Reservoir Computingを用いたログパターンの変化に適応可能なログベース異常検知システム

佐々木 康太[1,a)]　掛井 将平[1,b)]　白石 善明[2]　齋藤 彰一[1]

概要：システムに対する攻撃やバグなどによる異常の被害防止において，機械学習によるリアルタイム異常検知が有用である．システムの異常検知には，詳細な内部動作を記録するシステムログの活用が望ましいが，ソフトウェア開発におけるプログラムの継続的な更新が，ログパターンの変化に伴うモデル再学習のコストを高める．そこで本研究では，ログパターンの変化に伴うモデル更新における再学習コストの低減を目指して，Reservoir Computing の計算効率性を活かしたログパターンの変化に適応可能なログベース異常検知システムを提案する．多クラス識別手法の One vs Rest 方式をログテンプレートごとの個別学習に適用することで，未学習のログの出現やログの生起パターンの変化に関連する識別器のみの部分再学習を実現する．評価では，検知精度，計算資源使用量，実行時間を指標に，既存手法との比較を行った．また，時間経過によるログパターンの変化に伴う性能変化および学習効率を分析した結果，すべての識別器を学習することなく，性能が回復することを確認した．

キーワード：異常検知，機械学習，Echo State Network，Reservoir Computing，One vs Rest

Adaptive Log-based Anomaly Detection with Reservoir Computing for Log Pattern Changes

Kota Sasaki[1,a)]　Shohei Kakei[1,b)]　Yoshiaki Shiraishi[2]　Shoichi Saito[1]

Abstract: Real-time anomaly detection using machine learning can prevent damage caused by system attacks or bugs. For accurate anomaly detection, utilizing system logs that record detailed internal operations is desirable. However, the continuous updates in software development often lead to changes in log patterns, which increase the cost of retraining anomaly detection models. To address this issue, this study proposes a log-based anomaly detection system that adapts to changes in log patterns while reducing retraining costs by leveraging the computational efficiency of Reservoir Computing. The system applies the One-vs-Rest multi-class classification method to enable individualized training for each log template, allowing for partial retraining of only the relevant classifiers when new log messages or log pattern changes occur. We evaluate detection accuracy, resource usage, and execution time. Furthermore, our analysis of performance variation under evolving log patterns and learning efficiency revealed that performance can be restored without retraining all classifiers.

Keywords: anomaly detection, machine learning, Echo State Network, Reservoir Computing, One vs Rest

1. はじめに

システム開発において，脆弱性のないシステムを作ることは，システムの複雑化，未知の攻撃手法，開発者の限界，時間・コストの制約など複数の要因により容易ではない．

[1] 名古屋工業大学, Nagoya Institute of Technology
Gokiso-cho, Showa-ku, Nagoya, Aichi 466-8555, Japan
[2] 神戸大学, Kobe University
1-1 Rokkodai-cho, Nada-ku, Kobe 657-8501, Japan
[a)] k.sasaki.261@nitech.jp
[b)] kakei.shohei@nitech.ac.jp

そうした脆弱性は，適切なシステムメンテナンスや状態監視を怠ると見逃され，攻撃やバグによるシステム障害による金銭的・社会的な損失をもたらす．被害の拡大防止において，システムの異常動作を検知し運用者に通知する異常検知システムの活用が有効である．

ログには，システムの動作履歴が記録されるため，ログ解析が異常の発見に繋がる．しかし，システムログは膨大であることが多く，手作業や単純な解析では効果的に異常を検出できるとは言い難い．そこで，深層学習を利用したログベースの異常検知システムが注目されている．

Lehman は，ソフトウェアは有用性を維持するために継続的な更新が必要であり，更新がなければ次第に使われなくなると主張している [1]．そのため，システム開発者によるシステム更新は避けられないが，更新によりログの出力パターンが変化する可能性がある．Zhang らによる調査 [2] でも，Microsoft のオンラインサービスにおいて一か月間の間隔を空けて二日間のログデータを収集したところ，新しいログシーケンスがデータセット全体の 90%を超えることが分かっている．ログパターンが変化すると，利用している異常検知システムの学習済みモデルは正しく特徴を捉えられず検知精度が低下する．そのため，ログパターンの変化に合わせて，モデルを再学習することが望ましい．

ログベースの異常検知手法は, 学習データにより三種類に分類される．教師あり学習手法では, RobustLog[2] や LightLog[3] のように，正常ログと異常ログの両方で学習を行う．半教師あり学習手法では，DeepLog[4] や LogAnomaly[5] のように，正常ログのみを利用して学習を行う．さらに，正常や異常のラベルがついていないデータを利用する教師なし学習手法 [6] もある．教師あり学習手法は情報量が多いため異常検知の有効性が高く，教師なし学習手法はデータのラベル付けが不要であるためデータ取得が容易である．半教師あり学習手法はデータ取得の容易性と，検知の有効性のバランスを取った手法である．異常ログデータの取得には事前のログ解析が必要であることや，すでに異常が発生している必要があり，ログデータの中に異常が発生する確率は低いことなどの障壁が存在する．自作のシステムを開発した際，異常ログのデータを十分な数確保することは困難であり，異常ログデータを確保できるのであれば，開発時にその異常を修正すべきである．一方，正常ログデータはシステムのテスト時に取得すればよいため，比較的データ取得が容易である．本研究では，実運用を考慮し，正常ログのみで学習する半教師あり学習手法を採用する．

本研究では，ログパターンの変化によるモデル更新の再学習コストの低減を目指して，One vs Rest（OvR）方式を用いた，未知のログやログの出力順の変更による影響範囲のみを再学習する異常検知システムを提案する．異常検知システムによる被害拡大の防止には，異常発生から短い時間で異常を検知できるリアルタイム性が重要であるが，OvR 方式はクラスの数だけ識別器が必要であり，その数に比例して計算量が増大する．そこで，軽量な機械学習モデル Reservoir Computing の一種である Echo State Network（ESN）を利用し，学習対象の識別器を選定することで，計算コストを抑えている．

再学習対象の識別器は実運用で得られる誤検知の情報から特定する．ソフトウェアの更新情報に合わせてモデルの更新を行うことも考えられるが，複雑なシステムでは更新情報から変更のあったログとそのログに関連するログを見つけることが難しい．また，テストケースではカバレッジの問題があり，本番環境ではテスト時に予期していない正常動作が発生することがあるため，実運用を通してモデルを調整していくことが望ましい．

2. 関連研究

2.1 機械学習を用いたログベースの異常検知

DeepLog[4] は，Long-Short Term Memory（LSTM）に基づいて，入力されたシーケンスから次のログを予測する．入力に one-hot encoding を使用しており，学習データに存在しない未知のログは処理できず，誤検知やエラーの原因となっている．また，多くの実装 [7] では，出力層にソフトマックスを採用しているため，未知のログの対応には，新しいニューロンの追加と，入力層の形状変更が必要である．新しいニューロンのみの増分更新による再学習では，既存の確率分布が再分配による既存クラスの性能劣化が避けられず，未知のログを含むデータの再収集とモデル全体の再学習が必要となる．

LogAnomaly[5] は，Word2Vec によるログの特徴ベクトルやログの生起頻度を利用して，性能改善を示している．特徴ベクトルにより，未知ログを類似ログにマッチングさせることでログの変化に対応するが，新機能の実装などで新規ログが追加された場合，異なるログテンプレートが同一のものと判定され，検知性能が低下する．

RobustLog[2] は，ログ文の意味ベクトルとアテンション付き双方向 LSTM により，ログ変化に対する頑健性を持ち，再学習頻度の削減を目指している．正常と異常の二値分類を行う教師あり学習のため，出力層のクラス数は増減しないが，処理フローの大幅な変更や新しい処理フロー追加時には増分更新による再学習が必要となる．

2.2 DeepLog

半教師あり学習手法の一つである DeepLog による，ログシーケンスの異常検知モデルを説明する．DeepLog ではログをテンプレートと時刻情報を含んだパラメータに分割する．例えば `logging.info(f"{user} logged in from IP {address}.")` のようなログは，`{user}` や `{address}` のような変数部分（パラメータ）を抽象化した，`"* logged in from IP *"`のような文字列がログテンプレートとな

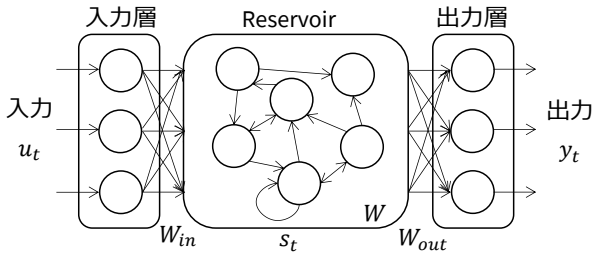

図 1 Echo State Network の基本構成 （[10] を参考に作成）
Fig. 1 Basic Structure of ESN (Based on [10])

る．この分割は，Spell[8] によって行われる．ログシーケンスの異常検知ではこのテンプレートを利用する．

ログテンプレートごとにログキーと呼ばれる ID を振り分ける．LSTM を用いて時系列データとして正常なログパターンを学習し，実行時のログパターンから次のログを予測する．予測と実際のログが一致しない場合，異常遷移と判定し，セッション全体を異常セッションと判定する．本研究でもログキーの遷移情報をもとに次のログを予測することで異常検知を行う．

2.3 Echo State Network

Reservoir Computing は入力層，Reservoir，出力層からなり，Reservoir によって時系列データを時空間パターンに変換し，出力層の学習アルゴリズムにより時系列データを学習する手法である [9].

ESN の基本的なモデル構造を図 1 に示す．一般的な Recurrent Neural Network の構造を利用しており，入力層と Reservoir の結合重みをランダムな初期値で固定する．時刻 t における Reservoir の出力 s_t は以下のように計算される．

$$s_t = (1-\alpha)\cdot s_{t-1} + \alpha \cdot \tanh(W_{in}u_t + W s_{t-1}) \quad (1)$$

α は漏れ率と呼ばれ，state の更新スピードを意味する．また，u_t は入力値であり，W_{in}, W は入力層および Reservoir の結合重みである．

3. 研究のアプローチ

DeepLog は LSTM の出力層でソフトマックス関数を利用して，次に発生するログキーごとの確率分布を推定している．そのため，ログパターンの変化により検知精度が低下し，特に過検知が増加する．プログラムの更新の度に再学習を行えばよいが，単一のモデルですべてのログキーを扱っているため全体の学習が必要となる．

そこで本研究では，OvR 方式を採用する．OvR 方式は多クラス分類において各クラスごとに識別器を用意し，それぞれの識別器を個別に学習させることで，各識別器は，そのクラスに当てはまるか（One），当てはまらないか（Rest）の二値問題を推論する手法である．性能劣化した識別器だけを再学習すればよいため，ログの削除・追加や出現パターンの変化が生じても必要最小限の再学習で対応できる．

OvR 方式は，クラス数の識別器が必要となるため，ログテンプレートの増加に比例して計算量や計算資源の消費量が増加する．そこで，本研究では軽量な機械学習モデル Reservoir Computing の一種である ESN を採用する．ESN は，RNN の出力層の結合重みのみを学習し，Reservoir 層の重みを固定しているため，各識別器で独立して Reservoir 層を持つ必要がなく，OvR 方式を効率的に適用できる．これにより，ログパターンの変化への柔軟な対応と計算コストの抑制を実現する．

4. 提案システム

4.1 システムの概要

提案システムの概要を図 2 に示す．本システムは学習フェーズ，検知フェーズ，再学習フェーズから構成される．学習フェーズでは，テスト動作などによって取得した初期の学習データをもとに異常検知モデルを学習する．検知フェーズでは，異常検知モデルが実際にシステムが出力したログから各セッションが正常か異常かを判定する．異常と判定された場合，そのセッション情報を異常セッションデータに加え，正常の場合は正常セッションデータに加える．再学習フェーズでは，システム運用者が異常セッションデータから誤検知データを抽出し，誤検知セッションデータに保存する．一定期間再学習データが蓄積されたり，システム運用者が再学習を指示した場合，直近の誤検知セッションデータと直近の正常セッションデータ，初期の学習データから再学習用のデータを作成する．誤検知セッションデータから再学習対象となるクラスの識別器を特定し，再学習用データで識別器の追加と再学習を行ったうえで，再度検知フェーズへ戻る．

DeepLog を参考に正常セッションのログキーシーケンスを用い，ESN と OvR 手法で学習・検知を行う．出力層にシグモイド関数による線形識別器を採用し，セッション終了のクラス 0 および各ログキーごとの識別器を用意する．未知ログキーに対応するため，one-hot encoding ではなくログキーの値そのものを入力とする．

4.2 学習フェーズ

本モデルはログシーケンスから次に生成されるログを予測するタスクであり，入力ログキー k_t により得られたリザバー状態 s_t を用いて次のログキー k_{t+1} を予測する．あるセッションにおける n 個のログキー系列を $K = \{k_1, k_2, \cdots, k_n\}$ とする．初期状態 s_0 をゼロベクトルとして系列 K を式 (1) の状態遷移式に基づき逐次処理することで，状態ベクトル系列 $S = \{s_1, s_2, \cdots, s_n\}$ を得る．

次に，状態ベクトル系列 S の各要素に対応する教師ラベルを生成する．時刻 t（$1 \leq t < n$）の状態 s_t に対するクラ

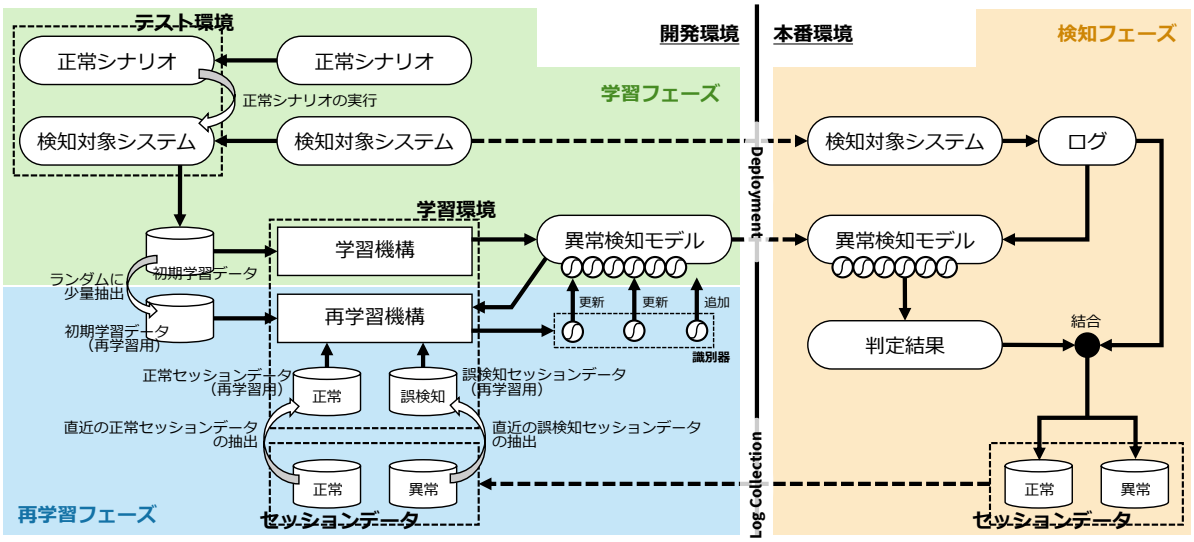

図 2　提案システムの概要図
Fig. 2　Overview of proposal

ス c の教師ラベル $y_{t,c}$ を，次時刻のログキー k_{t+1} を用いて $\delta(k_{t+1}, c)$ と定義する．ここで $\delta(\cdot,\cdot)$ はクロネッカーのデルタであり，二つの引数が等しければ 1，等しくなければ 0 を示す．系列の最終状態である s_n には次のログが存在しないため，セッション終了のクラス $c = 0$ のラベルを 1 とし，そのほかのクラスは 0 とする．つまり，式 (2) のように定義する．

$$y_{t,c} = \begin{cases} \delta(k_{t+1}, c) & (1 \le t < n) \\ \delta(0, c) & (t = n) \end{cases} \tag{2}$$

各クラス c の識別器は，リザバー状態 s_t から式 (3) に従ってロジット $o_{t,c}$ を算出する．ここで，$W_{out,c}$ および b_t はクラス c の識別器の学習対象パラメータである．

$$o_{t,c} = W_{out,c} s_t + b_c \tag{3}$$

ログデータはクラスごとの生起頻度が著しく不均衡であるため，損失関数には重み付きバイナリクロスエントロピー損失を用いる．クラス c に対する損失 L_c は式 (4) で表される．

$$L_c = -\frac{1}{N}\sum_{i=1}^{N}[w_c \cdot y_{i,c} \cdot \log(\sigma(o_{i,c})) + (1 - y_{i,c}) \cdot \log(1 - \sigma(o_{i,c}))] \tag{4}$$

N はセッションのログ数であり，$\sigma(\cdot)$ はシグモイド関数である．w_c はクラス c に対する重みであり，クラス c に属するサンプル数を N_c としたとき $w_c = (N - N_c)/N_c$ で計算される．これにより，出現頻度の低い少数派クラスほど大きな重みを与え，損失計算における少数派クラスの寄与度を増大させる．これは，モデルが多数派クラスに過度に適合することを防ぐ効果を持つ．学習プロセスでは，この損失 L_c を最小化するように，各識別器のパラメータを勾配降下法に基づき最適化する．

4.3　検知フェーズ

検知フェーズでは，未知のログキー系列 K' が与えられた際，学習時と同様に，状態ベクトル系列 S' を生成する．次に，各時刻 t の状態 s'_t を各識別器に入力することで，ロジット $o_{t,c}$ を出力する．これにシグモイド関数 $\sigma(\cdot)$ を適用することで，次時刻にクラス c のイベントが発生する予測確率 $p_{t,c}$ を算出する．

異常の判定は，モデルの予測と実際の観測結果を比較することによって行われる．時刻 t において，モデルが算出した実際の次のログキー k'_{t+1} の生起確率 $p_{t,k'_{t+1}}$ が、事前に設定された閾値 $\theta (0 < \theta < 1)$ を下回った場合、その遷移は異常と判定される。また，観測されたログキー k'_{t+1} が，学習データに一度も出現しなかった未知のクラスである場合，その生起確率を 0 とみなし，未知のクラスであることを記録したうえで異常と判定する．最終的に，セッション全体を通してこれらの異常が一つでも検知された場合，そのセッションは異常セッションとして判定される．

4.4　再学習フェーズ

ログパターンが変化することによる誤検知の軽減のため，再学習が必要となる．再学習に用いるデータは，初期の学習データからランダムに少量抽出したものに加え，直近の正常と判定されたセッションデータおよび誤検知であったセッションデータを組み合わせる．過去の学習結果の破壊を防ぐため，既存の識別器の再学習では学習率やエポック数を小さく設定し，正則化項の係数を大きくすることで，過去の情報を保持しつつ局所的な微調整を行う．ただし，新規に追加した識別器は，学習フェーズと同様の学習率・エポック数で学習を行う．

4.4.1　未知のログが登場するケース

検知フェーズでは，ログ文の変更や，新規機能によるロ

グの追加など，学習時に現れなかった未知のログが登場するケースがある．RobustLog は意味ベクトルやアテンション付き双方向 LSTM を用いることでこれに対応していた．

本研究では，未知のログを新規クラスとして扱い，ログテンプレートに ID を付与して出力層に識別器を追加する．未知のログが登場すると，対応する識別器が存在しないため異常検知モデルは異常と判定する．このとき，異常検知モデルは通常の異常判定ではなく，識別器が存在しないことによる異常判定であることも合わせて，異常セッションデータを保存する．保存する異常セッションデータには，異常セッションのログキー系列，異常遷移発生地点，未知のログ発生地点が含まれる．

システムの運用者が抽出した誤検知セッションデータから，再学習対象となる識別器を特定し，再学習を行う．この際，未知のログに対応する識別器も追加する．ESN では状態が過去入力に依存して決定されるため，再学習対象となる識別器は異常判定された遷移を含む後方のログキーとなる．例えば，ログキー系列 K の k_{n-2} で誤検知が発生した場合，k_{n-2}, k_{n-1}, k_n が再学習対象の識別器の ID となる．

4.4.2 ログ出力順が変化するケース

ソフトウェア開発においてログの生起順序が変更される場合も存在する．例えば，システムアルゴリズムの変更により，ログ実行順序が入れ替わる場合である．出力順の変化により，学習データとは異なるパターンが入力されると，モデルは正常セッションを誤って異常と判定する．この場合も，未知ログと同様に，誤判定された遷移以降のログキーの識別器を再学習対象として再学習を実施する．

4.5 ウィンドウサイズによる制約の排除

DeepLog や LogAnomaly に代表される半教師あり学習手法による既存研究の多くは，時間的依存性をモデル化するためにスライディングウィンドウ方式に準拠する．この方式では，連続するログ系列を固定長のサブ系列に分割し，入力系列と正解ラベルのペアを生成して学習する．ウィンドウサイズを h とし，時刻 t における過去 h 個のログキー系列 $sub(K,t) = \{k_{t-h+1}, \cdots, k_t\}$ から，次時刻のログキー k_{t+1} を予測する条件付き確率 $p(k_{t+1}|k_{t-h+1}, \ldots, k_t)$ を最適化する．

このアプローチは強力である一方，ウィンドウという境界を設定することによる制約がある．まず，最適なウィンドウサイズ h の選定が不可欠であり，これはドメイン知識や試行錯誤を要するハイパーパラメータとなる．h の値が不適切であれば文脈情報の不足やノイズの混入を招き，モデルの汎化性能を損なう．さらに，モデルが参照可能な履歴はウィンドウ内に限定されるため，それを超える長期的なイベント間の因果関係や複雑な依存性は原理的にとらえることができない．また，この制約により，セッション長が h に満たないデータを学習や検知段階での分析対象から除外するか，あるいは系列の先頭または末尾をゼロなどの特殊な値で埋めて長さを h に揃えるパディング処理を施す必要がある．しかし，パディングはモデルに人工的な情報を入力することとなり，予測の信頼性を損なう可能性がある．最後に，ウィンドウごとに中間層の出力を計算するため，検知時の計算量が多くなるという問題もある．

本研究で採用する ESN は過去の入力情報がリザバーの内部状態に保持され，時間経過に伴いその影響が指数関数的に減衰していくエコーステート性を持つ．この記憶の減衰率は式 (1) における，重み行列 W のスペクトル半径によって制御される．この特性により，ESN はセッション開始からの履歴情報を状態ベクトル s_t に自動的に蓄積するため，従来のスライディングウィンドウ方式のようなウィンドウサイズの設定が不要になる．これにより，ハイパーパラメータ選定の難しさから解放され、固定ウィンドウでは捉えられなかった長期的な依存関係もモデル化できる可能性がある．本手法にもスペクトル半径という調整すべきハイパーパラメータは存在するが，ウィンドウサイズがデータの物理的な区切りを強制するのに対し，スペクトル半径はモデルの内部的な記憶減衰速度を制御する柔軟なパラメータである．

ウィンドウサイズを設定する手法では，計算過程で同一の遷移情報が繰り返し現れるため，学習効率の向上が期待できる．一方，ウィンドウサイズを設定しない場合，各ログキー遷移の特徴を個別に計算するため、学習効率が低下する可能性がある．しかし，本研究では学習効率の高い ESN を採用している．さらに，学習対象となる識別器を選別することで，再学習対象のパラメータ数を抑えている．このため，ウィンドウサイズを排除しても学習効率の低下の抑制が期待できる．また，ウィンドウサイズを設定しないことで，各ログキー遷移の特徴の計算量が少なくなるため，検知効率の向上も期待できる．

また，セッション情報が記録されるためには少なくとも 1 つのログが記録されている必要があり，セッション終了を示すクラス 0 を追加したため，1 セッションに含まれるログキーの系列の長さは，2 以上となる．そのため，この最小系列長のセッションでも有効なデータとして扱うことができる．

5. 評価

本章では，提案した ESN と OvR 方式を組み合わせた異常検知システムの評価を行う．ログデータからログキーへの変換に必要なテンプレートの抽出は本研究の対象外であるため詳細の実装は省略し，将来的に追加されるログテンプレートも含めて全て既知であり，単純なマッチングで変換できると仮定して実装・評価を行う．実際に抽出する場合，DeepLog で利用されている Spell や，RobustLog で利

表 1　HDFS データセットのサンプリング
Table 1　Sampling of HDFS Dataset

Label	Total	Train	Test
Normal	558,221	4,855	553,366
Abnormal	16,838	145	16,693

用されている Drain[11] のようなものを利用すればよい．

5.1　既存手法との比較

DeepLog，RobustLog，LightLog，そして提案手法の 4 つの異常検知システムを比較評価する．また、DeepLog に OvR 方式を適用したシステムも評価対象とする．評価には、Hadoop 分散ファイルシステムのログである HDFS データセット [12], [13] を使用する．このデータセットは，558,221 の正常セッションと 16,838 の異常セッション，合計 575,059 セッションで構成されている．学習データは，これらのセッションから比率を保ったままランダムにサンプリングして作成する．サンプリングしたデータの詳細を表 1 にまとめる．

本研究におけるモデルの学習段階では, NVIDIA GeForce RTX 4060 Ti（VRAM 8GB）GPU を利用した．学習済みのモデルを用いた検知では，Intel Core i9-10900X CPU（メモリ 256GB）のみ利用した．実運用を想定した評価のため，検知は複数セッションをまとめて検知するバッチ処理を行わず，1 セッションずつ行う．

5.1.1　DeepLog

DeepLog は 2 章で述べた半教師あり学習手法によって，次のログを予測する異常検知システムである．one-hot encoding によってエンコードされた入力から，次のクラスの生起確率をソフトマックス関数によって推測し，確率上位のログキーを正常遷移の候補とする．本実験では，LSTM 層を 2 層，隠れ状態の次元を 64 とし，学習率 0.001，エポック数 50 で，最適化アルゴリズムに Adam を利用し学習を行った．また，ウィンドウサイズ 10 とし，確率上位 9 個のログキーを正常とした．OvR 手法を取り入れたシステムでは，提案システムと同様に中間層を共有した．学習時に登場したクラスの識別器を用意して学習し，生起確率が 0.02 を超えたクラスを正常と判定する．また，OvR 手法のときデータ全体で損失計算をすると精度が安定しなかったため，バッチサイズ 2048 のミニバッチ学習を行った．

5.1.2　RobustLog

RobustLog は，正常とラベル付けされたログと異常とラベル付けされたログから学習を行い，セッション全体の異常・正常を判定する教師あり学習手法の異常検知システムである．テンプレートに含まれる単語の意味ベクトルを利用し，テンプレートの意味ベクトルを作成し，異常検知システムに意味ベクトルの系列を入力する．本実験では単語の意味ベクトルを Common Crawl と Wikipedia を用いて fastText により学習したモデル [14] から単語ベクトルを取得し，TF-IDF によって 300 次元のログテンプレートの意味ベクトルを作成した．意味ベクトルの作成は Le らの研究 [7] で利用されていたコードを利用した．異常セッションの割合は偏っているため，損失関数の計算では異常シーケンスの頻度をもとに重みづけを行った．本実験では，LSTM 層を 2 層とし，隠れ状態の次元は 64 としたが，双方向であるため次元数は合計 128 となる．入力した意味ベクトルの系列からセッションが正常である確率を推定し，正常の確率が閾値 0.2 を下回った場合，そのセッションを異常と判定する．学習率は 0.001 とし，エポック数 50 で，最適化アルゴリズムに Adam を利用し学習を行った．

5.1.3　LightLog

LightLog は IoT 製品の異常検知を行うため，軽量な異常検知システムを構築することを目指した研究である．RobustLog と同様にテンプレートの意味ベクトルを入力とするが，高次元性を課題とし，PCA と後処理による次元圧縮を適用している．論文中では Word2Vec を利用しているが，本実験では RobustLog と同様の手順で，fastText による単語ベクトルの抽出による 300 次元の意味ベクトルを作成しており，次元圧縮により 20 次元まで圧縮している．異常検知モデルには軽量な時系列畳み込みネットワークを利用しており，残差ブロック数を 4 つとして，拡張率を $1, 2, 4, 8$ にしている．また，残差ブロックのカーネルサイズは 3 である．学習率は 0.001 とし，エポック数 500 で，最適化アルゴリズムに Adam を利用し学習を行った．データ全体で損失計算をすると精度が安定しなかったため，バッチサイズ 2048 のミニバッチ学習を行った．LightLog では閾値を 0.5 とし，正常である確率が閾値を下回った場合，そのセッションを異常と判定する．

5.1.4　提案手法

学習・検知の手順は 4 章の通りである．また，Reservoir の次元数は 500 とし，スペクトル半径を 0.99，漏れ率を 0.4 とした．学習率を 0.001，正則化項の係数を 10^{-5} とし，エポック数 1000 で，最適化アルゴリズムに Adam を利用し学習を行った．閾値は異常セッションの生起確率である約 $\frac{16838}{575059} \simeq 0.029$ とする．

5.1.5　検知精度

検知精度の評価指標として Accuracy, Precision, Recall, F1 スコアを利用する．異常を Positive，正常を Negative として各評価指標の計算を行う．

各異常検知システムの検知性能を表 2 に示す．提案システムは，他の異常検知システムに近い検知精度を記録しており，十分な検知性能を持っていると考えられる．特に，異常の見逃しの指標となる Recall が高く，異常を見逃しづらい検知システムだと言える．また，過検知の指標である Precision は他の手法に比べて低くなっており，改善の余地が残っている．

表 2　検知性能
Table 2　detecting performance

	Accuracy	Recall	Precision	F1
DeepLog	0.9954	0.9917	0.8705	0.9272
DeepLog（OvR）	0.9954	0.9917	0.8705	0.9272
RobustLog	0.9868	0.9033	0.9487	0.9445
LightLog	0.9960	0.9942	0.8839	0.9359
Proposal	0.9945	0.9995	0.8425	0.9143

表 3　実行時間
Table 3　Running Time

	Training Time		Detecting Time
	seconds	s/epoch	ms/session
DeepLog	74.086	1.482	4.935
DeepLog（OvR）	37.876	0.758	6.918
RobustLog	204.094	4.082	2.317
LightLog	184.164	0.368	1.011
Proposal	34.165	0.034	1.750

5.1.6　実行時間

各異常検知システムが学習および検知にかかった時間を計測した．ただし，データの読み込みにかかった時間は除外している．計測結果を表 3 に示す．

エポック数やバッチサイズが統一されていないため単純な比較はできないが，エポック数やバッチサイズが他のものに比べて大きな値となっている提案手法が学習時間が最も小さくなっている．また，1 エポック当たりの学習時間が特に小さくなっており，ウィンドウサイズを排除したことによる学習効率の低下を抑えることができた．また，検知時間は LightLog に次いで 2 番目に小さい値を記録しており，リアルタイム性に優れた異常検知システムであると言える．

5.1.7　計算資源使用量

各異常検知システムが学習および検知で利用した計算資源の使用量を計測した．学習フェーズでは GPU 使用率，GPU メモリ使用量，CPU 使用率，メモリ使用量を計測し，検知フェーズでは CPU 使用率，メモリ使用量を計測した．ただし，GPU の使用率はプロセスごとの計測が不可能なため，GPU を利用している他のプロセスの使用量が 0 に近似できることを確認したうえで，システム全体の GPU 使用率を計測した．その結果を表 4 に示す．

本研究で提案する手法は，既存手法と比較して，計算資源の利用効率において課題が残る．特に，検知時の CPU 利用率が高く，効率化の余地がある．この非効率性は，中間層である Reservoir の次元が他の手法に比べて大きいことに起因すると考えられる．より詳細なログイベントの情報を捉えるために次元数を 500 という高い値に設定したが，これが推論時の計算負荷を増大させていると考える．

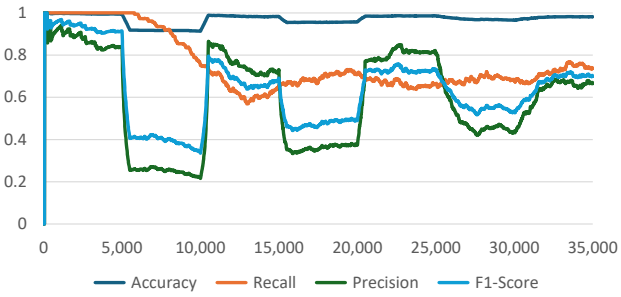

図 3　検知性能の遷移
Fig. 3　Transition of detection performance

5.2　ログパターン変化対応の評価

ログパターンの変化による学習効率を評価するため，疑似的にログパターンを 3 段階で変化させ，検知と学習を行った．1 回目は，特定のログキーを新しいログキーに置換し，ログ文の変更を再現した．2 回目は，特定のログパターンの直後に新しいログシーケンスを追加し，新しいログ群の追加を再現した．3 回目は特定の 2 つのログパターンの順序を丸ごと入れ替え，ログの生成順序の変化を再現した．5,000 セッションを基準に，10,000 セッションごとに変更を行い，合計 35,000 セッションで実験した．また，異常セッションの割合は常に約 2.9%とし，ランダムに配置する．10 セッションごとに直近 5,000 セッションの検知性能をプロットする．また，再学習は 500 セッションごとに，エポック数 100，学習率 0.001，正則化項の係数を 10^{-4} として行った．

実験の結果を図 3 に示す．ログパターンの変化直後には検知性能が著しく低下したが，再学習による回復が確認された．しかし，Recall の低下は十分に回復せず，再学習時に異常を見逃したセッションを正常として扱ってしまったため，異常セッションが正常と誤って学習されたことが原因だと考えられる．また，ログの生成順序を変更した場合，元の水準まで性能が回復しなかった．これは，再学習時のエポック数や学習率を小さく設定したことや，ログパターンの変化を繰り返すうちに，再学習で利用している初期学習データが実際のログと大きく乖離し，新しいパターンへの順応が不十分となったためだと推測される．

再学習で利用された識別器の数を図 4 に示す．学習対象の識別器の数は最大でも 16 であり，識別器が増加しても対象の数が大きく変わることはなかった．また，検知性能もある程度回復できていることを踏まえると，識別器の選定により再学習の効率を向上できたと考えられる．

6.　まとめ・今後の展望

本研究では半教師あり学習による異常検知モデルで ESN と OvR 方式を組み合わせることにより，ログパターンの変化に伴うモデル更新における再学習コストの低減を実現

表 4　検知時の計算資源使用量（平均/最大）
Table 4　Resource usage during Detecting (average/max)

	学習時				検知時	
	CPU (%)	mem (MB)	GPU (%)	GPU mem (MiB)	CPU (%)	mem (MB)
DeepLog	119/209	1312.29 / 1448.88	7 / 100	6190 / 6968	987/998	776.53 / 777.42
DeepLog（OvR）	120/212	1133.22 / 1126.13	7 / 16	405 / 470	992/999	776.08 / 776.44
RobustLog	120/210	2220.54 / 2399.02	5 / 100	3414 / 3554	660/889	41075.44 / 43870.75
LightLog	167/211	1376.68 / 1410.73	2 / 4	202 / 208	796/947	5340.47 / 5521.96
Proposal	108/212	963.35 / 1197.72	67 / 99	449 / 564	956/990	1959.62 / 2022.55

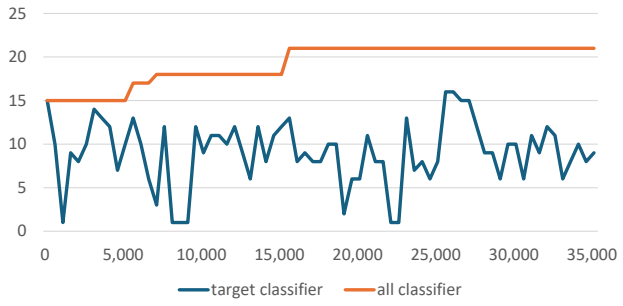

図 4　再学習対象の識別器
Fig. 4　Classifier for retraining

した．しかし，現在の異常検知システムではログの生起パターンによる異常のみ検知可能であるため，今後はパラメータを用いた異常検知も検討していく．また，RobustLog や LogAnomaly，LightLog といった異常検知システムのように入力をテンプレートの特徴を捉えたベクトルとすることで，検知性能や頑健性の向上を目指す．また，計算資源の使用量に課題が残るため，その改善も目指していく．

本研究で利用したデータセットは HDFS の 1 種類のみであった．また，比較に利用した異常検知システムも限定的である．今後は，他のデータセットを利用するだけでなく，実際にログの変更が発生したログデータによる評価も検討し，学習効率の向上が実世界でどの程度影響を与えるのか計測していきたい．

謝辞　本研究の一部は，JSPS 科研費 JP25K21199 の助成を受けたものです．

参考文献

[1] Lehman, M. M.: Programs, life cycles, and laws of software evolution, *Proceedings of the IEEE*, Vol. 68, No. 9, pp. 1060–1076 (2005).
[2] Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z. et al.: Robust log-based anomaly detection on unstable log data, *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pp. 807–817 (2019).
[3] Wang, Z., Tian, J., Fang, H., Chen, L. and Qin, J.: LightLog: A lightweight temporal convolutional network for log anomaly detection on the edge, *Computer Networks*, Vol. 203, p. 108616 (2022).
[4] Du, M., Li, F., Zheng, G. and Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning, *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 1285–1298 (2017).
[5] Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P. et al.: Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs., *IJCAI*, Vol. 19, No. 7, pp. 4739–4745 (2019).
[6] Farzad, A. and Gulliver, T. A.: Unsupervised log message anomaly detection, *ICT Express*, Vol. 6, No. 3, pp. 229–237 (2020).
[7] Le, V.-H. and Zhang, H.: Log-based Anomaly Detection with Deep Learning: How Far Are We?, *2022 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)* (2022).
[8] Du, M. and Li, F.: Spell: Streaming parsing of system event logs, *2016 IEEE 16th International Conference on Data Mining (ICDM)*, IEEE, pp. 859–864 (2016).
[9] Jaeger, H.: The “echo state” approach to analysing and training recurrent neural networks-with an erratum note, *Bonn, Germany: German national research center for information technology gmd technical report*, Vol. 148, No. 34, p. 13 (2001).
[10] 田中剛平. 中根了昌. 廣瀬明：リザバーコンピューティング 時系列パターン認識のための高速機械学習の理論とハードウェア，森北出版，東京都千代田区富士見 1-4-11 (2021).
[11] He, P., Zhu, J., Zheng, Z. and Lyu, M. R.: Drain: An online log parsing approach with fixed depth tree, *2017 IEEE international conference on web services (ICWS)*, IEEE, pp. 33–40 (2017).
[12] Xu, W., Huang, L., Fox, A., Patterson, D. and Jordan, M. I.: Detecting large-scale system problems by mining console logs, *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 117–132 (2009).
[13] Zhu, J., He, S., He, P., Liu, J. and Lyu, M. R.: Loghub: A large collection of system log datasets for ai-driven log analytics, *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, pp. 355–366 (2023).
[14] Grave, E., Bojanowski, P., Gupta, P., Joulin, A. and Mikolov, T.: Learning Word Vectors for 157 Languages, *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)* (2018).