فصلنامه سیستم‌های پردازشی و ارتباطی چندرسانه‌ای هوشمند
Intelligent Multimedia Processing and Communication Systems (IMPCS)

Winter 2023, 3 (4), 1-13
DOR: 20.1001.1.27832570.1401.3.4.1.5

Received: 8 Oct 2022
Accepted: 15 Nov 2022

مقاله پژوهشی

SDC-causing Error Detection and Mitigation Based on Failure Rate Prediction without Fault Injection

Moona Yakhchi[1], Mahdi Fazeli [2*], Seyed Amir Asghari Toochai[3]

1. Phd Student, Department of Computer, Borujerd Branch, Islamic Azad University, Borujerd, Iran, m.yakhchi@iaub.ac.ir
2. Associate Professor, School of Information Technology, Halmstad University, Halmstad, Sweden, *(Corresponding Author)* mahdi.fazeli@hh.se
3. Assistant Professor, Electrical and Computer Engineering Department, Kharazmi University, Tehran, Iran. asghari@khu.ac.ir

Abstract:

Introduction: Reducing the size of processing components and increasing the probability of failure even in ordinary components maintaining reliability has become a serious challenge of today's computer systems. The soft errors can lead to silent data corruption which seriously compromises the reliability of a system. The Silent data corruption is a fault that affects running software and leads to incorrect output. Detecting silent data corruption needed a profile of the instructions causing the silent data corruption to decide which instructions to be protected. Current approaches by machine learning algorithms predict the occurrence rate of silent data corruption for each instruction. While most of the existing algorithms suffer from inaccuracy. Most current detection techniques require sufficient data from fault injection for training, which is difficult to achieve due to high resource consumption, such as execution time and code size costs. However, as technology is downscaling toward Nano-scale sizes, multiple-bit soft errors are emerging as an important reliability challenge. Therefore, identifying and determining vulnerable points in the presence of fault has so important.

Method: Traditional solutions based on redundancies are very expensive in terms of chip area, energy consumption, and performance. Consequently, providing low cost and efficient approaches to cope with SDCs has received researchers' attention more than ever. Hence the lack of a high-precision method without fault injection becomes a research challenge. Utilizing fault injection methods in complex systems is costly; therefore, in identifying silent data corruptions, a method based on machine learning algorithm is used, which is not necessary to inject fault in all software. Multi-bit faults and silent data corruptions with instruction sources are also considered. For this goal, we have proposed the M5rule decision tree model to detect the silent data corruption error by calculating the importance of the instruction feature for the vulnerability. Then we have used the error detection method by copying the critical instructions with sort.

Results: Finally, we evaluated our model on Mibench benchmarks with multiple test programs. The results show an overhead of 58 % with data silent corruption coverage rate of about 99%.

Discussion: In order that in this paper not only the single-bit fault but also multiple-bit fault has been considered. In addition, fault had been injected into instruction and data. Consequently, the evaluation results show that our method achieves a better detection accuracy compared to other state-of-the-art methods.

Keywords: Silent Data Corruption, Fault Injection, Soft Errors, Multi-Bit Fault, Machin Learning.

فصلنامه سیستم‌های پردازشی و ارتباطی چندرسانه‌ای هوشمند
Intelligent Multimedia Processing and Communication Systems (IMPCS)
انجمن علمی تجارت الکترونیکی ایران

تشخیص و کاهش خرابی ساکت داده براساس پیش‌بینی نرخ رخداد خرابی بدون تزریق اشکال

تاریخ دریافت:۱۴۰۱/۰۷/۱۶ دورهٔ سوم، زمستان ۱۴۰۱
تاریخ پذیرش:۱۴۰۱/۰۸/۲۴ شمارهٔ چهارم، صص: ۱-۱۳

مونا یخچی[۱]، مهدی فاضلی[۲*]، سید امیر اصغری توچائی[۳]

۱. دانشجوی دکتری، گروه کامپیوتر، واحد بروجرد، دانشگاه آزاد اسلامی، بروجرد، ایران. m.yakhchi@iaub.ac.ir
۲-.دانشیار، گروه کامپیوتر ، دانشکده فناوری اطلاعات، دانشگاه هالمستاد، هالمستاد، سوند. (نویسندهٔ مسئول) mahdi.fazeli@hh.se
۲. استادیار، گروه کامپیوتر، دانشکده مهندسی برق و کامپیوتر، دانشگاه صنعتی خوارزمی، تهران، ایران. asghari@khu.ac.ir

چکیده: خرابی ساکت داده (SDC) به‌طورجدی قابلیت اطمینان یک سیستم را به مخاطره می‌اندازد. رویکردهای فعلی با استفاده از یادگیری ماشین نرخ رخداد SDC برای هر دستورالعمل را پیش‌بینی‌می‌کنند. در حالی‌که اکثر آن‌ها فاقد دقت مناسب و نیازمند مجموعه داده برای آموزش هستند و به دلیل مصرف منابع زیاد دستیابی به آن‌ها دشوار است. از سوی دیگر نرخ رخداد اشکالات چندبیتی در قطعات نیمه هادی افزایش چشمگیری داشته‌اند. لذا تشخیص دستورهای آسیب‌پذیر در حضور اشکال اهمیت یافته‌است. اما خلاء تحقیقات موجود عدم وجود یک روش نرم‌افزاری با دقت بالا بدون نیاز به تزریق اشکال است؛ به طوری‌که تشخیص اشکال در SDC با منشأ داده و دستورالعمل مورد بررسی قراربگیرد. بدین منظور، در این پژوهش با محاسبه نرخ رخداد SDC برای هر دستورالعمل‌ها، مدل درخت تصمیم‌گیری M5rule پیشنهاد گردیده‌است. سپس از روش تشخیص خطا، با کپی‌کردن دستورالعمل‌های حیاتی به‌وسیله مرتب سازی استفاده شده و در نهایت مدل ارائه شده بر روی معیار Mibench با برنامه‌های آزمایشی متعدد ارزیابی گردیده‌است. نتایج ارزیابی نشان‌می‌دهد روش ارائه‌شده در مقایسه با سایر روش‌های پیشرفته به دقت تشخیص بهتری با سربار در حدود ۹۹ درصد برای ۵۸ درصد نرخ پوشش SDC رسیده‌است.

واژه‌های کلیدی: خرابی ساکت داده، تزریق اشکال، خطاهای نرم، خطاهای چندبیتی، یادگیری ماشین.

۱. مقدمه

روند ارتقاء فناوری امکان کاهش اندازه ترانزیستورها امکان قرارگرفتن میلیون‌ها ترانزیستور در کنار هم را در دستگاه‌هایی که در دامنه گیگاهرتز کارمی‌کنند و از منابع ولتاژ پایین تغذیه‌می‌شوند، فراهم‌می‌کند با این حال، به دلیل محدودیت‌های حاشیه نویز و کاهش اندازه قطعات در مدارها منجر به کاهش قابلیت اطمینان[۱] می‌شود. خطاهای نرم[۲] عمدتاً به دلیل رخدادهای واژگونی[۳]هستند که در اثر تابش[۴]، نویزهای الکترومغناطیسی[۵] و اشکالات برق[۶] ایجاد می‌شوند. این نوع خطاها بسیار جدی هستند و می‌توانند عملکرد یک برنامه را با پیامدهای قابل‌توجهی مختل‌کنند. برخلاف خطاهایی که در هنگام طراحی یا ساخت رخ می‌دهد، نمی‌توان رفتار خطاهای نرم را از قبل پیش‌بینی‌کرد و تأثیر آن‌ها بر عناصر سیستم مانند ثبات‌ها[۷] کاملاً تصادفی است [۱].

از سوی دیگر سیستم‌های تعبیه‌شده[۸] به‌وفور مورد استفاده قرارمی‌گیرند و دامنه کاربرد آن‌ها از سیستم‌های سرگرمی تا ماشین‌آلات صنعت و سیستم‌های ایمنی است. این افزایش استفاده به دلیل روند رو به رشد تکنولوژی در کاهش اندازه ویژگی ترانزیستورها و کاهش ولتاژهای تغذیه بوده‌است. با کاهش اندازه ویژگی ترانزیستورها، ترانزیستورهای بیشتری را می‌توان در یک منطقه مشابه قرارداد و ریزپردازنده‌های[۹] قوی‌تر و اغلب سریع‌تر ایجادکرد.

روند خطاهای نرم را می‌توان در سطح سیستم به دو دسته طبقه‌بندی‌کرد [۲]: ۱) خطای جریان کنترل[۱۰]، که به هرگونه انحراف از جریان کنترل صحیح یک برنامه اشاره‌دارد. ۲) خطای داده[۱۱]، این خطاها به هر گونه دستکاری داده‌ها ناشی از خطاهای گذرا[۱۲] اشاره دارد. اگر داده‌ها محافظت‌نشوند، چنین خطاهایی ممکن است منجر به خرابی سیستم شود.

خطای جریان کنترل: هنگامی رخ‌می‌دهد که پردازنده دستورالعمل های غیرمنتظره‌ای را اجرامی‌کند و سپس آن را از جریان صحیح برنامه منحرف‌می‌کند. بنابراین، برنامه به یک مکان نادرست پرش‌می‌کند و توالی اجرا را تغییرمی‌دهد که ممکن است منجر به رفتار غیرقابل پیش‌بینی سیستم شود.

خطای داده: این خطا انحراف از خروجی صحیح است هنگامی که یک اشکال تک‌بیتی یا چندبیتی در زمان پردازش داده‌ها اتفاق‌می‌افتد. به عبارت دیگر این خطا در داده آشکارمی‌شود. نتایج سیستم به عنوان خروجی صحیح، خرابی/ قطع، SDC طبقه‌بندی شده‌است.

- خروجی صحیح: این خطاها هیچ تاثیری در نتیجه برنامه ندارند. آن‌ها ممکن است در طول اجرای برنامه پوشانده‌شوند یا راهی برای تأثیر بر نتایج خروجی پیدانکنند [۶-۳].
- خرابی/ قطع : هنگامی که یک خطای نرم باعث‌می‌شود سیستم وارد حالتی شود که در آن به هیچ رویدادی پاسخ‌ندهد، از آن به عنوان وضعیت خرابی/ قطع یادمی‌شود [۶-۳].
- *SDC* : خرابی ساکت داده به خطاهایی در داده‌های کامپیوتری اطلاق‌می‌شود که در طول خواندن، نوشتن، انتقال، ذخیره‌سازی یا پردازش اتفاق‌می‌افتد و به شکل غیرعمدی داده اصلی را تغییر می‌دهد. در حالت کلی، وقتی خرابی داده‌ای اتفاق‌می‌افتد، فایلی‌که شامل داده است توسط سیستم یا نرم‌افزاری استفاده‌می شود نتیجه غیرقابل انتظاری تولیدمی‌کند و محدوده نتیجه بین ازدست دادن داده کم تا شکست سیستم است [۶-۳].

بنابراین، ایجاد یک مدل تشخیص SDC برای افزایش قابلیت اطمینان و امنیت سیستم بسیار مهم است در سال‌های اخیر، تعدادی رویکرد برای شناسایی دقیق و کارآمد خطاهای SDC توسعه‌یافته‌است. به‌طور سنتی، تزریق خطا[۱۳]، عملوند مقصد دستورالعمل‌ها را به‌طور تصادفی تغییرمی‌دهد و سپس نتیجه اشتباه را به عنوان خطاهای SDC شناسایی‌می‌کند [۶-۵].

SDC مسبب خطاهایی است که در برنامه پخش شده و منجر به خروجی نادرست می‌گردد و به این ترتیب می‌توان با استفاده از شاخص‌های تشخیص خطا یا بررسی برنامه از رویداد آن جلوگیری کرد. یک تشخیص‌دهنده، سازگاری متغیرهای برنامه را بررسی‌می‌کند و در صورت عدم سازگاری، دستور خاتمه برنامه را صادرمی‌کند [۱].

با توجه به این رویکردها، چالش مهم این است که چگونه متغیرهایی از برنامه با پوشش SDC بالا و بودجه سربار کارایی[۱۴] مشخص شناسایی و محافظت شوند. یک راه شناسایی متغیرها برای حفاظت در نرم‌افزارها، استفاده از تزریق اشکال است که مرتباً برنامه تحت‌تأثیر تقلید اشکالات سخت‌افزاری قرارمی‌گیرد. سپس نتایجی که منجر به تولید SDC می‌گردند، مطالعه‌می‌شوند. اجرا این روش‌ها برای به‌دست‌آوردن تخمین چشم‌گیر نیازمند تعداد زیادی تزریق اشکال است که زمان زیادی مصرف‌می‌کند. مورد دیگر با تزریق اشکال این است که انطباق نتایج به برنامه و تصمیم‌گرفتن درباره این‌که چه متغیر(هایی) محافظت شوند با توجه به سربار کارایی تلاش زیادی لازم دارد. بنابراین سعی شده‌است رویکردی ارائه شود که با تزریق اشکال در یک نرم‌افزار مدل مطلوبی استخراج‌گردد تا دستورات مستعد SDC مشخص و سپس در دیگر نرم‌افزارها از آن استفاده‌شود.

به دلیل افزایش نویزها، پرتوهای کیهانی، کاهش اندازه قطعات سخت‌افزاری، استفاده بیشتر از ترانزیستورها اشکالات از حالت تک بیتی به سمت چندبیتی افزایش‌یافته‌است. تلاش بر این است که علاوه بر بررسی تأثیر اشکالات تک‌بیتی اقدام به بررسی اشکالات چندبیتی نیز شود. از سوی دیگر در پژوهش‌های ارائه‌شده تنها برروی خرابی داده‌ها تمرکزکرده‌اند. این در حالی است که خرابی دستورات نیز خود اثرات فاجعه باری به دنبال دارد. بنابراین در این مقاله اثرات اشکال بر روی داده و دستور درنظرگرفته‌می شود. با پیشرفت فناوری رویکرهای هوش مصنوعی با دقت بالا به‌وجودآمده‌اند، از این‌رو رویکردهای مورد اشاره در تشخیص دستورها و داده‌های مستعد خرابی استفاده‌خواهدشد. هدف نهایی این پژوهش تشخیص و کاهش رخداد خرابی ساکت در

سیستم‌های نرم‌افزاری است. رسیدن به این هدف یک فرآیند پنج مرحله‌ای شامل، تزریق اشکال، جمع‌آوری اطلاعات، آموزش، تشخیص داده و دستورات مستعد SDC و در نهایت بهبود پوشش اشکالات منجر به SDC خواهدبود. در بحث تزریق اشکال سه مسئله مدّنظر قرارخواهدگرفت. اولین مسئله که پیش از این کمتر مورد توجه واقع‌شده، خطاهای چندبیتی است. در دنیای امروز احتمال رخداد خطاهای چندبیتی به‌شدت افزایش‌یافته‌است. به‌همین دلیل توجه به این دسته از خطاها بسیار حائز اهمیت است. مسئله دیگر توجه به اشکالات در دستورالعمل است. در پژوهش‌های صورت‌گرفته تمرکز بر روی اشکال خطا در داده بوده‌است. در این مقاله تزریق اشکالات در دستورالعمل را نیز مدّنظر قرارخواهدگرفت. مسئله آخر حجم تزریق اشکال است. هدف این است که تنها به بخشی از برنامه‌ها تزریق اشکال شود تا در سیستم‌های نرم‌افزاری دیگر بدون تزریق اشکال از آن استفاده‌گردد.

جمع‌آوری اطلاعات با نظارت و کنترل برنامه تحت اجرای آزمایشی صورت‌خواهدگرفت. پس از تزریق اشکال به برنامه با جمع آوری اطلاعات رفتاری برنامه، داده های لازم جهت مرحله بعدی فراهم خواهدشد. نکته قابل‌توجه در این خصوص این است که در هر اجرا تنها یک تزریق اشکال، یک‌بیتی یا چندبیتی، بر روی برنامه صورت‌می‌گیرد.

روش‌های هوشمند قادر به یافتن پاسخ نزدیک به بهینه در زمانی کوتاه هستند. استفاده از این روش‌ها باعث افزایش دقت برای رسیدن به پاسخ بهینه خواهد شد. بنابراین از مجموعه داده جمع‌آوری شده در مرحله قبل جهت آموزش یک روش هوشمند استفاده‌خواهدشد.

بدون شک برای کاهش خرابی ساکت یک گام کلیدی تشخیص داده و دستورات مستعد SDC خواهدبود. در این مرحله با استفاده از روش آموزش دیده‌شده در مرحله قبل اقدام به تشخیص داده و دستورات مستعد SDC می‌شود. نکته حائزاهمیت در ایده پیشنهادی این پژوهش عدم نیاز به اجرای آزمایشی تمامی برنامه‌ها خواهدبود. به عبارت دیگر پس از آموزش ماشین براساس مجموعه داده جمع‌آوری شده، ماشین قادر به تخمین نرخ رخداد[15] دستورات مستعد SDC خواهدبود.

پس از تخمین نرخ رخداد دستورات مستعد SDC و سپس مرتب سازی مقادیر تخمین‌زده‌شده با روش‌های نرم‌افزاری نظیر دوتایی کردن[16] اقدام به اجرای مجدد برنامه محافظت‌شده خواهدشد. نتیجه برنامه کاهش نرخ SDC در برنامه است.

۲. معیار

۱،۲. مقاوم‌سازی[17] در برابر خطا

مقاوم‌سازی در برابر خطا به معنی افزایش توانایی تحمل (مقاومت) خطا در زمان رخداد است به طوری که ممکن است یک خطا در برنامه روی نتیجه اثر بگذارد یا نگذارد. در این موضوع مقاوم سازی در برابر خطا به افزایش احتمال اینکه نرم‌افزار یک نتیجه خراب، بعد از رخداد یک اشکال سخت‌افزاری نداشته‌باشد گفته‌می‌شود (به معنی شکست، هنگ[18] یا SDC). بنابراین، مقاوم‌سازی در برابر خطا یک ویژگی در سکو[19] و نرم‌افزار کاربردی[20] است. باید توجه‌شود که مقاوم‌سازی در برابر خطا احتمال شرطی یک برنامه است به این معنی که با توجه به اینکه اشکال رخداده‌است با هیچ خرابی مواجه‌نمی‌شود [۷].

۲،۲. نرخ رخداد SDC

نرخ رخداد SDC بر اساس تعداد اشکالاتی که به کد برنامه تزریق‌می‌شود و در نهایت تعدادی از آن اشکالات منجر به SDC می‌شوند، محاسبه‌می‌گردد [۱۱-۸][۵].

۳،۲. پوشش SDC

میزان پوشش SDC، تعداد SDCهایی است که پس از تزریق اشکال و شناسایی آن‌ها توسط تشخیص‌دهنده پوشش داده‌می‌شوند و منجر به خروجی نادرست نمی‌گردند [۵].

۴،۲. سربار کارایی[21] و زمان اجرا

سربار کارایی، میزان افت کارایی به علت استفاده از رویکردهای محافظت برنامه در برابر رخداد SDC است. مثلاً در روش‌های دوتایی کردن در ازای هر دستورالعمل مستعد SDC، دستوری که منجر به اجرای دوباره و مقایسه مقادیر برای جلوگیری از رخداد SDC می‌گردد دوتایی می‌شود و سربار کارایی بیش از ۱۰۰ درصد را به همراه دارد[۵، ۶].

کارایی و زمان اجرا به‌ترتیب از رابطه‌های (۱) و(۲) به‌دست می‌آیند:

$$Execution\ time = \frac{CPI^{22} \times Instruction\ time}{Frequency} \qquad (1)$$

$$Performance = \frac{1}{Execution\ time} \qquad (۲)$$

۵،۲. تکرار داده

تکنیک تکرار داده‌ها[23] رایج‌ترین روش نرم‌افزاری برای تحمل خطای سخت‌افزاری پیاده‌سازی‌شده توسط نرم‌افزار است که از ویژگی تصادفی بودن خطاهای نرم استفاده‌می‌کند؛ زیرا همان خطا مکرر رخ‌نمی‌دهد. مفهوم اساسی برای تکنیک تکرار داده‌ها این است که داده‌ها را کپی‌کرده و دو نسخه داده را در طول زمان اجرای برنامه نگه‌داشت. داده‌های اصلی نسخه اصلی را با داده‌های نسخه سایه مقایسه‌می‌شود. هنگامی که یک خطا رخ‌دهد تکنیک بازیابی فراخوان‌می‌شود. این تکنیک‌ها به‌طور گسترده‌ای برای مزایای آن در اجرای انعطاف‌پذیر و عمومی مورد استفاده قرارگرفته‌اند.

۶،۲. مدل درختی M5

مدل‌های تصمیم درختی[24] انواع مختلف دارد. هنگامی که خروجی یک درخت، یک مجموعه گسسته از یک مجموعه مقادیر ممکن است، به

آن طبقه‌بندی[25] درختی گفته‌می‌شـــود. هنگامی که بتوان خروجی درخت را یک عدد حقیقی درنظرگرفت، آن را رگرســـیون[26] درختی می‌نامند. به عبارت دیگر، اگر متغیرهای ورودی به ســـیســـتم عددی باشند، از رگرسیون درختی و اگر مطلق و قیاسی باشند، از طبقه‌بندی درختی اســـتفاده می‌شـــود. مدل M5 یک مدل درختی برای پیش‌بینی صـــفات عددی پیوســـته اســـت که در آن توابع رگرســـیونی خطی در برگ‌های این درخت ظاهرمی‌شـوند [۱۲]. فرآیند انشـعاب[27] در هر گره بارها تکرارمی‌شـود تا به گره پایانی (برگ) برســد که در برگ، مجموع مجذور انحراف از میانگین داده‌ها حدوداً به صـــفر می‌رســـد. با این کار درخت بزرگی توسعه پیداخواهدکرد.

روش‌های بر مبنای درخت یکی از روش‌های داده کاوی[28] است که در این روش‌ها خروجی به صورت یک مدل با سازه درختی با استفاده از داده‌های ورودی و خروجی است. درخت تصمیم‌گیری یک ابزار برای پشتیبانی از تصمیم است که از درخت‌ها برای مدل‌کردن استفاده‌می‌کند. درخت تصمیم به‌طور معمول در تحقیق‌ها و عملیات مختلف استفاده‌می‌شود. به‌طور خاص در آنالیز تصمیم، برای مشخص‌کردن استراتژی که با بیشترین احتمال به هدف برسد به‌کارمی‌رود. استفاده دیگر درختان تصمیم، توصیف محاسبات احتمال شرطی است. مدل درختی به عنوان ابزاری قدرتمند برای داده‌کاوی و پیش‌بینی به صورت عملی و گسترده در شاخه‌های مختلف علوم مورداستفاده قرارگرفته است. مدل‌های درختی که در طبقه‌بندی و رگرسیون درختی استفاده‌می‌شوند توسط کوین لان [۱۳] ارائه‌شده‌است. تئوری او به صورت یک روش توسعه‌یافته است. مدل وی در سال ۱۹۹۷ توسط ونگ و ویتن توسعه و M5 نامیده‌شد [۱۴].

۳. پیشینه پژوهش

روش‌های مورد بررســـی در این تحقیق به ســـه دســـته طبقه (دســـته) تقســـیم‌می‌گردند. روش‌های مبتنی بر مدل‌ســـازی، روش‌های مبتنی بر تزریق اشکال و در نهایت روش‌های مبتنی بر هوش مصنوعی است. که تنها به بررسی مطالعاتی در راستای مقابله با SDC با روش‌های مبتنی بر هوش مصنوعی است پرداخته‌شده‌است.

۱،۳. روش‌های مبتنی بر مدل‌سازی ریاضی

رویکردهای مدل سازی ریاضی از ساختارهای داده ویژه برای مدل‌سازی پارامترهای اولیه و همچنین مدل انتشـار خطاهای نرم در فرایند برآورد نرخ خطای نرم استفاده‌می‌کنند. ولی اثرات خطاهای نرم را بیش از حد ارزیابی‌می‌کند [۲۳-۱۵].

۲،۳. روش‌های مبتنی بر تزریق اشکال

رویکردهای تزریق اشکال بر اساس تزریق خطا و شبیه سازی منطقی اســـت و نتایج برآورد نرخ خطای نرم بســـیار دقیقی را ارائه‌می‌دهند، این رویکرد به زمان و تلاش زیادی برای دستیابی به نتایج آماری قابل‌توجه نیازدارد [۲۵،۲۴]. برای مثال می‌توان مطالعات [۳۱-۲۶][۳] را نام‌برد.

۳،۳. روش‌های مبتنی بر هوش مصنوعی

لو و همکاران در [۴][۳۲] دو مدل SDCtune و SDCAuto را برای کمیت سنجی استعداد SDC متغیرهای برنامه و توسعه روش مبتنی بر مدل حفاظت بالای انتخابی متغیرهای مســـتعد SDC ارائه‌کرده‌اند. متغیرهای مستعد SDC آن‌هایی است که یک اشکال در آن با احتمال بالایی منجر به نتیجه SDC می‌شـود و در نتیجه نیاز به حفاظت دارد. مدل‌های SDCTune و SDCAuto تنها از تحلیل‌های پویا و ایستا برای شناسایی متغیرهای مستعد SDC در برنامه بدون نیاز به تزریق اشکال استفاده می‌کنند، پس زمان زیادی ذخیره می‌شود. علاوه بر این مدل به کاربر اجازه‌ی تنظیم محافظت با توجه به ســـربار کارایی و تحمل آن اندازه که تمایل دارند را می‌دهند. رویکرد این مقاله ســـه بخش دارد. در ابتدا راهکار ابتکاری شناسایی می‌شود که به متغیرهای به شدت مستعد SDC برنامه وابسته است. راهکار ابتکاری با استفاده از تزریق اشکال برروی مجموعـه کوچکی از برنـامـه محـک کـه برای آموزش هـدف استفاده‌می شود، استخراج‌می شود. سپس این راهکار ابتکاری با تحلیل دستی و الگوریتم یادگیری ماشین برای ساخت مدل تجمیع‌می شود. SDCTune به صورت دستی تنظیم‌می‌گردد و SDCAutoها به واسطه مدل اتوماتیک با الگوریتم یادگیری ماشین ساخته شده‌است؛ بنابراین نیاز به تلاشـی از سـوی توسـعه‌دهندگان بر روی این بخش نیسـت. در شـــناســـایی اولیه راهکار ابتکاری در مدلSDCAuto و SDCTune از تزریق اشکال استفاده‌می شود در حالی که برای برنامه‌های جدید، مدل احتیاجی به تزریق اشکال ندارند. در این پژوهش خطاهای سخت‌افزاری گذرا درنظرگرفته شده‌است و بر روی تشخیص خطا، بیشتر از بازیابی خطا تمرکزشـــده‌اســـت. برای بازیابی از یک خطای گذرا از روش‌های شروع دوباره برنامه از نقطه بازر سی[29] استفاده شده‌است. SDCAuto و SDCTune برای شناسایی متغیرهای مستعد SDC و برای استخراج تشـخیص‌دهنده خطا برای متغیرها با توجه به سـربار کارایی مشـخص به‌کارگرفته شده‌است. تشخیص‌دهنده، مقادیر متغیرهای انتخاب شده را در برش عقبی[30] تکرارمی‌کند و مقادیر محاسبه شده تکراری را با مقدار اصلی مقایسه‌می‌کند. تشخیص‌دهنده با استفاده از الگوریتم مستخرج از تزریق اشکال برای استخراج مدل بر روی ۶ برنامه مختلف و با محدوده ســـربار کارایی از ٪۱۰ تا ٪ ۳۰ درج و ارزیابی می‌شـود و نتایج پوشـش تشـــخیص خطا به علت SDC با درنظرداشـــتن ســـربار کارایی را نشان‌می‌دهند.

در مرجع [۳۳] با استفاده از الگوریتم جنگل تصادفی[31] دستورالعمل عادی منجر به SDC را شناسایی و اقدام به استخراج ویژگی‌های پویا و ایســـتا برنامه برای شـــناســـایی این دســـتورها می‌نمایند. ایده اصـــلی پیش‌بینی استعداد SDC، داده‌های برنامه و سپس انتخاب آن‌ها با توجه به محدودیت ســـربار کارایی اســـت. وقتی مدل پیش‌بینی که بر اســـاس

درخت CART اســت ســاخته‌شــد دیگر نیازی به تزریق اشــکال برای شناسایی دستورالعمل‌ها نیست. این روش را SCDpredictor نامیده‌اند. برای ســاخت درخت، ویژگی‌هایی وزن‌دهی می‌شــوند که ارزش بالاتری در ســاخت یک درخت پیش‌بینی کننده با صــحت بالاتر دارند. تزریق اشــکال بر روی حافظه و حافظه نهان به دلیل محافظت توســط کد تصحیح خطا درنظرگرفته نشده‌است. برای منظور کردن اشکالاتی که منجر به اشــکال در جریان داده می‌شــوند، اشــکال در واحد تابعی و ثبات‌ها درنظرگرفته‌شــده‌اســت. اشــکال در کد دســتورالعمل‌ها نیز درنظرگرفته‌نشده‌است.

در مرجع [۳۴] روشی برای کاهش تعداد دفعات تزریق اشکال درجهت کاهش هزینه تزریق ارائه شده‌است که PVIns[۳۲] نامگذاری شده‌است. سپس دستورات آسیب‌پذیر برنامه را شناسایی‌می‌کنند. با تزریق اشکال جزئی داده آموزش برای SVM تولیدمی‌شود. این مجموعه داده براساس برنامه‌های هدف طبقه‌بندی‌می‌شوند. تزریق اشکال به صورت تک‌بیت بر روی فایل ثبات یا حافظه درنظرگرفته‌می شود با استفاده از Pin که یک چارچوب ابزار دقیق باینری برای مجموعه دســتورهای IA-۳۲ و ۶۴-x۳۲ اقدام به تزریق اشکال جزئی می شود [۳۵]. یک اشکال برای هر بار اجرا تزریق صــورت‌می‌گیرد و بر کد دســتورالعمل اشــکال تزریق نمی شود. اگر یک دستورالعمل به حد آستانه آسیب‌پذیری رسید به عنوان دســتورالعمل آســیب‌پذیر انتخاب‌می‌شــود. اگر نیاز بود در مرحله بعد طبقه‌بندی، ۵٪ ویژگی جدید اضافه‌می شود. دو دسته از دستورها مهم هســتند یکی اتصــالی مثل متغیرهای عمومی، پارامترها و مقادیر بازگشتی که اشکال در آن‌ها منجر به SDC می‌شود. دیگری دستورهای پرشی مثل دستور CMP ویژگی بعدی ثبات است و حد آستانه آن‌ها بر روی آسیب‌پذیری و حافظه است. نتایج نشان‌داده‌اند که با ۳۵٪ تزریق اشکال به میانگین ۸۰٪ صحت، ۸۵٪ یادآوری (بازخوانی) و سربارکارایی ۷۸,۳٪ رسیده‌اند.

فانگ و همکاران در مرجع [۳۶] مدل SDIFI را برای تشــخیص آســیب‌پذیری SDC که اهمیت ویژگی دســتورالعمل در آســیب‌پذیری SDC با استخراج ویژگی‌های دستورالعمل در برنامه و با در نظر گرفتن تأثیر ویژگی‌های دســتورالعمل بر نتایج پیش‌بینی‌شــده ارائه نموده‌اند و دستورالعمل‌هایی با آسیب‌پذیری SDC بالا برای محافظت انتخاب می شــوند. چگونگی ادغام اهمیت ویژگی دســتورالعمل در مدل پیش‌بینی آسیب‌پذیری SDC و کشف دستورالعمل آسیب‌پذیری بالا SDC برای حفاظت از اهمیت زیادی برخوردار است.

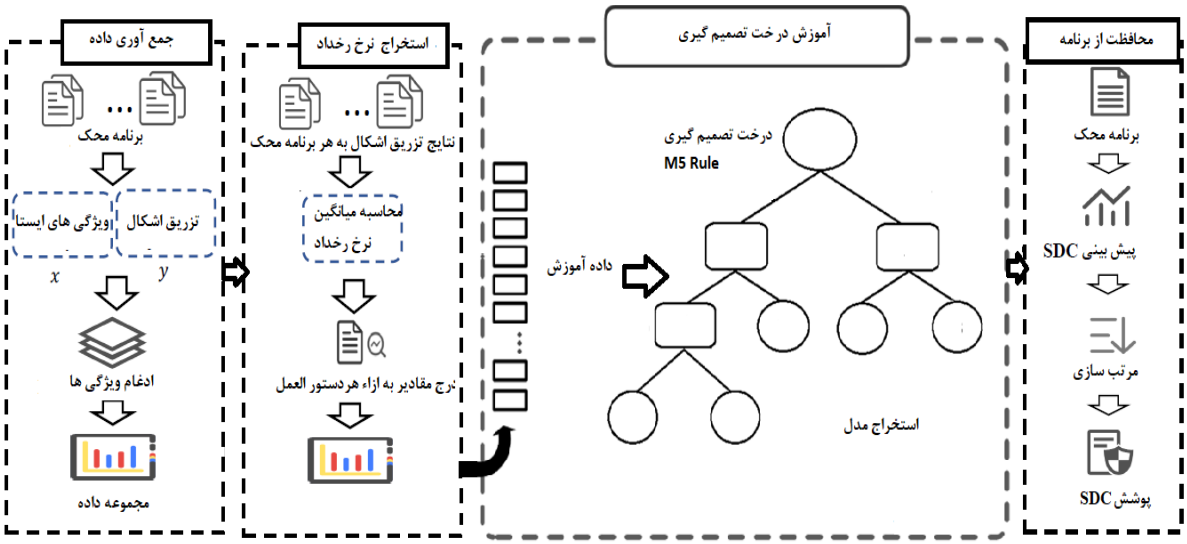
در مرجع [۳۷] یک مدل سبک وزن به نام افزونگی چند دانه‌ای مبتنی بر رگرسیون جنگل عمیق (DFRMR) برای بهبود نرخ تشخیص خطا و در عین حال کاهش مصــرف منابع بدون انجام تزریق خطا پیشــنهاد کرده‌اند. به‌طورکلی، با توجه به مســئله مصــرف زمان، DFRMR دســتورالعمل‌های آســیب‌پذیر را بســته به ویژگی‌های کد منبع به جای تزریق خطا شــناســایی‌می‌کند. علاوه بر این، روش به کاربران اجازه می‌دهد تا با توجه به هزینه‌ای که مایل به تحمل آن هســتند، تعداد دســتورالعمل‌هایی را برای محافظت انتخاب‌کنند. به‌طورخاص، ابتدا، از تجزیه و تحلیل برنامه برای اســتخراج ویژگی‌های دســتورالعملی که به شــدت به SDCها مرتبط هســتند، آســیب‌پذیری SDC را بســته به ویژگی‌های دســتورالعمل و دســتورالعمل‌ها تاحدّی با توجه به جزئیات داده شده تو سط کاربران محافظت می شوند، استفاده شده است و از تزریق اشکال برای جمع آوری داده‌ها استفاده‌شده‌است.

در مرجع [۳۸] برای شــناســایی SDC های ممیز شــناور یک شناسایی‌کننده ارائه‌کرده‌اند که مبتنی بر شبکه عصبی عمیق است. این مقاله بر روی نرم‌افزارهای HPC انجام‌شده‌است و از شبکه عصبی leNet و AlexNet-inspired با کمی تغییرات استفاده شده است. در ابتدا به بررسی خطا در تکرارهای مختلف برنامه پرداخته‌شده‌است.

در رویکرد ارائه‌شده مرجع [۳۹] بر روی خروجی SDC تمرکز شده‌است و آن را خرابی خروجی ساکت می‌خوانند. این رویکردها بر این مشاهده استوار است که، در اغلب کدهایHPC ، یک بخشی از خطاهای SDC به صورت طبیعی پوشیده‌می‌شوند و SOC تولید نمی‌کنند، به همین دلیل خروجی برنامه‌ها حتی در صورت رخداد خطا برای دانشمندان قابل‌قبول است.

۴. روش تحقیق

شکل ۱ چارچوب کلی رویکرد ارائه‌شده را نشان‌می‌دهد. عمدتاً شامل چهار جزء است: (۱) جمع‌آوری داده‌ها. برای به‌دست‌آوردن داده‌های آموزشی مورد نیاز، تجزیه و تحلیل برنامه برای استخراج ویژگی‌هایی که بسیار مرتبط با SDCها هستند انجام‌می‌شود. تزریق خطا در برنامه‌های محک برای محاسبه آسیب‌پذیری SDC انجام‌می‌شود. (۲) استخراج نرخ رخداد. پس از مرحله اول میانگینی از نرخ رخداد نتایج استخراج‌می‌گردد. (۳) آموزش درخت تصمیم‌گیری انتخاب‌شده. ابتدا تزریق اشکال برای استخراج ویژگی دستورالعمل‌ها انجام‌می‌شود. سپس مدل درخت تصمیم‌گیری M5 Rule برای آموزش پیش‌بینی‌کننده نرخ رخداد SDC هر دستورالعمل اعمال‌می‌شود.



شکل ۱: چارچوب رویکرد روش پیشنهادی

(۴) محافظت از برنامه. آسیب‌پذیری SDC دستورالعمل‌های یک برنامه جدید توسط پیش‌بینی‌کننده SDC پیش‌بینی‌می‌شود. در نهایت دستورالعمل‌ها بر اساس مقادیر پیش‌بینی‌شده مرتب‌می‌گردد و مواردی آسیب‌پذیرتر برای تکرار انتخاب‌می‌شوند. در ادامه جزئیات هر بخش مرحله به مرحله تشریح‌می‌شود.

۱،۴. جمع‌آوری داده

در این بخش، ویژگی‌های دستورالعمل استخراج می‌گردد. با در نظر گرفتن فرض امکان وجود خطا به دو صورت تک‌بیتی و چندبیتی در سیستم، خطا بر دستورها و داده‌ها تزریق‌می شود و احتمال رخداد هر نوع خطا از مرجع ابراهیمی سال ۲۰۱۶ [۴۰] پیروی‌می‌کند. ضرورت بررسی اثر انواع اشکال در یک برنامه در اختیار داشتن اطلاعات کاملی از نوع و محل اصابت اشکال و رفتار برنامه پس از آن است. دسترسی به این اطلاعات از طریق تزریق اشکال صورت‌می‌گیرد، جمع‌آوری اطلاعات اشکالات و برنامه با شبیه‌سازی اشکالات و تزریق هدفمند و تحت کنترل آن به برنامه با توجه به نتایج تأثیرات اشکال که شامل خروجی صحیح، شکست/هنگ و SDC است. اشکالات بر فایل اسمبلی هر برنامه تزریق‌شده و به دلیل استفاده از کامپایلر LLVM در اکثر مطالعات صورت‌گرفته در این حوزه این فایل با استفاده از کامپایلر فوق تولیدمی‌شود. همچنین ابزاری جهت تزریق اشکال به کد اسمبلی طراحی و در پایتون پیاده‌سازی‌گردید. به ترتیب از اولین تا آخرین دستورالعمل، به هر عملوند و هر عملگر هر کدام ۱۰۰ اشکال طبق توزیع مرجع [۴۰] تزریق‌می‌شود که این فرایند در شکل۳،۴ قابل‌مشاهده‌است.

مراحل تزریق اشکال برای اولین تا آخرین دستورالعمل هر یک از برنامه‌های محک به شرح زیر است:

- ۱۰۰ اشکال با توزیع مذکور و با تعیین تصادفی بیت مورد اشکال به عملگر تزریق‌می‌گردد.
- در صورت وجود عملوند ۱۰۰ اشکال با شرایط مشابه به هر عملوند تزریق می‌شود.
- در پایان هر تزریق اشکال ذخیره‌سازی فایل اسمبلی و بروزرسانی اطلاعات مربوط در فایل گزارش صورت‌می‌گیرد.
- در پایان مراحل تزریق اشکال و اجرای برنامه لازم است تعدادی ویژگی ایستای دیگر استخراج گردد. همانگونه که اشاره‌شد یکی از فیلدهای ذخیره‌شده در فایل گزارش، دسته‌بندی رکوردها به نتایج درست، شکست و SDC است.

۲،۴. نرخ رخداد SDC

در ادامه براساس این فیلد، اطلاعات زیر در فایل دیگری که به آن فایل دستورالعمل گفته‌می‌شود، ذخیره‌می‌گردد:

- دستورالعمل، در حقیقت بخش عملگر هر دستور العمل.
- نرخ نتایج درست، نسبت تعداد نتایج درست به کل تعداد تزریق اشکال صورت‌گرفته.
- نرخ SDC ، نسبت تعداد SDC به کل تعداد تزریق اشکال صورت‌گرفته.
- نرخ SDC با منشأ دستورالعمل، نسبت تعداد SDC که اشکال به بخش عملگر دستورالعمل تزریق گردیده به کل تعداد SDC .

لازم به ذکر است که کلیه نرخ‌های مذکور به تفکیک اشکال تک بیتی یا چند بیتی نیز محاسبه و به فایل دستورالعمل اضافه‌گردیده‌است.

هدف از اجرای این بخش پیش‌بینی نرخ رخداد SDC هر دستورالعمل از کد برنامه بدون تزریق اشکال است که سپس با توجه به سربار کارایی روش نرم‌افزاری تکرار دستورالعمل، اقدام به انتخاب دستورالعمل‌های مستعد SDC نموده تا با کمترین سربار بیشترین پوشش تشخیص اشکال ایجادگردد. برای دستیابی به هدف مذکور لازم است ابتدا الگویی برای نرخ رخداد SDC هر دستورالعمل در اختیار داشت. به منظور شناسایی الگوی رخداد SDC دستورالعمل‌های

مختلف، اقدام به تزریق اشـکال به تعدادی از مجموعه برنامه‌های محک شده‌ا ست. سپس برای هر د ستورالعمل در شرایط مختلف بر ا ساس نرخ‌های رخداد SDC در برنامه‌های متفاوت یک نرخ میانگین تعریف گردیده‌ا ست. به‌عنوان مثال نرخ رخداد جهت د ستورالعمل MOV برای عملوندهای دو بایتی و چهار بایتی متفاوت است.

اطلاعات مربوط به د ستورالعمل‌ها و کدهای هگزاد سیمال آن‌ها در یک پایگاه داده ذخیره شده‌ا ست. با کمک این پایگاه داده احتمال تبدیل هر یک از دستورالعمل‌های موجود در فایل به یک دستورالعمل مجاز با یک تزریق اشـــکال تک‌بیتی یا چندبیتی محاســـبه‌گردید. به‌عنوان مثال فرض‌کنید دســتورالعملی با کد ۱۰۱۰ با تزریق اشـکال به با ارزش‌ترین بیت به ۰۰۱۰ تبدیل‌می شود. بر ا ساس اطلاعات موجود در پایگاه داده وجود یا عدم وجود دســـتورالعمل مجاز با این کد بررســـی‌می‌گردد. اطلاعات حا صل از این برر سی به فایل د ستورالعمل ا ضافه‌می شود. در ادامه فایل‌های دســتورالعمل همه برنامه‌های محک را با یکدیگر ادغام و به ازای هر دستورالعمل یک رکورد ایجادگردید.

پس از تکمیل فایل دســـتورالعمل اطلاعات مربوط به هر یک از رکوردهای موجود در فایل گزارش بر اســـاس این فایل تکمیل‌می‌گردد. آخرین فیلدی که به فایل گزارش اضـافه‌می‌شـود وزن هر دسـتورالعمل است که بر اساس فایل اسمبلی محاسبه و افزوده‌می‌گردد. لازم به ذکر ا ست که وزن هر د ستورالعمل، ن سبت تعداد تکرار هر د ستورالعمل به کل تعداد دســـتورالعمل‌های برنامه باشـــد. جدول ۱ کلیه ویژگی‌های مستخرج از برنامه‌ها را نشان‌می‌دهد.

جدول ۱: ویژگی‌های مستخرج از برنامه‌ها که در درخت تصمیم‌گیری استفاده شده اند

نام	توضیح
Cod	کد هگزا دسیمال
Len	طول دستور بر حسب بایت (۲بایتی، ۴ بایتی، ...)
Wi	وزن دستور در برنامه
Pf	احتمال تبدیل دستور به یک دستور دیگر در اثر برخورد اشکال
Tcorrect	احتمال خروجی صحیح پس از برخورد اشکال
Scorrect	احتمال خروجی صحیح پس از برخورد اشکال تک بیتی
Sreg	احتمال بروز SDC پس از برخورد اشکال تک بیتی به پارامتر، ثبات

۳،۴. آموزش درخت تصمیم‌گیری

فایل گزارش تولیدشده بخش قبلی به‌عنوان ورودی الگوریتم‌های یادگیری ماشین استفاده‌می‌شود. هدف تخمین نرخ رخداد SDC با استفاده از ویژگی‌های شرح‌داده‌شده‌است، بخشی بر اساس خروجی‌های ابزار تزریق اشکال و پایگاه داده به‌دست‌آمده‌است و بخش دیگر آن از سطح برنامه جمع‌آوری‌می‌شود. در این پژوهش جهت تخمین نرخ رخداد SDC هر دستورالعمل از مدل درختی M5 نرم‌افزار WEKA[۳۳] که در دانشگاه Waikato نیوزلند توسعه داده‌شده، استفاده‌گردیده‌است [۴۱]. داده‌ها به شکل اعتبارسنجی[۳۴] متقابل با ۱۰ تا[۳۵] آموزش و آزموده شده‌اند. نتایج این یادگیری ماشین در بخش بعدی قابل‌مشاهده است. علت انتخاب M5 برای تخمین نرخ رخداد SDC هر دستورالعمل ویژگی‌های آن در دسته‌بندی است مثل قابلیت تفسیر و فهم ساده، هرس‌کردن ویژگی‌هایی که در اشتقاق درخت دخالت ندارند و سادگی در پیاده‌سازی است. موارد ذکرشده باعث بالاتر رفتن سرعت محاسبات می‌گردد. در نهایت باعث کاهش چشم‌گیر میانگین قدرمطلق[۳۶] خطا شده‌است. در M5 نتایج به صورت قوانین "اگر-آنگاه" ترجمه و ارائه‌می‌گردند. سپس کلیه فرمولاسیون دسته‌بندی M5 برای برنامه‌های محک استخراج‌شده و با توجه به تخمین نرخ رخداد SDC اقدام به انتخاب فرمولاسیونی با کمترین مقدار میانگین قدر مطلق خطای قابل‌مشاهده در روابط (۳) و (۴) به‌عنوان تابع تخمین نرخ رخداد SDC همان‌طور که در رابطه (۵) مشخص‌شده‌است، انتخاب می‌شود:

$$\bar{y}_m = \frac{1}{N_m}\sum_{i\in N_m} y_i \qquad (۳)$$

$$MAE = \frac{1}{N_m}\sum_{i\in N_m} |y_i - \bar{y}_m| \qquad (۴)$$

$$\begin{aligned}
&IF\ Len \le 2\ THEN\\
&\quad PSDC = 0.0002 \times Len + 0.0117 \times Wi\\
&\qquad + 0.0331 \times Tcorrect\\
&\qquad - 0.0439 \times Scorrect\\
&\qquad + 0.0034 \times Sreg\\
&ELSE\\
&IF\ Tcorrect\ 0.019\ \ AND\ \ Cod21\ \ THEN\\
&\quad PSDC = 0.0001 \times Cod + 0.0025 \times Pf\\
&\qquad + 0.0056 \times Sreg + 0.0161\\
&ELSE\\
&IF\ Cod > 7.5\ \ AND\ \ Wi \le 0.026\ \ THEN\\
&\quad PSDC = 0.0538 \times Wi - 1.9157\\
&\qquad \times Scorrect - 0.0024\\
&\qquad \times Sreg + 0.0418\\
&ELSE\\
&\quad PSDC = 0.4675 \times Wi - 0.0075
\end{aligned} \qquad (۵)$$

۲،۴. محافظت از برنامه

برای تعیین تعداد دستورالعمل‌ها جهت تکرار با توجه به محدودیت سربار به روشی نیاز است که درعین پوشش بالای SDC کمترین سربار را به سیستم تحمیل‌کند. از این رو از تکنیک دانه‌بندی دستورالعمل‌ها به‌ترتیب نرخ رخداد SDC استفاده‌می‌شود. پیش‌بینی نرخ رخداد SDC به واسطه مدل استخراجی با توجه به ویژگی‌های استخراج‌شده قبلی از برنامه است. هدف مقاله این است که دیگر نیازی به تزریق اشکال برای تعیین نرخ رخداد SDC در هر برنامه نباشد. به عبارت دیگر بر اساس آزمایشات اولیه بر روی تعدادی از برنامه‌های محک الگویی جهت دستورالعمل‌ها شناسایی‌گردد که بر اساس آن بتوان در سایر برنامه‌های محک بدون تزریق اشکال و صرفاً بر اساس اطلاعات برنامه و دستورالعمل نرخ رخداد SDC را تخمین‌زد.

۵. نتایج و بحث

برای ارزیابی رفتار سیستم‌ها در برابر خطاهای تک‌بیتی و چندبیتی در داده‌ها و دستورالعمل‌ها، ابزاری جدید برای تزریق اشکال در محیط Python و پایگاه داده MySQL طراحی و پیاده‌سازی‌شده‌است. به منظور دستیابی به نتایج دقیق‌تر و بدون وابستگی به نوع سیستم، تلاش‌شده تا برنامه‌های محک از دسته‌های مختلف انتخاب‌شود تا رفتار آن‌ها در برابر خطاهای گذرا ارزیابی‌شوند.

۱،۵. محیط برنامه

محیط اجرایی مورد استفاده در این پژوهش در جدول ۲ خلاصه‌شده‌است.

در ابزار تولیدی اشکالات به هر دو بخش عملوند و عملگر دستورالعمل تزریق‌می‌شود. این اشکالات به‌صورت تک‌بیتی و چندبیتی (۲ تا ۶ بیت) بر اساس نتایج محققان [۴۲] و [۴۰] که در شکل ۲ قابل‌مشاهده است، صورت‌می‌گیرد. نحوه انتخاب محل اصابت (به عملگر یا عملوند کدام دستور) اشکال به‌صورت تصادفی و با استفاده از توزیع یکنوا است. در نهایت برای بررسی همه‌جانبه سیستم‌های نرم‌افزاری مجموعه برنامه محک MiBench [۴۳] انتخاب و تحت تزریق اشکال قرارگرفته‌است. جزئیات برنامه محک مورد استفاده در این بخش پژوهش در جدول ۳ قابل‌مشاهده است.

جدول ۲: محیط اجرایی

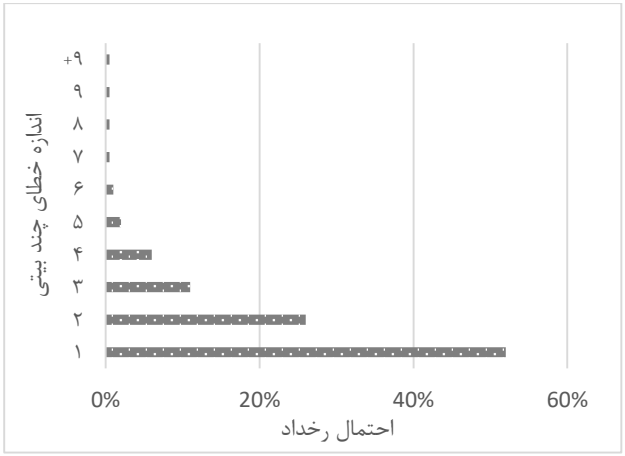
محیط برنامه	
سخت افزار	CPU i7 Intel 1.7–2.9 GHz RAM 6 G
سیستم عامل	Ubuntu 16.1 Linux 64 bit
نرم افزار	ابزار تزریق اشکال پیاده‌سازی شده با زبان پایتون
پایگاه داده	My sql برای جمع آوری جزئیات عملگرها و عملوندها و کد هگزادسیمال آن‌ها

جدول ۳: برنامه محک مورد استفاده در این پژوهش

MiBench			
Auto/Industrial	Network	Telecomm	Other
Q sort	Dijksttera (L)	FFT	DFS

هدف این پژوهش ارائه روشی برای پوشش اشکال‌های یک و چند بیتی بدون تزریق اشکال و با درنظرداشتن سربار کارایی است، لذا اقدام به انتخاب چند برنامه محک برای آزمون شد.

لازم به توضیح است که در روش‌های قبلی چند برنامه محک را به عنوان داده یادگیری با تزریق اشکال مستقیم درنظرگرفته‌اند و چند برنامه محک را بدون تزریق اشکال و به‌عنوان داده آزمون درنظرگرفته‌اند. اما در این پژوهش با تزریق اشکال به چند برنامه محک، نرخ میانگین رخداد SDC هر دستورالعمل محاسبه‌شده، سپس بعد از اضافه‌نمودن این میانگین برای هر دستورالعمل به برنامه‌های محک اقدام به استفاده از یادگیری ماشین شد.



شکل ۲: توزیع خطاهای چندبیتی در SRAMهای ۴۵ نانومتر [۴۰].

سپس با استفاده از مرتب‌کردن دستورها مبتنی بر نرخ رخداد SDC پیش‌بینی‌شده توسط درخت تصمیم‌گیری M5 rule اقدام به مشخص نمودن مجموعه دستورها می‌شود که تنها، خطای داده مدّنظر قرارگرفته‌شده‌است.

جدول ۴: معیارهای ارزیابی درخت M5 در ازای هر یک از برنامه‌های محک

برنامه محک	ضریب وابستگی	میانگین قدر مطلق خطا (برحسب درصد)	خطای جذر میانگین مربعات
DFS	۰،۹۵۴۳	۰،۲۰	۰،۰۰۴۵
Dijkstra	۰،۹۴۷۴	۰،۱۷	۰،۰۰۵۸
Fft	۰،۹۱۵۱	۰،۱۴	۰،۰۰۴۹
Qsort	۰،۸۸۱۱	۰،۱۸	۰،۰۰۴۷

همانگونه که در جدول ۴ مشاهده‌می شود، ضریب همبستگی[۳۷] در همه موارد بالاتر از ۸۸٪ است و میانگین قدر مطلق خطا حداکثر ۰،۱۷٪ است. با توجه به نکات ذکرشده همچنین سایر معیارهای ارزیابی موجود در جدول ۴ می‌توان نتیجه‌گرفت، درخت M5 دارای توانایی پیش‌بینی با دقت بالا به‌عنوان یک روش یادگیری ماشین است.

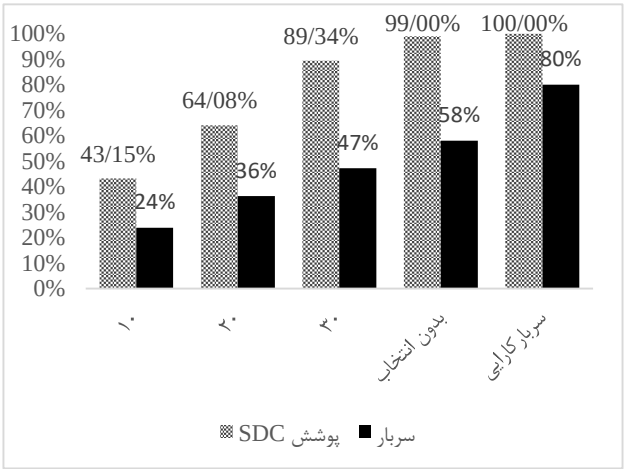
علاوه بر این، چهار روش پیشرفته به‌عنوان روش‌هایی برای مقایسه دقت پیش‌بینی نرخ رخداد انتخاب‌گردید که در جدول ۵ قابل‌مشاهده هستند، از جمله SEDSVR[۴۴] برای پیش‌بینی آسیب‌پذیری SDC توسط مدل SVR، SDCPpredictor[۳۳] برای شناسایی دستورالعمل‌ها با آسیب‌پذیری SDC توسط Random Forest، SDIFI [۳۶] و DFRMR [۳۷] یک مدل رگرسیون جنگل عمیق بهبودیافته‌است و ثابت شده‌است که روش پیشنهادی از اکثر روش‌های عمومی مبتنی بر یادگیری عمیق بهتر عمل‌می‌کند و نیاز به تنظیم مجدد توسط کاربر و مبتنی بر برنامه‌های مختلف را ندارد.

برای ارزیابی بهتر عملکرد الگوریتم، از شاخص ارزیابی کلاسیک ریشه میانگین مربعات خطا [۳۸] برای انتخاب پارامترهای SED-SVR، SDCPredictor، DFRMR و SDIFI استفاده‌می‌شود. هرچه امتیاز کمتر باشد، دقت پیش‌بینی بالاتر است. پارامترهای مورد استفاده در همه مدل‌ها با جستجوی شبکه یا آزمایش تنظیم‌می‌شوند. در SEDSVR

ضریب مجازات c 1.0، نوع هسته rbf و پارامتر هسته آن σ 0.003 است. SDCPpredictor بر اساس جنگل تصادفی از ۸۰۰ درخت تشکیل‌شده‌است. در حداکثر عمق ۸ و نرخ یادگیری ۰،۰۹، در DFRMR، پنجره ها ۵، تلورانس ۰ است. در روش SDIFI ، د قت پیش‌بینی را با تنظیم پارامترهای تعداد برگ و نرخ یادگیری بهبود می دهند. اما روش پیشنهادی این پژوهش نیازمند تنظیم هیچ پارامتری نیست.

جدول ۵: عملکرد خطای جذر میانگین مربعات برنامه‌های آزمایشی

معیار	محک	SED-SVR	SDC Predictor	DFRMR	SDIFI	روش پیشنهادی
خطای جذر میانگین مربعات	DFS	۰،۱۵۰۳	۰،۰۸۵۹	۰،۰۸۸۳	۰،۰۶۳۹	۰،۰۰۴۵
	Dijkstra	۰،۱۹۹۱	۰،۱۴۹۹	۰،۱۳۹۱	۰،۱۲۲۲	۰،۰۰۵۸
	Fft	۰،۳۴۹۵	۰،۳۹۸۲	۰،۳۴۳۲	۰،۳۰۱۸	۰،۰۰۴۹
	Qsort	۰،۲۰۰۶	۰،۰۹۲۵	۰،۱۶۱۵	۰،۰۴۶۶	۰،۰۰۴۷



شکل ۳: میانگین پوشش اشکال در سطوح دانه بندی مختلف و سربار کارایی روش پیشنهادی با تکرار کامل

در شکل ۳ م شاهده‌می شود که تکنیک دوتایی کامل به‌طور میانگین حدود ۸۰٪ سربار کارایی را به سیستم تحمیل‌می‌کند درحالی‌که این پژوهش موفق‌شده با میانگین ۵۸٪ سربار کارایی حدود ۹۹٪ یک پوشش اشکال کامل را به‌دست‌آورد. از سوی دیگر کمترین میزان پو شش ا شکال در بین نتایج کار این پژوهش ۴۳،۱۵٪ پو شش با سربار ۲۴٪ ا ست به این معنی که در مقایسه با دوتایی کامل با تحمیل حدود یک چهارم سربار کارایی نزدیک به ن صف پو شش ا شکال به‌د ست‌آمده است. باید متذکرشد که در این پژوهش تنها به پوشش اشکالات خطای داده پرداخته‌شده و اشکالات خطای کنترل جریان به دلیل روش‌های مقابله ارائه‌شده خارج از بررسی این پژوهش است. سربار کارایی دوتایی کردن کامل نیز با درنظرگرفتن این فرض می‌با شد تا عدالت در مقایسه رعایت‌گردد.

در مرحله بعد، به‌طور تصادفی ۳۰۰۰ خطا بر ۴ محک تزریق و قابلیت تشخیص خطا مدل پیشنهادی را با پوشش SDC و سربار کارایی حاصل از تشخیص خطا SDC ارزیابی‌می‌گردد. آسیب‌پذیری SDC پیش‌بینی شده د ستورالعمل‌ها رتبه‌بندی‌می شود، اولین د ستورالعمل‌ها بر اساس دانه‌بندی ۱۰، ۲۰ و ۳۰ درصد به ترتیب برای افزونگی انتخاب و پوشش SDC و سربار کارایی به‌دست‌آمده با روش ارائه‌شده DFRMR و SDIFI مقایسه‌می‌گردد [۱۳].

مطالعه Fang و همکارانش [۳۶] با نام SDIFI و Li و همکارانش [۳۷] با نام DFRMR بسیار به کار این پژوهش شبیه است. زیرا از هوش مصنوعی درخت تصمیم‌گیری LightGBM استفاده‌کرده‌اند و محیط شبیه‌سازی و برنامه‌های محک مورد استفاده همانند پژوهش پیش رو است. لذا در مقایسه با کار Fang و همکارانش و Liu و هم کارانش که جزو تلاش هایی است که منجر به ن تایج مطلوبی شده‌است. صرف‌نظر از خلأ بررسی خطاهای چندبیتی در کنار خطاهای تک‌بیتی، در این دو مقاله همچون پژوهش پیش رو ارزیابی بر اساس سطوح دانه‌بندی ۱۰٪، ۲۰٪ و ۳۰٪ بالاترین نرخ رخداد بوده، در جدول ۶ قابل‌مشاهده است.

همانگونه که م شاهده‌می شود هر دوکار در دانه‌بندی ۳۰٪ با تحمیل سربار کمتر موفق به ارائه پوشش اشکال مناسبی شده‌اند اما در مقایسه با کار این پژوهش در دانه‌بندی‌های مشابه درحالی‌که پوشش بهتری داشته‌است سربار کارایی کمتری نیز تحمیل‌نموده‌است. با این تفاوت که روش ارائه‌شده دیگر نیازی به تزریق اشکال مجدد ندارد و در ضمن کلیه ا شکال تک و چندبیتی و ا شکالات به کلیه داده‌ها و د ستورالعمل ها مدّنظربوده‌اند. باید متذکرشد یکی از فاکتورهایی که در این مطالعه برر سی‌نشده‌ا ست امکان تبدیل یک د ستور معتبر به یک د ستور معتبر دیگر در صورت برخورد انواع خطا به بیت‌های مختلف است. ممکن ا ست یک د ستور که افزونگی بالایی هم دا شته‌ا ست در صورت برخورد خطا به یک د ستور نامعتبر تبدیل‌گردد و برنامه دچار خرابی یا شکست گردد. که در پژوهش حال حاضر فاکتور احتمال

جدول۶: نتایج روش‌های قابل‌مقایسه

سطح دانه بندی	SDIFI		DFRMR		ایده پیشنهادی	
	پوشش SDC	سربار	پوشش SDC	سربار	پوشش SDC	سربار
٪۱۰	٪۴۱,۶	٪۲۵,۱	٪۳۹,۵	٪۲۲,۵	٪۴۳,۱۵	۲۳,۸
٪۲۰	٪۷۱,۳	٪۳۶,۹	٪۶۸,۳	٪۴۱,۲	٪۶۴,۰۸	۳۶,۳
٪۳۰	٪۸۸,۴	٪۴۷,۸	٪۸۶,۲	٪۵۰,۳	٪۸۹,۳۴	۴۷,۲

تبدیل یک دستور در صورت بروز خطا به دستورالعمل دیگر نیز به عنوان یک ویژگی مد نظر واقع‌شده‌است.

در آخر به شکل مختصر می‌توان مقایسه کلی از روش ارائه‌شده در این پژوهش با دیگر رویکردهای ارائه شده در جدول ۷ ملاحظه‌نمود.

جدول ۷: مقایسه مختصر روش ارائه‌شده در پژوهش پیش‌رو با روش‌های پیشین

روش ارائه شده در این پژوهش	روش های پیشین
در نظر گرفتن اشکالات تک بیتی و چند بیتی(تا ۶ بیت)	در نظر گرفتن اشکالات تک بیت
تزریق اشکال به تمام بخش‌های یک برنامه	تزریق تنها به بخشی از برنامه برای مثال فقط به دستورات بارگذاری و ذخیره سازی، به فایل ثبات، دستورات مقصد
شناسایی بدون تزریق اشکال به برنامه جدید	شناسایی دستورات با تزریق اشکال به بخشی از آن برنامه یا کد مشابه
تکرار دستورات مبتنی بر ویژگی های ایستا	تکرار دستورات مبتنی بر ویژگی های ایستا و پویا

۶. نتیجه‌گیری

هدف اصلی این مقاله تشخیص و کاهش SDC بود. بدون شک لازمه دستیابی به یک نتیجه قابل‌قبول، بررسی جامع تلاش‌های صورت گرفته قبلی و تکمیل آن‌ها با مواردی است که به آن‌ها توجه نشده‌است. مطالعات و بر رسی کارهای پی شین ن شان‌داد که در تلاش‌های صورت گرفته قبلی تمرکز بر روی SDC با منشأ خطاهای تک‌بیتی بوده‌است. لذا به برر سی ا شکالات چندبیتی همرا ه با تک‌بیتی پرداخته شده‌ا ست. نکته حائز اهمیت بعدی هدف تزریق اشکال بود، در اغلب مطالعات قبلی تزریق اشکال تنها بر روی بخش از پیش‌تعیین‌شده‌ای از داده دستورات، به‌عنوان مثال دستورات بارگذاری و ذخیره‌سازی صورت گرفته‌بود و یا بخ شی از د ستورات به‌طورکلی کنار گذا شته شده‌اند. اما این مقاله تفاوتی بین دستورات بعنوان هدف تزریق اشکال قائل نشده‌است و به همه دستورات اشکال تزریق شده‌است. از سوی دیگر در کارهای پیشین اشکال‌ها تنها به بخش عملوندی یک دستورالعمل تزریق‌می‌گردید. که این رو یه بخشی از استعداد SDC را در یک دستورالعمل نادیده می‌گیرد. با توجه به اینکه با تعیین استعداد SDC دستورالعمل های برنامه، کاهش نرخ رخداد خرابی‌ها افزایش خواهدیافت. در نهایت جهت هر دستورالعمل یک‌سری ویژگی‌های ایستا مشخص‌شد که در کنار چند پارامتر پویا نظیر وزن دستورالعمل در برنامه، میزان استفاده از نتیجه دستورالعمل در برنامه، مکان و حضور یا عدم حضور دستورالعمل در حلقه و ... تخمینی از استعداد SDC هر د ستورالعمل به‌د ست‌آمد. به عبارت دیگر جهت تخمین ا ستعداد SDC دستورالعمل‌های یک برنامه جدید نیازی به تزریق اشکال به برنامه مذکور نیست نتایج تجربی نشان‌داد که روش ارائه‌شده از دقت پیش‌بینی آسیب پذیری SDC بالاتری نسبت به روش های دیگر برخوردار است. در نهایت پوشش خرابی داده ساکت بالاتر و سربار کارایی کمتری را با دانه‌بندی نرخ رخداد SDC به‌دست‌آورد و تحمل خطای بهتری داشت.

References

[1] S. A. Asghari, M. Binesh Marvasti, and M. Daneshtalab, “A software implemented comprehensive soft error detection method for embedded systems,” Microprocess. Microsyst., vol. 77, p. 103161, Sep. 2020, doi: 10.1016/J.MICPRO.2020.103161.

[2] S. A. Asghari, H. Taheri, H. Pedram, and O. Kaynak, “Software-based control flow checking against transient faults in industrial environments,” IEEE Trans. Ind. Informatics, vol. 10, no. 1, pp. 481–490, Feb. 2014, doi: 10.1109/TII.2013.2248373.

[3] B. Sangchoolie, K. Pattabiraman, and J. Karlsson, “One Bit is (Not) Enough: An Empirical Study of the Impact of Single and Multiple Bit-Flip Errors,” Proc. - 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, DSN 2017, pp. 97–108, Aug. 2017, doi: 10.1109/DSN.2017.30.

[4] Q. Lu, G. Li, K. Pattabiraman, M. S. Gupta, and J. A. Rivers, “Configurable Detection of SDC-causing Errors in Programs,” ACM Trans. Embed. Comput. Syst., vol. 16, no. 3, Mar. 2017, doi: 10.1145/3014586.

[5] M. Yakhchi, M. Fazeli, and . A. Asghari, “Investigation of the Effect of Burst Multi-bit Soft Errors on Control Flow and Data Error Behaviors of Embedded Systems,” J. Soft Comput. Inf. Technol., vol. 10, no. 2, pp. 68–81, 2021.

[6] M. Yakhchi, M. Fazeli, and S. A. Asghari, “Silent Data Corruption Estimation and Mitigation Without Fault Injection,” IEEE Can. J. Electr. Comput. Eng., vol. 45, no. 3, pp. 318–327, 2022.

[7] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, “GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications,” ISPASS 2014 - IEEE Int. Symp. Perform. Anal. Syst. Softw., pp. 221–230, 2014, doi: 10.1109/ISPASS.2014.6844486.

[8] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, “Quantifying the accuracy of high-level fault injection techniques for hardware faults,” Proc. Int. Conf. Dependable Syst. Networks, pp. 375–382, Sep. 2014, doi: 10.1109/DSN.2014.2.

[9] FengShuguang, GuptaShantanu, AnsariAmin, and MahlkeScott, “Shoestring,” ACM SIGARCH Comput.

Nucl. Sci., vol. 67, no. 1, pp. 321–327, Jan. 2020, doi: 10.1109/TNS.2019.2959975.

[25] M.-L. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou, "Understanding the propagation of hard errors to software and implications for resilient system design," p. 265, 2008, doi: 10.1145/1346281.1346315.

[26] V. B. Thati, J. Vankeirsbilck, J. Boydens, and D. Pissort, "Selective Duplication and Selective Comparison for Data Flow Error Detection," 2019 4th Int. Conf. Syst. Reliab. Safety, ICSRS 2019, pp. 10–15, Nov. 2019, doi: 10.1109/ICSRS48664.2019.8987731.

[27] F. Ayatolahi, B. Sangchoolie, R. Johansson, and J. Karlsson, "A study of the impact of single bit-flip and double bit-flip errors on program execution," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 8153 LNCS, pp. 265–276, 2013, doi: 10.1007/978-3-642-40793-2_24/COVER.

[28] B. Sangchoolie, F. Ayatolahi, R. Johansson, and J. Karlsson, "A study of the impact of bit-flip errors on programs compiled with different optimization levels," Proc. - 2014 10th Eur. Dependable Comput. Conf. EDCC 2014, pp. 146–157, 2014, doi: 10.1109/EDCC.2014.30.

[29] B. Sangchoolie, K. Pattabiraman, and J. Karlsson, "An Empirical Study of the Impact of Single and Multiple Bit-Flip Errors in Programs," IEEE Trans. Dependable Secur. Comput., 2020.

[30] N. Narayanamurthy, K. Pattabiraman, and M. Ripeanu, "Finding Resilience-Friendly Compiler Optimizations Using Meta-Heuristic Search Techniques," Proc. - 2016 12th Eur. Dependable Comput. Conf. EDCC 2016, pp. 1–12, Dec. 2016, doi: 10.1109/EDCC.2016.26.

[31] S. K. S. Hari, S. V. Adve, and H. Naeimi, "Low-cost program-level detectors for reducing silent data corruptions," Proc. Int. Conf. Dependable Syst. Networks, 2012, doi: 10.1109/DSN.2012.6263960.

[32] Q. Lu, K. Pattabiraman, M. S. Gupta, and J. A. Rivers, "SDCTune: A model for predicting the SDC proneness of an application for configurable protection," 2014 Int. Conf. Compil. Archit. Synth. Embed. Syst. CASES 2014, Oct. 2014, doi: 10.1145/2656106.2656127.

[33] Liu, L., Ci, L., Liu, W., Yang, H., 2019,"Identifying SDC-causing Instructions based on Random forests algorithm", KSII Transactions on Internet and Information Systems. Vol. 13.

[34] N. Yang and Y. Wang, "Identify Silent Data Corruption Vulnerable Instructions Using SVM," IEEE Access, vol. 7, pp. 40210–40219, 2019, doi: 10.1109/ACCESS.2019.2905842.

[35] N. A. Rink and J. Castrillon, "Trading fault tolerance for performance in AN encoding," ACM Int. Conf. Comput. Front. 2017, CF 2017, pp. 183–190, May 2017, doi: 10.1145/3075564.3075565.

[36] W. Fang, J. Gu, Z. Yan, and Q. Wang, "SDC Error Detection by Exploring the Importance of Instruction Features," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 12937 LNCS, pp. 351–363, 2021, doi: 10.1007/978-3-030-85928-2_28/COVER.

[37] C. Liu, J. Gu, Z. Yan, F. Zhuang, and Y. Wang, "SDC-causing Error Detection Based on Lightweight Vulnerability Prediction." PMLR, pp. 1049–1064, Oct. 15, 2019. Accessed: Aug. 22, 2022. [Online]. Available: https://proceedings.mlr.press/v101/liu19c.html

[38] C. Wang, N. Dryden, F. Cappello, and M. Snir, "Neural Network Based Silent Error Detector," Proc. - IEEE Int. Conf. Clust. Archit. News, vol. 38, no. 1, pp. 385–396, Mar. 2010, doi: 10.1145/1735970.1736063.

[10] S. K. Sastry Hari, R. Venkatagiri, S. V. Adve, and H. Naeimi, "GangES," ACM SIGARCH Comput. Archit. News, vol. 42, no. 3, pp. 61–72, Jun. 2014, doi: 10.1145/2678373.2665685.

[11] G. Li, Q. Lu, and K. Pattabiraman, "Fine-Grained Characterization of Faults Causing Long Latency Crashes in Programs," Proc. Int. Conf. Dependable Syst. Networks, vol. 2015-September, pp. 450–461, Sep. 2015, doi: 10.1109/DSN.2015.36.

[12] M. Pal, "M5 model tree for land cover classification," http://dx.doi.org/10.1080/01431160500256531, vol. 27, no. 4, pp. 825–831, 2007, doi: 10.1080/01431160500256531.

[13] A. Adams and L. Sterling, "AI '92," pp. 1–410, Dec. 1992, doi: 10.1142/9789814536271.

[14] Y. Wang and I. H. Witten, "Induction of model trees for predicting continuous classes," 1996, Accessed: Aug. 22, 2022. [Online]. Available: https://researchcommons.waikato.ac.nz/handle/10289/1183

[15] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," ACM SIGARCH Comput. Archit. News, vol. 35, no. 2, pp. 460–469, Jun. 2007, doi: 10.1145/1273440.1250719.

[16] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," Proc. - Int. Symp. High-Performance Comput. Archit., pp. 243–247, 2005, doi: 10.1109/HPCA.2005.37.

[17] B. Ghavami and M. Raji, "Soft Error Rate Estimation of VLSI Circuits," Soft Error Reliab. VLSI Circuits, pp. 9–23, 2021, doi: 10.1007/978-3-030-51610-9_2.

[18] G. Li, K. Pattabiraman, S. K. S. Hari, M. Sullivan, and T. Tsai, "Modeling Soft-Error propagation in programs," Proc. - 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, DSN 2018, pp. 27–38, Jul. 2018, doi: 10.1109/DSN.2018.00016.

[19] G. Li and K. Pattabiraman, "Modeling Input-Dependent error propagation in programs," Proc. - 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, DSN 2018, pp. 279–290, Jul. 2018, doi: 10.1109/DSN.2018.00038.

[20] J. Ma, Z. Duan, and L. Tang, "A Methodology to Assess Output Vulnerability Factors for Detecting Silent Data Corruption," IEEE Access, vol. 7, pp. 118135–118145, 2019.

[21] B. Fang, Q. Lu, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "EPVF: An enhanced program vulnerability factor methodology for cross-layer resilience analysis," Proc. - 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, DSN 2016, pp. 168–179, Sep. 2016, doi: 10.1109/DSN.2016.24.

[22] A. Thomas and K. Pattabiraman, "Error Detector Placement for Soft Computing Applications," ACM Trans. Embed. Comput. Syst., vol. 15, no. 1, Jan. 2016, doi: 10.1145/2801154.

[23] X. Wei, R. Zhang, Y. Liu, H. Yue, and J. Tan, "Evaluating the soft error resilience of instructions for GPU applications," Proc. - 22nd IEEE Int. Conf. Comput. Sci. Eng. 17th IEEE Int. Conf. Embed. Ubiquitous Comput. CSE/EUC 2019, pp. 459–464, Aug. 2019, doi: 10.1109/CSE/EUC.2019.00091.

[24] B. James, H. Quinn, M. Wirthlin, and J. Goeders, "Applying Compiler-Automated Software Fault Tolerance to Multiple Processor Platforms," IEEE Trans.

[42] A. Banaiyanmofrad, M. Ebrahimi, F. Oboril, M. B. Tahoori, and N. Dutt, “Protecting caches against multi-bit errors using embedded erasure coding,” Proc. - 2015 20th IEEE Eur. Test Symp. ETS 2014, Jun. 2015, doi: 10.1109/ETS.2015.7138735.
[43] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, “MiBench: A free, commercially representative embedded benchmark suite,” 2001 IEEE Int. Work. Workload Charact. WWC 2001, pp. 3–14, 2001, doi: 10.1109/WWC.2001.990739.
[44] J. Gu, W. Zheng, Y. Zhuang, and Q. Zhang, “Vulnerability Analysis of Instructions for SDC-Causing Error Detection,” IEEE Access, vol. 7, pp. 168885–168898, 2019, doi: 10.1109/ACCESS.2019.2950598..
Comput. ICCC, vol. 2018-September, pp. 168–178, Oct. 2018, doi: 10.1109/CLUSTER.2018.00035.
[39] I. Laguna, M. Schulz, D. F. Richards, J. Calhoun, and L. Olson, “Ipas: Intelligent protection against silent output corruption in scientific applications,” in 2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), 2016, pp. 227–238.
[40] M. Ebrahimi, P. M. B. Rao, R. Seyyedi, and M. B. Tahoori, “Low-Cost Multiple Bit Upset Correction in SRAM-Based FPGA Configuration Frames,” IEEE Trans. Very Large Scale Integr. Syst., vol. 24, no. 3, pp. 932–943, Mar. 2016, doi: 10.1109/TVLSI.2015.2425653.
[41] E. Frank et al., “Weka-a machine learning workbench for data mining,” in Data mining and knowledge discovery handbook, Springer, 2009, pp. 1269–1277.

[1] Reliability
[2] Soft Errors
[3] event Upset
[4] Radiation
[5] Electromagnetic Noise
[6] Power Supply Disturbance (PSO)
[7] Registers
[8] Embedded System
[9] Micro-processor
[10] Control Flow Error
[11] Data Error
[12] Transient Error
[13] Fault Injection
[14] Performance overhead
[15] Event rate
[16] Duplication
[17] resilience
[18] hang
[19] platform
[20] Application
[21] Performance Overhead
[22] Clock per instruction
[23] Data Duplication
[24] decision tree
[25] classification
[26] regression
[27] branch
[28] Data Mining
[29] Check pointing
[30] Back slice
[31] Random Forest
[32] Identify program vulnerable instructions
[33] Waikato Environment for Knowledge Analysis
[34] cross validation
[35] fold
[36] Mean absolute Error
[37] correlation coefficient
[38] root mean squared error