Tabriz Journal of Electrical Engineering (TJEE), vol. 55, no. 2, 2025 Serial no. 112
DOI: 10.22034/tjee.2024.57018.4645

Bug detection Using model transformations and deep learning

Leila Yousofvand [1], Seyfollah Soleimani[2]*, Sajad Esfandyari [3]

[1] Department of Computer Engineering, Faculty of Engineering, Lorestan University, Khorramabad, Iran
[2] Department of Computer Engineering, Faculty of Engineering, Arak University, Arak, Iran
[2] Department of Computer Engineering, Malayer University, Malayer, Iran
E-mails: yousofvand.l@lu.ac.ir; s-soleimani@araku.ac.ir; esfandyari@malayeru.ac.ir;
* Corresponding author

Abstract
In recent years, software bug detection techniques have become widely used by application developers to improve code quality and ensure robustness. One of the most popular approaches is static analysis, which relies on pattern-based methods. These techniques involve manually creating patterns by experts to identify known bugs. Despite the extensive efforts to create a large set of patterns for different bugs, there are still numerous bugs that slip through all the available filters. In response to this limitation, this paper introduces a new method for automatic bug detection specifically designed for JavaScript code. The proposed approach involves mapping both buggy and bug-free code to graph representations. These graphs are then fed into a deep learning model that has been trained to classify the code as either buggy or bug-free. The model's ability to work with graph data allows it to capture more complex relationships within the code. Furthermore, this approach can adapt to new, previously unseen bugs, enhancing its robustness. Evaluation results show that this method is able to detect a broader range of bugs and outperforms previous detection techniques in terms of accuracy and coverage.

Keywords
Bug detection, deep learning, graph classification, convolutional networks, abstract syntax tree (AST).

1- Short Introduction

Automatic bug detection techniques are widely used by developers. Although various bug detectors have been created and are widely used, there are many bugs that pass through all existing filters.This paper proposes a new learning-based method for detecting bugs in JavaScript applications. The main goal is to identify a wide range of programming errors, complex bugs that require adding or removing commands, which were not addressed in previous works. The approach uses the abstract syntax tree to extract an equivalent graph for each input code. A machine learning model is then trained to predict the correct label using the graph structure and its node characteristics. The method can predict labels for previously unseen graphs.

2- Proposed Work and Methodology (including comprision, simulation/experimental results and discusion)

In the proposed method, we model the bug detection problem as a graph classification problem. First, we convert the input program code into a graph according to the abstract syntax tree, and then we use the graph classification method and deep learning to classify buggy graphs and bug-free graphs. In this article, Patchy-SAN method is used. It is used for graph classification. The general goal of this method is to learn a function from a set of graphs that gives us a general representation of graphs. These representations can then be used for any operation such as graph classification. We implemented the proposed method in this research with Python programming language and tested it on the dataset of JavaScript codes. The results show that we have achieved an accuracy of 73% for 96268 samples. Compared to the previous works, the proposed method detects a wider range of bugs, while in the previous methods, the bugs have been evaluated separately, and in most of the methods, the bugs are specifically in the datasets and codebases and specific codes have been examined while the proposed method has been tested on the codes available in GitHub.

3- Conclusion

In this paper, we have presented a solution based on deep learning on graphs for the problem of detecting bugs in programming codes. Graphs are rich and flexible structures that can accurately represent the interaction between different parts of many complex systems. In this paper, we have modeled the bug detection problem as a graph classification problem. Comparing the results of implementation and execution of this method shows that the proposed method detects a wider range of bugs compared to previous works. In the future, we can consider the combination of graph node classification methods with graph edge classification methods to detect the location of bug.

کشف خودکار باگ‌های برنامه با استفاده از تبدیلات مدل و یادگیری عمیق

لیلا یوسفوند
استادیار، گروه مهندسی کامپیوتر، دانشگاه لرستان، خرم آباد، ایران

سیف اله سلیمانی
دانشیار، گروه مهندسی کامپیوتر، دانشگاه اراک، اراک، ایران

سجاد اسفندیاری
استادیار، گروه مهندسی کامپیوتر، دانشگاه ملایر، ملایر، ایران.

چکیده
در سال‌های اخیر، تکنیک‌های شناسایی باگ نرم‌افزاری به‌طور گسترده‌ای توسط توسعه‌دهندگان برنامه‌ها مورد استفاده قرار گرفته‌اند تا کیفیت کد را بهبود بخشیده و از پایداری آن اطمینان حاصل کنند. یکی از محبوب‌ترین روش‌ها، تحلیل ایستا (Static Analysis) است که بر پایه‌ی الگو عمل می‌کند. در این تکنیک‌ها، الگوهای شناسایی باگ به‌صورت دستی توسط کارشناسان طراحی می‌شوند تا خطاهای شناخته‌شده را تشخیص دهند. با وجود تلاش‌های گسترده برای ایجاد مجموعه‌ای بزرگ از الگوها برای انواع مختلف باگ‌ها، هنوز هم باگ‌های زیادی وجود دارند که از تمام فیلترهای موجود عبور می‌کنند و شناسایی نمی‌شوند.
در پاسخ به این محدودیت، این مقاله روشی جدید برای شناسایی خودکار باگ در کدهای جاوااسکریپت ارائه می‌دهد. رویکرد پیشنهادی شامل نگاشت کدهای دارای باگ و بدون باگ به ساختارهای گرافی است. سپس این گراف‌ها به یک مدل یادگیری عمیق داده می‌شوند که برای طبقه‌بندی کدها به دو دسته‌ی دارای باگ و بدون باگ آموزش دیده است. توانایی مدل در کار با داده‌های گرافی به آن امکان می‌دهد تا روابط پیچیده‌تری را در ساختار کد تشخیص دهد. علاوه بر این، این رویکرد قابلیت سازگاری با باگ‌های جدید و ناشناخته را نیز دارد که باعث افزایش پایداری و اثربخشی آن می‌شود. نتایج ارزیابی نشان می‌دهند که این روش می‌تواند دامنه‌ی وسیع‌تری از باگ‌ها را شناسایی کند و از نظر دقت و پوشش عملکرد بهتری نسبت به روش‌های پیشین دارد.

کلمات کلیدی
کشف باگ، یادگیری عمیق، طبقه‌بندی گراف‌ها، شبکه‌های کانولوشن، درخت خلاصه نحوی (AST).

نام نویسنده مسئول: دکتر سیف اله سلیمانی
ایمیل نویسنده مسئول: s-soleimani@araku.ac.ir

تاریخ ارسال مقاله: ۱۴۰۲/۰۳/۱۷
تاریخ (های) اصلاح مقاله: ۱۴۰۲/۰۷/۰۷
تاریخ پذیرش مقاله: ۱۴۰۳/۰۵/۱۱

۱- مقدمه

تکنیک‌های تشخیص باگ خودکار به‌طور گسترده توسط توسعه‌دهندگان مورداستفاده قرار می‌گیرد. در سال‌های اخیر، توسعه‌دهندگان به دلیل افزایش اندازه و پیچیدگی کدها از ابزارهای خودکار برای شناسایی و رفع باگ‌ها استفاده می‌کنند. به‌عنوان‌مثال می‌توان به Tricorder [۱] در گوگل، Getafix [۲] و Zoncolan در فیس بوک و IntelliCode در مایکروسافت ویژوال استودیو اشاره کرد.

تکنیک‌های اساسی این ابزارها را می‌توان به‌طورکلی به دو دسته طبقه‌بندی کرد: تکنیک‌های مبتنی بر قاعده[1] مانند [۱] و تکنیک‌های مبتنی بر داده[2] مانند [۳ , ۴]. تکنیک‌های دسته اول از قوانینی که به‌صورت دستی نوشته‌شده‌اند برای استخراج الگوهای نادرست کد استفاده می‌کنند و کل کد را برای این دسته از باگ‌ها اسکن می‌کند. یکی از مشهورترین تکنیک‌های مبتنی بر قاعده، تجزیه‌وتحلیل ایستا است که نمونه‌هایی از الگوهای شناخته‌شده باگ را جستجو می‌کند. معمولاً، این تجزیه‌وتحلیل‌ها به‌عنوان بخشی از یک چارچوب ایجاد می‌شوند. این چارچوب‌ها مجموعه‌ای از الگوهای باگ یا قوانین را پشتیبانی می‌کنند مانند چارچوب‌های Google Error Prone [۵] و Find-Bugs [۶]. هرکدام این چارچوب‌ها گاهی اوقات شامل ده‌ها و یا حتی صدها الگو هستند که این الگوها توسط افراد خبره نوشته و تنظیم‌شده‌اند. تکنیک‌های دسته دوم یاد می‌گیرند که چگونه کد غیرعادی را از یک مجموعه بزرگی از کدها با استفاده از شبکه‌های عصبی عمیق تشخیص دهند. باوجوداین که پیشرفت‌های بزرگی در این زمینه به‌دست‌آمده، اما هر دو نوع ابزار به‌طورکلی محدود هستند زیرا الگوهای باگ را در دیتابیس‌های خاصی هدف

[1] Rule-based techniques
[2] Data-based techniques

قرار می‌دهند یا انواع خاصی از باگ‌ها را هدف قرار می‌دهند. به‌عنوان‌مثال، قوانین زونکولن (Zoncolan) به‌گونه‌ای طراحی‌شده‌اند که فقط برای پایگاه داده‌هایی که کدهای فیس بوک را نگهداری می‌کنند قابل‌استفاده است و همچنین مدل‌های یادگیری عمیق باگ‌های خاصی را هدف قرار می‌دهند [۷] (مثلاً در نام‌گذاری متغیرها [۳] یا عبارات باینری [۴]).

باوجوداینکه آشکارسازهای باگ (باگ یاب‌ها) مختلفی ایجادشده و به‌طور گسترده مورداستفاده قرارگرفته‌اند، اما باگ‌های بسیاری وجود دارد که از همه فیلترهای موجود عبور می‌کنند. یک دلیل اصلی برای عدم تشخیص بسیاری از باگ‌ها این است که ساخت یک آشکارساز باگ به‌صورت دستی مشکل است. چالش خاصی که در این زمینه وجود دارد این است که هر آشکارساز باگ باید با دقت تنظیم شود و با استفاده از روش‌های اکتشافی توسعه یابد تا بتواند به‌صورت عملی مورداستفاده قرار گیرد. به‌عنوان‌مثال، یک آشکارساز باگ که اکنون بخشی از چارچوب Google Error Prone است، همراه با روش‌های اکتشافی مختلفی بکار گرفته‌شده تا بتواند تعداد باگ‌های بیشتری را شناسایی کند [۸]. ما برای پاسخ به این چالش و تشخیص باگ‌های مختلف، یادگیری ماشین را بکار گرفته‌ایم. اگرچه یادگیری ماشین در زمینه رفع مشکلات مختلف توسعه نرم‌افزار کمک کرده است [۹، ۱۰] اما مسئله تشخیص باگ مبتنی بر یادگیری همچنان یک چالش باز است [۴]. یک دلیل اصلی برای چالش عدم وجود آشکارسازهای باگ مبتنی بر یادگیری این است که یادگیری مؤثر ماشین به مقدار زیادی داده آموزش نیاز دارد. برای آموزش مدلی که کد باگ دار را شناسایی کند، تکنیک‌های یادگیری به نمونه‌های زیادی معمولاً هزاران یا حتی میلیون‌ها نمونه کد درست و نادرست نیاز دارند؛ بااین‌حال، اکثر کدهای موجود صحیح هستند و یا دقیقاً مشخص نیست که کدام قسمت از آن نادرست است. درنتیجه، آشکارسازهای باگ موجود که اطلاعات مربوط به کد را استنباط می‌کنند عملیات یادگیری را فقط از نمونه صحیح انجام می‌دهند [۱۱] و سپس کدی را که غیرمعمول است به‌عنوان احتمالاً نادرست علامت‌گذاری می‌کنند و یا تناقضات را در یک برنامه واحد جستجو می‌کنند [۱۲]. یک رویکرد تشخیص باگ مبتنی بر مدل برای برنامه‌های یادگیری عمیق، با استفاده از متا مدل‌سازی و تبدیلات گراف در [۱۳] ارائه‌شده است. در [۱۴] عملیات تشخیص باگ و تولید تصحیح برای باگ را در نظر گرفته‌اند و تصحیح‌های پیش‌بینی‌شده برای تشخیص باگ دار بودن و یا بدون باگ بودن استفاده کرده‌اند.

در این مقاله، ما یک روش جدید مبتنی بر یادگیری برای یافتن خودکار باگ در برنامه‌های جاوا اسکریپت پیشنهاد می‌دهیم. جاوا اسکریپت یک زبان اسکریپت نویسی است که برای توسعه برنامه‌های وب طراحی‌شده است و در ۹ سال گذشته محبوب‌ترین زبان برنامه‌نویسی در گیت‌هاب بوده است. تشخیص باگ در کد جاوا اسکریپت یک چالش منحصربه‌فرد است زیرا باگ‌ها به دلیل ویژگی‌های غیرمعمول زبان و عدم پشتیبانی ابزار به شکل‌های مختلفی ظاهر می‌شوند؛ بنابراین، هدف اصلی رویکرد ما تشخیص طیف گسترده‌ای از خطاهای برنامه‌نویسی، مانند باگ‌های عملکردی، استفاده از عملوندها یا شناسه‌های اشتباه، دسترسی به ویژگی‌های تعریف‌نشده، مدیریت نادرست دامنه‌های متغیر، ناسازگارهای نوع و غیره است. همچنین رویکرد ما توانایی شناسایی باگ‌های پیچیده‌تر و معنایی تری را دارد یعنی باگ‌هایی که نیاز به افزودن یا حذف دستورات از یک برنامه دارند. این نوع باگ‌ها در کارهای قبلی در نظر گرفته نشدند. در این پژوهش، ما با تکنیکی که هم از کد صحیح و هم از باگ دار یاد می‌گیرد، به مشکل تشخیص خودکار کد باگ دار می‌پردازیم. در روش پیشنهادی ابتدا برای هر کد ورودی، گراف معادل آن را از روی درخت خلاصه نحوی استخراج می‌کنیم با این هدف که مدل یادگیری ماشینی را آموزش دهیم که از ساختار گراف داده‌ها و ویژگی‌های موجود گره‌های گراف، برای پیش‌بینی برچسب صحیح گرافی که قبلاً دیده نشده، استفاده کند. ما در این روش متد طبقه‌بندی گراف‌ها را برای پیش‌بینی برچسب صحیح به کار گرفته‌ایم.

در ادامه مطالب این مقاله به این صورت سازماندهی‌شده است، در بخش ۲ پیش‌زمینه لازم، ازجمله مطالبی درباره باگ‌های موجود در کدهای جاوا اسکریپت، درخت خلاصه نحوی، شبکه‌های عصبی پیچشی و شبکه‌های پیچشی گراف ارائه می‌دهیم. در بخش ۳ به توضیح کامل روش ارائه‌شده می‌پردازیم. در بخش ۴، روش پیشنهادی را مورد آزمایش قرارداده و نتایج حاصله را ارائه می‌دهیم و در پایان، در بخش ۵ نتیجه‌گیری و همچنین پیشنهادهایی را ذکر می‌کنیم.

۲- ادبیات موضوعی

در این بخش پیش‌زمینه لازم ارائه‌شده است. ما مفاهیم زیر را توضیح داده‌ایم: باگ در کدهای جاوا اسکریپت، درخت خلاصه نحوی، شبکه عصبی کانولوشن، شبکه کانولوشن گراف و طبقه‌بندی گراف‌ها.

۲-۱- باگ در کدهای جاوا اسکریپت

زبان برنامه‌نویسی جاوا اسکریپت به‌طور واقعی زبان برنامه‌نویسی وب در سطح جهانی و بیشترین زبان مورداستفاده در گیت‌هاب است. جاوا اسکریپت به دلیل سرعت پاسخگویی بالا و سادگی، به‌طور گسترده در برنامه‌های وب سمت مشتری مورداستفاده قرار می‌گیرد. در سال‌های اخیر، این زبان به دلیل انعطاف‌پذیری بالا، به‌طور فزاینده‌ای برای توسعه برنامه‌های وب سمت سرور نیز مورداستفاده قرارگرفته است که این امر منجر به ایجاد برنامه‌های وب کاملی با استفاده از این زبان شده است [۱۵]. بسترهای نرم‌افزاری مانند Node.js[3] به توسعه‌دهندگان این امکان را می‌دهد که سمت کاربر و سمت مشتری برنامه‌ها را کاملاً در جاوا اسکریپت توسعه دهند. علیرغم محبوبیت فراوان، خصوصیات ذاتی جاوا اسکریپت مانند تایپ ضعیف، ارزیابی زمان اجرا و تابع eval این زبان را به یکی از زبان‌های مستعد خطاهای برنامه‌نویسی تبدیل کرده است. تابع eval یک رشته را به‌عنوان قطعه کد تفسیر و اجرا می‌کند و به‌عنوان منبع اصلی اشکالات و آسیب‌پذیری‌ها شناخته‌شده است. یک نمونه از باگ‌های موجود در این زبان باگ‌های بزرگ نوشتن حروف است. جاوا اسکریپت به حروف کوچک و بزرگ حساس است. این بدان معناست که متغیرهایی که ایجاد می‌شوند باید هر بار که از آن‌ها استفاده می‌شود با همان صورت که ایجادشده‌اند مورداستفاده قرار گیرند. همچنین به این معناست که برای درست عمل کردن برنامه، حروف در نام توابع باید به‌درستی به‌صورت بزرگ نوشته شوند. یکی از متداول‌ترین مکان‌ها برای مشاهده این نوع باگ، استفاده از متد innerHTML از شیء Document است؛ که معمولاً آن را به نادرست به‌صورت InnerHTML می‌نویسند. در شکل ۱ مثالی از کاربرد نادرست این متد آورده شده است.

```
function clearEmployeeListOnLinkClick(){
document.querySelector("a").addEventListener("click",
function(event){
document.querySelector("ul").InnerHTML = ""; }
);  }
```

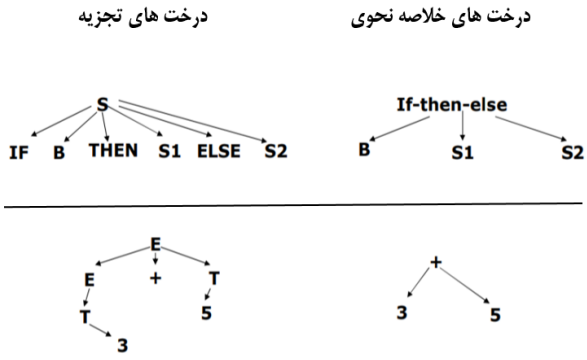
شکل۱- InnerHTML باید به innerHTML تبدیل شود.

[3] https://nodejs.org/en/

۲-۲- درخت خلاصه نحوی

درخت خلاصه نحوی[۴] درخت تجزیه‌ای است که در آن ترم‌های میانی حذف‌شده‌اند. درواقع شکل خلاصه‌شده‌ای از درخت تجزیه است؛ درخت تجزیه درختی است که نشان‌دهنده ساختار دستوری (نحوی) یک رشته است. معمولاً این نمایش با توجه به دستور زبان‌های رسمی است. همان‌طور که در شکل ۲ می‌بینیم در یک درخت تجزیه گره‌های داخلی، نقش‌های دستوری (غیر پایانه‌ها) و برگ‌ها یا همان گره‌های خارجی کلمات مربوط به آن نقش هستند. برای تبدیل درخت تجزیه به درخت خلاصه نحوی باید به‌صورت زیر عمل نماییم:

- پرانتزها که به‌عنوان جداکننده و نشانگر اولویت هستند را از درخت تجزیه حذف می‌کنیم .
- تک فرزندها را جایگزین پدرشان می‌نماییم .
- ترم‌های میانی باقیمانده را با عملگرهایی جایگزین می‌کنیم که فرزند آن ترم باشد.



شکل ۲- درخت‌های تجزیه و درخت‌های خلاصه نحوی معادل

۲-۳- شبکه عصبی کانولوشن

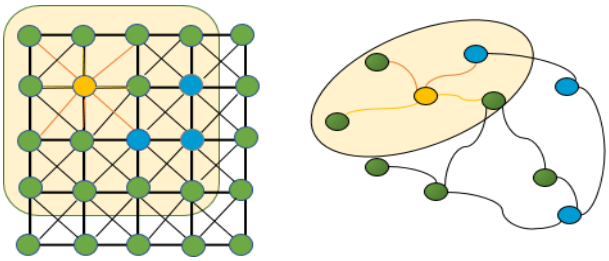
در طول دهه گذشته، شبکه‌های عصبی پیچشی (کانولوشن)[۵] (CNN) به یک استاندارد عملی برای عملیات مختلف بینایی کامپیوتر و یادگیری ماشین تبدیل‌شده‌اند. این شبکه عصبی ورودی را به قطعات کوچک‌تر تقسیم کرده و سپس استخراج ویژگی را انجام می‌دهند؛ به‌عنوان‌مثال در مسئله طبقه‌بندی به این صورت عمل می‌کند که بخش‌های مهمی از ورودی را استخراج می‌کند که می‌تواند برای تصمیم‌گیری استفاده شود.

CNNها شامل تعدادی لایه کانولوشن[۶] و لایه تجمیع[۷] هستند. لایه‌های کانولوشن یک فیلتر را از روی تصویر منبع عبور می‌دهند و اطلاعات مهم را از هر قطعه استخراج می‌کنند [۱۶]. لایه‌های تجمیع اطلاعات استخراج‌شده را نمونه‌برداری می‌کنند و فقط مهم‌ترین اطلاعات را در خود نگه می‌دارند. وقتی داده‌های اساسی استخراج می‌شوند، از یک لایه کاملاً متصل عبور داده می‌شوند تا به تصمیم نهایی طبقه‌بندی برسند.

CNNهای دوبعدی عمیق با بسیاری از لایه‌های پنهان و میلیون‌ها پارامتر توانایی یادگیری آبجکت‌های پیچیده و الگوهای پیچیده را دارند به این شرط که بتوانیم آن‌ها را روی یک پایگاه داده با تعداد داده‌های زیاد و با برچسب‌های حقیقی آموزش دهیم. با یک آموزش مناسب، این توانایی منحصربه‌فرد شبکه‌های عصبی پیچشی آن‌ها را به ابزاری اصلی برای کاربردهای مختلف مهندسی برای سیگنال‌های دوبعدی مانند تصاویر و فریم‌های ویدیویی تبدیل می‌کند. بااین‌حال، شبکه‌ها پیچشی دوبعدی ممکن است در برنامه‌های که از سیگنال‌های یک‌بعدی استفاده می‌کنند و همچنین هنگامی‌که داده‌های آموزش کم هستند گزینه مناسبی نباشند. برای حل این مسئله، اخیراً شبکه‌های عصبی پیچشی یک‌بعدی پیشنهادشده‌اند. این شبکه‌ها در مسائلی چون طبقه‌بندی داده‌های پزشکی و تشخیص زودهنگام بیماری، تشخیص و شناسایی خطاها در برق الکترونیک و تشخیص خطای موتور به عملکرد بالایی دست‌یافته‌اند

۲-۴- شبکه کانولوشن گراف

در علوم کامپیوتر، گراف ساختمان داده‌ای است که از گره‌ها و یال‌ها تشکیل‌شده است. گراف‌ها ساختارهای بسیار مفیدی هستند که می‌توانند برای مدل‌سازی پدیده‌های دنیای واقعی استفاده شوند مانند شبکه‌های اجتماعی، ساختارهای مولکولی، ساختارهای معنایی، مدل‌های جغرافیایی یا فیزیکی [۱۷]. همچنین اخیراً از این ساختمان داده برای مدل‌سازی کدهای برنامه‌نویسی و تشخیص خطاهای برنامه‌نویسی استفاده می‌شود. یک شبکه عصبی کانولوشنال گراف[۸] که به آن GCN [۱۸] نیز گفته می‌شود، به‌جای اینکه عملیات کانولوشن را روی یک تصویر انجام دهد، این عملیات را روی یک گراف انجام می‌دهد. همان‌طور که در شکل ۳ مشاهده می‌شود دقیقاً مشابه CNN که قصد دارد مهم‌ترین اطلاعات را از تصاویر برای طبقه‌بندی تصویر استخراج کند، GCN نیز با عبور یک فیلتر از روی گراف، به دنبال گره‌ها و یال‌های اساسی است که می‌تواند به طبقه‌بندی گره‌ها در گراف کمک کند.



شکل ۳- کانولوشن دوبعدی در مقابل کانولوشن گراف [۱۵]

۲-۵- طبقه‌بندی گراف

یادگیری با نظارت اصلی‌ترین مسئله در پردازش داده‌های گراف است. در طبقه‌بندی گراف‌ها فرض بر این است که مقادیر هدف تعداد مشخصی از گراف‌ها یا قسمت خاصی از گراف به‌عنوان مجموعه داده‌های آموزشی در دسترس است و هدف استخراج مقادیر سایر گراف‌ها یا قسمت باقیمانده از گراف است. کاربردهای طبقه‌بندی گراف‌ها بسیار زیاد است ازجمله تعیین آنزیمی بودن یا نبودن پروتئین در بیوانفورماتیک، طبقه‌بندی اسناد در پردازش زبان طبیعی (NLP)، تحلیل شبکه‌های اجتماعی.

عملیات کانولوشن بر روی گراف‌ها در ۳ دامنه تعریف‌شده است: دامنه طیفی، دامنه فضایی و دامنه جاسازی. روش ارائه‌شده توسط نیپرت و همکاران با نام Patchy-SAN [۱۹] در دسته الگوریتم‌های دامنه فضایی قرار می‌گیرد. در این مقاله از روش Patchy-SAN برای طبقه‌بندی گراف استفاده‌شده است. هدف

[4] Abstract Syntax Tree
[5] Convolutional Neural Networks
[6] Convolution layer
[7] Pooling layer
[8] Graph Convolutional Networks

کلی این روش یادگیری تابعی از روی دسته‌ای از گراف‌ها است که به ما یک نمایش کلی از گراف‌ها می‌دهد. سپس می‌توان از این نمایش‌ها برای هر عملیاتی مانند طبقه‌بندی گراف‌ها استفاده کرد.

چالش‌های اعمال کانولوشن به‌طور مستقیم بر روی گراف‌ها شامل دو مشکل اصلی است که باید قبل از اعمال کانولوشن روی گراف‌ها به آن‌ها پرداخت.

- چالش ۱: تصاویر را می‌توان به‌عنوان یک گراف معمولی در نظر گرفت که در آن هر پیکسل با یک گره مشخص می‌شود و دو پیکسل مجاور در تصویر می‌توانند توسط یک لبه به هم متصل شوند. به‌این‌ترتیب، می‌توانیم گراف منظم روی جملات داشته باشیم. داده‌های این حوزه‌ها (تصاویر، متن، گفتار) دارای نظم مکانی هستند. منظور ما از نظم مکانی این است که با توجه به یک تصویر، پیکسل اول، پیکسل دوم و غیره وجود دارد، یعنی ترتیب منحصربه‌فردی دارد که در آن پیکسل‌ها پردازش می‌شوند. بااین‌حال، این مورد در مورد داده‌های ساختاریافته گراف نیست. توالی که در آن گره‌ها پردازش می‌شوند منحصربه‌فرد نیست.
- چالش ۲: داده‌های تصویری یا داده‌های متنی (این داده‌های ساختاریافته به‌عنوان داده‌های حوزه اقلیدسی نیز نامیده می‌شوند)، دارای چهار همسایه مشخص هستند - پیکسل چپ، پیکسل راست، پیکسل بالا و پیکسل پایین. ازاین‌رو، می‌توانیم یک فیلتر اندازه ثابت برای همه پیکسل‌ها صرف‌نظر از جایی که در تصویر رخ می‌دهد، داشته باشیم. بااین‌حال، یک گره در گراف می‌تواند همسایه‌های زیادی داشته باشد که از ۱ تا حداکثر درجه گراف را شامل می‌شوند و ازاین‌رو هیچ همسایه سمت چپ یا همسایه بالایی یک گره وجود ندارد. به دلیل متفاوت بودن تعداد همسایگان، نمی‌توانیم از یک فیلتر اندازه ثابت برای همه گره‌ها استفاده کنیم.

Patchy-SAN چارچوبی برای یادگیری نمایش گراف است، یعنی برای یادگیری توابعی که گراف را به نمایش‌های برداری نگاشت می‌کنند. عمدتاً برای مسئله طبقه‌بندی گراف استفاده می‌شود.Patchy-SAN برای هر گراف ورودی ابتدا دنباله‌ای از گره‌ها را بر اساس یک روش برچسب‌گذاری گراف [۲۰] تعیین می‌کند. سپس برای هر گره در این دنباله، همسایگی از دقیقاً k گره استخراج و نرمال‌سازی می‌شود. سپس تعداد ثابتی از همسایگان مرتب‌شده را انتخاب نموده و یک شبکه عصبی کانولوشنال را برای طبقه‌بندی گراف ورودی اعمال می‌کند.

مرحله ۱: برچسب‌گذاری گراف و انتخاب گره

نیپرت و همکاران با استفاده از الگوریتم برچسب‌گذاری گراف، به چالش ۱ یعنی نداشتن ترتیب گره‌های منحصربه‌فرد می‌پردازند. به هر گره برچسب اختصاص می‌دهد به‌گونه‌ای که وقتی گره‌ها را بر اساس این برچسب مرتب می‌کنیم، مجموعه‌ای از گره‌های مرتب‌شده منحصربه‌فرد را دریافت می‌کنیم. ازنظر ریاضی یک تابع است، یعنی مجموعه‌ای از گره‌ها را می‌گیرد و مجموعه‌ای مرتب از گره‌ها را در خروجی می‌دهد. الگوریتم برچسب‌گذاری گراف می‌تولند درجه گره باشد که در آن گره‌هایی با درجه یکسان به یک برچسب اختصاص داده می‌شوند. Patchy-SAN از الگوریتم رنگ‌آمیزی گراف به نام الگوریتم Weisfeiler-Lehman (WL) [۲۰] برای برچسب‌گذاری گراف استفاده می‌کند. به‌عنوان‌مثال، در شکل ۴، مجموع N گره در گراف ورودی وجود دارد، پس از اعمال الگوریتم برچسب‌گذاری گراف، مجموعه‌ای مرتب از گره‌ها را دریافت می‌کنیم. این مجموعه با یک آرایه نمایش داده می‌شود.

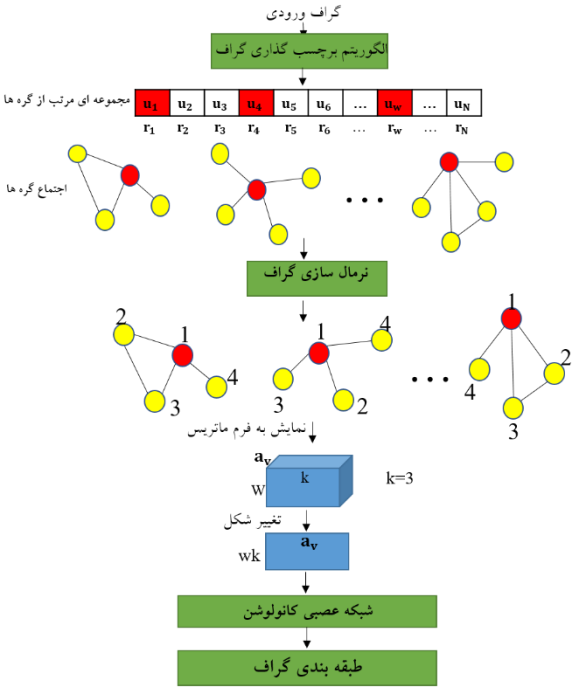
مرحله ۲: اجتماع گره

این الگوریتم زیر گراف همسایگی فقط w گره انتخاب‌شده را برای پردازش بیشتر در نظر می‌گیرد. همسایه‌ها با استفاده از الگوریتم جستجوی سطح اول انتخاب می‌شوند.

مرحله ۳: نرمال‌سازی گراف

در تصاویر، هر پیکسل تعداد همسایگان ثابتی دارد و ازاین‌رو، اندازه فیلتر ثابت است. برای استفاده از مفهوم مشابه در گراف‌ها و پرداختن به چالش ۲، نویسندگان تعداد گره‌های همسایگی در نظر گرفته‌شده برای پردازش را به مقدار k که به‌عنوان فیلد دریافتی نامیده می‌شود، محدود می‌کنند.

اگر گره‌های همسایگی بیشتر از k باشند، رتبه‌بندی نسبی همسایه‌ها در گره مرکزی برای حذف همسایه‌های اضافی در نظر گرفته می‌شود. به‌عنوان‌مثال، در شکل ۴، فیلد دریافتی 3 k = است و زیرگراف ۴ همسایه دارد، بااین‌حال، ما فقط ۳ همسایه بالا را که نزدیک‌ترین به گره ریشه هستند را نگه‌داری می‌کنیم. تعریف نزدیکی به این صورت است: برای یک گره مرکزی معین u و دو گره همسایه vi و vj، اگر dist(u,vi)<dist(u,vj) درنتیجه rank(vi)>rank(vj) است و vi در نظر گرفته می‌شود و vj در نظر گرفته نمی‌شود.



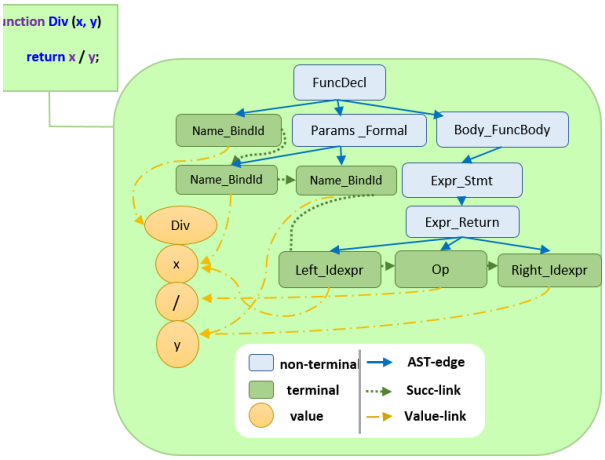
شکل ۴- معماری Patchy-SAN

در پایان ما w گره داریم که هر گره k همسایه دارد. گراف تبدیل‌شده ما به شکل یک تنسور (w, k, av) است که w تعداد گره‌های انتخاب‌شده برای گراف است و k تعداد ثابت همسایه‌های هر گره انتخاب‌شده است و av بردار ویژگی گره است. بقیه معماری را می‌توان به‌صورت اختیاری انتخاب کرد؛ و از لایه‌های کانولوشن استفاده کرد و طبقه‌بندی با نظارت را اعمال نمود. شکل ۴ معماری Patchy-SAN را نشان می‌دهد.

۳- روش پیشنهادی

برنامه‌هایی که به زبان‌های سطح بالا نوشته می‌شوند دارای ساختارهای مفیدی هستند. محققان برای استخراج این ساختار، نمایش‌های مبتنی بر گراف را ارائه داده‌اند [۳].

ابتدا کد منبع (سورس کد) برنامه را به‌عنوان گراف نشان می‌دهیم و از انواع لبه‌های مختلف برای مدل‌سازی روابط نحوی بین نشانه‌های مختلف استفاده می‌کنیم. قسمت اصلی گراف برنامه، درخت خلاصه نحوی (AST) است که ساختار نحوی برنامه را مشخص می‌کند. این درخت از گره‌های نحوی (مربوط به ترم‌های میانی در دستور زبان برنامه‌نویسی) و توکن‌های نحوی (مربوط به پایانه‌ها) تشکیل‌شده است. همان‌طور که در شکل ۵ مشاهده می‌کنیم گره‌های نحوی را از دستور زبان برنامه با نام غیر پایانه برچسب‌گذاری می‌کنیم، درحالی‌که توکن‌های نحوی با رشته‌ای که آن‌ها نشان می‌دهند برچسب‌گذاری می‌شوند. ما از لبه‌های AST edge برای اتصال گره‌ها مطابق با AST استفاده می‌کنیم. ازآنجاکه این امر نظم و ترتیب گره‌های نحوی را به فرزندان القا نمی‌کند، ما علاوه بر این یال‌ها، یال‌های Succ Token را که هر توکن نحوی را به جانشین آن متصل می‌کند را اضافه می‌کنیم. درواقع گره‌های برگ را به هم متصل می‌کنیم. همچنین گره‌هایی با مقدار اضافه می‌کنیم که محتوای واقعی گره‌های برگ را ذخیره می‌کنند. سپس گره‌های برگ با یال Value Link به مقادیر واقعی متصل می‌کنیم.

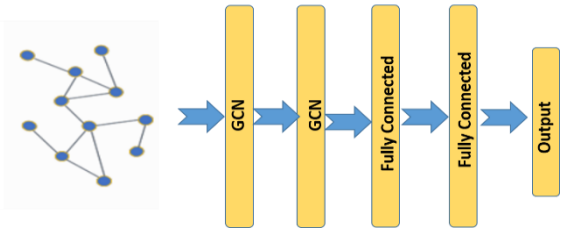


شکل ۵- یک نمونه کد برنامه و گراف معادل آن

در روش پیشنهادی، ما مسئله کشف باگ را به‌عنوان یک مسئله طبقه‌بندی گراف‌ها مدل می‌کنیم. پس از تبدیل کد برنامه ورودی به گراف از متد طبقه‌بندی گراف‌ها و یادگیری عمیق و روش Patchy-SAN ذکرشده در بخش قبل برای طبقه‌بندی گراف‌های باگ دار و گراف‌های بدون باگ استفاده می‌کنیم.

۴- ارزیابی

روش پیشنهادی در این پژوهش را با زبان برنامه‌نویسی پایتون پیاده‌سازی کرده و روی دیتاست کدهای جاوا اسکریت [۱۴] مورد آزمایش قراردادیم. برای این کار از ۳۲۰۸۹۳ نمونه‌ی مجموعه داده ذکرشده استفاده نموده‌ایم که از بین آن‌ها ۲۲۴۶۲۵ نمونه برای آموزش و ۹۶۲۶۸ نمونه برای تست (۷۰ درصد از نمونه‌ها برای آموزش و ۳۰ درصد برای تست) استفاده‌شده است.. بخش آموزش مجموعه داده شامل ۱۱۲۱۹۷ کد باگ دار و ۱۱۲۴۲۸ کد بدون باگ است. علاوه بر این، بخش تست مجموعه داده شامل ۴۸۰۸۴ کد باگ دار و ۴۸۱۸۴ کد بدون باگ است. پیکربندی‌های مختلف با لایه‌های مختلف برای مدل پیشنهادی آزمایش شد و به این نتیجه رسیدیم که هایپرپارامترهای زیر می‌توانند بهترین دقت را برای این مسئله ارائه دهند. شکل ۶ پیکربندی نهایی مدل پیشنهادی را نشان می‌دهد. دو لایه کانولوشن گراف به ترتیب با ۱۶ و ۸ واحد برای هر لایه پنهان استفاده‌شده است. سپس دو لایه کاملاً متصل و لایه Dropout با نرخ.5 استفاده‌شده است. تعداد دوره‌های آموزشی ۱۵۰ دوره در نظر گرفته شد. آنتروپی متقاطع باینری نیز به‌عنوان تابع ضرر استفاده شد و شبکه توسط بهینه‌ساز Adam آموزش داده شد. ما روش پیشنهادی را روی سیستمی با مشخصات RAM 12 GB و CPU corei7 و GEFORCE 920MX اجرا نموده‌ایم.



شکل ۶- پیکربندی مدل پیشنهادی

برای ارزیابی نتایج از معیار ارزیابی ماتریس درهم‌ریختگی[9] (CM) استفاده‌شده است. در این ماتریس مؤلفه CM_{ij} بیانگر این است که چند عنصر از کلاس i ام به‌عنوان عضوی از دسته j برچسب خورده است. جدول ۱ این ماتریس را نشان می‌دهد. فرمول دقت و سایر معیارها بر اساس این ماتریس نیز در جدول۲ نشان داده‌شده است.

جدول ۱- جدول درهم‌ریختگی

		کلاس پیش‌بینی‌شده	
		بدون باگ	باگ دار
کلاس واقعی	باگ دار	FN	TP
	بدون باگ	TN	FP

جدول ۲- معیارهای ارزیابی

$accuracy = \frac{TP+TN}{TP+TN+FP+FN}$	$Recall = \frac{TP}{TP+FN}$
$Precision = \frac{TP}{TP+FP}$	$F1 = \frac{2}{\frac{1}{Recall}+\frac{1}{Precision}}$

نتایج نشان می‌دهد که ما به دقت ۷۹.۲۲٪ برای ۶۴۰۰۰ داده به دقت ۷۰..۷۳٪ برای ۹۶۲۶۸ دست‌یافته‌ایم. در گذشته فقط در یک کار [۱۴] (Hoppity) از این دیتاست استفاده‌شده و آن‌ها دقت ۴۷.۸۱٪ را روی ۶۴۰۰۰

[9] Confusion Matrix

داده و دقت ۳۸.۵۰٪ را روی ۹۶۲۶۸ داده کسب کرده‌اند. در جدول ۳ روش پیشنهادی با این روش مقایسه شده است.

جدول۳- مقایسه روش پیشنهادی با Hoppity –حداکثر تعداد گره ۵۰۰

حجم دیتاست (نمونه)	دقت (٪)	روش
٦٤٠٠٠	٤٧.٨١	Hoppity
٦٤٠٠٠	٧٩.٢٢	روش پیشنهادی
٩٦٢٦٨	٣٨.٥٠	Hoppity
٩٦٢٦٨	٧٣.٧٠	روش پیشنهادی

نتایج ارزیابی مدل ما بر اساس CM بر روی داده‌های تست در جدول ۴ نشان داده‌شده است. همان‌طور که در این جدول مشاهده می‌شود، درصد مثبت واقعی (کدهای باگ داری که به‌درستی به‌عنوان باگ دار پیش‌بینی‌شده‌اند) ۰.۵۹ و منفی واقعی (کدهای بدون باگ که به‌درستی به‌عنوان بدون باگ پیش‌بینی می‌شوند) ۰.۸۳ است درحالی‌که درصد مثبت کاذب و درصد منفی کاذب به ترتیب ۰.۱۷ و ۰.۴۱ هستند. این نتایج نشان می‌دهد که روش پیشنهادی با پارامترهای تعیین‌شده بسیار امیدوارکننده است.

جدول ۴- نتایج پیش‌بینی با حداکثر تعداد گره ۵۰۰

	کلاس پیش‌بینی‌شده		
		باگ دار	بدون باگ
کلاس واقعی	باگ دار	۰.۵۹	۰.٤١
	بدون باگ	۰.۱۷	۰.۸۳

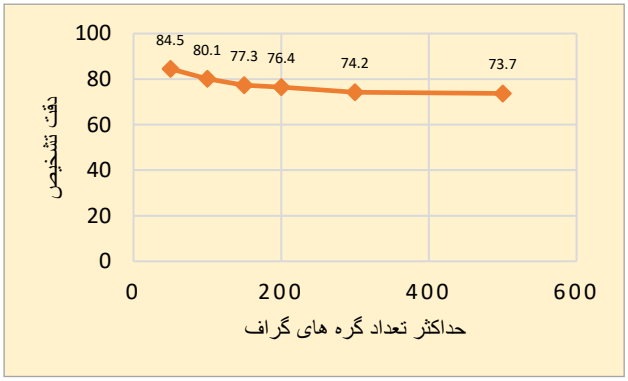
در مقایسه با کارهای قبلی، روش پیشنهادی طیف گسترده‌تری از باگ‌ها را تشخیص می‌دهد ازجمله ویژگی تعریف‌نشده، اضافه کردن کلمه کلیدی async، کاربرد نادرست کمیت سنج‌های const/let/var، افزودن کلمه کلیدی export، varnaming، varmisuse و ... . درحالی‌که در [۴] چهار نوع باگ به تفکیک مورد ارزیابی قرارگرفته‌اند و همچنین در اکثر روش‌ها باگ‌ها به‌طور اختصاصی در دیتاست‌ها و پایگاه کدهای خاصی موردبررسی قرارگرفته‌اند درحالی‌که روش پیشنهادی بر روی کدهای موجود در گیت‌هاب مورد آزمایش قرارگرفته‌اند و طیف بسیار گسترده‌تری از باگ‌ها را پوشش می‌دهد. در [۴] هر نوع باگ به‌صورت جداگانه تحلیل می‌شود، به این صورت که برای هر نوع یک مدل جداگانه آموزش داده می‌شود و عملیات تست فقط برای تشخیص آن نوع خاص انجام می‌شود و علت به دست آوردن درصدهای بالای دقت در این روش به همین دلیل است به‌عنوان‌مثال برای تشخیص باگ‌های مبتنی بر نام دقت ۹۷٪ را گزارش دادند. نتایج سایر معیارها در جدول ۵ نشان داده‌شده است. همان‌طور که در این جدول مشاهده می‌شود، دقت روش پیشنهادی در داده‌های مورد آزمایش ۰.۷۳۷ و امتیازهای Recall، Precision و F1 به ترتیب ۰.۵۹۴، ۰.۸۳۲ و ۰.۶۹۳ است. هرچه مقدار این معیارها به ۱ نزدیک‌تر باشد رویکرد دقیق‌تری داریم. این نتایج مجدداً مناسب بودن روش پیشنهادی ما را تأیید می‌کند.

جدول ۵- نتایج پیش‌بینی با حداکثر تعداد گره ۵۰۰

روش	F1	Precision	Recall	Acc (%)
پیشنهادی	0.693	0.832	0.594	73.7%

در شکل ۷ تأثیر تعداد گره‌های گراف را بر دقت پیش‌بینی می‌بینیم. همان‌طور که انتظار می‌رود، اندازه گراف و دقت رابطه معکوس دارند. ازآنجایی‌که معماری Patchy-SAN بر اساس GCN است و GCN اطلاعات گره را در نظر می‌گیرد. با افزایش تعداد گره‌ها، ویژگی‌های گره‌ها افزایش می‌یابد و این باعث افزایش پیچیدگی مدل و در نتیجه کاهش دقت می‌شود.

–



شکل ۷- دقت در مقابل تعداد گره‌های گراف

۵- نتیجه‌گیری

در این مقاله، برای مسئله کشف باگ در کدهای برنامه‌نویسی راهکاری مبتنی بر یادگیری عمیق گراف‌ها ارائه داده‌ایم. گراف‌ها، یا شبکه‌ها، ساختارهای غنی و انعطاف‌پذیری هستند که می‌توانند تعامل بین قسمت‌های مختلف بسیاری از سیستم‌های پیچیده طبیعی و ساخته‌شده توسط بشر را با دقت نشان دهند. یکی از عملیات مهم در گراف‌ها، طبقه‌بندی گراف‌ها است که هدف پیش‌بینی دسته‌ای است که به آن تعلق دارند. در این مقاله، مسئله کشف باگ‌ها را به‌عنوان یک مسئله طبقه‌بندی گراف‌ها مدل‌سازی کرده‌ایم. مقایسه نتایج پیاده‌سازی و اجرای این روش نشان می‌دهد که روش ارائه‌شده در مقایسه با کارهای قبلی طیف گسترده‌تری از باگ‌ها را شناسایی می‌کند. در آینده می‌توان ترکیب روش‌های طبقه‌بندی گره‌های گراف با روش‌های طبقه‌بندی

یال‌های گراف برای تشخیص مکان باگ را مدنظر قرارداد.

مراجع

[1] Caitlin Sadowski, Jeffrey van Gogh, Ciera Jaspan, Emma Söderberg, and Collin Winter. “Tricorder: Building a program analysis ecosystem”. In Proceedings of the 37th International Conference on Software Engineering, 2015.

[2] Andrew Scott, Johannes Bader, and Satish Chandra. “Getafix: Learning to fix bugs automatically”. In Proceedings of the ACM on Programming Languages. 2019.

[3] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. “Learning to represent programs with graphs”. International Conference on Learning Representations, 2018.

[4] Michael Pradel and Koushik Sen. “Deepbugs: A learning approach to name-based bug detection”. ACM Program. Lang, 2(OOPSLA), 2018.

[5] Edward Aftandilian, Raluca Sauciuc, Siddharth Priya, and Sundaresan Krishnan. 2012. “Building Useful Program Analysis Tools Using an Extensible Java Compiler”. In 12th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2012, Riva del Garda, Italy, September 23-24, 2012. 14–23.

[6] David Hovemeyer and William Pugh. “Finding bugs is easy”. In Companion to the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA). ACM, 132–136, 2004.

[7] L. Yousofvand, S. Soleimani and V. Rafe, “Automatic bug localization using a combination of deep learning and model transformation through node classification”, Software Quality Journal, 2023.

[8] Andrew Rice, Edward Aftandilian, Ciera Jaspan, Emily Johnston, Michael Pradel. “Detecting Argument Selection Defects”. In OOPSLA, 2017.

[9] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. “Deep API learning”. In Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016. 631–642.

[10] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade.. “DeepFix: Fixing Common C Language Errors by Deep Learning”. InAAAI, 2017.

[11] Sudheendra Hangal and Monica S. Lam.. “Tracking down software bugs using automatic anomaly detection”. In International Conference on Software Engineering (ICSE). ACM, 291–301, 2002.

[12] Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, and Benjamin Chelf.. “Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code”. In Symposium on Operating Systems Principles (SOSP). ACM, 57–72, 2001.

[13] Amin Nikanjam, Houssem Ben Braiek, Mohammad Mehdi Morovati, and Foutse Khomh. “Automatic Fault Detection for Deep Learning Programs Using Graph Transformation”. ACM Transactions on Software Engineering and Methodology, 2021.

[14] E. Dinella, H. Dai, Z. Li, M. Naik, L. Song, and K. Wang, “Hoppity: Learning graph transformations to detect and fix bugs in programs”. In International Conference on Learning Representations, 2021.

[15] Péter Gyimesi, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Árpád Beszédes, Rudolf Ferenc, Ali Mesbah. “BUGSJS: a benchmark and taxonomy of JavaScript bugs”. Software Testing, Verification and Reliability, e1751,2019.

[16] Maryam Moradi, Reza Yousefian, Vahid Rafe, “Providing a solution to deal with the state space explosion problem in graph transformation systems using bird algorithms and gravitational search”, Journal of Engineering of Tabriz University of Electrical Engineering, Volume 45, Number 4, Pages 163-177, 2015.

[17] A. Darvish and S. Shamekhi, “ A hybrid multi-scale CNN-LSTM deep learning model for the identification of protein-coding regions in DNA sequences”. Tabriz Jornal of Electrival Engineering, 2022.

[18] Kipf, T. N., and Welling, M. “Semi-supervised classification with graph convolutional networks”. In The International Conference on Learning Representations (ICLR), 2017.

[19] M. Niepert, M. Ahmed and K. Kutzkov, “Learning convolutional neural networks for graphs”, InProceedings of The 33rd International Conference on Machine Learning, PMLR , 2016.

[20] Shervashidze, N.; Schweitzer, P.; Leeuwen, E. J. v.; Mehlhorn, K.and Borgwardt, K. M. “Weisfeiler-lehman graph kernels”. Journal of Machine Learning Research 12(Sep):2539–2561 , 2011.