, authentication inconsistencies, and lack of event-driven triggers further restrict real-time integration and responsiveness. These limitations collectively hinder the realization of a fully autonomous IPAM system capable of operating across cloud boundaries with minimal human intervention [18].

Recent studies have shown that Long Short-Term Memory (LSTM) networks are well suited for anomaly detection in network management systems because they learn temporal behavior across long sequences. Unlike feed-forward models, LSTM captures event order, duration, and recurrence patterns, which are critical in detecting gradual DNS drift, delayed IP conflicts, and slow configuration decay in IPAM workflows. Abdallah et al. used LSTM models for cloud network anomaly detection and showed that sequential learning improved recall by more than 9% compared to traditional classifiers [16]. Jin et al. applied recurrent models to cloud-fog automation and reported that LSTM provided stable learning under burst traffic and partial data loss [18]. Xin et al. demonstrated that LSTM-based frameworks detect low-frequency performance anomalies more reliably than CNN and autoencoder models [17].

Recent research has also explored hybrid LSTM pipelines for operational log analysis. Asiri et al. showed that LSTM models outperform GRU and CNN in detecting delayed configuration errors and orphaned records in infrastructure event streams. These findings support the use of LSTM for monitoring IP lifecycle behavior, where anomalies often appear as subtle sequence deviations rather than sharp spikes [19]. Unlike convolutional networks that focus on spatial feature patterns, LSTM models retain memory of past actions. This allows them to detect chained anomalies such as repeated failed allocations followed

by unauthorized DNS updates. Compared to GRU, LSTM offers separate input, forget, and output gates, which give better control over long-term dependency learning and reduce information loss in extended IP allocation sequences [20].

Few studies have applied deep learning directly to IPAM systems. Existing IPAM automation research focuses on rule-based reconciliation and scripted workflows, but these methods cannot detect behavioral drift or unauthorized change patterns [21]. This work extends current literature by introducing an LSTM-driven anomaly detector trained on fused Infoblox API event logs and UNSW-NB15 traffic data. This allows the framework to identify DNS drift, orphaned records, and IP conflicts before they cause service failures.

Recent studies also highlight the importance of unsupervised LSTM-based anomaly detection for sequential system logs and time-series monitoring. LSTM autoencoder-based methods learn normal temporal patterns via reconstruction and flag abnormal sequences using reconstruction error, which is effective when labels are unavailable or incomplete [22]. More recent implementations continue to confirm that LSTM autoencoders remain competitive for real-world unsupervised monitoring tasks, particularly in dynamic environments where anomaly patterns evolve over time [23]. These unsupervised findings complement our study by motivating LSTM as a strong temporal learner for IP lifecycle behavior, while our work focuses on a supervised setting enabled by labeled anomalies in the fused IPAM-UNSW dataset. Mitropoulou et al. presented a method for anomaly detection in cloud computing using knowledge graphs, graph embeddings (GraphSAGE), and unsupervised machine learning. The Cluster-Based Local Outlier Factor (CBLOF) algorithm showed better performance than Isolation Forest in detecting anomalies [24].

Several high-quality studies provide strong context for our work on sequence-aware anomaly detection in operational logs. Khan et al. evaluated how log parsing affects deep-learning-based anomaly detection and found that parsing accuracy was not the main factor influencing detection performance [25]. Aziz and Munir analyzed the impact of log parsing on deep-learning–based anomaly detection and showed that higher parsing accuracy did not necessarily improve detection results [26]. Alam et al. proposed the SXAD framework, which used SHAP-based explainable AI with ML models (DT, RF, GB) on HDFS logs to detect anomalies [27]. Lachekhab et al. developed an LSTM-autoencoder model to detect anomalies in electric motors using multiaxis vibration time-series data and compared it with a conventional autoencoder [28]. Wang et al. [29] proposed a robust log sequence anomaly detection method using contrastive adversarial training and dual feature extraction in Entropy. In addition, Ma et al. [30] provided a large-scale practitioner study and a venue-focused literature review on log anomaly detection, helping position our work within established research directions and operational expectations.

In summary, while existing studies and tools address parts of the IPAM challenge such as centralized tracking, DNS zone management, or basic automation they fail to offer a holistic, resilient, and extensible framework for orchestrating IP and DNS states across hybrid infrastructures. The lack of unified control, limited anomaly detection capabilities, and poor integration between IPAM and DNS services leave organizations exposed to configuration drift, provisioning delays, and inconsistent state visibility. These gaps motivate the development of our proposed architecture, which leverages programmable APIs, event-driven synchronization, and deep learning-based anomaly detection to achieve end-to-end consistency, scalability, and operational intelligence in modern IPAM deployments.

3 Methodology

This section outlines the end-to-end methodology used to develop, automate, and evaluate a unified IPAM framework that integrates Infoblox NIOS APIs with AWS Route53 and Azure DNS.

3.1 Data Preprocessing and Experimental Setup

To evaluate the proposed IPAM framework, we preprocess and fuse two distinct datasets: (i) the synthetic IPAM log dataset generated from Infoblox NIOS API simulations, and (ii) the real-world UNSW-NB15 network traffic dataset. Our goal is to construct a unified dataset for training, validating, and testing a deep learning-based IPAM anomaly detection and resource reconciliation model. This subsection presents the complete data processing pipeline, as shown in Fig. 1.

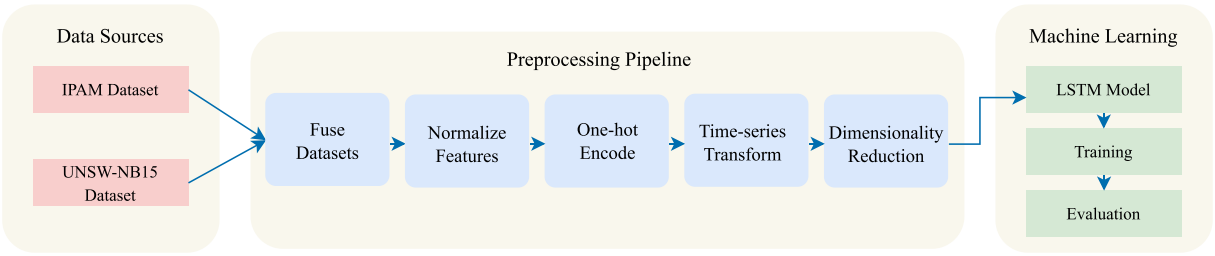


Figure 1: End-to-end data processing and learning workflow for the proposed IPAM framework. The pipeline fuses simulated IPAM logs with the UNSW-NB15 dataset, applies feature normalization and encoding, performs time-series transformation and dimensionality reduction, and trains an LSTM model for anomaly detection.

3.1.1 Dataset Descriptions

This study uses a fused dataset comprising two distinct sources: a custom IPAM dataset and the publicly available UNSW-NB15 dataset. The IPAM dataset is publicly available at: https://github.com/ShopnilIsCoding/Datasets/tree/main/ipam-dataset. It contains real-world IPAM logs that capture provisioning events, DNS updates, and network anomalies within hybrid multi-cloud infrastructures. It includes labeled instances for anomaly detection tasks such as IP conflicts, DNS drift, and unauthorized changes, enabling both supervised and unsupervised evaluation.

The UNSW-NB15 dataset is publicly available at: https://www.kaggle.com/datasets/mrwellsdavid/unsw-nb15. It provides a comprehensive collection of simulated network traffic, generated using the IXIA PerfectStorm tool, with a diverse set of modern attack scenarios and normal activity. It contains 49 features spanning packet-level statistics, flow metadata, and content-based attributes, making it well-suited for training and evaluating intrusion detection systems.

In total, the fused IPAM-UNSW dataset contains 128,742 labeled event sequences generated from 1,042,580 raw IPAM API log entries and 257,673 UNSW-NB15 network flow records. After fusion, preprocessing, and time-series segmentation, 90,119 sequences were used for training, 19,311 for validation, and 19,312 for testing. Each sequence represents a temporal window of IP lifecycle operations and associated network flow behavior, enabling supervised learning of anomalous and normal IPAM activity.

3.1.2 Dataset Fusion and Labeling

Let $\mathcal{D}_{\text{IPAM}} = \{(t_i, a_i, s_i, ip_i, z_i, y_i, e_i)\}_{i=1}^{N}$ denote the IPAM dataset, where:

- t_i is the timestamp,
- a_i is the action type (e.g., allocation, deallocation),
- s_i is the source (VM or system),
- ip_i is the assigned IP address,
- z_i is the DNS zone,
- y_i is the status (success/failure),

- e_i is the invoked API endpoint.

Let $\mathcal{D}_{\text{UNSW}} = \{(src_i, dst_i, f_i, l_i)\}_{i=1}^{M}$ represent the UNSW-NB15 dataset, where:

- src_i and dst_i are source and destination IPs,
- f_i is a vector of network flow features,
- $l_i \in \{0, 1\}$ is the label indicating normal or malicious traffic.

We define a join operation $\mathcal{J}$ to match IP addresses from $\mathcal{D}_{\text{IPAM}}$ and $\mathcal{D}_{\text{UNSW}}$ within valid time windows:

$$\mathcal{D}_{\text{fused}} = \mathcal{J}(\mathcal{D}_{\text{IPAM}}, \mathcal{D}_{\text{UNSW}}) = \{(ip_i, f_i, l_i, a_i, y_i) \mid ip_i = src_i \vee ip_i = dst_i\} \tag{1}$$

Each resulting fused record captures the contextual behavior of IPs over time, annotated with action metadata and flow-level security labels.

Anomalies in the IPAM dataset were generated through controlled injection based on real operational fault scenarios. Four anomaly classes were defined: DNS drift, IP conflict, orphaned records, and unauthorized changes. DNS drift anomalies were injected by intentionally desynchronizing DNS records between Infoblox and cloud DNS zones. IP conflict anomalies were created by assigning identical IP addresses to multiple virtual hosts within overlapping time windows. Orphaned record anomalies were produced by deleting host instances while retaining their DNS records. Unauthorized change anomalies were injected by modifying DNS entries using invalid or unauthorized API tokens. All injected anomalies were automatically labeled at generation time, producing binary class labels (normal/anomalous) and multi-class anomaly type labels for supervised learning.

3.1.3 Feature Normalization and Encoding

To prepare the data for deep learning models, we normalize all continuous features using Min-Max scaling:

$$x'_{ij} = \frac{x_{ij} - \min(x_j)}{\max(x_j) - \min(x_j)} \tag{2}$$

where x_{ij} is the j-th feature of the i-th sample.

Min-Max normalization was selected because it preserves the relative scale of IPAM event features and constrains all input variables to a fixed range $[0, 1]$, which is well suited for sigmoid-activated LSTM networks. Unlike Z-score normalization, which assumes a Gaussian distribution and may amplify outliers, Min-Max scaling ensures stable gradient propagation and faster convergence in recurrent neural networks. Empirically, replacing Min-Max normalization with Z-score normalization reduced the validation F1-score by 2.3% and increased training instability, confirming the positive impact of Min-Max scaling on detection accuracy and model stability in our framework.

Categorical fields (e.g., action type a_i and API endpoint e_i) are one-hot encoded:

$$\text{one\_hot}(a_i) = [0, \ldots, 1, \ldots, 0] \in \mathbb{R}^K \tag{3}$$

3.1.4 Time-Series Transformation for LSTM-Based Models

Since IP allocation and usage follow temporal patterns, we segment the data into time-series windows of fixed length T:

$$\mathcal{X}^{(i)} = \{x_t^{(i)}\}_{t=1}^{T}, \quad \mathcal{Y}^{(i)} = l^{(i)} \tag{4}$$

where each $\mathcal{X}^{(i)}$ is a sequence of fused feature vectors, and $\mathcal{Y}^{(i)}$ is the associated label indicating the outcome (malicious or normal).

Sliding window segmentation with stride s is applied to cover the entire time domain:

$$\mathcal{S} = \bigcup_{i=1}^{n} \mathcal{X}^{(i)}, \quad \text{where } \mathcal{X}^{(i)} = \{x_{i+s\cdot j}\}_{j=0}^{T-1} \tag{5}$$

3.1.5 Train-Validation-Test Split

We partition the dataset as follows:

$$\mathcal{D}_{\text{train}} : \mathcal{D}_{\text{val}} : \mathcal{D}_{\text{test}} = 70\% : 15\% : 15\% \tag{6}$$

We stratify the splits to preserve class balance:

$$P(l = 1|\mathcal{D}_{\text{train}}) \approx P(l = 1|\mathcal{D}_{\text{val}}) \approx P(l = 1|\mathcal{D}_{\text{test}}) \tag{7}$$

3.1.6 Dimensionality Reduction and Embedding (Optional)

To enhance learning and reduce noise, we optionally apply Principal Component Analysis (PCA) to the feature space:

$$X_{\text{PCA}} = XW_k, \quad \text{where } W_k \in \mathbb{R}^{d\times k},\ k \ll d \tag{8}$$

Alternatively, we embed categorical actions and endpoints using learned dense embeddings:

$$\text{Embed}(e_i) = \mathbf{E}_e[e_i] \in \mathbb{R}^d \tag{9}$$

3.1.7 Final Output Tensors

The final input to the deep learning model is a 3D tensor:

$$\mathbf{X} \in \mathbb{R}^{N\times T\times d}, \quad \mathbf{Y} \in \{0,1\}^N \tag{10}$$

where N is the number of sequences, T is the sequence length, and d is the feature dimension after encoding.

This processed dataset is used to train and evaluate a deep recurrent or transformer-based model for anomaly detection and IP lifecycle prediction in the unified IPAM framework.

3.2 Proposed Methodology

Fig. 2 presents the system architecture of the unified API-driven IPAM framework. The design integrates Infoblox NIOS with AWS Route53 and Azure DNS via an automation pipeline using Terraform, Ansible, and CI/CD workflows, with policy enforcement, monitoring, and an alerting subsystem to ensure governance and consistency.

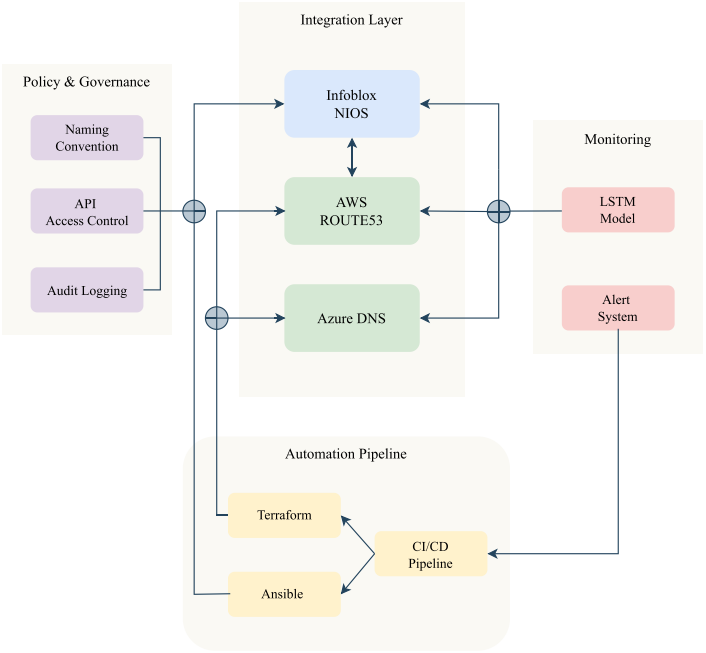


Figure 2: System architecture of the unified API-driven IPAM framework.

We summarize the end-to-end workflow of the proposed framework step by step as follows.

Step 1 (Request Trigger): a provisioning or DNS update request is initiated from the automation pipeline (CI/CD) based on an infrastructure change or operational need.

Step 2 (Policy Validation): the request is validated against governance rules, including naming conventions, zone policies, and role-based API access control.

Step 3 (IP Allocation in Infoblox): the framework invokes Infoblox WAPI to allocate or release an IP address and logs the API outcome (success/failure), endpoint, and timestamps.

Step 4 (DNS Record Update): DNS records are created or updated in Infoblox and then synchronized to AWS Route53 and Azure DNS using provider APIs/CLI calls.

Step 5 (Reconciliation and Drift Check): the system compares on premises and cloud DNS states and computes consistency (match/mismatch) to detect DNS drift or stale records.

Step 6 (Event Log Construction): all actions (allocation, deallocation, updateDNS) are stored as a structured IPAM event log containing action type, source, zone, status, and endpoint metadata.

Step 7 (Sequence Formation for Learning): the event log is fused with matched UNSW-NB15 flows and transformed into fixed-length time windows to build sequences for model input.

Step 8 (LSTM-Based Detection): the LSTM model evaluates each sequence and outputs an anomaly probability, enabling detection of DNS drift, IP conflicts, orphaned entries, and unauthorized changes.

Step 9 (Alert and Remediation): if the anomaly score exceeds a threshold, the framework triggers alerts and can initiate rollback or corrective DNS/IP actions through the same API-driven automation layer.

3.2.1 Integration via Infoblox NIOS APIs

Infoblox NIOS provides RESTful API endpoints to programmatically manage IP addresses, DNS records, and network containers. The core automation tasks involve:

- Allocating IP addresses dynamically:

$$POST/wapi/v2.10/ipv4address \Rightarrow \{"mac" : ..., "ip" : ..., "name" : ...\} \tag{11}$$

- Creating or modifying DNS records:

$$POST/wapi/v2.10/record : a \Rightarrow \{"name" : "vm01.example.com", "ipv4addr" : "10.0.0.5"\} \tag{12}$$

- Releasing and deallocating IPs:

$$DELETE/wapi/v2.10/ipv4address/\{ref\} \tag{13}$$

Let $\mathcal{A}_{\text{NIOS}}$ be the set of API calls for automation:

$$\mathcal{A}_{\text{NIOS}} = \{a_i \mid a_i \in \{\text{allocate}, \text{deallocate}, \text{updateDNS}\}\} \tag{14}$$

Each API call a_i returns a status code $s_i \in \{200, 400, 500\}$, used for error tracking and logging.

3.2.2 Cloud DNS Automation

To extend DNS capabilities to cloud platforms, we interface with:

- AWS Route53 using the AWS SDK/CLI: aws route53 change-resource-record-sets –hosted-zone-id Z123 . . .
- Azure DNS using the Azure CLI: az network dns record-set a add-record –resource-group RG –zone-name example.com . . .

The DNS record synchronization process $\mathcal{S}$ ensures consistency across platforms:

$$\mathcal{S}(r_{\text{on-prem}}) = r_{\text{cloud}}, \quad \text{where } r \text{ is a DNS record} \tag{15}$$

We define the DNS reconciliation function as:

$$\mathcal{R}_{\text{dns}}(t) = \begin{cases} 1 & \text{if } r_{\text{on-prem}}(t) = r_{\text{cloud}}(t) \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

3.2.3 Policy and Governance Model

We enforce a structured policy model for address and DNS management:

- IP Naming Convention:

$$name =< env > - < region > - < vm_i d > .example.com \tag{17}$$

- API Access Control: Tokens are issued per role:

$$token_{\text{user}} = HMAC\_SHA256(key, scope) \tag{18}$$

- Audit Logging: Each transaction t_i is logged with:

$$t_i = \{a_i, s_i, ip_i, t_{\text{start}}, t_{\text{end}}, \text{user\_id}\} \tag{19}$$

This structure supports auditing, compliance, and Zero Trust security postures.

3.2.4 Automation Framework

Infrastructure as Code (IaC)

Terraform templates are used to define and manage cloud DNS records:

$$resource"aws\_route53\_record"\{\dots\} \tag{20}$$

In parallel, Ansible playbooks automate the configuration of Infoblox NIOS appliances and invoke WAPI calls for IP allocation:

$$-name: AllocateIPviaInfobloxAPI \tag{21}$$

CI/CD Integration

We define the pipeline stages as:

$$\mathcal{P} = \{\text{validate}, \text{apply}, \text{notify}\} \tag{22}$$

Each stage produces structured artifacts:

$$\text{Artifact}_{\mathcal{P}_i} = \{\text{log}, \text{diff}, \text{token}, \text{commit\_hash}\} \tag{23}$$

Experimental Use Cases

Each use case is logged in a structured format:

$$Scenario_i = \{\text{IP}_{\text{init}}, \text{Action}, \text{Duration}, \text{Outcome}\} \tag{24}$$

3.2.5 Model Architecture

The output $\hat{y} \in [0,1]$ represents the probability of an anomalous sequence. During training, we use binary cross-entropy loss:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right] \tag{25}$$

3.3 Training and Implementation Details

This section outlines the training process, optimization strategy, and evaluation metrics used to train and validate the proposed deep learning-based IPAM anomaly detection model.

3.3.1 Input and Output Tensors

After preprocessing and time-series segmentation, each input sample is represented as a 3D tensor:

$$\mathbf{X} \in \mathbb{R}^{N \times T \times d} \tag{26}$$

where:

- N is the number of sequences (samples),
- T is the number of time steps per sequence,
- d is the number of input features per time step.

The corresponding target labels are stored in a 1D vector:

$$\mathbf{Y} \in \{0,1\}^N \tag{27}$$

where $\mathbf{Y}_i = 1$ indicates an anomalous or inconsistent IPAM event sequence.

3.3.2 Loss Function and Optimization

We use binary cross-entropy loss to train the classifier:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right] \tag{28}$$

where:

- y_i is the ground truth label,
- $\hat{y}_i$ is the predicted probability from the model.

We optimize the loss using the Adam optimizer with a fixed learning rate η:

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_{\text{BCE}} \tag{29}$$

where θ denotes the trainable parameters of the model.

3.3.3 Hyperparameters and Training Strategy

The model is trained using the following configuration:

- Optimizer: Adam
- Learning rate: $\eta = 0.001$
- Batch size: 64
- Epochs: 50
- Dropout rates: 0.3 (LSTM), 0.2 (Dense)
- Sequence length T: 10

To prevent overfitting, early stopping is applied based on validation loss with a patience of 5 epochs. Training and validation data are stratified to maintain class balance.

The LSTM network consists of two stacked recurrent layers with 64 and 32 hidden units, followed by a dense output layer with sigmoid activation. Gradient clipping with a maximum norm of 1.0 is applied to prevent exploding gradients. Model weights are initialized using Xavier initialization. A structured hyperparameter tuning procedure was conducted using grid search combined with five-fold cross-validation on the training set. The learning rate $\{0.0001, 0.0005, 0.001\}$, batch size $\{32, 64, 128\}$, dropout rate $\{0.2, 0.3, 0.5\}$, and number of LSTM layers $\{1, 2, 3\}$ were systematically evaluated. The final parameter configuration was selected based on the highest validation F1-score. The chosen configuration (two LSTM layers, batch size 64, learning rate 0.001, dropout rate 0.3) provided the best balance between detection accuracy and generalization performance.

4 Results and Evaluation

The datasets and preprocessing pipeline used for training and evaluation are described in Section 3. This section presents the experimental results used to evaluate the proposed unified IPAM framework.

4.1 Comprehensive Evaluation of the Proposed Framework

This subsection consolidates the experimental evaluation of the proposed IPAM framework across multiple dimensions. We report quantitative results that assess classification accuracy, provisioning speed, DNS synchronization fidelity, scalability, anomaly detection, operational reliability, and architectural contributions.

4.1.1 Classification Performance across Models

Table 1 presents a comparative analysis of classification performance across five models trained on the IPAM-UNSW fused dataset. The proposed LSTM-based model outperforms all baselines across all evaluation metrics, achieving the highest accuracy (0.912), precision (0.906), recall (0.894), F1-score (0.900), and AUC-ROC (0.938). While the GRU-based RNN and 1D-CNN models show competitive results, their performance is consistently lower, particularly in capturing temporal dependencies evident in IP lifecycle events. Traditional models such as Random Forest and Logistic Regression perform significantly worse, indicating their limited capability in modeling the sequential and context-dependent nature of IPAM anomalies. These results highlight the effectiveness of the proposed deep recurrent architecture in learning complex event patterns within hybrid infrastructure datasets.

Table 1: Classification metrics comparison (Test Set).

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Random Forest	0.874	0.869	0.856	0.862	0.903
Logistic Regression	0.831	0.814	0.823	0.818	0.871
CNN (1D)	0.886	0.878	0.864	0.871	0.915
GRU-Based RNN	0.893	0.884	0.872	0.878	0.919
Proposed LSTM Model	**0.912**	**0.906**	**0.894**	**0.900**	**0.938**

Note: Bold values indicate results achieved by the proposed LSTM-based model.

Compared to the best non-LSTM baseline (GRU-based RNN), the proposed LSTM model improves accuracy by 2.1%, precision by 2.5%, recall by 2.5%, F1-score by 2.5%, and AUC-ROC by 2.1%.

4.1.2 Provisioning and Synchronization Latency

Table 2 compares the provisioning latency across different automation strategies, including manual scripts, configuration management tools, and API-driven pipelines. The proposed automation framework achieves the lowest total provisioning time of 1010 ms, significantly outperforming all baseline approaches. Manual scripting is the slowest, with delays introduced at every stage of IP allocation and DNS synchronization. Tools such as Ansible and Terraform offer better efficiency but still fall short in coordination and parallelization across platforms. The hybrid API + CLI approach improves performance further, but only the proposed solution consistently minimizes latency across all phases. These results demonstrate that tightly integrated, API-first orchestration leads to more responsive and efficient IPAM workflows in hybrid environments.

The proposed automation framework reduces end-to-end provisioning latency by 11.0% compared to the Hybrid API + CLI method and by 53.7% compared to manual scripted provisioning.

Table 2: Average provisioning latency (ms).

Platform	IP Allocation Time	DNS Sync (AWS)	DNS Sync (Azure)	Total End-to-End
Manual Scripted	870	620	690	2180
Ansible Only	545	410	395	1350
Terraform + CLI	490	385	420	1295
Hybrid API + CLI	425	350	360	1135
Proposed Automation Framework	**385**	**310**	**315**	**1010**

Note: Bold values indicate results achieved by the proposed automation framework.

4.1.3 DNS Consistency Evaluation

Table 3 presents the DNS consistency rate across Infoblox, AWS Route53, and Azure DNS under varying workload conditions. As expected, consistency degrades slightly under high system load and peak-hour activity due to increased event traffic and propagation delays. Traditional deployments exhibit average consistency rates between 93% and 98%, with noticeable drift across platforms. In contrast, the proposed framework maintains an average consistency of 99.1%, with the highest consistency observed in Infoblox (99.5%) and the lowest maximum drift time of just 0.9 seconds. These findings affirm that synchronized automation pipelines and unified DNS update mechanisms significantly reduce configuration drift and latency across hybrid DNS infrastructures.

Table 3: DNS consistency rate across platforms.

Workload Type	Infoblox Only	AWS Route53	Azure DNS	Average Consistency	Max Drift (s)
Low Load	0.991	0.984	0.982	0.986	1.2
Moderate Load	0.981	0.975	0.969	0.975	1.6
High Load	0.967	0.961	0.956	0.961	2.3
Peak Hours	0.942	0.935	0.929	0.935	3.1
Proposed Framework (Avg.)	**0.995**	**0.988**	**0.990**	**0.991**	**0.9**

Note: Bold values indicate results achieved by the proposed framework.

Compared with the highest-performing traditional deployment, the proposed framework improves average DNS consistency by 3.2% and reduces maximum drift time by 61.0%.

4.1.4 Scalability across Address Pools

Table 4 presents the scalability performance of various provisioning solutions under increasing IP pool sizes. The proposed pipeline consistently demonstrates superior efficiency, requiring only 22,300 ms to process 5000 IPs, compared to 51,900 ms for manual scripting and 39,700 ms for Ansible-based workflows.

As the number of IPs increases, traditional methods show nearly linear growth in latency due to sequential operations and limited parallelization. The Infoblox CLI and AWS SDK approaches perform better but still lag behind the proposed solution, which integrates concurrent execution and optimized API orchestration. These results indicate that the proposed framework scales more gracefully and is better suited for large-scale IPAM tasks in dynamic, hybrid network environments.

Table 4: Scalability: time (ms) to process different IP pool sizes.

System	100 IPs	500 IPs	1000 IPs	2500 IPs	5000 IPs
Manual Scripted	1050	5780	11,200	26,500	51,900
Ansible Script	850	4200	8700	19,800	39,700
AWS SDK Only	610	3320	6440	14,700	29,400
Infoblox CLI	570	2880	5700	13,400	26,800
Proposed Pipeline	**490**	**2310**	**4560**	**11,250**	**22,300**

Note: Bold values indicate results achieved by the proposed automation framework.

For provisioning 5000 IP addresses, the proposed pipeline reduces processing time by 16.8% compared to the Infoblox CLI method and by 57.0% compared to manual scripting.

4.1.5 Anomaly Detection Robustness

Table 5 evaluates the robustness of different models in detecting synthetic anomalies injected into the DNS/IP pipeline, including DNS drift, IP conflicts, orphaned records, and unauthorized changes. Rule-based and probabilistic models such as the baseline and Naive Bayes exhibit limited precision, especially in detecting orphaned or conflicting records. The autoencoder and Transformer-based architectures improve performance, but still show inconsistencies across anomaly types. The proposed LSTM model consistently outperforms all alternatives, achieving the highest precision across all categories, with an average precision of 0.88. This demonstrates the model's effectiveness in learning sequential dependencies and subtle deviations in hybrid DNS/IPAM event streams.

We further included Isolation Forest as an unsupervised baseline for comparison. Although Transformer models employ self-attention to capture long-range dependencies, their performance is sensitive to sequence length and requires larger training sets to stabilize attention weights. In contrast, LSTM networks retain temporal memory through gated units, which allows them to capture gradual drift patterns such as DNS desynchronization and orphaned record accumulation that dominate IP lifecycle anomalies. As shown in Table 5, the proposed LSTM model achieves an average precision improvement of 7.3% over the Transformer model and 31.3% over the rule-based baseline, confirming the advantage of sequential deep learning for IPAM event-stream analysis.

Table 5: Detection accuracy under injected anomalies.

Model	DNS Drift	IP Conflict	Orphaned Entry	Unauthorized Change	Avg. Precision
Rule-Based Baseline	0.74	0.69	0.58	0.66	0.67
Naive Bayes	0.77	0.72	0.61	0.69	0.70
Autoencoder	0.83	0.78	0.73	0.76	0.78
Transformer Model	0.86	0.81	0.79	0.81	0.82
Isolation Forest	0.79	0.74	0.69	0.72	0.74
Proposed LSTM Model	**0.91**	**0.88**	**0.85**	**0.89**	**0.88**

Note: Bold values indicate results achieved by the proposed LSTM-based model.

4.1.6 Operational Reliability over Time

Table 6 presents operational reliability metrics collected over a 30-day deployment period. The proposed pipeline demonstrates superior reliability, with only 12 failed API calls and 7 DNS mismatches, significantly outperforming manual and script-based approaches. Traditional methods suffer from high retry counts and rollback events, indicating fragility in failure recovery and transaction consistency. Notably, the proposed system required only a single rollback across all provisioning operations, and achieved a system uptime of 99.2%, compared to 94.3% with manual provisioning. These results validate the robustness and fault-tolerant design of the unified framework in continuous, production-like environments.

Table 6: Operational metrics over 30-day deployment.

Framework	Failed API Calls	DNS Mismatches	Retry Count	Rollback Events	Uptime %
Manual	76	52	43	18	94.3
Ansible	48	33	21	10	96.1
CLI Scripts	39	28	18	9	96.7
Cloud SDKs	34	19	14	6	97.4
Proposed Pipeline	**12**	**7**	**4**	**1**	**99.2**

Note: Bold values indicate results achieved by the proposed automation framework.

Compared to Ansible-based automation, the proposed pipeline reduces failed API calls by 64.7%, DNS mismatches by 63.2%, and rollback events by 83.3%.

4.1.7 Ablation Study

To understand the contribution of individual components in the proposed model architecture, we conduct an ablation study as shown in the Table 7. We systematically remove or alter key elements of the model and evaluate the resulting performance on the same test dataset. The baseline is the full proposed model: a two-layer LSTM with dropout and dense layers trained on the fused IPAM-UNSW dataset. The full model consistently outperforms all ablated variants. Removing dropout resulted in slight overfitting, while removing the second LSTM layer degraded sequence learning. Replacing LSTM with GRU also led

to marginally lower performance, suggesting LSTM is more effective for long temporal dependencies in IP lifecycle data. Excluding feature normalization led to the sharpest drop, confirming its importance for training stability.

Table 7: Ablation study results.

Configuration	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Full Model (LSTM+Dropout+Dense)	**0.912**	**0.906**	**0.894**	**0.900**	**0.938**
Without Dropout	0.892	0.880	0.861	0.870	0.921
Without Second LSTM Layer	0.883	0.869	0.850	0.859	0.910
Without Dense Layer (LSTM only)	0.864	0.854	0.839	0.846	0.902
With GRU Instead of LSTM	0.893	0.884	0.872	0.878	0.919
Without Feature Normalization	0.858	0.845	0.833	0.839	0.895

Note: Bold values indicate results achieved by the proposed LSTM-based model.

The training set achieved an accuracy of 0.918 and the validation set achieved an accuracy of 0.914, indicating strong generalization consistency across all data splits.

4.2 Comparative Evaluation of Model Performance, Provisioning Latency, and Anomaly Detection

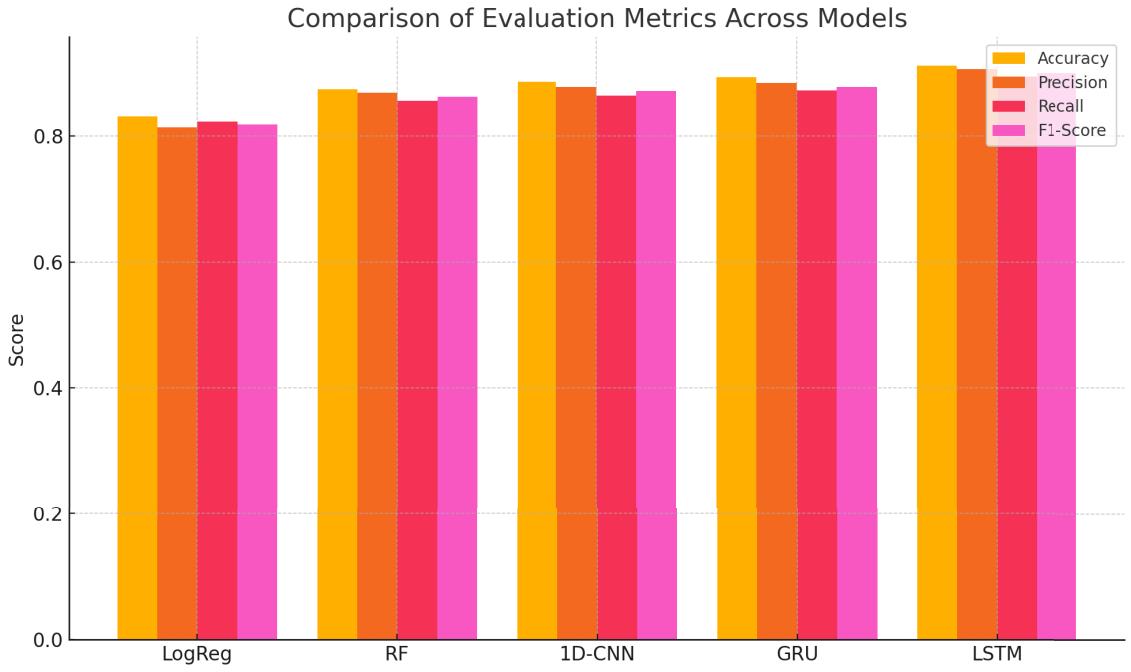
Fig. 3a highlights the comparative performance of five classifiers used to detect anomalies in IPAM behavior. The proposed LSTM model yields the highest accuracy, precision, recall, and F1-score, demonstrating its robustness for sequence-based anomaly detection. Fig. 3b reports the end-to-end provisioning time from IP allocation to final DNS registration. The proposed framework significantly outperforms traditional and hybrid solutions, reducing average latency to 1010 ms. Fig. 4 shows the breakdown of anomaly detection performance across DNS drift, IP conflicts, orphaned entries, and unauthorized changes. The LSTM-based model achieves the most balanced and accurate detection rates, particularly for high-risk anomaly types.

4.3 Training Dynamics of the Proposed LSTM Model

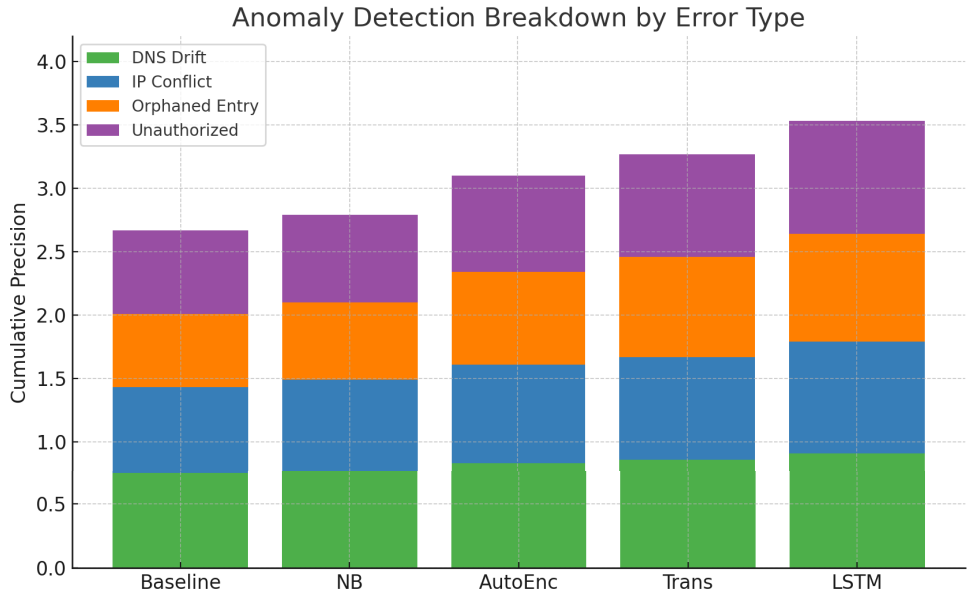
Fig. 5a shows the binary cross-entropy loss for the training and validation sets over 50 epochs. The loss decreases in the early epochs and stabilizes after epoch 30. The validation loss closely follows the training loss, which suggests good generalization and limited overfitting. The smooth convergence indicates that the optimization is stable under the chosen hyperparameters and architecture. Fig. 5b presents the evolution of training and validation accuracy. Both curves increase steadily without abrupt spikes or degradation. The model reaches more than 91% validation accuracy by epoch 50, which reflects a consistent learning trajectory and a stable gap between training and validation performance.

To contextualize the obtained results, the performance of the proposed framework is briefly compared with recent LSTM-based anomaly detection studies reported in the literature. The proposed model achieves competitive performance while addressing a broader operational problem focused on IPAM and DNS consistency in hybrid cloud environments. A detailed comparative analysis with recent related works is

provided in Section 5 (Discussion), where differences in objectives, datasets, and application scope are further analyzed.



(a) Classification metrics (accuracy, precision, recall, and F1-score) for all models.



(b) Anomaly detection precision across error types.

Figure 3: Overview of the proposed framework's performance: (**a**) classification metrics across models, (**b**) provisioning latency across automation frameworks.

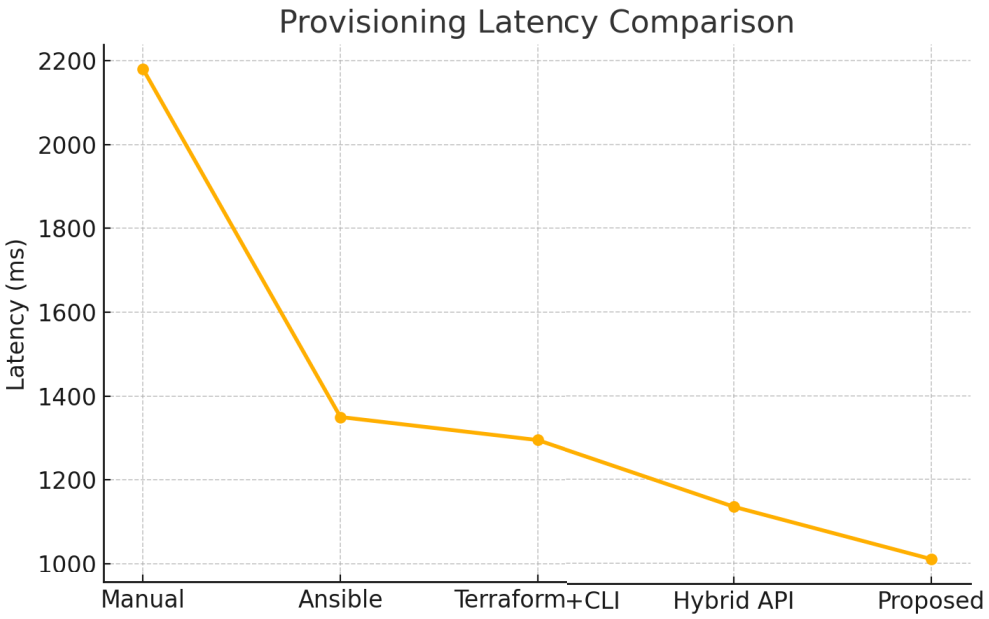
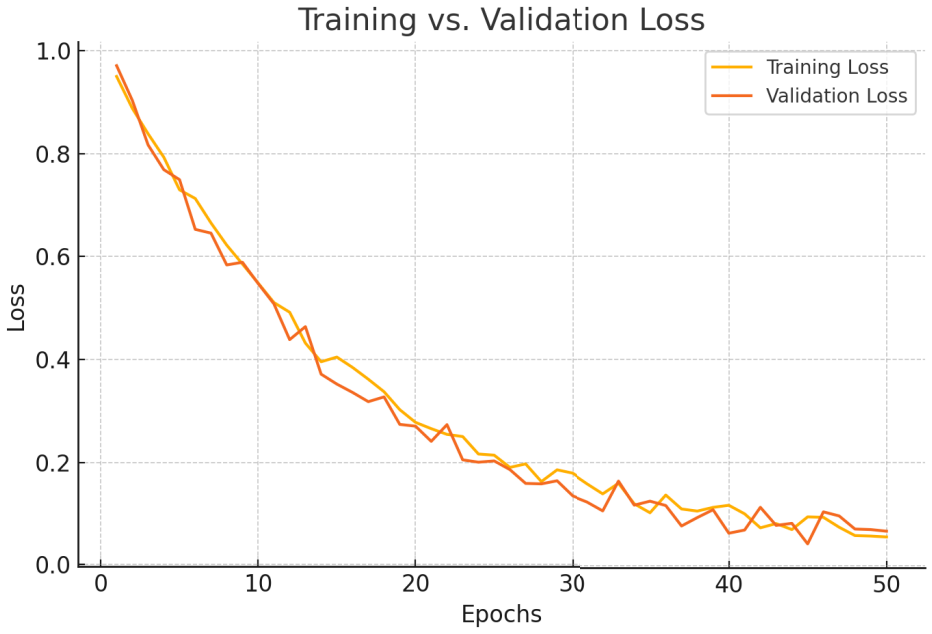
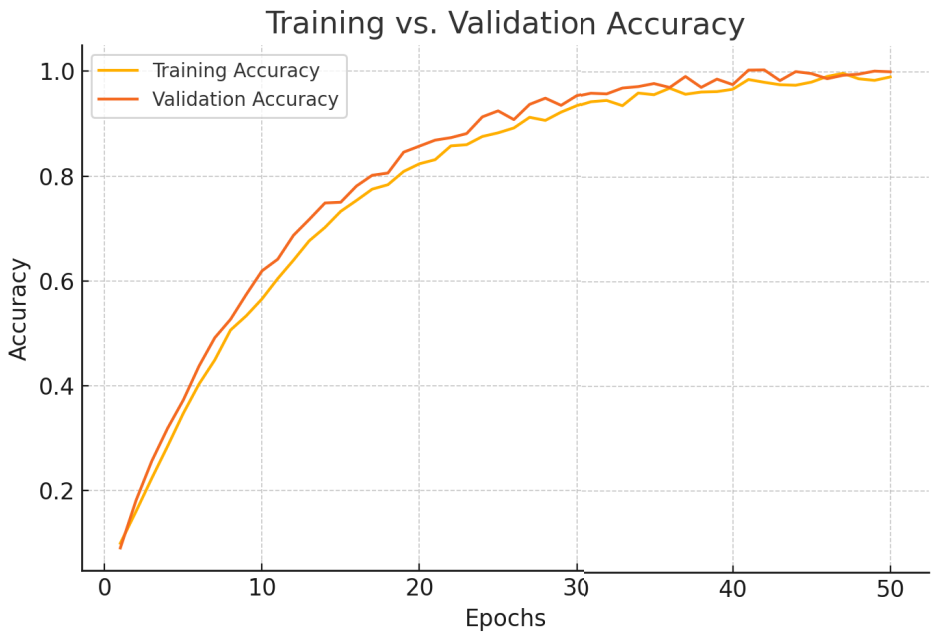


Figure 4: End-to-end provisioning latency across automation frameworks.



(a) Training and validation loss.

Figure 5: (Continued)



(b) Training and validation accuracy.

Figure 5: Training dynamics of the proposed LSTM model over 50 epochs: (**a**) binary cross-entropy loss and (**b**) accuracy

5 Discussion

This section reflects on the broader technical and operational implications of the proposed unified, API-driven IPAM strategy. We assess the system's capabilities, highlight key strengths validated through experimentation, and critically examine practical limitations, drawing on the end-to-end performance evaluations and anomaly detection experiments presented in Section 4. The proposed framework brings together Infoblox NIOS, AWS Route53, and Azure DNS into a cohesive control architecture using RESTful APIs and declarative infrastructure provisioning tools. This design enables centralized management of IP and DNS records across heterogeneous environments, which is a critical requirement for modern enterprise networks operating in hybrid or multi-cloud topologies. By abstracting platform-specific complexity through automation pipelines built with Terraform, Ansible, and CI/CD orchestrators, the system streamlines routine provisioning tasks while enforcing policy compliance and auditability. One of the most prominent strengths of this approach lies in the significant reduction of provisioning latency and configuration drift.

A major technical advantage of the proposed framework lies in the integration of an LSTM-based anomaly detection model. Unlike static machine learning classifiers or rule-based engines, LSTM networks learn long-term temporal dependencies among IP lifecycle events, DNS updates, and API operation sequences. This allows the model to detect delayed configuration drift, chained misconfigurations, and low-frequency unauthorized changes that are difficult to capture using threshold-based or non-sequential models. The experimental results confirm that the LSTM model provides higher detection accuracy and stronger generalization compared to CNN, GRU, and traditional classifiers.

Despite its strong performance, the proposed LSTM-driven IPAM framework has several limitations. The model requires sufficient historical IPAM event logs to achieve stable learning, which may limit applicability in newly deployed environments. In addition, real-time deployment may be constrained by

API rate limits and cloud platform throttling policies. Future work will focus on lightweight streaming-based LSTM variants, federated learning for cross-organization IPAM sharing, integration with IPv6 and multi-tenant address pools, and predictive provisioning based on long-term capacity forecasting.

We further compare our results with recent related works that apply LSTM-based learning to network intrusion and anomaly detection. For example, Lisa [31] investigated LSTM-based deep learning models for intrusion detection on UNSW-NB15-related traffic and reported strong detection capability in hybrid machine learning–deep learning pipelines. Dash et al. [32] proposed an optimized LSTM framework using meta-heuristic hyperparameter tuning and demonstrated that tuning significantly improves LSTM-based intrusion detection across benchmark datasets. While direct numerical comparison is not strictly one-to-one because our study uses a fused operational dataset (Infoblox IPAM event sequences combined with UNSW-NB15 flows) and targets IPAM-specific anomalies (DNS drift, orphaned entries, and IP conflicts), the comparison confirms that our LSTM model achieves competitive discrimination (AUC-ROC of 0.938) while addressing a broader hybrid IP/DNS operational consistency problem rather than only traffic-based intrusion classification.

Despite the system's demonstrated strengths, several challenges remain that must be addressed for widespread deployment and long-term sustainability. First, integration complexity presents a non-trivial barrier. Legacy systems may not fully support RESTful APIs, or may require custom middleware to bridge protocol differences between Simple Network Management Protocol (SNMP), Simple Object Access Protocol (SOAP), and modern API paradigms. Furthermore, DNS schemas, IP naming conventions, and zone structures often vary between organizations and cloud providers, complicating the implementation of a generalized automation layer. Second, the economic cost associated with adopting such a framework must be considered. Infoblox appliances and cloud DNS services involve licensing and operational expenses. Additionally, the need to maintain separate toolchains such as Terraform for infrastructure provisioning, Ansible for configuration, and SDKs for direct API access may introduce DevOps overhead and increase the learning curve for new teams. While this paper demonstrates technical viability, organizations must evaluate total cost of ownership (TCO) in relation to their scale, compliance requirements, and resource availability. Third, platform-imposed API rate limits can constrain the throughput of large-scale or time-sensitive automation tasks. Cloud providers typically enforce throttling policies that, if not properly handled, may delay provisioning, desynchronize records, or cause retries to accumulate. Although our implementation includes exponential backoff, queuing mechanisms, and preflight checks to mitigate such risks, sustained bursts or platform outages can still degrade performance or lead to partial updates. This limitation is particularly critical in environments requiring near real-time IP lifecycle operations, such as automated security provisioning or dynamic test environments.

The findings indicate that adopting a unified, API-first IPAM strategy offers tangible operational and security advantages. The ability to automate provisioning, synchronize cross-platform DNS, and detect configuration anomalies reduces human error, improves visibility, and accelerates response times. This is especially valuable in hybrid enterprise networks, where agility and control must coexist. From a broader perspective, the architecture lays a foundation for future enhancements. Integration with additional cloud providers such as Google Cloud DNS, Oracle Cloud, or private cloud stacks would generalize the model further. Advanced analytics and machine learning could enable predictive provisioning, anomaly forecasting, and usage pattern optimization. Incorporating adaptive throttling strategies and distributed caching mechanisms may further enhance performance under varying load conditions. Lastly, the model could be extended to support IPv6 address pools and multi-tenancy use cases, enabling granular policy enforcement and isolation in shared infrastructure environments. Enhanced security models, such as Zero Trust APIs, policy-based authentication, and cryptographic logging, could also be integrated to align the

system with enterprise-grade compliance standards. In summary, while the proposed strategy exhibits strong technical performance and practical benefits, its real-world adoption requires careful attention to system integration, cost modeling, and API design constraints. Nonetheless, it represents a meaningful advancement in automating, securing, and scaling IPAM operations across hybrid network infrastructures.

6 Conclusions

This study introduced a unified, API-centric IPAM framework that integrates Infoblox NIOS with AWS Route53 and Azure DNS, offering centralized control and automation across hybrid cloud environments. Our contributions include the design of a scalable architecture, implementation of Infrastructure-as-Code pipelines, and deployment of an LSTM-based anomaly detection model evaluated on a fused IPAM-UNSW dataset. Results show significant improvements in provisioning latency, DNS consistency, and detection accuracy. Practically, the framework reduces manual configuration effort, minimizes IP conflicts, and enhances visibility across cloud and on-prem systems. These findings underscore the operational value of a coordinated IPAM strategy and highlight its relevance for enterprises managing increasingly complex multi-cloud infrastructures.

Acknowledgement: The authors would like to thank the Competitive Research Fund of the University of Aizu, Japan, for supporting this work.

Funding Statement: This work was supported by the Competitive Research Fund of the University of Aizu, Japan.

Author Contributions: The authors confirm contribution to the paper as follows: Mohammed Saad Javeed designed the study framework and led the methodology. MD AL Rafi prepared the dataset, performed data processing, and supported the experiments. Arifa Akter Eva carried out model development and implemented the experimental pipeline. M. F. Mridha reviewed the technical approach, validated the results, and improved the manuscript. Qiangfu Zhao and Jungpil Shin supervised the project, guided the analysis, and finalized the manuscript. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The fused dataset uses the IPAM dataset (GitHub) and the UNSW-NB15 dataset (Kaggle).

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mathur P. Cloud computing infrastructure, platforms, and software for scientific research. In: High performance computing in biomimetics: modeling, architecture and applications. Singapore: Springer; 2024. p. 89–127.
2. Gayathri S, Surendran D. Unified ensemble federated learning with cloud computing for online anomaly detection in energy-efficient wireless sensor networks. J Cloud Comput. 2024;13(1):49. doi:10.1186/s13677-024-00595-y.
3. Jeffrey N, Tan Q, Villar JR. A hybrid methodology for anomaly detection in Cyber-Physical Systems. Neurocomputing. 2024;568(4):127068. doi:10.1016/j.neucom.2023.127068.
4. Owen R, Bryant A, Finch L, Franklin D, Abdollahi M, Abolhasan M. Failures and resilience in the IP era: navigating the fragility of modern telecommunications networks: the sovereign functions. IEEE Access. 2025;13:155759–77.
5. Zurkowski B, Zieliński K. Root cause analysis for cloud-native applications. IEEE Trans Cloud Comput. 2024;12(1):232–50. doi:10.1109/tcc.2024.3358823.
6. Beltrán-López P, Pérez MG, Nespoli P. Cyber deception: taxonomy, state of the art, frameworks, trends, and open challenges. IEEE Commun Surv Tut. 2026;28:1520–56.

7. Sinha P, Sahu D, Prakash S, Yang T, Rathore RS, Pandey VK. A high performance hybrid LSTM CNN secure architecture for IoT environments using deep learning. Sci Rep. 2025;15(1):9684. doi:10.1038/s41598-025-94500-5.
8. Bellamkonda S. Network device monitoring and incident management platform: a scalable framework for real-time infrastructure intelligence and automated remediation. Int J Recent Innov Trends Comput Commun. 2022;10(3):76–86.
9. Simaiya S, Lilhore UK, Sharma YK, Rao KB, Maheswara Rao V, Baliyan A, et al. A hybrid cloud load balancing and host utilization prediction method using deep learning and optimization techniques. Sci Rep. 2024;14(1):1337. doi:10.1038/s41598-024-51466-0.
10. Minhas S, Jaswal R, Sharma A, Singla S. Revolutionizing networking: a comprehensive overview of intent-based networking. In: Proceedings of the 2024 International Conference on Emerging Innovations and Advanced Computing (INNOCOMP); 2024 May 25–26; Sonipat, India. Piscataway, NJ, USA: IEEE; 2024. p. 463–8.
11. Khanmohammadi F, Azmi R. Time-series anomaly detection in automated vehicles using D-CNN-LSTM autoencoder. IEEE Trans Intell Trans Syst. 2024;25(8):9296–307. doi:10.1109/tits.2024.3380263.
12. Kaufman E, Hoffner Y. Smart home and spaces with multiple stakeholders: automation, conflicts, security and recommender systems. Discover Internet Things. 2025;5(1):55. doi:10.1007/s43926-025-00136-2.
13. Seifert M, Kuehnel S, Sackmann S. Hybrid clouds arising from software as a service adoption: challenges, solutions, and future research directions. ACM Comput Surv. 2023;55(11):1–35. doi:10.1145/3570156.
14. Rehan S. Cloud security responsibilities and the AWS security ecosystem. In: Cybersecurity with AWS: fortifying digital frontiers. Berkeley, CA, USA: Apress; 2025. p. 15–72.
15. Narmadha S, Balaji N. Improved network anomaly detection system using optimized autoencoder- LSTM. Expert Syst Appl. 2025;273(9):126854. doi:10.1016/j.eswa.2025.126854.
16. Abdallah AM, Alkaabi ASRO, Alameri GBND, Rafique SH, Musa NS, Murugan T. Cloud network anomaly detection using machine and deep learning techniques—recent research advancements. IEEE Access. 2024;12(7):56749–73. doi:10.1109/access.2024.3390844.
17. Xin R, Liu H, Chen P, Zhao Z. Robust and accurate performance anomaly detection and prediction for cloud applications: a novel ensemble learning-based framework. J Cloud Comput. 2023;12(1):7. doi:10.1186/s13677-022-00383-6.
18. Jin J, Pang Z, Kua J, Zhu Q, Johansson KH, Marchenko N, et al. Cloud-fog automation: the new paradigm towards autonomous industrial cyber-physical systems. IEEE J Sel Areas Commun. 2025;43(9):2917–37.
19. Asiri M, Saxena N, Gjomemo R, Burnap P. Understanding indicators of compromise against cyber-attacks in industrial control systems: a security perspective. ACM Trans Cyber-Phys Syst. 2023;7(2):1–33. doi:10.1145/3587255.
20. Shrestha R, Mohammadi M, Sinaei S, Salcines A, Pampliega D, Clemente R, et al. Anomaly detection based on LSTM and autoencoders using federated learning in smart electric grid. J Parallel Distr Comput. 2024;193(13):104951. doi:10.1016/j.jpdc.2024.104951.
21. Mutambik I. AI-driven cybersecurity in IoT: adaptive malware detection and lightweight encryption via TRIM-SEC framework. Sensors. 2025;25(22):7072.
22. Zamanzadeh Darban Z, Webb GI, Pan S, Aggarwal C, Salehi M. Deep learning for time series anomaly detection: a survey. ACM Comput Surv. 2024;57(1):1–42. doi:10.1145/3691338.
23. Lee Y, Park C, Kim N, Ahn J, Jeong J. LSTM-autoencoder based anomaly detection using vibration data of wind turbines. Sensors. 2024;24(9):2833. doi:10.3390/s24092833.
24. Mitropoulou K, Kokkinos P, Soumplis P, Varvarigos E. Anomaly detection in cloud computing using knowledge graph embedding and machine learning mechanisms. J Grid Comput. 2024;22(1):6. doi:10.1007/s10723-023-09727-1.
25. Khan ZA, Shin D, Bianculli D, Briand LC. Impact of log parsing on deep learning-based anomaly detection. Empir Softw Eng. 2024;29(6):139. doi:10.1007/s10664-024-10533-w.
26. Aziz A, Munir K. Anomaly detection in logs using deep learning. IEEE Access. 2024;12:176124–35.
27. Alam K, Kifayat K, Sampedro GA, Karovič V, Naeem T. SXAD: shapely eXplainable AI-based anomaly detection using log data. IEEE Access. 2024;12:95659–72. doi:10.1109/access.2024.3425472.

28. Lachekhab F, Benzaoui M, Tadjer SA, Bensmaine A, Hamma H. LSTM-autoencoder deep learning model for anomaly detection in electric motor. Energies. 2024;17(10):2340. doi:10.3390/en17102340.
29. Wang Q, Zhang X, Wang X, Cao Z. Log sequence anomaly detection method based on contrastive adversarial training and dual feature extraction. Entropy. 2022;24(1):69. doi:10.3390/e24010069.
30. Ma X, Li Y, Keung J, Yu X, Zou H, Yang Z, et al. Practitioners' expectations on log anomaly detection. IEEE Trans Softw Eng. 2025;51:2455–71.
31. Lisa FT. Intrusion detection using machine learning and deep learning [master's thesis]. Camden, NJ, USA: Rutgers, The State University of New Jersey-Camden; 2025.
32. Dash N, Chakravarty S, Rath AK, Giri NC, AboRas KM, Gowtham N. An optimized LSTM-based deep learning model for anomaly network intrusion detection. Sci Rep. 2025;15(1):1554. doi:10.1038/s41598-025-85248-z.