Solidity ソースコード内に潜む脆弱性を大規模言語モデルで検知する手法の提案

安田一覚[1,a)]　三村 守[1]

概要：近年，スマートコントラクトの普及に伴い，Solidity コードに起因する深刻なセキュリティ被害が相次いで報告されており，開発段階における脆弱性の自動検出手法の確立が重要となっている．従来は静的解析や機械学習を用いた検出手法が提案されてきたが，これらは既知のパターンやルールに依存するため，関数間の依存関係やコード全体の文脈を考慮した複雑な脆弱性の検出には限界がある．本研究では，コードの文脈や依存関係を包括的に捉えられる大規模言語モデル（LLM）を用い，Solidity コードに内在する脆弱性を検知する手法を提案した．実験コストを考慮した固定長シーケンスによる学習と，コード全文を入力する学習の分類性能を既存手法と比較評価した．実験の結果，固定長シーケンスで学習したモデルは既存手法に比べ精度が劣ったが，全文を入力してコード全体の文脈を考慮した LLM は精度が向上し，最良の F-measure は 0.879 を記録した．本研究は，LLM を用いた Solidity コードに対する脆弱性検知の有効性と限界を明らかにするものである．

キーワード：大規模言語モデル，自然言語処理技術，スマートコントラクト， Transformer，Solidity

Detecting Vulnerabilities in Solidity Source Code Using Large Language Models

Ikkaku Yasuda[1,a)]　Mamoru Mimura[1]

Abstract: In recent years, with the spread of smart contracts, serious security breaches caused by Solidity code have been reported one after another, making it important to establish automated vulnerability detection methods during the development phase. Previously, detection methods using static analysis and machine learning have been proposed, but because they rely on known patterns and rules, they have limitations in detecting complex vulnerabilities that take into account inter-function dependencies and the overall code context. In this study, we propose a method to detect vulnerabilities inherent in Solidity code using large-scale language models (LLMs), which can comprehensively capture code context and dependencies. We compared the classification performance of training using fixed-length sequences, which takes into account experimental costs, and training using full-text code input, with existing methods. Experimental results showed that the model trained using fixed-length sequences performed less accurately than existing methods, while LLMs, which input the full text and consider the overall code context, improved accuracy, achieving the best F-measure of 0.879. This study clarifies the effectiveness and limitations of vulnerability detection using LLMs in Solidity code.

Keywords: Large Language Model, Transformer, Solidity，Natural Language Processing, Smart Contracts

1. はじめに

近年，イーサリアムをはじめとするスマートコントラクトの普及により，さまざまな分野で分散型アプリケーショ

[1] 防衛大学校研究科
National Defense Academy of Japan
[a)] em64010@nda.ac.jp

ンが実用化されている [11]．スマートコンタクトはブロックチェーン上にソースコードがデプロイされ，そのコードによって自律的に動作するシステムが構築される．しかしながら，開発段階で脆弱性を含んだソースコードがそのままデプロイされると，意図した動作が実現できないだけでなく，悪意ある攻撃者により脆弱性を突かれるリスクが生じる．実際に，スマートコントラクトの脆弱性を利用した攻撃によって，数十億ドル規模の被害が発生していると報告されている [10]．Open Worldwide Application Security Project でも，スマートコントラクトに特有の脆弱性と攻撃手法について「Smart Contract Top 10」として整理，公開しており，その深刻さが広く認識されている [8]．

このような背景から，Solidity コードに内在する脆弱性の自動検出に関する研究が数多く行われてきた [14,16]．特に，機械学習や静的解析技術を用いたアプローチが注目されている [1,2]．Feist らは，静的解析ツールである Slither を提案し，Solidity コードにおける既知の脆弱性パターンを高速かつ高精度に検出する手法を実現した．Slither は定義済みの検出ルールに基づき，構文的，構造的な特徴に基づく解析を行うため，明示的に記述された脆弱性パターンの検出に強みを持つ [3]．また，Large Language Model（LLM）を用いた脆弱性検出においては，プロンプトベースによる手法が近年提案されている．Chen らの研究では，LLM を用いた脆弱性分類タスクにおいて，プロンプトベースのアプローチとファインチューニングの有効性が示されており，LLM がコードの意味的理解に優れていることが確認されている [6]．

既存研究の課題として，Feist らの研究では，既知のパターンやルールに依存するため, 関数間の依存関係やコード全体の文脈を考慮した複雑な脆弱性の検出では限定的である．また，Chen らの研究では LLM でのプロンプトベースアプローチによる分類において最良のモデルの F-measure が 0.776 と性能が高いとは言い難い．

そこで本研究では，Solidity コードの長さに関係なく，すべてのサンプルを学習することの実験コストを考慮し，LLM に Solidity コードを固定長のトークンシーケンスに分割して入力し，学習およびテストを行うことで，既存手法との比較評価を行う．また，Solidity コード全体を一括で LLM に読み込ませて学習し，コード全体の文脈を考慮した脆弱性検出能力を検証し，LLM の特性がどの程度精度向上に寄与するかを明らかにすることを目的とする．

本研究の研究課題 (RQ) は以下のとおりである．

(RQ1)LLM は既存の検知モデルと比較してどの程度の Solidity コード脆弱性を検知できるのか？

(RQ2)LLM による全文コンテキストでの脆弱性検出は分類性能を向上することができるのか？

(RQ3)LLM による Solidity コードの脆弱性検知における実験の限界は何か？

本研究では，これらの研究課題を解決するため，Solidity ソースコードが書かれたデータセットを用いて検知精度を比較した．その結果，提案手法は固定長のトークン数で学習した場合，知精度が F-measure が 0.707 という結果となった．また，コード全体の文脈を考慮した脆弱性検出能力の検証では，最良の F-measure は 0.879 となった．本研究の主な貢献は以下のとおりである．

(1) Solidity コードを固定長のトークン数で学習した場合，提案手法の精度は F-measure が 0.707 という結果となった．

(2) Solidity コードを全文 LLM に読み込ませて学習，テストした手法は，既存の検知モデルと比較して精度が向上することを確認した．

(3) Solidity コードのトークン数増加に伴い，LLM モデルの学習，推論時間が大幅に増加することを確認し，本手法のスケーラビリティに関する限界を示した．

本研究の構成を以下に示す．第２節では，主にスマートコントラクトについての概要及び脆弱性について述べ，第３節では Solidity コードの脆弱性検知に関する関連研究について述べ，第４節では LLM に関する関連技術について紹介する．第５節では提案手法について説明し，第６節では検証実験の手法およびその結果について述べる．第７節では実験結果について考察する．最後に第８節では，本研究のまとめを述べる．

2. スマートコントラクト

スマートコントラクトは，ブロックチェーン上で条件が満たされると自動実行されるプログラムである [11]．契約者と被契約者との間で仲介者を介さず契約を履行でき，信頼性と透明性が高い．主にイーサリアム等の分散型プラットフォームで利用され，金融，ゲーム，NFT などに応用されている．一方，プログラムの欠陥や設計ミスにより重大なセキュリティ問題が生じるため，コードの正当性と安全性の検証が重要である．

2.1 Solidity 言語

Solidity は，イーサリアム向けに開発された高水準言語で，JavaScript や C++に類似した構文を持つ．状態管理やイベント処理などの機能を備える一方，デバッグの難しさや学習コストの高さが課題であり，安全なコーディングと脆弱性対策が求められる．

2.2 脆弱性

スマートコントラクトは自動的に契約を実行する便利な仕組みである一方で，設計ミスや不適切な記述によって深刻な脆弱性を引き起こす可能性がある [9]．特に Solidity では，call() 関数を用いた外部送金処理が典型的な攻撃対象になっている．これは外部コントラクトに Ether を送金す

る際に (bool sent,)=addr.call{value:mount}("") のように記述されるが，この時外部コントラクトが元の関数を呼び出した際，処理が繰り返されされ資金が不正に引き出される恐れがある（rerntrancy 脆弱性）．また，call() の戻り値を確認せずに送金を行うと，送金が失敗しても処理が続行される危険性がある．これらの脆弱性に対して，静的解析ツールの活用，コードの監査などによる多層的な対策が求められる．

3. 関連研究

本節では，機械学習モデルを用いた Solidity コード脆弱性検知に関する研究および LLM を用いた脆弱性検知モデルに関する研究を調査し，本研究との違いを示す．

3.1 機械学習モデルによる脆弱性検出手法

Solidity スマートコントラクトの脆弱性検出においては，従来より多様な機械学習手法が適用されてきた．初期の研究では，コードから抽出した静的特徴量を用いて分類を行うアプローチが主流であった．例えば Chen らは，Solidity コードから命令数や制御構造，関数呼び出しパターンなどの静的特徴を抽出し，これらを入力として Support Vector Machine（SVM）や Random Forest（RF）を用いた 2 値分類を実施した．この研究では，従来型機械学習手法による一定の検出制度が確認され，機械学習の有効性が示された [1]．これらの手法では，特徴量設計の巧拙が性能に大きく影響するという課題が指摘されている．

また，Zhang らは Word2Vec と FastText という 2 種類の単語埋め込み手法と，CNN および BiGRU を組み合わせたハイブリッドモデル CBGRU を提案した [7]．本手法は，CNN による局所特徴抽出能力と BiGRU による文脈依存性の学習能力を特徴量として融合することによってスマートコントラクトの多様な脆弱性を高精度に検出できる点が特徴である．既存の単一モデルより高い精度と F1 スコアを達成した．この研究では，データセットの多様化やモデルの軽量化，Transformer 系モデルとの比較が課題として挙げられている．また，単一の既知脆弱性クラスを対象とした分類を行っているため，複数存在する脆弱性では適応範囲が限定される．その結果，未知の脆弱性を検知するためには，学習済みツールを繰り返し使用する必要があり，実運用上の負荷となるという課題がある．

加えて，静的解析による検出手法のなかでも，Feist らによって開発された Slither は代表的なツールとして広く利用されている [3]．Slither は Solidity コードを中間表現に変換し，制御フロー解析やデートフロー解析を通じて既知の脆弱性パターンを高精度かつ高速に検出する．定義済みの検出ルールに基づいて分析を行うため，明示的なパターンには非常に強い一方で，コード全体の意味的文脈を理解する力には限界がある．すなわち，構文的な構造や静的な特徴のみに依存するため，複雑なコードパスや文脈依存の脆弱性に対応しにくいという課題がある．

上記の課題を解決するため，本研究では LLM モデルを使用し，コード全体の意味的文脈を理解させることで，複雑なコードや文脈依存関係を考慮した分類を可能とすることができる．また，LLM モデルを学習することによって未知の脆弱性に対応することができることを明らかにする．

3.2 LLM モデルによる脆弱性検出手法

LLM を用いた研究としては，プロンプトベースの脆弱性検知モデルに関する研究が挙げられる．Peter らは，DetectLlama を提案し，Solidity スマートコントラクトに対して，LLM モデルを用いた脆弱性検出を行う手法である．特に GPT-3.5 を脆弱性検出用にファインチューニング（GPT -3.5FT）したモデルが最も高い性能を示し，F1 スコア 0.776 を達成した．加えて，Meta の CodeLlama ベースのモデルを微調整した DetectLlama-Foundation では F1 スコア 0.68，GPT-4 及び GPT-4Turbo はそれぞれ 0.66 及び 0.675 と，事前学習モデルのままでは性能が劣ることが示された．この研究では，ファインチューニングの有効性と，プロンプトベースによる脆弱性分類タスクへの可能性を明らかにしている [6].

これに対し本研究では，ファインチューニングされた LLM を用いて Solidity コードをそのまま入力し，脆弱性分類を既存手法と比較し，有効性を確認する．また，LLM モデルの分類性能の向上手法を提案し，高い分類性能の実現を試みる．

4. 関連技術

本節では，本研究で使用する LLM モデルおよびファインチューニングの技術について述べる．本研究では，Transformer モデルを使用した事前学習済みモデルである Llama を用いた分類タスクに対して，Low-Rank Adaptation of Large Language Models (LoRA) を用いてファインチューニングを実施する．

4.1 Llama

Llama（Large Language Model Meta AI）は，Meta 社が開発した大規模言語モデルであり，Transformer アーキテクチャを基盤としている [13,15]．Transformer は従来の RNN や CNN の代わりに Attention 機構を用いて，文脈理解を効率的に行うモデルであり，自然言語処理における主流技術となっている．Llama はその高い処理性能を活かし，様々な言語タスクに対応可能である．さらに，ソースコード処理に特化した派生モデルとして CodeLlama が開発されており，コード生成，補完，バグ修正，自然言語からのコード変換などに利用できる．本研究では，Solidity コードに特化した脆弱性検出を目的として CodeLlama を

用いる．CodeLlama は構文や文脈の理解に優れており，精度の高いコード解析が可能である．

4.2 LoRA

Low-Rank Adaptation（LoRA）は，大規模言語モデルの効率的なファインチューニング手法として提案された [5]．従来の全パラメータ更新方式では膨大な計算資源とメモリが必要であったが，LoRA では既存のモデルパラメータを固定し，一部の層に低ランク行列を追加して学習を行う．これにより、学習すべきパラメータ数が大幅に削減され，計算コストメモリ使用量を抑えつつ，高い性能を維持したモデル適応が可能となる．

5. 提案手法

本節では，本研究で提案する Solidity コードの脆弱性の検知手法について詳細を述べる．

5.1 概要

図 1 に提案手法の概要を示す．

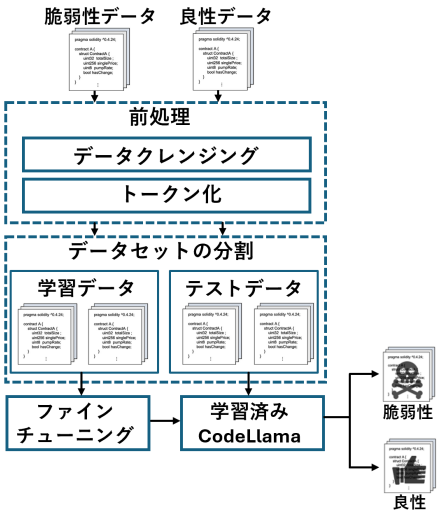

図 1 提案手法

提案手法は，主にデータセットの前処理および LLM のファインチューニング手法から構成されている．以下に提案手法の手順を示す．

1. Solidity コードをトークンに変換し，トークンの数によってデータセットを分ける．
2. 前処理を実施し，1 で作成したデータセットを使用して LoRA で LLM のファインチューニングを実施する．
3. ファインチューニングした LLM で Solidity コードを分類する．

本研究では，LLM のタスクを Transformer の LlamaForSequenceClassification を用いて脆弱性の有無について，2 値分類タスクに設定した．その後，LoRA を用いてモデルのパラメータ効率を向上させるためのファインチューニングを実施する．

5.2 前処理

モデルの学習及び評価の精度を向上させるために，Solidity コードに対してデータクレンジング処理を実施した．処理した内容を**表 1** に示す．

また，Solidity コードのトークンへの変換には，Transformers ライブラリの LlamaTokenizer を用いた．トークンへの変換には，Bite-Pair Encoding (BPE) [13] が用いられている．BPE とは，頻出する文字のペアを結合し，単語をサブワードに分割してトークンに変換する手法である．これにより，未知のトークンに対しても柔軟に対応することが可能である．

表 1 データクレンジング

処理	説明
コメントの削除	//や/*...*/で記述されたコメントの削除
空行の削除	意味を持たない空行を削除
余白の正規化	行頭，行末の余白部分やスペースやタブ文字の正規化
重複データの除去	完全一致するコードの除去

5.3 ファインチューニング

LLM のトレーニングでは，LoRA を用いて一部のレイヤーを低ランクのパラメータに変換し，Transformers のライブラリから Trainer を使用してファインチューニングを実行する．本研究ではチューニングするレイヤーを q_proj と v_proj に指定した．損失関数にはクロスエントロピー誤差を用いた．

5.4 実装

提案手法は，python3.8.9 を用いて実装した．実装に使用した主なライブラリを**表 2** に示す．

表 2 提案手法の実装に用いた主なライブラリ

Transformer	4.46.3
TensorFlow	2.13.0
Scikit-learn	1.3.2
peft	0.13.2

機械学習モデルの実装には，Transformers および Tensorflow を使用した．ファインチューニングライブラリは，peft を使用し，行列のランク r=16，lora_alpha=16，lora_dropout=0.05 と設定した．パラメータ設定に関しては，パラメータ毎に 3 回の精度を比較し，最も良い精度を示した値を採用した．ファインチューニングの epoch 数は 1 回とした．シーケンス長は，50〜400 の範囲で 50 刻みで

精度を比較し，精度と検知時間のバランスが良い 200 に設定した．

6. 検証実験

6.1 概要

本節では，データセット述べた後，実験内容及び結果，分析を通じて，提案手法の Solidity コード脆弱性検知の有効性について述べる．

実験 1 では CodeLlama による Solidity コードの脆弱性の検知制度を確認し，既存手法と比較を行うことで有効性を言及する．

実験 2 では，ソースコード全文を入力することで LLM の強みであるコードの構造や文脈情報を保持したまま学習することの有効性について言及する．

6.2 データセット

本研究では，スマートコントラクトの脆弱性検出のため，Behaviour-Centric Cybersecurity Cente(BCCC) から入手した BCCC-SCsVul-2024 データセットを採用した [4]．このデータセットは，Smart Bugs，ESCs，SmartScan など複数のソースから収集された Solidity ソースコードを基に構築されている．各コントラクトがリエントランシー，整数オーバーフロー，サービス拒否など，特定の脆弱性タイプごとに詳細なラベルが付与されている．良性データ 26,914 件，脆弱性データが合計 84,983 件，総サンプル数は 111,897 件で構成される．各データセットのサンプル数を**表 3** に示す．本稿では，データセットに含まれる Weak access mod の脆弱性 959 件について，その記述が意味を持たない数字のみで構成されていたため，分析対象から除外した．

表 3 BCCC-SCsVul-2024

Label	No.of Samples
Total data	111897
External bug	3604
Gas exception	6879
Mishandled exception	5154
Timestamp	2674
Transaction order dependence	3562
Unused return	3229
Weak access mod	1918
Call to unknown	11131
Denial of service	12394
Integer underflow/overflow	16740
Re-entrancy	17698
Secure	26914

6.3 実験環境

実験環境を**表 4** に示す．

表 4 実験環境

CPU	IntelCore i7-13700KF 3.40GHz
GPU	NVDIA GeForce RTX 3090
Memory	64GB
OS	Windows11 Home
Cuda	12.1
使用言語	Pyhon3.8.9

6.4 比較モデル

本研究では，CodeLlama の分類性能を評価するための比較対象として，以下の５つの代表的な機械学習・深層学習モデル及び関連研究との比較として CBGRU [7] を選定した．Random Forest（RF），Support Vector Machine（SVM），Deep Neural Network（DNN），Long Short-Term Memory（LSTM），eXtreme Gradient Boosting（XGboost），及び CBGRU モデル（Word2Vec ＋ CNN と FastText ＋ BiGRU を組み合わせたハイブリッドモデル）である [7]．これらは従来から様々な分類タスクで高い性能を示しており，本実験においても基準モデルとして適切であると判断した [1]．各分類モデルのパラメータを**表 5** に示す．

LLM モデルの比較として，Mistral7b を採用した．Mistral 社が開発したオープンソースの LLM モデルであり，高効率かつ高精度な推論が可能である [12]．Transformer アーキテクチャに基づき，推論速度とメモリ使用量を最適化する Grouped Query Attention や長いシーケンスを効率的に処理しつつ文脈長を拡大する Sliding Window Attention を導入することで，従来モデルと比較してメモリ効率と推論速度が向上している．本研究では，7B（70 億）パラメータを有する Mistral 7B を使用し，コード特化型である CodeLlama がいかにソースコードへの脆弱性検知に優れているかの比較対象モデルとして採用する．

表 5 分類モデルのパラメータ

モデル	パラメータ
RF	Sklearn, 決定木=100，random_state=42
SVM	Sklearn， kernel=’rbf’ (Radial Basis Function)
DNN	Keras，Optimization=Adam, epochs=10
LSTM	Keras, Optimization=Adam, epochs = 10
XGboost	XGboost, 決定木=100，random_state=42
CBGRU	Keras,Optimization=Adam, epochs=50

6.5 実験１

実験１では，第６節に示す分類モデルと比較し，提案手法の有効性を検証した．脆弱性データセットを 65880 件と 18144 件に分け，65880 件からランダムアンダーサンプリングによって 8800 件にしたデータを学習用データとした．内訳は，各脆弱性からランダムに 800 件ずつ抽出し，8800 件の脆弱性データを生成した．良性データからも 8800 件

抽出し，均衡なデータセットを生成した．テストデータは脆弱性データ 18144 件及び良性データ 18144 件の均衡データでテストを実施した．データセットのサンプル数を**表 6** に示す．分類タスクにおける検出結果は，0 が良性の判定，1 が脆弱性ありの判定を示す．

6.6 実験 2

実験 2 では，Solidity スマートコントラクトの全文を入力として Code Llama を学習させることで，コード全体の文脈を考慮した脆弱性検出能力を評価することを目的とする．ソースコードの長さによる学習，分類性能の変化について検証する．

Solidity コードを LlamaTokenizer を用いてトークン化し，表 6 に示すとおり各コントラクトを 0〜2000 トークンまで 500 トークンずつ 4 つのセグメントに分割して入力データとした．学習データセットは，ランダムアンダーサンプリング手法を用いて構築した．学習データには，11 種類の脆弱性クラスからそれぞれ 100 件ずつ，計 1100 件の脆弱性データを抽出した．また，同数である 1100 件の良性（非脆弱）データを抽出し，合計 2200 件の均衡データセットを作成した．同様に，テストデータについても，各脆弱性クラスから 50 件ずつ，計 550 件の脆弱性データを抽出し，良性データも同数用意し，合計 1100 件からなる均衡なテストデータセットを構築した．本実験においては，データセット内に含まれる Weak access mod の脆弱性に関しては，サンプル数が不足しているため，学習：テストを 8：2 の割合で抽出した．

表 6 実験データセット

		学習データ		テストデータ	
		良性	悪性	良性	悪性
実験 1		8800	8800	18144	18144
実験 2	0-500SQ	1015	1015	504	504
実験 2	501-1000SQ	1037	1010	510	510
実験 2	1001-1500SQ	1100	1100	543	543
実験 2	1501-2000SQ	1100	1100	550	550

6.7 実験結果

実験 1 の結果を**図 2** に示す．実験 2 の結果は各トークン数における F-measure を**図 3** に示す．また，サンプル 1 件当たりに要する時間を**表 9** に示す．

実験 1 において，提案手法は比較している既存手法よりも精度が低いことを確認した．実験 2 においては他のモデルと比較し，各シーケンスにおいても F1 値が高く，分類精度が高いことを確認した．

学習時間に関しては，表 9 に示す．Code Llama は 1 件あたりに要する時間は Solidity コードシーケンス長が長くなるにつれ学習時間が長くなり，1501-2000 シーケンス長では，学習 1 件あたり 261.7 秒であることを確認した．

表 7 実験 1 結果

	Accuracy	Precision	Recall	F1
CodeLlama	0.754	**0.839**	0.655	0.736
Mistral7b	0.741	0.815	0.625	0.707
SVM	0.757	0.761	0.757	0.756
RF	**0.797**	0.798	0.797	**0.797**
DNN	0.767	0.768	0.767	0.767
LSTM	0.797	0.702	0.692	0.689
XGboost	0.796	0.803	**0.797**	0.796
CBGRU [7]	0.678	0.663	0.719	0.690

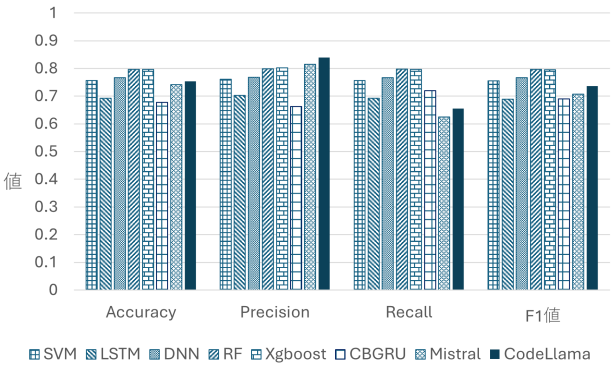

図 2 実験 1 結果

表 8 各トークン数における検知精度

モデル	範囲	Accuracy	Precision	Recall	F1
CodeLlama	0-500	**0.877**	**0.867**	**0.889**	**0.879**
	501-1000	**0.804**	0.750	**0.809**	**0.796**
	1001-1500	**0.750**	0.657	**0.864**	**0.781**
	1501-2000	**0.786**	0.635	**0.802**	**0.738**
Mistral7b	0-500	0.861	0.859	0.865	0.862
	501-1000	0.761	**0.771**	0.764	0.774
	1001-1500	0.755	0.742	0.780	0.760
	1501-2000	0.743	0.640	0.791	0.730
SVM	0-500	0.817	0.823	0.817	0.816
	501-1000	0.680	0.685	0.680	0.678
	1001-1500	0.722	0.725	0.722	0.720
	1501-2000	0.720	0.723	0.719	0.718
RF	0-500	0.820	0.822	0.820	0.820
	501-1000	0.723	0.724	0.723	0.723
	1001-1500	0.740	**0.761**	0.758	0.757
	1501-2000	0.664	0.665	0.664	0.663
DNN	0-500	0.714	0.731	0.714	0.705
	501-1000	0.609	0.627	0.609	0.595
	1001-1500	0.746	0.755	0.746	0.744
	1501-2000	0.662	0.669	0.662	0.657
LSTM	0-500	0.499	0.350	0.499	0.335
	501-1000	0.508	0.481	0.508	0.384
	1001-1500	0.492	0.368	0.492	0.336
	1501-2000	0.500	0.400	0.500	0.337
XGboost	0-500	0.845	0.845	0.845	0.845
	501-1000	0.715	0.715	0.715	0.715
	1001-1500	0.736	0.737	0.736	0.736
	1501-2000	0.727	**0.728**	0.727	0.727
CBGRU [7]	0-500	0.620	0.619	0.626	0.623
	501-1000	0.632	0.637	0.613	0.625
	1001-1500	0.596	0.589	0.637	0.612
	1501-2000	0.546	0.555	0.469	0.504

7. 考察

7.1 提案手法の検知精度

検証実験の結果，LLM を用いた提案手法の検知精度は，

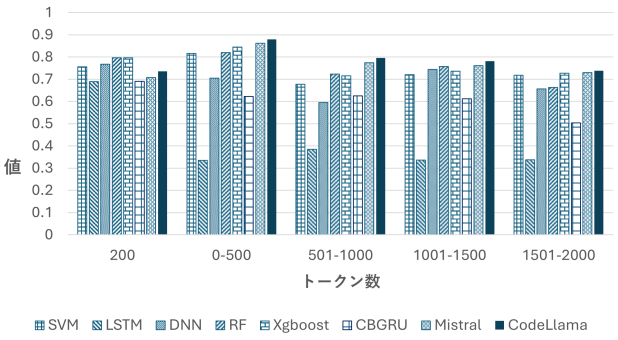

図 3 各トークン数における F-measure

表 9 提案手法の学習時間

トークン数	実行時間 (s)	1 件当たりの時間 (s)
0-500	74377	85.9
501-1000	114646	109.5
1001-1500	292820	133.1
1501-2000	575832	261.7

既存手法および従来モデルに比べて劣ることが明らかとなった．実験 1 における F1 スコアを図 2 に示す．提案手法の F1 スコアは 0.736 であり，最も高精度であったモデル（RF）と比較しても，精度が劣る結果となった．実験 1 では，各脆弱性クラスおよび非脆弱性クラスからコードサンプルをランダムに抽出し，固定されたシーケンス長でモデルの学習および評価を実施した．本実験では，シーケンス長を 200 トークンに設定したが，Solidity コードの構造的特性を考慮すると，このアプローチにはいくつかの課題が存在する．第一に，実際のスマートコントラクトは 200 トークンを大幅に超える長さであることが多く，脆弱性はコード全体に散在している可能性が高い．よって，脆弱性が必ずしもその範囲内に含まれるとは限らないため，短いシーケンス長では，その一部しか捉えられない可能性がある．第二に，LLM の強みであるコード全体の文脈的関係性の把握が，短いシーケンスに限定されることで十分に活かされていないと考えられる．以上の点から，提案手法による脆弱性検出性能が限定的となった要因の一つとして，シーケンスの抽出方法およびその長さの設定に起因する情報損失が挙げられる．

7.2 LLM による全文コンテキストでの脆弱性検出

コードサンプル内のトークン数に応じて，各データに対して全文を読み込ませた状態で学習，テストを行った結果，提案手法はすべてのシーケンス長において最も高い F1 スコアを示した（表 8）．これは，LLM がコード全体の文脈を把握できるという特性を活かした結果であり，既存手法と比較して分類性能が向上したと考えられる．具体的には，CodeLlama を用いた提案手法において，以下のような F1 スコアが得られた．0–500 トークンのサンプルでは 0.879，501–1000 トークンでは 0.796，1001–1500 トークンでは 0.781，1501–2000 トークンでは 0.738 であり，いずれの区間においても他のモデルよりも高い性能を示した．Recall の値に関してもほかの比較モデルに比べて高い性能を示し，開発者にとって Solidity コードの脆弱性を見逃さない高い検出性能であることを確認した．これらの結果は，LLM が持つ長距離依存関係の把握能力および文脈理解能力が，Solidity コード中の脆弱性検出において有効であることを示唆している．従って，固定長のシーケンス抽出による断片的な学習よりも，コード全文を処理対象とするアプローチの方が，LLM の潜在的な能力をより効果的に活かすことができることを明らかにした．

7.3 コードの長さによる学習時間の変化

検証実験の結果，Code Llama の学習時間は表 9 で示す通り一番長い 1501-2000 トークン数での実験では 1 件あたり 261.7 秒であった．Solidity コードは図 4 で示す通り本実験で使用した 2000 トークン以上に現実に存在しているソースコードが長い傾向があることを考慮すると，学習時間に要する時間は大幅に増加すると考察できる．しかしながら，Solidity スマートコントラクトに内在する脆弱性は，コードのデプロイ前，すなわち開発段階で検出されることが望ましい．よって，リアルタイムでの対応を要求される場面は限定的であり，むしろ検出手法に求められるのは脆弱性の検知精度であることを考慮すると，必ずしもリアルタイム性を前提とした設計が必要とされるわけではない．

7.4 関連研究との比較

本研究で提案した手法と既存研究との比較を通じて，有効性および限界について検討する．

Zhang らが提案した CBGRU [7] では，最良の F-measure が 0.623 であり，本研究の手法と比較して性能が劣る結果となった．なお，Zhang らの研究では，各脆弱性ごとに多数のサンプルを用いて学習が行われており，本研究と比較して 5 倍以上のデータ量の差がある．このようなデータ量の差異により，同一条件下での比較では学習が不十分となり，性能に影響を及ぼした可能性があると考察する．

一方，Feist らが開発した Slither [3] は，静的解析を用いて Solidity コード中の脆弱性を検出するツールであり，特定のルールに基づいて，脆弱性の種類とその位置を明示的に報告する機能を有している．したがって，Slither は脆弱性の有無を分類する本研究のタスクとは根本的に目的が異なっており，出力形式にも違いがある．そのため，本研究のモデルと Slither との間で直接的な性能比較を行うことは適切ではないと判断した．

7.5 研究倫理

本研究で使用したデータセットはすべて公開されている

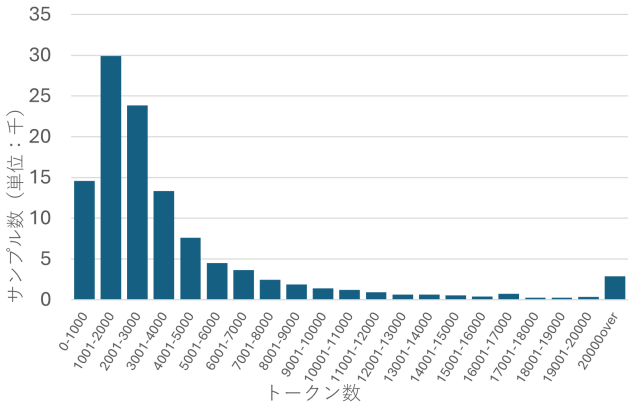

図 4　トークン数ごとのサンプル数

か，あるいは公開サイトの情報に基づいて収集したものである．また，収集時期や収集手段についても明記しており，研究の再現性は確保されているものと考えられる．

8. おわりに

本研究では，LLM モデルを用いて言語的特徴から Solidity コードの脆弱性を検知する手法を既存手法と比較し，LLM の性能向上手法について提案した．固定長のシーケンス抽出による検証実験の結果，LLM を用いた提案手法は機械学習モデルと比較して精度が劣ることを確認した．しかし，コード全文を処理対象とするアプローチでは，LLM の強みであるコード全体の文脈的関係性の把握を活かし，他の機械学習モデルと比較し，高い分類性能を示した．特に，ソースコード解析・生成に特化した CodeLlama は，汎用 LLM である Mistral と比較して，コード構文やプログラミングパターンの理解に優れ，Solidity コードに対する分類精度で優位性を示した．

今後の課題としては，次の 2 点である．1 つ目は，学習コストである．本実験では，2000 トークンでの学習に約 7 日費やした．Ethereum で使用されている Solidity コードは本実験で使用した 2000 トークン以内にとどまらず，現実では 10000 トークン以上のコードも存在する．学習コストが膨大になるという課題がある．2 つ目は，本実験では対象コード内に脆弱性の有無に関する 2 値分類であるため，具体的な脆弱性箇所を特定することはできない．よって，開発者がソースコードの修正を行う際に，原因箇所を特定することはできないため，行単位での検知能力が必要とされる．

参考文献

[1] Fadi Al-Mousa and Rawan Mohammad A. Alnsour. Smart contract security analysis based on machine learning techniques. *Computer Systems Science and Engineering*, 46(2):1427–1440, 2023.

[2] Nami Ashizawa, Naoto Yanai, Jason Paul Cruz, and Shingo Okamura. Eth2Vec: Learning contract-wide code representations for vulnerability detection on Ethereum smart contracts. *arXiv preprint arXiv:2101.02377*, 2021.

[3] Josselin Feist, Gustavo Grieco, and Alex Groce. Slither: a static analysis framework for smart contracts. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB@ICSE 2019, Montreal, QC, Canada, May 27, 2019*, pages 8–15. IEEE / ACM, 2019.

[4] S. HajiHosseinKhani, A. H. Lashkari, and A. M. Oskui. Unveiling smart contract vulnerabilities: Toward profiling smart contract vulnerabilities using enhanced genetic algorithm and generating benchmark dataset. *Blockchain: Research and Applications*, 6(2):100253, 2025.

[5] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[6] Peter Ince, Xiapu Luo, Jiangshan Yu, Joseph K. Liu, and Xiaoning Du. Detect llama - finding vulnerabilities in smart contracts using large language models. In Tianqing Zhu and Yannan Li, editors, *Information Security and Privacy - 29th Australasian Conference, ACISP 2024, Sydney, NSW, Australia, July 15-17, 2024, Proceedings, Part III*, volume 14897 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2024.

[7] Guiping Liu, Jing Wang, Jing Wang, and Chao Li. Cbgru: A detection method of smart contract vulnerability based on a hybrid model. *Electronics*, 12(1):177, 2023.

[8] OWASP Foundation. OWASP Smart Contract Top 10. `https://owasp.org/www-project-smart-contract-top-10/`, 2025. Accessed: [Current Date, e.g., 2025-07-10].

[9] Noama Fatima Samreen and Manar H. Alalfi. A survey of security vulnerabilities in ethereum smart contracts. *CoRR*, abs/2105.06974, 2021.

[10] David Siegel. Understanding the dao attack. *Medium*, 2016.

[11] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.

[12] Parity Technologies. Parity wallet hacking incident: The aftermath and lessons learned. *Parity Technologies Blog*, Nov 2017. Accessed: [Current Date, e.g., 2025-07-15].

[13] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[14] Trail of Bits. Echidna: A Fast Fuzzer for Ethereum Smart Contracts. `https://blog.trailofbits.com/2018/10/05/echidna-a-fast-fuzzer-for-ethereum-smart-contracts/`, 2018. Accessed: [Current Date, e.g., 2025-07-10].

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[16] Hengyan Zhang, Weizhe Zhang, Yuming Feng, and Yang Liu. SVScanner: Detecting smart contract vulnerabilities via deep semantic extraction. *JOURNAL OF INFORMATION SECURITY AND APPLICATIONS*, 75, 2023.