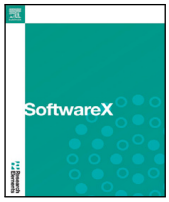
SoftwareX 27 (2024) 101748

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

Identifying safety–critical concerns in unmanned aerial vehicle software platforms with SALIENT

Sajad Khatiri [a,b], Andrea Di Sorbo [c], Fiorella Zampetti [c], Corrado A. Visaggio [c], Massimiliano Di Penta [c], Sebastiano Panichella [b,*]

[a] Università della Svizzera italiana, Lugano, Switzerland
[b] Zurich University of Applied Sciences, Zurich, Switzerland
[c] University of Sannio, Benevento, Italy

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.6207783, https://github.com/spanichella/SALIENT-TOOL

Keywords:
Unmanned aerial vehicles
Issue management
Safety issues
Machine learning
Empirical study

ABSTRACT

Safety-related concerns may emerge during the operation of unmanned aerial vehicles (UAVs), reported by users and developers in the form of issue reports and pull requests. To help UAV developers identify safety-related concerns, we propose SALIENT, a machine learning (ML)-enabled tool that analyzes individual sentences composing the issue reports and automatically recognizes those describing a safety-related concern. The assessment of the classification performance of the tool on the issues of popular open-source UAV-related projects demonstrates that SALIENT represents a viable solution to assist developers in timely identifying and triaging safety–critical UAV issues, outperforming baselines based on ChatGPT and Google's Bard.

Code metadata

Current code version	V1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00004
Permanent link to Reproducible Capsule	https://github.com/spanichella/SALIENT-TOOL/releases/tag/V.1.0
Legal Code License	Pre-approved GNU GPL-3.0 license as reported in https://github.com/spanichella/SALIENT-TOOL.
Code versioning system used	Git
Software code languages, tools, and services used	Python 3.9+ (97.0%) and Dockerfile (3.0%)
Compilation requirements, operating environments & dependencies	Tested within Mac OS and Linux (Ubuntu)
If available Link to developer documentation/manual	https://github.com/spanichella/SALIENT-TOOL/blob/main/README.md
Support email for questions	spanichella@gmail.com

1. Motivation and significance

Unmanned aerial vehicles (UAVs), also known as drones, are autonomous or teleoperated aircrafts adopted in a wide variety of application fields, including agriculture, disaster management, and surveillance [1]. Since UAVs leverage sensors-based analysis to continuously sense the changes in the physical environment [2,3] in which they are operating and take physical actions to react to the sensed changes [4], they are part of safety–critical cyber–physical systems (CPSs), enabling long-range operations at relatively low costs [5]. However, due to the cyber–physical nature of UAV systems, ensuring that the system always operates as *expected* is hard, especially in continuously varying scenarios [6]. Specifically, autonomous systems can enter *unexpected states* leading to *unexpected behaviors* potentially harmful to humans [2,3,7]. One of the main reasons behind this is represented by the *Reality Gap* problem characterizing safety–critical CPSs [8–11], arising from disparities between the testing and real-world environments, leading to inaccuracies in testing results and unsafe behaviors in real world. Linked to this is the challenge of replicating and identifying bugs [2,3,7] of safety–critical CPSs early in the development process [8,9,12].

* Corresponding author.
E-mail address: spanichella@gmail.com (S. Panichella).

https://doi.org/10.1016/j.softx.2024.101748
Received 2 January 2024; Received in revised form 27 March 2024; Accepted 19 April 2024

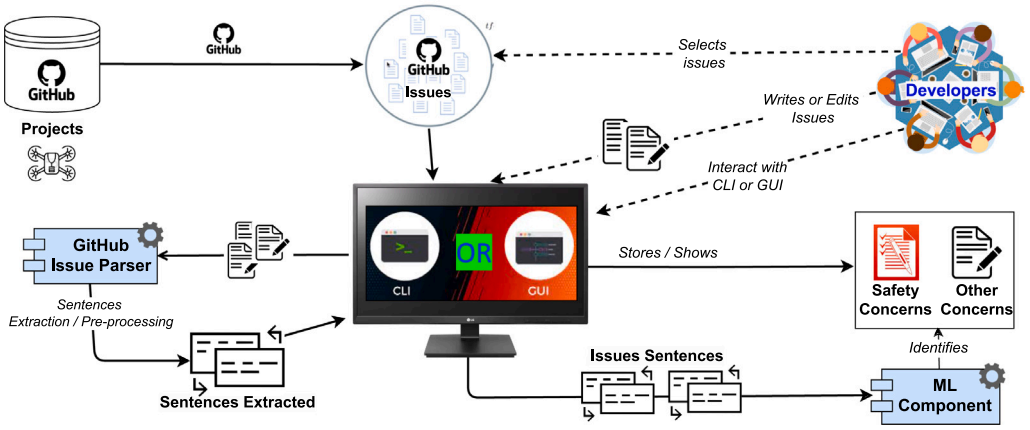


Fig. 1. SALIENT's architecture overview.

We argue that it is essential for UAVs that *"the possibility of harm to persons or of property damage is reduced to, and maintained at or below, an acceptable level through a continuous process of hazard identification and safety risk management"* [13,14]. To this aim, the aeronautical industry and the software engineering community [2,3] is constantly concerned with enhancing the efficiency, safety, and automation of such vehicles. Since the types of possible safety-related concerns that might happen are not all a priori known, the identification of safety requirements can be performed iteratively [15], expecting that safety-related concerns emerge during the operation phase of UAV systems provided by users or developers in the form of issue reports and pull requests [16]. On the one hand, popular projects receive thousands of reports of varying types and quality [17,18] (*e.g.,* in the last 12 months, Ardupilot [19] received more than 3000 among issue reports and pull requests). On the other hand, given the criticality of safety-related issues, UAV developers should pay special attention to them and promptly identify safety issues for fast triage and resolution.

To help developers analyze user-reported issues occurring in UAV platforms, in this paper, we present SALIENT (**SA**fety-critica**L** **I**ssue id**ENT**ifier), a tool that automatically detects safety-related concerns disclosed in issue reports. The tool implements an approach proposed in previous work [20], which analyzes issue reports to recognize safety–critical sentences contained in them and provides developers with proper support during the issue triaging and resolution phase. When dealing with cyber–physical systems, safety-related concerns might be highly critical. Therefore, SALIENT aims to help developers early capture if an issue or pull request contains safety-related implications by automatically analyzing it. We focus on both issue reports and pull requests, as beyond users that can directly report issues and problems, GitHub supports pull-based development, where developers contribute to a repository by submitting pull requests rather than directly modifying the source code [21]. After careful revision, such pull requests can be either merged into the master branch or rejected.

Although researchers devised tools able to automatically analyze [22], classify [17,23], or summarize [24,25] user-reported issues, the purpose and classification granularity of SALIENT are different, aiming to analyze individual sentences composing the issues and recognize those describing a safety-related concern. We showcase that SALIENT can achieve promising results to effectively aid developers in this task, outperforming baseline strategies based on ChatGPT and Google's Bard (see Section 3). SALIENT is available either in a Graphical User Interface (GUI) version or as a Command-Line Interface (CLI).

2. The SALIENT tool

In this section, we overview SALIENT's architecture and its main components (see Fig. 1); and we discuss in detail the approach, and the technologies behind SALIENT.

2.1. SALIENT *architecture overview*

As shown in Fig. 1, SALIENT supports two main usage modes thanks to its *CLI* and its *GUI*. The GUI is designed for general users (not necessarily developers), while the CLI is useful for UAV developers interested in automating the issue management process toward safety-related issues. Both the CLI and GUI are designed to allow developers/users to select issues as well as write and edit issues. Then, CLI commands and specific GUI events support the developers/users in the automated identification (or classification) of sentences in issues concerning safety-related aspects of UAVs.

The CLI and the GUI of SALIENT enable the aforementioned steps by leveraging (or interacting with) two components: the `GitHub Issue Parser` and the `ML Component`. The users/developers (i) interact with the `GitHub Issue Parser` via the CLI or the GUI, with the component able to preprocess issues downloaded from Github in different formats (explained in later sections) and preprocess them (*e.g.,* splitting the issue text logically in sentences) for the next analysis. Once the data are pre-processed, (ii) the users/developers interact with the `ML Component` via the CLI (or the GUI) of SALIENT, which allows them to identify sentences with safety-related implications. Specifically, in this process, the data pre-processed by the `GitHub Issue Parser` are then provided as input to the SALIENT's `ML Component` which identifies (or classifies) the individual sentences that are safety-related. To support developers/users during the issue management process, the `ML Component` uses a model we trained using a manually curated dataset as described in our previous work [20] to classify safety-related sentences in newly reported UAV issues in GitHub. The training, tuning, and accuracy of the ML model employed by SALIENT are summarized in Section 3 and detailed in previous work [20].

2.2. *Tool approach and technology overview*

SALIENT is developed to classify sentences contained in GitHub issues of UAVs that are critical from a safety point of view. For this we need a reference taxonomy or set of safety concerns to consider for automating the classification process. In the context of our study, we considered safety–critical issues from recent work [20]. Specifically, Table 1 describes some of the safety issues developers of UAVs need to address (a complete list can be found in our recent work [20]), which are the target of the SALIENT identification process. As described in [20], these categories emerged from a systematic manual analysis of sentences occurring in 304 safety-related closed issues and pull requests of three popular UAV-related GitHub projects (further details on the data collection and labeling steps are provided in Section 3).

As shown in Fig. 1, the `GitHub Issue Parser` is responsible to prepare the text for the analysis (*i.e.,* text cleaning, sentence splitting,

Table 1
Reference Safety-related issue categories of UAVs [20].

Category	Description
Undesired behavior on failsafe or error condition/configuration	Unexpected behavior of the system in the scenarios where failsafe operations or error conditions occur, *e.g.,* low default speed allowed when radio contact is lost, undesired throttle behavior in case of GPS lock.
Inadequate checks	Parameters, input data from sensors, actions and/or events not checked before activating a specific flight mode or before enabling a specific action, *e.g.,* battery status not checked before takeoff.
Improper parameter setting/initialization/configuration	System-level parameters, *e.g.,* min-throttle, not or wrongly initialized, configured and set, *e.g.,* improper calibration or incorrect conversions.
Undesired hardware behavior	Unexpected behavior, *e.g.,* failures, of the hardware components affecting the overall behavior of the system, *e.g.,* throttle unclamped when disarmed.
Timing/timeout/ synchronization issue	Inappropriate handling of synchronization across different functionalities and timing issues when various components, both hardware and software, must communicate.

etc.). Our `GitHub Issue Parser` exploits the functionalities provided by the *nltk* library to analyze the natural text of issues given as input and apply a sentence-splitting strategy. In particular, for sentence splitting, the `GitHub Issue Parser` leverages the *nltk* sentence tokenizer. To identify sentence boundaries, it relies on punctuation marks and regular expressions, including a list of common abbreviations (e.g., Dr. or Mr.) to avoid incorrectly splitting sentences. Issue data can be provided as input via the CLI or the GUI of SALIENT. As demonstrated in the video and detailed in our repository, the SALIENT 's GUI allows to upload a text file (or CSV file) containing the text of the UAV issue, or the users/developers can directly copy and paste the text of the issue in the GUI's form. Via the command line tool, it is possible to specify, providing as input a *JSON config file* (with arguments as summarized in Table 2), the location of the file (containing the text of the issue) the developers/users are interested in or, alternatively, they can directly copy and paste (or type) the text of an existing (or new) issue directly in the JSON file.

Once the UAV issue text is provided as input and its text is divided into sentences, SALIENT enacts for each sentence its classification approach: the `ML Component` leverages *FastText* [26] to classify each sentence as safety- or non-safety-related with the aim of supporting the UAV issue management process. *FastText* is a library created by Facebook's AI Research lab that leverages word embeddings for text classification. By default, *FastText* uses a set of hyperparameters that can be further customized.[1] *FastText* is an extension of Word2Vec, and operates at a more granular level by breaking words into several n-grams (sub-words) [26]. This approach allows for better representing rare and out-of-vocabulary words, thereby improving classification performance. We decided to use *FastText* since it has been successfully used for identifying the correct labels to assign to GitHub issues [17]. Indeed, recent research has shown that *FastText* is one of the best-performing classifiers in issue type prediction tasks [27]. Compared to other ML models, *FastText* also achieves the best results when classifying UAV safety-related concerns [20]. SALIENT's `ML Component` obtains sub-word embeddings for each input sentence by leveraging *FastText*. Based on such a representation, the `ML Component` (i) predicts the likelihood that sentences in a new UAV issue concern safety- or non-safety-related aspects, (ii) assigns the class (i.e., safety or non-safety) with the highest probability to each sentence, and (iii) visualizes such information via the GUI (or the CLI). Hence, developers/users have the possibility to save the recommended concerns in local files, to prioritize the issues that are more critical to address from a safety perspective.

2.3. *Using* SALIENT

This section discusses the main SALIENT commands to use it and clarifies in an abstract level what the software can do or can enable (for this also refer to Sections 3 and 4).

To analyze UAV issues and identify UAV safety concerns, SALIENT can be used as a Python command-line utility, which needs proper setup and configurations for execution.[2] To simplify this process, we have included a Dockerfile that sets up all the requirements, runs the container, and opens the bash to interact with the tools, with the following commands:

```
docker build . -t salient_tool
docker run -it salient_tool bash
```

Analyzing UAV issues requires to invoke SALIENT within the container, with the configuration file `config.json`, a command is provided:

```
python Fasttext-Model-prediction-on-safety-
    unseen-data.py --infile config.json
```

As demonstrated in the video and detailed in our repository, SALIENT's `config.json` file contains different arguments (see Table 2), that allow specifying the issue data to be processed and classified, with results stored in the specified output file (path specified with the file `config.json`). It is important to note that, if multiple *Input types* (see Table 2) are provided as input with the `config.json` file, all of them are analyzed/classified and stored in the specified output file. SALIENT can also be used via its GUI interface (it requires installing required packages/libraries, as detailed in our repository):

```
cd salient_src
python salient_gui_tkinter.py #it opens the GUI
```

With the SALIENT's GUI it is possible to classify text of issues concerning safety aspects automatically, with relevant sentences highlighted in the specified color (the default color is orange). The GUI also allows storing the results locally.

To allow UAV developers or testers pririotizing issues, SALIENT provides different options to output the results with its GUI and its CLI.

[1] For more information, refer to https://fasttext.cc/docs/en/options.html.

[2] (Video) Demo URL: https://www.youtube.com/watch?v=cvlDJRW5Oj8.

Table 2
SALIENT CLI/GUI arguments to be provided in the config.json file.

Argument	Type	Description
`path_model`	String	The SALIENT's pre-trained model (`trained_model.bin`) required for the classification.
`path_test_set`	String	(Input type 1) Path of the CSV file (in our demonstration `test-set.csv`) containing the sentences of the issue(s) to be classified.
`path_text_issue`	String	(Input type 2) Path of the txt file (in our demonstration `text_issue.txt`) containing the text of the issue to be classified .
`text_as_input`	String	(Input type 3) String (in our demonstration *"AP_Arming: pre-arm check if ... to disable the first one"*) of the issue to be classified.
`path_results`	String	The path of the File (in our demonstration ``fasttext_predicted_labels_test_dataset.txt'') where the results of SALIENT's classification step are stored.

Table 3
Dataset characteristics (detailed sampling and labeling process is reported even further in [20]).

Project	# Issues Closed	# Safety Issues	# Pull Requests Closed	# Safety Pull Requests
Ardupilot	4,529	153	11,815	78
dRonin	691	25	1,452	37
PX4-Autopilot	5,623	3	11,195	8
TOTAL	10,843	**181 (1.67%)**	24,262	**123 (0.51%)**

Table 4
GPT-3.5 and PaLM 2: Confusion Matrices for predicted safety and non-safety issues.

		GPT-3.5		PaLM 2	
		Predicted class		Predicted class	
		safety	non-safety	safety	non-safety
Actual class	safety	27	13	40	0
	non-safety	18	42	42	18

As shown below, the output of SALIENT corresponds to a structured file (in CSV format) containing the sentence ID, the text of the sentence classified, the label assigned to the sentence (*i.e.,* "Yes" or "No", depending on the fact that the sentence concerns safety aspects), and the probability that the predicted label is actually correct.

```
1 Sentence,Predicted_Label,
      Probability_of_Predicted_Label
2 0,AP_Arming: pre-arm check if compass1 is
      disabled but 2 or 3 are enabled.,"('
      __label__YES',)",0.670190691947937
3 ...
```

In a broader setting, the SALIENT tool can be used as fundamental starting point for developing (i) UAV issue prioritization strategies (this to aid UAV developers in mitigating recurring safety issues); (ii) researchers/developers can utilize SALIENT to pinpoint safety-related concerns for testing and maintenance of UAV software [2,3]; finally, (iii) SALIENT can be experimented to explore safety issue categorization beyond UAV domains, to enabling automated hazard identification as well as human into-the-loop analysis of identified issues [14].

3. Evaluation and impact

We collected a dataset containing issue reports and pull requests from open-source UAV software platforms hosted on GitHub by querying for projects with topics *drone* and/or *drones* and filtering out any projects that did not have safety-related labels. This filtering resulted in three UAV-related projects.

Table 3 provides an overview of each project included in our dataset, showing the total number of closed issues and pull requests, as well as, among them, how many are labeled as reporting a safety concern. We, therefore, considered the 304 closed issues and pull requests labeled with safety-related labels by the original developers and manually classified each of the 1916 sentences appearing only in the titles and descriptions of such 304 documents as safety- or non-safety-related as we aim to enable early identification of safety-related concerns. To reduce subjectivity, a strict labeling protocol was followed. It included a pilot labeling task involving all coders, independent labeling by multiple annotators for each sentence, and resolution of disagreements with a third annotator [20]. At the end of the procedure, the coders identified 837 (43.7%) of the sentences as discussing safety concerns, while the remaining 1079 did not. For more details on the methodology followed to construct the dataset, please refer to the original work behind our tool [20] while the *dataset* used for training and evaluating the `ML Component` is accessible on Zenodo [28].

We are aware of the potential threats deriving from the usage of Large Language Models (LLMs) in software engineering research including issues with closed-source models, possible leakage between LLM training data and research evaluation, and the reproducibility of LLM-based findings [29]. However, we decided to explore whether a general-purpose and pre-trained AI-enabled solution based on LLM technologies can be used to automatically identify safety-related sentences appearing in UAV issues. To this aim, we selected a smaller sample of 17 issues from our dataset/oracle, comprising at least one issue from each of the considered projects for a total of 100 sentences, 40 of which are safety-related. We decided to experiment with GPT-3.5 and PaLM 2 (*i.e.,* the models behind OpenAI's ChatGPT and Google's Bard) to classify each sentence in the sample, using the prompt: `Is the following sentence safety-related? '<sentence_here>' Response format: YES/NO`. Our choice of experimenting with GPT-3.5, and not GPT-4, is motivated by the fact that GPT-4 is a paid resource that not everyone can have free access to, restricting the replicability of our results.

As shown in Table 4, both pre-trained models obtained low classification performance (*i.e.,* a precision that is less than or equal to 60%), motivating the need for a specialized model able to achieve better results to be useful in practical contexts.

Since we aimed to understand if pre-trained AI-enabled solutions could be used as they are for identifying safety-related concerns in the UAV context, it is worth noticing that we used pre-trained generative models. We did not perform any prompt engineering or use Retrieval-Augmented Generation (RAG) or reinforcement learning to enhance the LLM tools for this specific application. Instead, Table 5 reports the classification performance achieved by the fastText model

Table 5
ML classification of safety-related sentences (Pr: Precision, Rc: Recall, F_1: F1-score).

Model	Safety			Non-Safety			Avg
	Pr	Rc	F_1	Pr	Rc	F_1	F_1
fastText	0.78	0.80	0.79	0.85	0.81	0.83	0.81

(leveraged by the tool's internal `ML Component`) when applying 10-fold cross-validation on the dataset of labeled issue sentences described above. In particular, the model achieves an average F1-score of 81%, with reasonably high recall and precision results in automatically detecting safety-related sentences. These results demonstrate that SALIENT can represent effective support to aid developers in identifying safety-related issues promptly during the issue management process.

Implications and Impact. Our tool opens the way to utilizing automated methodologies to assist UAV *developers* in pinpointing safety-related issues within problem reports. In particular, SALIENT can help them identify, triage, and prioritize safety-related concerns that require attention, possibly reducing the time needed for implementing the corresponding solutions. Moreover, SALIENT can be integrated into the software development lifecycle to perform automated checks for safety-related sentences. This can enhance the overall quality of UAV software platforms by ensuring that safety considerations are consistently addressed and monitored throughout the development process.

As for *researchers*, our tool propels exploration into the enhanced analysis of safety-related issues occurring in UAVs. Researchers can use the tool to gather large volumes of data on safety issues. This data can be invaluable for empirical studies and trend analysis aimed at advancing research on best practices, guidelines, and standards for enhancing safety in UAV systems. Researchers can leverage our work and the manually labeled data provided to delve into automated methods for categorizing safety-related concerns and assist UAV developers in mitigating recurring safety-related issues. Our data can also be leveraged for developing approaches that aid developers in allocating appropriate resources to various types of UAV-specific safety concerns.

4. Conclusion

Issues related to safety in UAV systems can have serious consequences, emphasizing the necessity of maintaining a continuous process for identifying hazards and managing safety risks. However, the large volume of issue reports that a UAV software platform may receive poses a challenge for developers in promptly identifying and addressing safety–critical concerns. To address this challenge, we introduce SALIENT, a tool empowered by machine learning to automatically detect sentences within issue reports that discuss safety-related issues. Our tool offers developers tangible assistance in streamlining the prioritization of safety–critical problems in UAV software systems, facilitating the creation of comprehensive platforms to aid UAV developers in mitigating recurring safety issues. Researchers can utilize SALIENT to pinpoint safety-related concerns, thereby encouraging further investigation into the testing and maintenance of UAV software [2,3,7].

Future endeavors seek to expand upon this research by applying it in diverse contexts, including closed-source environments, and exploring safety issue categorization beyond UAV domains. The objective is to evaluate the adaptability of models across domains and ascertain whether retraining is necessary due to unique safety-related characteristics. Furthermore, to enhance fault localization and resolution processes, we intend to augment support for safety issue prioritization, for instance, by enabling automated hazard identification as well as human into-the-loop analysis of identified issues [14].

CRediT authorship contribution statement

Sajad Khatiri: Writing – review & editing, Software. **Andrea Di Sorbo:** Writing – review & editing, Writing – original draft, Validation, Resources, Project administration, Methodology, Investigation, Data curation, Conceptualization. **Fiorella Zampetti:** Writing – review & editing, Methodology, Data curation, Conceptualization. **Corrado A. Visaggio:** Writing – review & editing, Methodology, Investigation, Conceptualization. **Massimiliano Di Penta:** Writing – review & editing, Methodology, Investigation. **Sebastiano Panichella:** Writing – review & editing, Writing – original draft, Validation, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

https://doi.org/10.5281/zenodo.6207783 and https://github.com/spanichella/SALIENT-TOOL.

Acknowledgments

We thank the Horizon 2020 (EU Commission) support for the project COSMOS Project No. 957254, the Innosuisse support for the project ARIES (Project No. 45548.1), and the Hasler Foundation project (Project No.23064) entitled "Bridging the Reality Gap in Testing Unmanned Aerial Vehicles".

References

[1] Balestrieri E, Daponte P, De Vito L, Picariello F, Tudosa I. Sensors and measurements for UAV safety: An overview. Sensors 2021;21(24):8253. http://dx.doi.org/10.3390/s21248253.
[2] Khatiri S, Saurabh P, Zimmermann T, Munasinghe C, Birchler C, Panichella S. SBFT tool competition 2024 - CPS-UAV test case generation track. In: IEEE/ACM international workshop on search-based and fuzz testing. 2024.
[3] Khatiri S, Panichella S, Tonella P. Simulation-based testing of unmanned aerial vehicles with aerialist. In: International conference on software engineering. 2024.
[4] Zampetti F, Kapur R, Di Penta M, Panichella S. An empirical characterization of software bugs in open-source cyber–physical systems. J Syst Softw 2022;192:111425. http://dx.doi.org/10.1016/j.jss.2022.111425, URL https://www.sciencedirect.com/science/article/pii/S0164121222001315.
[5] Wojciechowska A, Frey J, Sass S, Shafir R, Cauchard JR. Collocated human-drone interaction: Methodology and approach strategy. In: International conference on human-robot interaction. IEEE; 2019, p. 172–81. http://dx.doi.org/10.1109/HRI.2019.8673127.
[6] Lindvall M, Porter A, Magnusson G, Schulze C. Metamorphic model-based testing of autonomous systems. In: International workshop on metamorphic testing. IEEE Computer Society; 2017, p. 35–41. http://dx.doi.org/10.1109/MET.2017.6.
[7] Khatiri S, Panichella S, Tonella P. Simulation-based test case generation for unmanned aerial vehicles in the neighborhood of real flights. In: IEEE conference on software testing, verification and validation. 2023, p. 281–92. http://dx.doi.org/10.1109/ICST57152.2023.00034.
[8] Wang D, Li S, Xiao G, Liu Y, Sui Y. An exploratory study of autopilot software bugs in unmanned aerial vehicles. In: Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering. 2021, p. 20–31.
[9] Afzal A, Katz DS, Le Goues C, Timperley CS. Simulation for robotics test automation: Developer perspectives. In: Conference on software testing, verification and validation. IEEE; 2021, p. 263–74.
[10] Ngo A, Bauer MP, Resch M. A multi-layered approach for measuring the simulation-to-reality gap of radar perception for autonomous driving. In: 24th IEEE international intelligent transportation systems conference. IEEE; 2021, p. 4008–14. http://dx.doi.org/10.1109/ITSC48978.2021.9564521.
[11] Reway F, Hoffmann A, Wachtel D, Huber W, Knoll AC, Ribeiro EP. Test method for measuring the simulation-to-reality gap of camera-based object detection algorithms for autonomous driving. In: IEEE intelligent vehicles symposium. IEEE; 2020, p. 1249–56. http://dx.doi.org/10.1109/IV47402.2020.9304567.

[12] Afzal A, Le Goues C, Hilton M, Timperley CS. A study on challenges of testing robotic systems. In: International conference on software testing, validation and verification. IEEE; 2020, p. 96–107.
[13] International Civil Aviation Organization (ICAO). Safety management manual; 3rd ed.; Doc 9859. Montréal, Quebec: ICAO; 2013.
[14] Birchler C, Mohammed TK, Rani P, Nechita T, Kehrer T, Panichella S. How does simulation-based testing for self-driving cars match human perception? In: ACM international conference on the foundations of software engineering. 2024.
[15] Cleland-Huang J, Vierhauser M. Discovering, analyzing, and managing safety stories in agile projects. In: Requirements engineering conference. 2018, p. 262–73. http://dx.doi.org/10.1109/RE.2018.00034.
[16] Wang D, Li S, Xiao G, Liu Y, Sui Y. An exploratory study of autopilot software bugs in unmanned aerial vehicles. In: European software engineering conference and symposium on the foundations of software engineering. 2021, p. 20–31. http://dx.doi.org/10.1145/3468264.3468559.
[17] Kallis R, Di Sorbo A, Canfora G, Panichella S. Predicting issue types on GitHub. Sci Comput Program 2021;205:102598. http://dx.doi.org/10.1016/j.scico.2020.102598.
[18] Panichella S, Canfora G, Di Sorbo A. "*Won't We Fix this Issue?*" qualitative characterization and automated identification of wontfix issues on GitHub. Inf Softw Technol 2021;139:106665. http://dx.doi.org/10.1016/j.infsof.2021.106665.
[19] Ardupilotorg. Open source drone software. Versatile, trusted, open. ArduPilot. 2023, URL https://ardupilot.org/. [Accessed on May 5th, 2023].
[20] Di Sorbo A, Zampetti F, Visaggio A, Di Penta M, Panichella S. Automated identification and qualitative characterization of safety concerns reported in UAV software platforms. ACM Trans Softw Eng Methodol 2023;32(3). http://dx.doi.org/10.1145/3564821.
[21] Azeem MI, Panichella S, Di Sorbo A, Serebrenik A, Wang Q. Action-based recommendation in pull-request development. In: ICSSP '20: international conference on software and system processes. 2020, p. 115–24. http://dx.doi.org/10.1145/3379177.3388904.
[22] Di Sorbo A, Visaggio CA, Di Penta M, Canfora G, Panichella S. An NLP-based tool for software artifacts analysis. In: IEEE international conference on software maintenance and evolution. 2021, p. 569–73. http://dx.doi.org/10.1109/ICSME52107.2021.00058.
[23] Panichella S, Di Sorbo A, Guzman E, Visaggio CA, Canfora G, Gall HC. Ardoc: app reviews development oriented classifier. In: International symposium on foundations of software engineering. 2016, p. 1023–7. http://dx.doi.org/10.1145/2950290.2983938.
[24] Di Sorbo A, Panichella S, Alexandru CV, Visaggio CA, Canfora G. SURF: Summarizer of user reviews feedback. In: International conference on software engineering, companion volume. 2017, p. 55–8. http://dx.doi.org/10.1109/ICSE-C.2017.5.
[25] Rastkar S, Murphy GC, Murray G. Automatic summarization of bug reports. IEEE Trans Softw Eng 2014;40(4):366–80. http://dx.doi.org/10.1109/TSE.2013.2297712.
[26] Bojanowski P, Grave E, Joulin A, Mikolov T. Enriching word vectors with subword information. Trans Assoc Comput Linguist 2017;5:135–46. http://dx.doi.org/10.1162/tacl_a_00051.
[27] Herbold S, Trautsch A, Trautsch F. On the feasibility of automated prediction of bug and non-bug issues. Empir Softw Eng 2020;25(6):5333–69. http://dx.doi.org/10.1007/s10664-020-09885-w.
[28] Di Sorbo A, Zampetti F, Visaggio CA, Di Penta M, Panichella S. Dataset of the paper "automated identification and qualitative characterization of safety concerns reported in UAV software platforms". Zenodo; 2021, http://dx.doi.org/10.5281/zenodo.6207783.
[29] Sallou J, Durieux T, Panichella A. Breaking the silence: The threats of using LLMs in software engineering. In: ACM/IEEE 46th international conference on software engineering - new ideas and emerging results. ACM/IEEE; 2024, URL https://conf.researchr.org/home/icse-2024.