GNNsを用いた静的プログラム解析に関する研究

M2022SE004 飯盛天翔
指導教員：野呂昌満

1　はじめに

機械学習技術の発展に伴って，その技術を用いたプログラム解析の研究も盛んに行われている [3] [10]．他方，CDI(CoDe Inspection) ツール等の従来の静的プログラム解析ツールは，アルゴリズミック (決定論的) にフォールトを解析している．決定論的な従来の静的プログラム解析ツールでは，予め仕様で決めたフォールトだけしか検出できない．NN(Neural Networks) の学習過程を従来の生成過程とみなすことができる．すなわち，訓練データが仕様を暗黙に定義していると考えれば，NN をそのデータで学習させることで，NN を用いたフォールト検出器が生成できる．NN を用いた静的プログラム解析の研究では，主に RNN(Recurrent Neural Network)[3]，Transformer[2]，GNNs[10] が用いられる．GNNs を用いた静的プログラム解析では，プログラムをグラフとして処理する．プログラムの本質的な構造は AST(Abstract Syntax Tree) で表現されるグラフ構造 (木構造) である．この構造を積極的に扱うという観点から，GNNs を用いたフォールト解析が可能であるという着想を得た．

本研究では，GNNs を用いた静的プログラム解析の可能性の考察を目的とする．

ソフトウェア工学の事例研究課題として以下を定義した．

RQ1. 予測学習モデルの設計と試作．

GNNs を用いたフォールト予測モデルを設計・試作する．AST から得られる構造を NN の構造としたフォールト検出器が生成できるかを確認する．

RQ2. 検出精度の考察．

正解率，AUC(Area Under the ROC Curve)，F 値を用いて得られたモデルの評価結果を基に，フォールト検出の可能性の考察を行う．

RQ3. 適用可能性範囲の考察．

提案するモデルが他の種類のフォールトも予測できるのかを考察する．定義したフォールトとは異なる種類のフォールトデータセットを提案モデルに適用して実験を行い，その結果の考察を行う．

2　関連技術と先行研究

GNNs を用いた研究は近年盛んに行われている [8][11]．GNNs とは，グラフ分類，ノード分類，リンク予測等の応用に使用される．GNNs は，GCN(Graph Convolutional Networks) と GRN(Graph Recurrent Networks) に分類される．

GCN は，GNNs に畳み込みの概念を取り入れた NN である．GCN の畳み込みには，スペクトラルアプローチと空間アプローチの 2 つのアプローチがある．スペクトラルアプローチとは，グラフをグラフフーリエ変換し，変換したグラフを畳み込むアプローチであり，分子構造分析や推薦システムに用いられる．空間アプローチとは，グラフ構造を直接畳み込むアプローチであり，グラフマッチングやソーシャルネットワーク分析に用いられる．

他方，GRN はグラフのリンクをたどり，解釈実行しながら動的に推論を行う GNNs である．

2.1　Menzies らの研究 [7]

Menzies らは，Maccabe IQ(プログラム複雑度)，Halstead E(プログラムの理解可能性)loc(コード行数) を用いて，データマイニングによるフォールトの有無を推測していた．この手法は，一定の成果を出しているがソフトウェアの構造を陽に扱っていないので，プログラムのフォールトの有無しか検知できなかった．

2.2　Zhou らの研究 [12]

Zhou らは，GNNs を用いたソフトウェアの構成を示した GNN パイプラインアーキテクチャ(図 1) を提案した．

入力コンポーネントでは，入力とするデータのグラフ構造を決定する．損失関数コンポーネントでは，ノードレベル，エッジレベル，グラフレベルのどのレベルで分類や予測を行うかの決定と学習方法の決定を行う．グラフの表現学習コンポーネントのパイプラインは複数の計算モジュールを持つ．グラフの表現学習コンポーネント中の伝播モジュール (図 1 中の橙色で塗られたコンポーネント) は置き換え可能なモジュールである．伝播モジュールには，グラフのノード間の関係から畳み込み演算を行う GCN とグラフを直接解釈実行する再帰演算を行う GRN がある．

2.3　Borandag の研究 [3]

Borandag は,RNN を用いたフォールト予測モデル，RNNBDL(RNN-Based Deep Learning) を提案した．RNNBDL は，LSTM 1 層，BiLSTM 1 層，全結合層 2 層で構成されているモデルである．RNNBDL は，Borandag の研究で作られた SFP-TDD データセットを適用して訓練した．この研究は，データが暗黙の仕様とすることを示した．

2.4　Šikić らの研究 [10]

Šikić らは，GCN を用いて，end-to-end SDP(Software Defect Prediction) モデルの作成している．Šikić らは研究課題を達成するのに，AST を入力とし，GCN を用いたモデル,DP-GCNN(Defect Prediction GCNN) を提案した．GCN から作成されたスペクトルの形状からフォールとの有無を判断しようとした．総体的には他のモデルと比較して良い精度となっていた．実用性には欠けるが，プログラム解析器の生成としての成果が得られた結果となった．

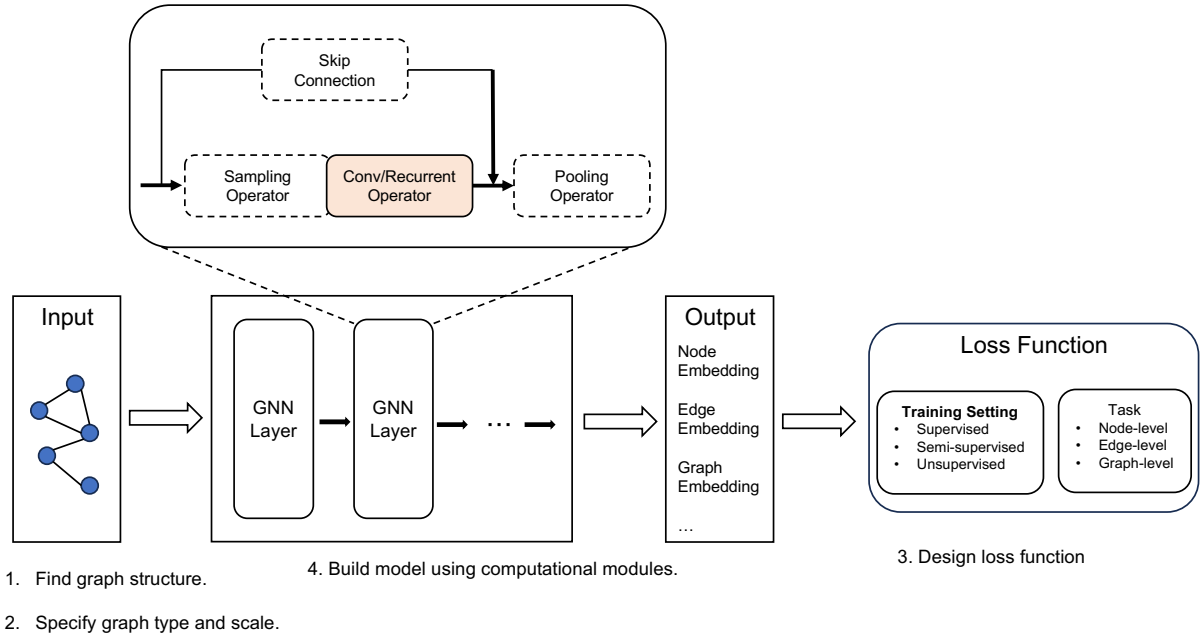


図 1: GNN モデルパイプラインアーキテクチャ ([12] 中の図 2 を再作図)

3 問題解決へのアプローチ

本研究は，GCN で生成系が構成可能かどうかを考える．我々は，対象とするフォールトを構造フォールトとする．Beizer ら [1] は構造フォールトを制御フローや式操作に関わる実装の誤りと定義している．

我々は，Šikić らに倣い，入力を AST とすることを選択した．AST は，静的な構造のみを持つ中間表現である．AST のノード間の関連やノードの属性から，プログラムのフォールトパターンを推論可能だと考える．クラス名やメソッド名などの AST の属性からどの様な機能のプログラムかを判断することが可能であり，その機能に必要とされる AST のノードの有無によってフォールトパターンを発見可能だと考えた．

フォールトは AST のパターンとして現れるので，AST の構造を NN の構造とする GNNs を用いる．具体的なモデルは，Zhou らのアーキテクチャ(図 1) を基に設計する．本研究では，GCN で生成系が構成可能か確認する第一歩として，グラフ全体の特徴量からプログラム中にフォールトがあるかどうかを予測する方法を採用する．

提案モデルの評価を行うために，提案モデルの学習後に NN の 2 値分類で一般的に用いられる正解率，AUC(Area Under the ROC Curve)，F 値を用いる．AUC は，横軸に偽陽性率 (FPR)，縦軸に真陽性率 (TPR) を取る曲線の面積からモデルを評価する．AUC は，モデルの性能を定量的に評価でき，データの偏りや不均衡の影響を受けにくい評価指標となっている．F 値は，再現率と適合率の調和平均であり，モデルが陽性サンプル (フォールト) をどれだけ正確に特定できるかと，その陽性サンプル (フォールト) の中でどれだけ再現できるかを同時に評価する．

4 提案モデル

4.1 グラフ定義

3 節で述べた通り，入力としてプログラムの中間表現である AST を用いる．入力とする AST から，プログラム中のフォールトのパターン認識をするのに必要だと考えられる特徴量を決定しなければならない．定義した特徴量を以下に述べる．

- AST ノードの型
  AST のノードの型とは，if, while, for, assignment 等のプログラムの構造や制御フローを示す特徴量である．対象とするプログラムの構造によるフォールトを予測するのに重要な特徴量となる．例えば，メソッド宣言がされているのに，そのメソッドが使用されていなかったりする時に，フォールトがあるのではないかという指標となる．
- ノードの属性
  ノードの属性とは変数名，値，演算子等のプログラムの変数や定数の利用，演算子の種類に関係する特徴量である．プログラムの動的意味を推論するのに必要な特徴量だと考える．クラス名や変数名，for 文，while 文の条件式などからプログラムをどの様に動作させたいのかを捉えることができると考える．

4.2 GNN モデルの設計

Zhou らの GNN モデルパイプラインアーキテクチャ(図 1) を基に提案モデルの設計を行う．既存技術を組み合わせて，フォールト検出器の自動生成を目指す．

- グラフ構造
  入力データとして上述の AST を用いる．
- グラフタイプとスケール
  入力とする AST を静的有向同種グラフとする．有向

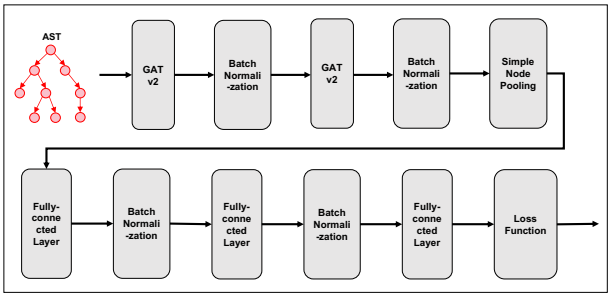


図 2: FP-GAT のアーキテクチャ

グラフの場合，メソッド間のメッセージ送信を表現可能である．同じ意味のエッジによって関係を示している同種グラフとなる．時系列で，AST が変化するわけではないので静的グラフとなる．これらのことから，AST は静的有向同種グラフとなる．

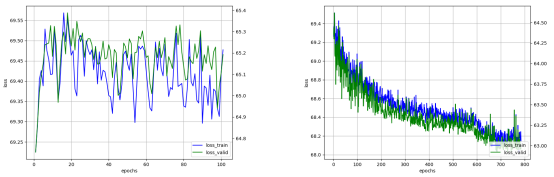
- 損失関数の設計
  グラフレベルの損失関数を用い，学習方法は教師あり学習とする．データがどのラベルに属しているか明確であるので，教師あり学習を用いる．
- 計算モジュールを用いてモデルの作成
  伝播モジュールとして GATv2(Graph Attention Networks v2)[4] を用いる．GATv2 は,，アテンション機構を伝播ステップに反映させた伝播モジュールである GAT を改善したモデルである．アテンション機構は本研究のフォールト検出器を生成するのに最適であると考える．アテンション機構によって，近傍ノードの重要度を考慮して集約を行うので，AST を用いてプログラムの構造からフォールトを予測可能であると考える．

図 2 に作成したモデル，FP-GAT(Fault Prediction GAT) のアーキテクチャの一つを示す．畳み込み層の数は，2 層以下とし，プーリング層 1 層，全結合層 3 層とした．畳み込み層を 2 層以下とした．プーリング層として一般的に用いられる，全てのノードの特徴量に対して総和の演算を行うプーリングを採用した．過学習の防止や妥当な結果を得るために，全結合層は手始めに 3 層とした．

4.3 データセット

データセットは，我々が定義した構造フォールトを多く含むオープンデータセットである Project CodeNet[9] とソースプログラムを AST に変換する javalang[6] を用いて作成した．本研究で対象とする言語を Java とする．理由は以下の 3 点である．

- Java はシステム開発でよく用いられており，最終的に実システムに適用させることを想定している．
- Java はオブジェクト指向であり，Java の言語特性である各 Java ファイルに対してクラスを書かなくてはならないので, 定義したノードの特徴量を得やすいと考えた．
- CBO(Coupling Between Object classes) などのオブジェクト指向のメトリクスを最終的にノードの特徴量として追加したいと考えた．



(a) 畳み込み層 1 層の学習過程 **(b)** 畳み込み層 2 層の学習過程

図 3: 学習過程

5 考察

5.1 学習過程の実験結果

3 種類のハイパーパラメータである hidden_channels, in_heads, hidden_feature を変更させて実験を行なった．hidden channels，in heads は GATv2 のハイパーパラメータである．hidden channels は，隠れ特徴量の次元数であり，in heads は，アテンションが何回使用されるかを表している．hidden feature は，全結合層の隠れ特徴量の次元数である．実験結果の一部である (hidden_channels, in_heads, hidden_feature) = (128, 8, 128) の時の畳み込み層 1 層と畳み込み層 2 層の学習過程を図 3 に示す．

5.2 RQ1. 学習モデルに関する考察

図 3a のように，畳み込み層が 1 層だと学習がうまくいかず，損失 (loss) のグラフが右肩下りにならず上下に振動していた．畳み込み層 2 層のハイパーパラメータが (hidden_channels, in_heads, hidden_feature) = (128, 8, 128) と (hidden_channels, in_heads, hidden_feature) = (256, 8, 256) の時，図 3b のように損失のグラフが右肩下がりになっており学習がうまくいった．畳み込み層が 1 層の場合，深い AST が存在するときに学習ができないことが原因ではないかと考える．畳み込み層が 2 層の場合，深い AST でも畳み込み層 1 層より対応可能である．これらのことから，畳み込み層が 2 層のアーキテクチャの方が様々なフォールトに対応可能なアーキテクチャとなっている．

提案モデルのアーキテクチャでは，全結合層を手始めに 3 層としたが，全結合層の層数についても考える必要がある．データセットのデータ数が少量の場合，全結合層を深くし過ぎてしまうと過学習をしてしまう恐れや，データ数に対してモデルが複雑になるのでうまく学習ができなくなり，モデルの精度に影響が出る可能性がある．全結合層の深さを変化させ，データ数に対してどのくらいの深さの全結合層が良いかを検証しなければならないと考える．

本研究では，オブジェクト指向言語である Java を対象としてグラフの定義を行なった．AST は言語によって，ノードの型が異なったりするが，本質的には同じグラフ構造となっているので，本研究の提案モデルは他の言語にも適用可能であると考える．

5.3 学習済みモデルの評価結果

学習がうまくいった，畳み込み層 2 層のモデルの評価結果は，正解率はおそよ 55 から 60%と高い精度となっていないが，F 値はおよそ 68%となっており 7 割近い値となっていた．AUC は，約 60% と約 55% となっていた．

5.4 RQ2. 検出精度の考察

評価結果で得られた値は，GNNs を用いたフォールト予測の応用可能性を示唆する値となっている．F 値が 7 割近い値となっているのに対して，正解率が 6 割近い値となっていることは，本研究のモデルは全体の正解率は低いものの，正確に構造フォールトを持つプログラムと判断する能力は高いと言うことがわかる．提案モデルへの入力とする特徴量の種類が不十分であったことと，特徴量を変換させる word2vec の学習がうまくできていない可能性がある．提案モデルへの入力とする特徴量を 4.1 節で定義したが，この特徴量のみでのプログラムの構造フォールトを判断するのが難しい．word2vec を用いて各ノードの属性をベクトル化した時に未知語の場合，全ての値が 0 のベクトルに変換されてしまう．特徴量をベクトル化させる際に，ベクトル化がうまくできずにデータセットがかなり小さくなってしまった問題もある．これらの問題を解決する方法を考える必要がある．

5.5 RQ3. 適用可能性の考察

5.5.1 実験の妥当性

テストデータに含まれているフォールトパターンが学習データに含まれておらず，フォールトのパターンを網羅的に認識することができず，精度が低くなってしまったと考える．以上より，提案したアーキテクチャが実験で得られた精度の低さに影響していないと考える．

5.5.2 前提条件に関する妥当性

本研究で作成したデータセットは，プログラミング問題の解答から作成された Project CodeNet[9] を用いて，コンパイルが可能であり，単純に答えが合うプログラムかどうかでフォールトがあるかどうかで作成しており，フォールトデータセットとして提供されているデータセットを用いていない．Project CodeNet は，プログラミング問題の解答から作成されたデータセットであるので，実システムのプログラムとは異なっている．これらのことから，実システムを用いたプログラムの構造フォールトのデータセットを作成する必要がある．

特徴量をベクトル化させるのにデータセットの AST をコーパスとして学習させた word2vec を用いた．ベクトル化させるモデルを一般化させるために，大規模なコーパスで学習させたモデルを使用する必要があると考える．

本研究では 1 つのデータセットを用いてしか実験を行なっていない．RQ3. に答えるために，他の種類のフォールトデータセットを用いて実験を行う必要性がある．

6 おわりに

本研究では，GNNs を用いた静的プログラム解析の応用可能性を考察を目的とした．その目的を達成するために，GCN を用いたプログラムのフォールトを予測可能な FP-GAT モデルの提案を行なった．入力として AST を採用した．AST の構造を GNNs の構造として処理することができるので，GNNs を用いた．

実際に，従来の静的プログラム解析ツールの自動生成が可能かどうか調査するのに，GCN を用いた提案モデル，FP-GAT のプロトタイプを作成し，データセットに適用して実験を行なった．実験結果から GNNs を用いたフォールト検出器の生成が可能であると結論づけた．

グラフ構造を直接解釈実行する GRN を用いたモデルによるフォールト予測に関しては今後の課題とする．

参考文献

[1] Boris Beizer, et al., "*Bug taxonomy and statics*," Technical report, Softw. Eng. Mentor, 2630 Taastrup, 2001.

[2] Berkey Berabi, et al., "TFix: Learning to fix coding errors with a text-to-text transformer," 38th Int. Conf. Machine Learning, PMLR 139, 2021.

[3] Emin Borandag, "*Software Fault Prediction Using an RNN-Based Deep Learning Approach and Ensemble Machine Learning Techniques*," Appl. Sci., Jan. 2023.

[4] Shaked Broady, et al., "*How attentive are graph attention networks?*," Proceedings of ICLR, 2022.

[5] Ziniu Hu, et al., "*Gpt-gnn: generative pre-Training of graph neural networks*," Proceedings of KDD, pp. 1857-1867.

[6] javalang: `https://github.com/c2nes/javalang/blob/master/javalang/tree.py`

[7] Tim Menzies, et al., "*Defect prediction from static code features: Current results, limitations, new approaches*," Autom. Softw. Eng., Vol. 17, no.2, pp.375-407, Dec. 2010.

[8] Atsuyuki Nakao, et al., "*Exploring the quantum chemical energy landscape with GNN-guided artificial force*," American Chemical Society, J. Chem. Theory Comput., Jan. 2023, 19, 3, 713–717.

[9] Project CodeNet: `https://github.com/IBM/Project_CodeNet`

[10] Lucija Šikić, et al., "*Graph nueral networks for source code defect prediction*," IEEE Access, vol. 10, Jan. 2022, pp. 10402-10415.

[11] Kesu Wang, et al., "*Unified abstract syntax tree representation learning for cross-language program classification*," ICPC '22: Proceedings of the 30th IEEE/ACM Int. Conf. Program Comprehension, May 2022, pp. 390-400.

[12] Jie Zhou, et al., "*Graph neural networks: A review of methods and applications*," AI Open, vol. 1, 2020, pp. 57-81.