大規模言語モデルによるソフトウェア脆弱性の検出

高知大学 理工学部情報科学科 B4

横川武典

1. 研究背景

言語モデルを用いたコーディングが普及

→セキュリティに関する指示をしない場合**脆弱なコードが生成**

図1. コーディングに使われる言語モデルが含まれるツールの例

Webアプリケーションは公開範囲が広い

→開発者自身が**脆弱性に気づかないまま公開**

　→**コードの安全性**の担保が課題

言語モデルを脆弱性検出に活用

→ **文脈**を考慮可能

　→ルールベースでは発見の難しい脆弱性も検出可能

一方で、以下の問題も存在

- **誤検知**
- **検出漏れ**

→ 言語モデルの使い方次第で検出性能が変化

→ 利用方法の違いが脆弱性検出に与える影響の整理は不十分

2. 研究目的

言語モデルで脆弱性を**自動で検知**する際、
モデルの利用方法の違いが検出に与える影響を調べる

以下の3種類で脆弱性検出を行い、**精度**/**安定性**を評価

- 素の言語モデル
- **FineTuning**を行った言語モデル
- **RAG**を利用した言語モデル

3. 既存手法

従来の脆弱性検出手法は大きく2つに分けられる

静的解析

- 実行せずにコードを解析
- ルールベース中心
- **誤検知が多い**

動的解析

- 実行時の挙動を解析
- 実環境に近い検出
- **検出漏れが発生しやすい**

→ 既存手法では**コード全体の文脈**を考慮した検出が困難

→ **文脈理解**による検出手法として**言語モデルの活用**が期待

セキュリティ分野での言語モデルの活用

言語モデルは**セキュリティ関連タスク**に既に活用

- **脆弱性の説明・要約**
  - 脆弱性レポートやアラートの理解支援
- **脆弱性検出**
  - コードの文脈を考慮した指摘が可能

→ **言語モデルの出力の信頼性**が重要

特に脆弱性検出では**誤検知・検出漏れが実運用に影響**

4. 関連研究

LLMs in Software Security: A Survey of Vulnerability Detection Techniques and Insights [Ze Sheng+2025]

- **コードの断片**から脆弱性を検出できる言語モデルが存在
- **リポジトリ**単位では限定的な脆弱性の検出のみ可能
- メモリ関連の脆弱性は検出精度が高い
- C/C++に関する研究が多い

→ 既存研究は主に **C/C++** が中心

Web言語を対象に**LLMの利用方法の違い**に着目して評価

5. 提案手法

脆弱性のデータベースを**参照**/**学習**に利用

→ 脆弱性の発見精度を改善

1. 脆弱性のデータベースを取得/作成
2. データベースを元に**RAG**の作成/**FineTuning**
3. 各モデルでソースコードから脆弱性を探す

RAG(Retrieval-Augmented Generation)とは

RAGは**事前学習していない外部知識を検索しその情報に基づいて文章を生成する手法**

以下の3段階で機能

1. クエリと**類似する情報**を
   データベースから検索
2. **得られた情報**をプロンプトに追加
3. LLMにプロンプトを入力して推論

→ 最新の**脆弱性知識**を柔軟に反映

FineTuningとは

既存の学習済みモデルに**追加の学習を行い特定のタスク用に調整する**手法

メリットは以下の3つ

1. **コストが安い**
2. 最新の情報への対応
3. 特定用途に**特化したモデル**を作成可

→ **脆弱性検出タスク**への適応

6. 実験設定

1. 脆弱性の**データベースを取得/作成**
   → 脆弱性データ数: 19800件
      脆弱性のある記述例: 750件
2. データベースを元に**RAGの作成/FineTuning**
3. ソースコードから**脆弱性を探す**
   → LLMが生成したソースコード: 10件

7. 評価方法

脆弱性の検出結果を**指標**として数値化

- 対象：PHPで記述されたWebアプリケーション
- 比較対象：素のLLM (以下**Base**)
  FineTuningを行ったLLM (以下**FT**)
  RAGを接続したLLM (以下**RAG**)
  - 各構成に同一コードを入力し、脆弱性の検出結果を取得
- 評価指標：**Precision** / **Recall** / **F1-score** / **AUC**
  ※ AUC：**閾値に依存しない**検出性能指標

8. 実験結果

図4より、F1-scoreは

- **FT**が最も正しく検出
- **RAG**は**Base**より悪化

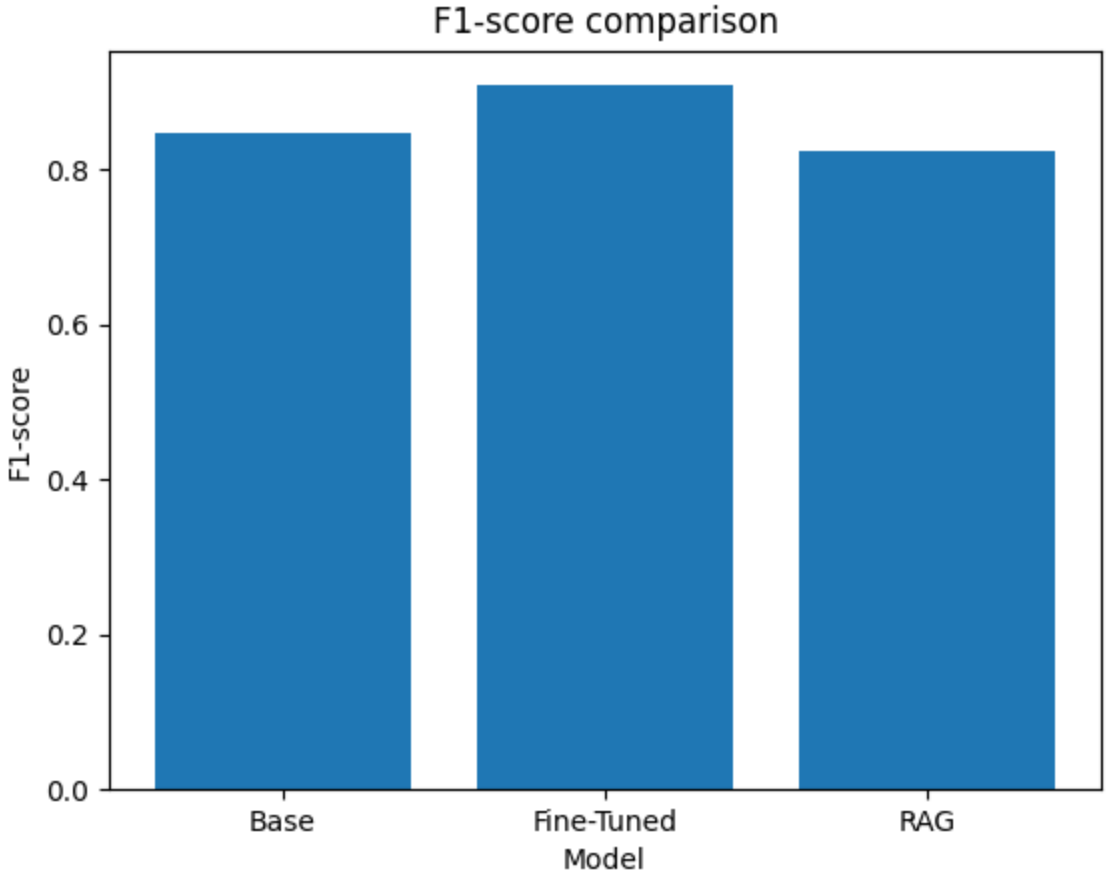


図4. F1-scoreの測定結果

8. 実験結果

図5より、AUCは

- **FT**が最も検出性能が高い
- 次点で**RAG**が高い

次に、結果が生じた要因を
誤検知（FP）・検出漏れ（FN）
から分析する

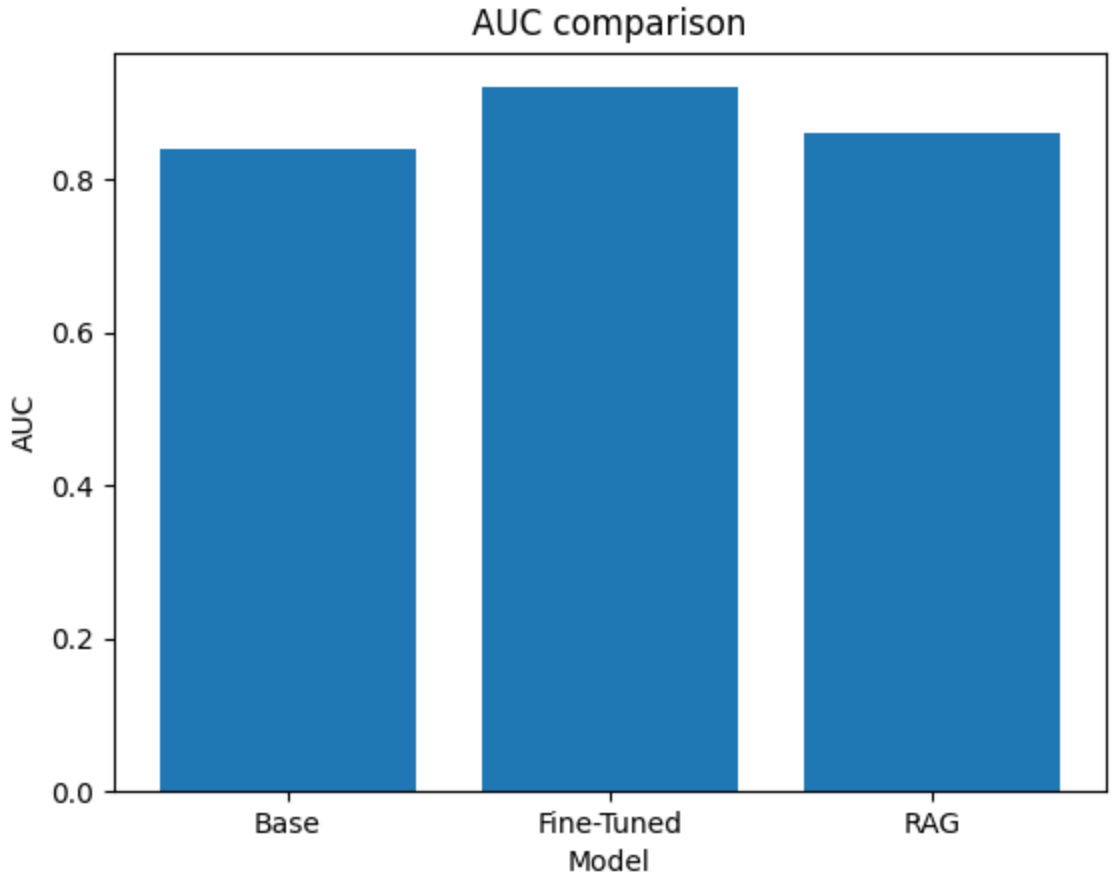


図5. AUCの測定結果

誤検知(FP)の傾向

Base において多く発生、**RAG** でも一定数発生

- Baseでは文脈理解が不十分で脆弱性を**過剰**に指摘
- RAGでは**関連知識の誤適用**/**コンテキスト長の増加**が存在

XSS を検出するタスクで多く発生

- 過剰に出力をエスケープ
  → **文脈を理解した検出**は不十分

検出漏れ(FN)の傾向

Base において多く発生

- 文脈理解が不十分なまま安全側に判断

FT / RAG では脆弱性知識の獲得により改善

- ただし完全には解消されず

9. 結論

LLMはWebアプリケーションの脆弱性検出に**活用可能**
ただし**利用方法により検出性能は大きく異なる**

- FTは**検出精度の安定化**に有効
- RAGは**知識依存度の高い脆弱性**で効果を示した

→ 利用方法の違いが検出性能に影響することを確認

10. 今後の課題

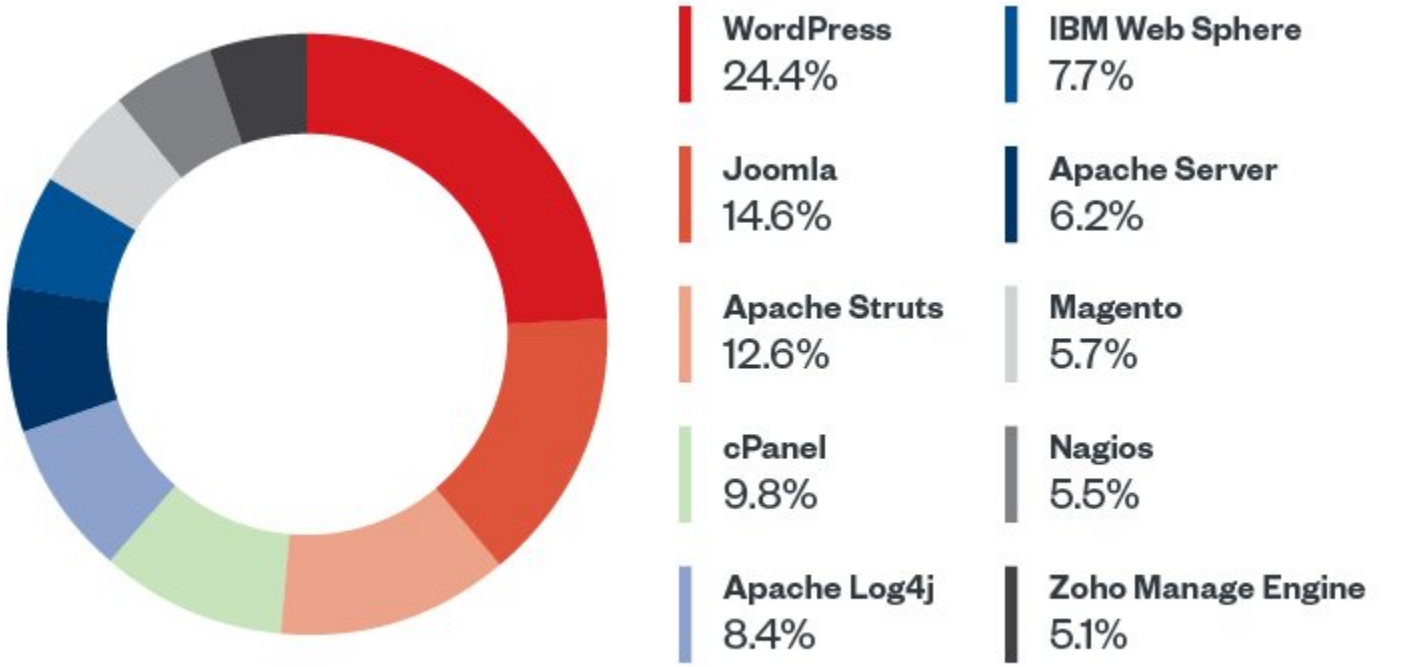
- より大規模・多様なコードでの検証
- 脆弱性検出結果の**根拠や判断理由の可視化**
- 誤検知パターンの**体系的分析**

参考文献

- [Ze Sheng+2025] LLMs in Software Security: A Survey of Vulnerability Detection Techniques and Insights https://arxiv.org/html/2502.07049v2
- JVN iPedia - 脆弱性対策情報データベース https://jvndb.jvn.jp/
- CVE: Common Vulnerabilities and Exposures https://www.cve.org/CVERecord

Appendix A. WEB言語の脆弱性の傾向

2022年に脆弱性が悪用された技術としてWordPressがトップ
全体の1/4を占めた



Appendix B. データベースの活用方法

FineTuningの場合

OSSの言語モデルに脆弱性のデータベースで**追加学習**

この処理は"unsloth", "trl"等のライブラリを使用

RAGを作成する場合

自然言語で書かれた脆弱性のデータベースを**ベクトル**に変換

この処理は"LangChain", "LanceDB"等のライブラリを使用

ベクトル化には"infloat_multilingual-e5-large"を使用