Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2023-IC-001

2023-IC-001

A Comparison of transformer and AR-SI Oracle For Control-CPS Software Fault Localization

Shiyu Zhang, Wenxia Liu, Qixin Wang, Lei Bu, Yu Pei

Technical Report 2023

A Comparison of transformer and AR-SI Oracle For Control-CPS Software Fault Localization

Shiyu, Zhang
The Hong Kong Polytechnic University
Hong Kong, China
shiyu-comp.zhang@connect.polyu.hk

Wenxia, Liu
Nanjing University
Nanjing, China
mf21330055@smail.nju.edu.cn

Qixin, Wang
The Hong Kong Polytechnic University
Hong Kong, China
qixin.wang@polyu.edu.hk

Lei, Bu
Nanjing University
Nanjing, China
bulei@nju.edu.cn

Yu, Pei
The Hong Kong Polytechnic University
Hong Kong, China
qixin.wang@polyu.edu.hk

***Abstract*—Control-CPSs are usually safety or mission critical, hence they demand thorough debugging. As nowadays control-CPSs reaching millions of lines of source code, traditional human-flesh debugging is no longer sufficient. We need automated *software fault localization* (SFL) to assist the debugging. In automated SFL, automatically generated test cases are fed to the control-CPS (or the simulator of the control-CPS), to generate thousands of cyber-subsystem code traces and physical-subsystem trajectories. Next, another automated program, aka *oracle*, is needed to label the correctness of these physical-subsystem trajectories (and hence cyber-subsystem code traces), even without knowing if there is a bug in the cyber-subsystem. Control-CPS oracle design is a known hard problem. To our best knowledge, AR-SI oracle (denoted as *AO* in the following) is the most widely adopted control-CPS oracle so far. On the other hand, recently, transformer emerges as a major game changer in the domain of time series prediction. As AO is also time series prediction based, people naturally wonder if transformers can also be used as control-CPS oracles; and if so, can it outperform AO. In this paper, we answer this question by comparing AO with an intuitive design of transformer control-CPS oracle (simplified as *TO* in the following). Our comparison results show that in terms of SFL accuracy and latency, the TO does not significantly outperform the AO; in terms of false positive rate, the AO performs significantly better; and in terms of false negative rate, the TO performs significantly better.**

***Index Terms*—Software Fault Localization, control-CPS, transformer, Oracle**

I. Introduction

Control-CPS refers to those software systems that interact with the physical world [1]. With the development of Industry 4.0, control-CPS is increasingly common in our life [2]. Typical control-CPSs include smart grids, autonomous automobile systems, medical monitoring, industrial control systems, robotics systems, etc [3]. The wide application of these control-CPS in everyday life may lead to some safety concerns. For example, an out-of-control autonomous automobile is likely to pose a safety threat to pedestrians on the road. Therefore, developers should conduct comprehensive debugging for these systems to ensure there are no safety or other severe problems [4].

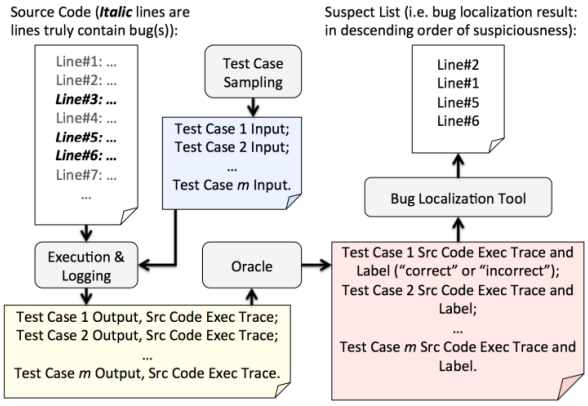


Fig. 1. Typical work flow of (program spectrum, statistics, etc) bug localization (aka software fault localization, simplified as "SFL") tools (quoted from [8])

As nowadays control-CPSs reaching millions of lines of source code, traditional human-flesh debugging is no longer sufficient. We need automated debugging tools. One major category of such tools is those for *software fault localization* (SFL), i.e. the tools to locate suspicious buggy lines in the source code. The SFL tools can be further categorized into various families [5]. Two of the main stream families are the program spectrum analysis SFL [6] and statistical analysis SFL [7]. Both families use the big data idea. A typical working flow of SFL and oracle is in Fig. 1, quoted from [8]. First we generate a large number (i.e. big data) of code traces (files contain blocks that program pass through; The blocks are usually in form of line numbers or function names, etc). Then, we examine these large number of code traces: if a program block often appears in the buggy code traces, but rarely appears in the correct code traces, then the program block is suspected to be buggy. A natural question then is, how to determine a code trace is buggy or correct. The tools to make such decisions is called *oracles* [9].

Oracle design difficulty is heavily application dependent.

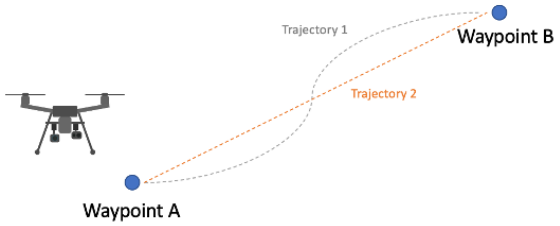


Fig. 2. Oracle of control-CPS output

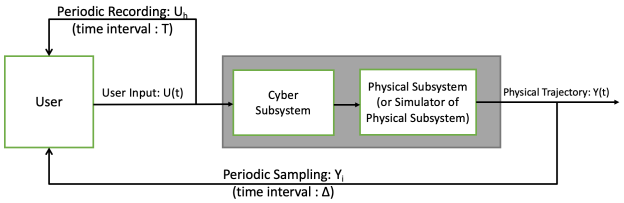


Fig. 3. Control-CPS Model

For some applications (such as sorting), oracles are very easy to design. But for most other applications, oracles are extremely difficult to design (this is the well known *oracle problem* [9]). Control-CPS oracle designs, unfortunately, usually belong to the latter. For example, for a control-CPS to fly a drone from point A to point B, the output of the control-CPS is the trajectory of the drone (see Fig. 2). The oracle needs to judge merely from this trajectory, the correctness of the control-CPS execution (note there could be many ways to fly from A to B, and not every buggy execution leads to drone crash).

So far, control-CPS oracle design is still mostly an open problem space. To our best knowledge, *auto-regressive system-identification* (AR-SI) oracle [10] is the most widely adopted control-CPS oracle (another alternative is to use human to label the correctness of control-CPS code traces but this is infeasible in the context of big data based automated SFL).

The AR-SI oracle believes that a buggy control-CPS will generate jitters in the outputted physical-subsystem trajectories (simplified as "*trajectories*" in the following). The method uses AR-SI to identify the math model of the control-CPS. Then, it uses the identified model to predict the future trajectory. If the difference between the actual trajectory and the predicted trajectory is smaller than a preset threshold, then the trajectory (and the corresponding code trace) is marked correct; otherwise, marked buggy.

AR-SI plays the function of time series prediction in the oracle approach mentioned above. Time series prediction, however, is also a focus application of AI. Recently, transformer [11] emerges as a game changer in the domain of AI based time series prediction [12]. Naturally, people wonder if transformers can be used to build control-CPS oracles, and how well they can perform compared to the AR-SI control-CPS oracles.

This paper aims to make an initial attempt to answer the above concerns.

We propose a transformer control-CPS oracle (simplified as *TO* in the following), and compare it with the AR-SI control-CPS oracle (simplified as *AO* in the following). Specifically, our TO assumes the existence of an earlier well-debugged version of the control-CPS. We use this well-debugged version to train a transformer based trajectory time series predictor. We use this predictor to predict the trajectories of a new version of the control-CPS (the to-be-debugged version). If the predicted trajectory is significantly different from the actual trajectory, the corresponding actual trajectory (and the corresponding code trace) is labeled buggy; otherwise, correct.

We tested the TO, and compared its performance with the AO. The results show that: i) in terms of SFL accuracy and latency, the TO and the AO are similar; ii) in terms of false positive rate, the AO performs significantly better; and iii) in terms of false negative rate, the TO performs significantly better.

The rest of the paper is structured as follows: Section II specifies the control-CPS context; Section III presents our TO solution; Section IV compares TO with AO; Section V discusses related work; and Section VI concludes the paper.

II. Problem Context

A. Control-CPS Model

In this paper, we focus on a control-CPS as shown in Fig. 3. We can instrument the source code in the cyber-subsystem of the control-CPS (specifically, add logging instructions to record the code traces), but we do not understand the source code, at least not before the automated SFL gives us a list of suspected buggy source code blocks.

Users send operation instructions to control-CPS. Receiving the operation instructions, the cyber-subsystem senses and actuates the physical-subsystem (or the simulator of the physical-subsystem). During operation, users are able to get the status information of control-CPS.

We regard the operation instructions given by the user as system input $U(t)$, $t \in [0, +\infty)$. User operation instructions are recorded before sending to the control-CPS, with a time interval of T. We denote U_h as the hth sampled $U(t)$. $U(t)$ stays unchanged during $[hT, (h+1)T)$. In other word, user send an operation instruction to control-CPS every time interval T.

$Y(t)$ is the physical trajectory generated by the system. We sample it periodically, and use Y_i to represent the ith sampled $Y(t)$. The time interval is Δ.

B. Assumptions

Assumption 1: With output $Y(t)$ frequency domain upper bound $F_{max} < \infty$(Hz), $Y(t)$ can be considered as continuous and differentiable. Corresponding to Assumption 1, control-CPS design has the following empirical rule on how to set Δ [10].

Assumption 2: The cyber system has to sample quickly enough ($\Delta \leq 1/(20F_{max})$) for the physical system to treat it like a continuous signal processor.

To repeat the experiment, we created a software (named "monkey") that would automatically send commands to the control-CPS to move to the target position every T time period. The "monkey" will also log the trajectory of the control-CPS at a time t while operating. To make the experiment easy to repeat, we make the control-CPS running in simulator. After each execution, related code trace and associated physical trajectory will be recorded. We need to oracle the correctness of the physical trajectory. Labeling the correctness of physical trajectories leads to the control-CPS oracle problem discussed in Section I.

III. Solution

A. Heuristics

A typical control-CPS model is Fig. 3. The input is U_h and output is Y_i. We can apply a sliding window with the size of p to Y_i since they are dispersed trajectories dots sampled from continuous trajectory $Y(t)$. As illustrated in Section II, $Y(t)$ represents trajectory and $U(t)$ represents user input. In our experiment, $U(t)$ is the target state (aka *reference state*) that users want the physical-subsystem to reach at time t. $U(t)$ hence is also called the *reference trajectory*.

For any $Y(t)$, there must be a relevant $U(t)$. By concatenating $Y(t)$ and $U(t)$, we will get $Y'(t) \stackrel{\text{def}}{=} (Y(t), U(t))$. Suppose the window size is p, then we can define a small part of trajectory containing waypoint information:

$$T_q \stackrel{def}{=} \{Y'_{q-p+1}, Y'_{q-p+2}, ..., Y'_q\}, Y' \in \mathbb{R}^d, p \geq q \quad (1)$$

T_q here represents the qth sliding window of trajectory (together with waypoint information). A transformer $f_\theta : \mathbb{R}^{n\times d} \rightarrow \mathbb{R}^{n\times d}$ "transforms" a collection of n objects in $\mathbb{R}^d$ to another collection of n objects in $\mathbb{R}^d$ [13]. We built a transformer to predict the next sliding window of trajectory using the previous one

$$\hat{T}_{q+1} = f_\theta(T_q) \quad (2)$$

$$\hat{T}_{q+1} \stackrel{def}{=} \{\hat{Y}'_{q-p+2}, \hat{Y}'_{q-p+2}, ..., \hat{Y}'_{q+1}\}, Y' \in \mathbb{R}^d, p \geq q \quad (3)$$

In (3), $\hat{T}_{q+1}$ represents the predicted sliding window, $\hat{Y}'$ represents the predicted items in the sliding window. $\hat{Y}_{q+1}$ represents the predicted trajectory combing with the relevant waypoint. Taking our the first few dimensions of $\hat{Y}'$, we will get Y_{q+1}, which represents the predicted trajectory point. When Y_{q+1} is available, we know the

$$transformer\ prediction\ error : e_{q+1} \stackrel{def}{=} \hat{Y}_{q+1} - Y_{q+1} \quad (4)$$

Heuristics 1: F_{max}^{actual} stands for the actual maximal frequency component of the continuous output of control-CPS. Normally, a control-CPS's output should have $F_{max}^{actual} \leq \frac{F_{max}}{10}$, or equivalently $\frac{10}{F_{max}} \leq \frac{1}{F_{max}^{actual}}$ [10]. For instance, despite the fact that an engine may shake at a maximum frequency of 100Hz, a typical design would not permit such high shaking (under controlled sounds). The design must keep the shaking to under 10Hz.

Heuristics 2: When $F_{max}^{actual} < \infty(HZ)$, $Y(t)$ can be considered as linearizable in small enough sliding time window ($p\Delta \leq 1/(20F_{max}^{actual})(sec)$) [14] [5]. After training, the transformer model will predict a reasonable state value based on the training set. the transformer prediction error magnitude (when $p\Delta \leq 1/(20F_{max}^{actual})$) should be small; a magnitude outlier suggests that something unusual—and hence probably buggy is taking place. The transformer prediction error can serve as an oracle in this regard.

Now let us discuss the setting of p. With **Heuristics 1**, **Heuristics 2**, and **Assumption 2**, we have if $p = 10$, then $p\Delta \leq 10/(20F_{max}) \leq 1/(20F_{max}^{actual})$. In this way, the prerequisite for **Heuristics 2** holds. Hence, by setting p to 10, we can apply **Heuristics 2** to the oracle for control-CPS SFL.

B. transformer Oracle: solution

Based on the heuristics of Section III-A, we proposed the oracle and code trace preparation methodology (following the overall design of the AR-SI oracle approach proposed in [10]). Our new approach is called "*transformer Oracle Solution*", as shown in Fig. 4.

Step1 **Task1**: Simulate the control-CPS using the simulation platform of Fig. 3. Log the physical trajectory $\{Y_i\}$, user input trace $\{U_h\}$, and code trace θ.
Task2: Run the pre-trained transformer model in parallel with the control-CPS simulate. For each new physical trajectory sample Y_{i+1} from the simulation, calculate the transformer prediction error e_{i+1} via Exp. 4 and log it.

Step2 When the simulation ends, check if the transformer prediction error trace $\{e_i\}$ contains $outlier(s)$ [15] [16]. If so, label θ as "incorrect"; otherwise label θ as "correct".

Step3 If enough code traces are collected, terminate; otherwise go to Step1.

Fig. 4. transformer Oracle Solution

Another classic automatic oracle for control-CPS is "*AR-SI oracle approach*" [10]. The only difference is replacing transformer in **Step1-Task2** with AR-SI, as shown in Fig. 5.

IV. Evaluation

Ardupilot is a very popular test-bed in the CPS field. It has more than 300,000 lines of code and is a very complicated open-source CPS [17]. Applicable devices for Ardupilot include Multirotor drones, Helicopters, Rovers, Boats, etc. ArduCopter is the multirotor drone part of Ardupilot. In this

Step1 **Task1**: Same as Fig. 4-**Step1-Task1**
Task2: Run AR-SI in parallel with the control-CPS simulation. For each new physical trajectory sample Y_{i+1} from the simulation, calculate the AR-SI prediction error e_{i+1} via Exp. 4 and log it.
Step2 Same as Fig. 4-**Step2**
Step3 Same as Fig. 4-**Step3**

Fig. 5. AR-SI Oracle Solution

section, we evaluate the proposed Transformer-based oracle (*TO*) and up-to-date ARSI-based oracle (*AO*) on ArduCopter with three SFL tools, Tarantula, Crosstab and Ochiai. All oracles are implemented in Python. Guided by Table I, we designed 27 bugs to inject into the ArduCopter cyber subsystem. Totally, we generate 37 versions (i.e. 37 subjects) of buggy ArduCopter systems. Each of 27 buggy subjects contains one of the candidate artificial bugs and each of the rest 10 subjects contains 3 candidate artificial bugs.

A. Evaluation Method

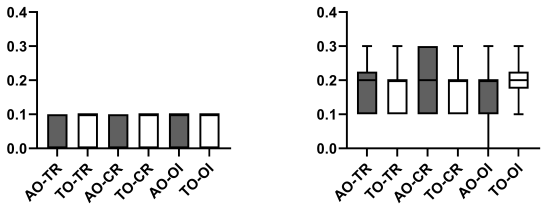
For each buggy subject, we generate 10000 code traces from ArduCopter emulation platform. The traces will be labeled by oracle, and then sent to three SFL tools, Tarantula, Crosstab and Ochiai (TR, CR and OI). In later evaluation, for example, TO-TR stands for Transformed-based oracle with Tarantula for SFL, while AO-CR stands for ARSI-based oracle with Crosstab for SFL.

We use box plot to show the experimental results clearly. A box plot, also known as a box and whisker plot, is a standardized way of displaying the dataset based on the five-number summary: the minimum, the maximum, the sample median, and the first and third quartiles [19]. We compare TO and AO for the control-CPS testbed ArduCopter from the following aspects.

Following the design evaluation matrix of [10], we choose accuracy and latency as our evaluation matrix. SFL tools will give the users a suspect list $S = (s_1, s_2, \cdots, s_l)$, where $s_i (i = 1, \cdots, l)$ is the ith suspected (s_1 is the most suspected) source code block. Given that B is the set of truly buggy blocks. Then *accuracy* refers to $|\{s_i | s_i \in B\}|/l$; *latency* refers to $min\{i | s_i \in B\}$.

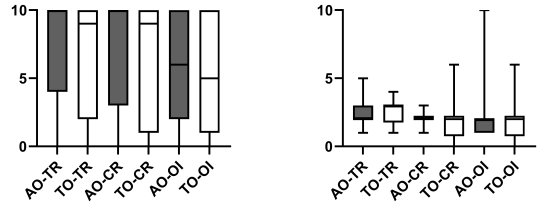
TABLE I
COMMON BUGS-IN-THE-FIELD [18]

Type	Description
WPFV	Wrong Variable used in Parameter of Function call
WVAV	Wrong Value Assigned to Variable
MVAE	Missing Variable Assignment using Expression
MFC	Missing Function Call
MIA	Missing IF construct Around statements
MVIV	Missing Variable Initialization using a Value
MVAV	Missing Variable Assignment using a Value
MIFS	Missing IF construct plus Statements
MIEB	Missing IF construct plus statements plus ELSE Before statements
MLC	Missing a Logic Clause in branch condition
MLPA	Missing small and Localized Part of the Algorithm
WAEP	Wrong Arithmetic Expression in Parameter of function call



(a) 1-bug subjects' trials (b) 3-bug subjects' trials

Fig. 6. Accuracy (the higher the better)



(a) 1-bug subjects' trials (b) 3-bug subjects' trials

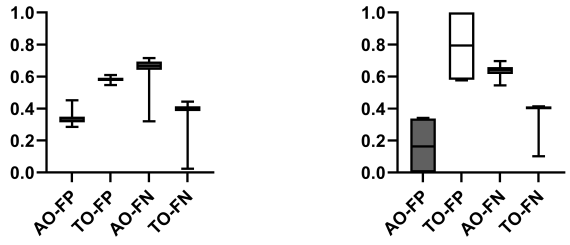
Fig. 7. Latency (the lower the better)

RQ1: SFL quality. How do TO and AO impact the quality of SFL from the perspective of accuracy and latency?
RQ2: Raw oracle quality. How do TO and AO impact the quality of raw oracle from the perspective of false positive rate and false negative rate?
RQ3: Significant difference. What is the significant difference between TO and AO through T-test?

1) Study 1: SFL quality.: The results related to accuracy and latency are plotted in Fig.6 and Fig.7 respectively. From the perspective of accuracy, our results demonstrate that TO and AO perform similarly in 1-bug subjects' trials. In 3-bug subjects' trials, with TR and CR, AO seems to have a slightly better performance, while with OI, TO is slightly better than AO. From the perspective of latency in Fig.7, it is clear that TO has a slightly better performance than AO. For 1-bug subjects' trials, the range of TO and AO are all from 0 to 10. Besides, the median of TO is smaller than AO in each of three SFL tools, which indicates that TO can find bugs minutely faster in some cases than AO. For 3-bug subjects' trials, TO has a little bit advantage over AO with CR and OI.

Answer to RQ1: AO and TO perform similarly in SFL quality, but TO has a slightly better performance than AO in terms of latency (not significant).

2) Study 2: Raw oracle quality.: The result related to the false positive rate and the false negative rate is plotted in Fig.8. The labels on X-axis represent the oracle and relative quality metrics. FP: false positive rate. FN: false negative rate. For



(a) 1-bug subjects' trials (b) 3-bug subjects' trials

Fig. 8. Oracle false positive rate and false negative rate

example, TO-FP stands for Transformer-based oracle and false positive rate. It is obvious that AO has a lower false positive rate and TO has a lower false negative rate, which implies that TO is more willing to label the traces as buggy.

Answer to RQ2: AO has a lower false positive rate and TO has a lower false negative rate.

3) Study 3: Significant difference.: The significance of TO and AO are quantified by the p-values [20] [21] and effect size (ES) values [22] in Table II. P-values can measure the probability that an observed difference could have occurred just by random chance. The lower the p-value, the greater the statistical significance of the observed difference. Effect size can compare the magnitude of differences. We calculate by Cohen's d [23], a measure relating the mean difference to variability. We regard the absolute value of effect size over 0.4 as at least medium difference magnitude. As shown in Table II, except for three cells in light gray, the p-values are mostly above 5%, and except four cells in drak gray, the absolute value of effect size are below 0.4, hence there is no significant difference between TO and AO.

Answer to RQ3: There is no significant difference between AO and TO in terms of accuracy and latency.

TABLE II
QUALITY OF AO VS TO

Metric		1-bug subjects		3-bug subjects	
		p-value	ES	p-value	ES
Accuracy	TR	10.3%	-0.15	67.8%	0.08
	CR	18.5%	-0.11	8.1%	0.21
	OI	32.7%	-0.04	19.3%	-0.21
Latency	TR	13.0%	0.14	83.2%	-0.05
	CR	23.1%	0.12	71.6%	0.08
	OI	27.5%	0.06	57.6%	0.14
FPR		<0.1%	-0.98	6.4%	-0.86
FNR		<0.1%	0.91	<0.1%	0.84

B. transformer Training

The structure of transformer basically the design of [11]. The only difference is, our transformer didn't embed $\hat{Y}_i'$ discussed in Section III-A. Instead, it treat $\hat{Y}_i'$ as attention values directly. We generated 3000 trajectories using ArduCopter3.6.10 (a release version). Each trajectory consists of 240 Y_i', we can organize these data into 230 (T_q, T_{q+1}) groups. In total, we got 690,000 pairs of training data. We trained our transformer model with these data for 30 epoches. We chose MSEloss [24] as our loss function. After 30 epoches of training, the loss dropped from 58.96 at the beginning and stabilized at around 0.2.

V. RELATED WORK

In this paper we focus on automatic oracle tools for control-CPS. The key is to identify whether the output trajectory of control-CPS is inline with user's expect (user's input). Using AR-SI as the automatic oracle for control-CPS [10] is a successful attempt, it uses AR-SI and sliding window to predict trajectory and then check outlier. Another successful trial is Mithra [17], using mature 3-step anomaly detection as control-CPS oracle. The result shows it has better performance than AR-SI approach. However, implementing Mithra requires more complicated steps than AR-SI, AR-SI is more convenient and thus more universal.

Metamorphic testing [25] is also very efficient as an oracle. The main idea is using the metamorphic relationships (the relationship between the software input change and output change during multiple program executions) as the oracle. Metamorphic testing was used in [26] to detect faults in embedded software. It's also an effective solution for oracle problems.

VI. CONCLUSION

In this short paper, we proposed an automatic oracle method based on transformer. We compared the transformer oracle (TO) and AR-SI oracle (AO) on classic control-CPS test-beds with SFL of injected bugs. The results show, TO and AO perform similarly in SFL quality, but TO has a slightly better performance than AO in terms of latency; AO has a lower false positive rate and TO has a lower false negative rate. As for overall performance, there is no significant difference between TO and AO. AO can achieve good performance without pre-training, and it is very convenient to use; In order for TO to defeat AO, it may need better design or more training data. New research opportunities may include combining the physical laws with user input to make the transformer Oracle's judgments more accurate.

VII. ACKNOWLEDGEMENT

We thank Dr. Zhijian He from The Hong Kong University of Science and Technology and Mr. Shaopeng Xing from Nanjing University for the legacy AR-SI oracle. We thank Ardupilot for the convenient open-source flight control software. The research projects related to this paper

are supported in part by the Hong Kong RGC Theme-Based Research Scheme (TRS) under Grant T22-505/19-N (P0031331, RBCR; P0031259, RBCP), in part by RGC GRF under Grants PolyU 152002/18E (P0005550, Q67V) and PolyU 152164/14E (P0004750, Q44B), in part by RGC Germany/HK under Grant G-PolyU503/16 (P0001326, RAA8), in part by HKJCCT P0041424 (ZB5A), and in part by the Hong Kong Polytechnic University Fund under Grants P0045578 (CE1C), P0043884 (CD6R), P0043634 (TAB2), P0043647 (TABF), 49NZ, P0042701 (CE09), P0042699 (CE55), P0036469 (CDA8), P0042721 (ZVG0), P0013879 (BBWH), P0031950 (ZE1N), P0033695 (ZVRD), P0000157 (BBWC), P0009706 (YBXW), 8B2V, Douyin Group (HK) Limited and CCF-Huawei Populus Grove Fund. The authors from Nanjing University are supported in part by the Leading-edge Technology Program of Jiangsu Natural Science Foundation (No. BK20202001), the National Natural Science Foundation of China (No. 6223200862172200), and the Fundamental Research Funds for the Central Universities (No.020214380101).

We also thank anonymous reviewers for their comments to improve this paper.

REFERENCES

[1] Lui Sha, Sathish Gopalakrishnan, Xue Liu, and Qixin Wang. Cyber-physical systems: A new frontier. In *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pages 1–9, 2008.

[2] Martin W Hoffmann, Somayeh Malakuti, Sten Grüner, Soeren Finster, Jörg Gebhardt, Ruomu Tan, Thorsten Schindler, and Thomas Gamer. Developing industrial cps: A multi-disciplinary challenge. *Sensors*, 21(6):1991, 2021.

[3] Junyan Hu, Barry Lennox, and Farshad Arvin. Robust formation control for networked robotic systems using negative imaginary dynamics. *Automatica*, 140:110235, 2022.

[4] Takeru Kuroiwa, Yusuke Aoyama, and Noriyuki Kushiro. Testing environment for cps by cooperating model checking with execution testing. *Procedia Computer Science*, 96:1341–1350, 2016.

[5] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016.

[6] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, pages 89–98, 2007.

[7] Ben Liblit, Mayur Naik, Alice X Zheng, Alex Aiken, and Michael I Jordan. Scalable statistical bug isolation. *ACM SIGPLAN Notices*, 40(6):15–26, 2005.

[8] Zhijian He et al. Mpc system identification method based oracle for control-cps software fault localization. 2018.

[9] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2014.

[10] Zhijian He, Yao Chen, Enyan Huang, Qixin Wang, Yu Pei, and Haidong Yuan. A system identification based oracle for control-cps software fault localization. In *41st ACM/IEEE International Conference on Software Engineering*, pages 116–127, 2019.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[12] Ze Sui, Yue Zhou, Xu Zhao, Ao Chen, and Yiyang Ni. Joint intention and trajectory prediction based on transformer. In *IEEE International Workshop on Intelligent Robots and Systems*, pages 7082–7088, 2021.

[13] John Thickstun. The transformer model in equations. 2021.

[14] F Gene, J Da Powell, Abbas Emami-Naeini, Roosevelt Braun, and Jaqueline Flatley. *Feedback Control of Dynamic Systems (7th Global Edition)*. Pearson, 2015.

[15] William F. Guthrie. *NIST/SEMATECH e-Handbook of Statistical Methods*. National Institute of Standards and Technology, 2020.

[16] Outlier, 2023. https://en.wikipedia.org/w/index.php?title=Outlier&oldid=1144860671.

[17] Afsoon Afzal, Claire Le Goues, and Christopher Steven Timperley. Mithra: Anomaly detection as an oracle for cyberphysical systems. *IEEE Transactions on Software Engineering*, 48(11):4535–4552, 2021.

[18] Roberto Natella, Domenico Cotroneo, Joao A Duraes, and Henrique S Madeira. On fault representativeness of software fault injection. *IEEE Transactions on Software Engineering*, 39(1):80–96, 2012.

[19] Box plot, 2018. https://en.wikipedia.org/wiki/Boxplot/, Last accessed on 2023-04-10.

[20] P-value, 2018. https://en.wikipedia.org/wiki/P-value, Last accessed on 2023-04-10.

[21] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Academic press, 2013.

[22] Effect size, 2018. https://en.wikipedia.org/wiki/Effectsize, Last accessed on 2023-04-10.

[23] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[24] Mseloss — pytorch 2.0 documentation, 2023. https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html.

[25] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys*, 51(1):1–27, 2018.

[26] Fei-Ching Kuo, Tsong Yueh Chen, and Wing K Tam. Testing embedded software by metamorphic testing: A wireless metering system case study. In *2011 IEEE 36th Conference on Local Computer Networks*, pages 291–294, 2011.