Automated Identification and Qualitative Characterization of Safety Concerns Reported in UAV Software Platforms

ANDREA DI SORBO, University of Sannio, Italy
FIORELLA ZAMPETTI, University of Sannio, Italy
CORRADO A. VISAGGIO, University of Sannio, Italy
MASSIMILIANO DI PENTA, University of Sannio, Italy
SEBASTIANO PANICHELLA, Zurich University of Applied Sciences, Switzerland

Unmanned Aerial Vehicles (UAVs) are nowadays used in a variety of applications. Given the cyber-physical nature of UAVs, software defects in these systems can cause issues with safety-critical implications. An important aspect of the lifecycle of UAV software is to minimize the possibility of harming humans or damaging properties through a continuous process of hazard identification and safety risk management. Specifically, safety-related concerns typically emerge during the operation of UAV systems, reported by end-users and developers in the form of issue reports and pull requests. However, popular UAV systems daily receive tens or hundreds of reports of varying types and quality. To help developers timely identifying and triaging safety-critical UAV issues, we (i) experiment with automated approaches (previously used for issue classification) for detecting the safety-related matters appearing in the titles and descriptions of issues and pull requests reported in UAV platforms, and (ii) propose a categorization of the main hazards and accidents discussed in such issues. Our results (i) show that shallow machine learning-based approaches can identify safety-related sentences with precision, recall, and F-measure values of about 80%; and (ii) provide a categorization and description of the relationships between safety issue hazards and accidents.

CCS Concepts: • **Software and its engineering → Maintaining software**; **Error handling and recovery**; **Software safety**; • **General and reference → Empirical studies**.

Additional Key Words and Phrases: Unmanned Aerial Vehicles, Issue Management, Safety Issues, Machine learning, Empirical study

ACM Reference Format:
Andrea Di Sorbo, Fiorella Zampetti, Corrado A. Visaggio, Massimiliano Di Penta, and Sebastiano Panichella. 2022. Automated Identification and Qualitative Characterization of Safety Concerns Reported in UAV Software Platforms. *ACM Trans. Softw. Eng. Methodol.* , (July 2022), 36 pages.

1 INTRODUCTION

Unmanned aerial vehicles (UAVs), also known as drones, are autonomous controlled aircrafts used to perform a wide variety of tasks and are at the heart of the latest and upcoming advancements in various fields. Specifically, with the continuous improvements in UAV technology, drones, either autonomous or teleoperated, are nowadays

Authors' addresses: Andrea Di Sorbo, University of Sannio, Benevento, Italy, disorbo@unisannio.it; Fiorella Zampetti, University of Sannio, Benevento, Italy, f.zampetti@unisannio.it; Corrado A. Visaggio, University of Sannio, Benevento, Italy, visaggio@unisannio.it; Massimiliano Di Penta, University of Sannio, Benevento, Italy, dipenta@unisannio.it; Sebastiano Panichella, Zurich University of Applied Sciences, Zurich, Switzerland, panc@zhaw.ch.

1049-331X/2022/7-ART $15.00
https://doi.org/

used in a plethora of application fields, *e.g.,* agriculture [104], disaster management [26], archaeology [149], delivery [32], and surveillance [12]. This is because drones offer undisputed advantages as they enable long-range operations at relatively low costs and provide access to dangerous or hostile environments without endangering humans [11, 176]. With the global UAV market size expected to reach 25.13 billion USD by 2027, showing a compound annual growth rate of 12.23% compared to 10.72 billion USD in 2019 [51], UAVs usage is expected to make a real impact on the citizens' quality of life.

UAVs fall in the category of cyber-physical systems (CPSs), as they (i) continuously sense the changes in the physical environment in which they are operating through sensors-based analysis [122, 179], (ii) make decisions based on a pre-programmed logic, and (iii) take physical actions to react to the sensed changes [144, 158]. Given the safety-critical dimension of UAVs, the aeronautical industry has constantly focused on improving the efficiency, safety, comfort, and automation of these vehicles. Specifically, the UAV community has successfully developed and maintained open access projects for the software (firmware) and hardware (flight controller, sensors) for the drones, such as Ardupilot [5], PX4 [129] (*i.e.,* widely used open-source autopilot software systems), and Pixhawk [142] (*i.e.,* open standards for drone hardware).

To build reliable UAV systems, developers must (i) consider the changes to the environment proactively, as the inputs of a UAV can be dynamically fluctuating and uncontrollable depending on environmental changes, and (ii) deal with the various configurations of hardware devices (*e.g.,* parameters and range of values) [170]. Moreover, considering that UAV software platforms are typically designed for multiple types of hardware, such as copters, planes, and VTOLs (Vertical Take-Off and Landings), the mapping between the various functions and the different hardware models need to be verified by developers. Also, 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"* [105].

The automated testing and the real-time monitoring of UAVs to ensure their "desired" behavior are technical and research challenges that are still open [163, 166]. Due to the cyber-physical nature of UAV systems, it is difficult to determine whether an autonomous system behaves as *expected* [121], which hinders the development and maintenance of UAVs' underlying software. Specifically, if one or multiple physical variable(s) are out of their expected range for some time during a scenario, the autonomous system can enter an *unexpected state* [157] leading to *unexpected behavior* that can be potentially harmful to humans (*e.g.,* physical instability, rotating or hovering in place for too long, terminating the mission immaturely, miss-calculation of positions).

For dealing with such safety-related concerns, there is an increasing interest in adopting agile development paradigms for safety-critical systems [163], where the identification of hazards and the elicitation of safety requirements can be performed iteratively [28]. Specifically, these paradigms expect that safety-related concerns might emerge during the operation phase of UAV systems, provided by users/developers in the form of issue reports and pull requests [170]. In cyber-physical contexts, developers should pay particular attention to safety-related issues [6] and promptly react to them as they might lead to harm or damage to people, properties, or the environment. However, especially in popular projects, tens or hundreds of issues of different types (*e.g.,* signaling bugs or asking questions) and quality are reported daily [109, 138], making it hard to quickly identify safety-related concerns.

This paper aims to provide developers with support in analyzing user-reported issues occurring in UAV platforms, and identifying sentences describing safety-related concerns. In the last decades, researchers proposed general-purpose automated solutions aimed at easing the issue management and fixing processes, with approaches able to classify [100, 109, 139], and prioritize [162, 165], or summarize [39, 40] user-reported issues, and techniques to detect potential misclassifications [3, 101] and duplications [173] of bug reports. With respect to the existing body of work, our purpose and classification granularity is different. That is, while previous work automatically classifies issues along different categories (*e.g.,* bugs, enhancement, etc.), we aim to analyze individual sentences composing the issues and recognize those describing a safety-related concern. The rationale would be to highlight such sentences and therefore provide developers with proper support during the issue triaging and management phase.

Also, while the sentence classification approach described in this paper would not be strictly tied to the UAV domain, to the best of our knowledge this is the first work aimed at identifying safety-related concerns in issue reports related to the UAV domain.

As a first contribution of the paper, we have experimented with several machine-learning (ML)-based approaches previously used in the context of issue classification to evaluate whether and the extent to which such approaches can identify safety-related sentences in issue reports. When a user or a developer is reporting an issue, every time a sentence (or even the issue title itself) is related to safety-critical problems, such a sentence is flagged, as well as the whole bug report. By properly classifying safety-related issues, and highlighting specific sentences within these issues, projects can properly triage issues, *e.g.,* by prioritizing safety-critical ones over others, and then developers can focus on specific sentences when performing hazard analysis, *i.e.,* the analysis of situations or conditions that have the potential to lead to mishaps [115].

To further aid such a hazard analysis in the context of UAV systems, through a manual inspection of 273 safety issues and pull requests reported in three different open-source UAV platforms, we propose a categorization of the main hazards and accidents (*i.e.,* unexpected and unwanted events that could cause injury or damage to persons, equipment, materials, or environment) discussed in such issues. Such a categorization is complementary to the ones proposed in recent studies concerning drones, which mainly focused on the root causes of UAV-specific bugs [160, 170] and the uses of bounding functions in UAV systems [120].

In summary, the original contributions of this paper are the following:

- an evaluation of ML-based approaches (previously used in the context of issue labeling) when leveraged for identifying safety-related sentences contained in user-reported issues of UAV platforms;
- a categorization of the main hazards and accidents occurring in safety-related issues reported in open-source UAV platforms; and
- a manually-labeled dataset (available for research purposes) of 304 safety issues extracted from three different UAV platforms hosted on GitHub.

The results of the experiments with different machine learning models for identifying safety-related concerns in issue reports of UAV software platforms show that Random Forest (RF) obtains precision, recall, and F-measure values of about 80%, with the model considering hardware-related keywords and text features about concrete states (or actions) of the UAVs as the most important ones. The RF results are similar to the ones obtained by using the default implementation of fastText (state-of-art approach for issue classification [108, 109]) in a 10-fold validation setting. Results of our study also provide insights into the nature of UAV issues, and, in particular, we found that in the great majority of the considered UAV issue reports, the hazardous circumstances depend on software-related defects. The most recurrent signaled accidents are related to crashes/collisions, while, in about 40% of the cases, hazards are due to undesired behaviors occurring when the system enters failsafe mode or to the missing inhibition of certain actions when the system is in specific states.

Replication package. Our replication package, available on Zenodo [43], contains (i) the scripts developed to pre-process the data used for training the ML models (RQ_1) and (ii) the results of the manual validation of issues (both RQ_1 and RQ_2).

Paper structure. The rest of the paper is organized as follows. Section 2 briefly provides the background knowledge about safety-critical systems and UAVs. Section 3 describes our empirical study, including the data, the research questions, and the methodology followed to answer such questions. The achieved results are reported and discussed in Section 4, while Section 5 reports the threats to our study's validity. Finally, Section 6 discusses the related work, while Section 7 concludes the paper outlining future research directions.

2 BACKGROUND

This section provides background notions about (i) concepts related to safety-critical systems, and (ii) concepts and terminologies about UAVs.

2.1 Safety-Critical Systems

In the following, we provide relevant definitions related to safety-critical systems (SCSs) that will be the subject of our study such as the definitions of *accident* and *hazard*. At same time, we only briefly elaborate on emerging types of SCSs, since a full coverage of background on SCSs is available in previous seminal studies [113–119].

A SCS can be considered any system (composed of hardware and/or software) whose failure may cause (i) death or injury to people, or (ii) damage to equipment/property, or to the environment [152, 167]. Given this definition, almost any system could be safety-critical [115]. For this reason, the key point in the safety-critical analysis is not to ensure 100% safety, but, rather, to reduce the risk that a *hazardous event* occurs. In this context, emerging SCSs can be represented by robotic systems [151], real-time systems [10], autonomous systems [178], and more in general CPSs [141].

SCSs complexity poses challenges on scaling agile practices [45, 156] and performing requirements engineering [126] for SCSs. Important SCS definitions related to the safety of these systems are *hazard* and *accident*. An *hazard* is defined as *"a set of conditions that can lead to an accident, given certain environmental conditions"* [115]. For instance, a hazard in a self-driving vehicle can be due to a sensor failing to detect an obstacle. An *accident*, also referred to as a *mishap*, refers to *"events, or series of events, that result in death, injury, occupational illness, damage to or loss of equipment or property, or environmental harm"*. [115]. For instance, the hazardous event reported above can lead to the vehicle hitting a pedestrian (*i.e.,* accident). In this paper, we focus primarily on shed some more light on the types of hazards and accidents affecting a specific type of autonomous system such as UAV.

2.2 Unmanned Aerial Vehicles Development Cycle

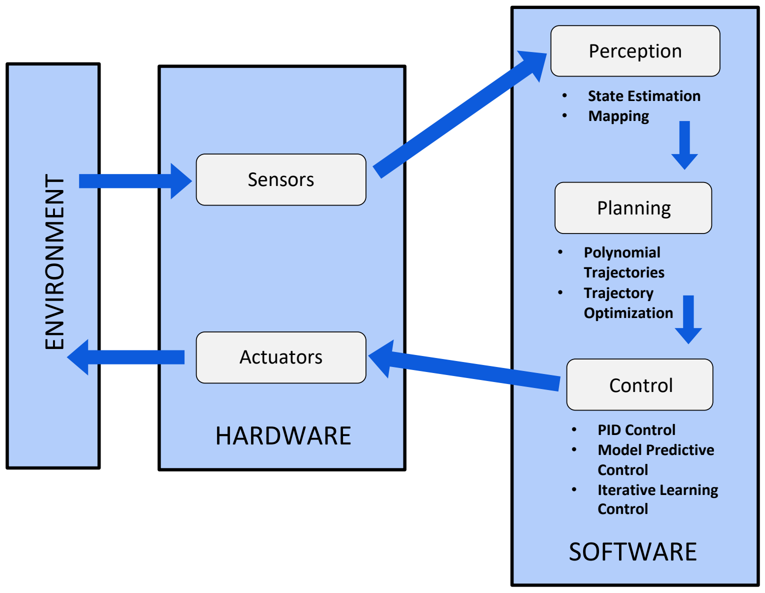

Fig. 1. Overview of development cycle of UAVs

Figure 1 shows the development cycle of UAVs [140], which includes three software phases (*i.e.,* Perception, Planning, and Control) as well as the hardware components that interact with the environment.

The **Perception phase** includes *state estimation* [47, 50, 134, 143] and *mapping* [103, 136] steps. The state estimation step enables navigation and autonomous driving of UAVs [143]. The mapping step consists of computing obstacles distance information and, complementary to the state estimation step, it generates a 3D model of the environment (with 3D range measurements [103]) and models unmapped areas at run-time [136].

The **Planning phase** consists of finding the right trajectory between a starting point and a destination, with state-of-the-art approaches in this phase leveraging *polynomial trajectories*[130, 133, 148] and *trajectory optimization* [49] strategies. Finally, the **Control phase** identifies the flight control commands to be executed by UAVs to safely navigate the environment.

To enable the usage of UAVs in different contexts, their software is equipped with functions and checks enabling different autonomous or teleoperated flight modes [25, 48, 140, 166]. These are enabled by the use of a Ground Control Station (GCS), consisting of a set of hardware and software components that enable UAV operators to communicate with/control the UAVs.

In the following, we elaborate on some of the main flight modes relevant for the conducted research:

- Position Mode: The UAV maintains a consistent location and heading, with manual control of the throttle, which controls the vertical motion of the UAV.
- Altitude Mode: Ensures that the UAV will level and maintain the current altitude (*e.g.,* in windy conditions).
- Manual/Stabilized Mode: This allows the operators to manually fly a UAV, with self-levels keeping the stability of pitch axis.
- Failsafe Mode: The UAV lands safely back to the take-off point based on GPS signals (the failsafe mode is activated when the connection between the UAV and the controller is disconnected).
- Return to Launch (RTL) Mode: The UAV returns to the home position.
- Fence Mode: It consists in geofencing an UAV, which means that the UAV flight will be restricted to a certain area.
- Follow-me Mode: Instructs the UAV to continuously follow an object of interest (or a person).
- Hold Mode: The UAV keeps its current position.
- Return Mode: Mode used to fly a UAV to safety from a difficult path to a safe destination.
- Mission Mode: The UAV performs a predefined autonomous mission.
- Takeoff Mode: The UAV to take off to a specified altitude position and wait for further input.
- Land Mode: The UAV lands from a certain altitude position to the ground.
- Offboard Mode: The UAV will apply the position, velocity, or altitude setpoint provided over MAVLink, *i.e.,* a protocol for communicating with small unmanned vehicle.

3 EMPIRICAL STUDY DESIGN

The *goal* of our study is to analyze safety issues occurring in open-source UAV-related software. The *perspective* is of researchers interested to build and improving recommender systems aimed at identifying, prioritizing, and supporting the issue fixing and management processes for this kind of system. In particular, we (i) investigate the extent to which automated strategies (previously used in the context of issue classification) can be used for identifying safety-related concerns occurring in issues reported in UAV platforms, and (ii) qualitatively discuss the types of hazards and accidents emerging from safety issues in UAV software. The study *context* consists of 304 safety issues sampled from three well-known open-source UAV software systems.

3.1 Research Questions

Based on the aforementioned goal, our study aims at answering the following research questions:

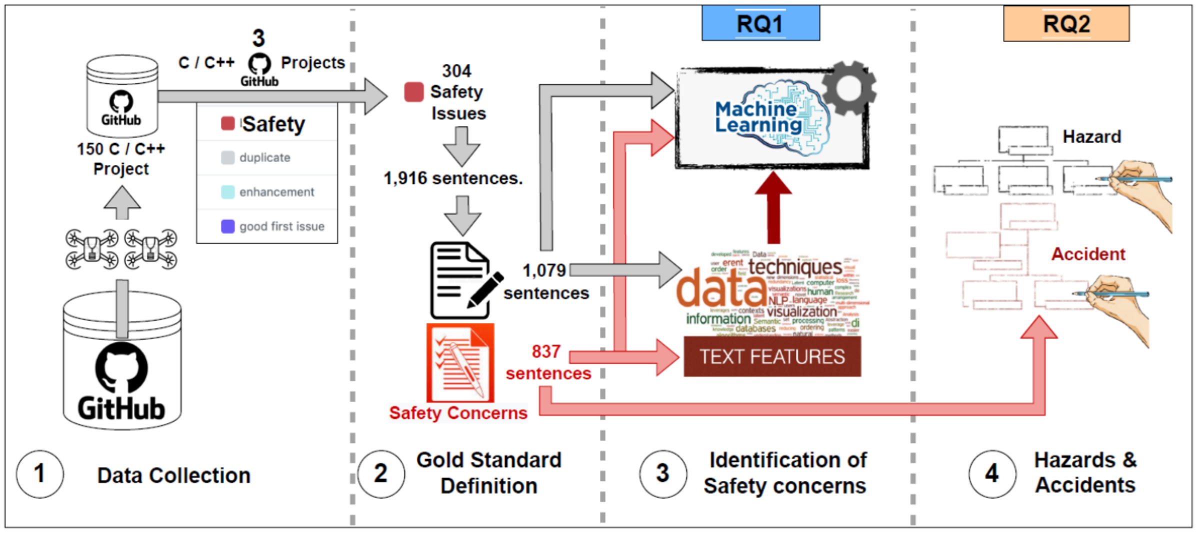

Fig. 2. Research Approach Overview

- *RQ*$_1$***: To what extent can machine learning models automatically identify safety-related concerns in issue reports of UAV software platforms?*** In this research question, we want to explore whether and the extent to which automated strategies proposed for issue report classification purposes can help UAV developers to recognize safety issues. Therefore, we experiment with approaches that leverage textual analysis and ML techniques to identify safety-related sentences in the titles and descriptions of reported issues. We argue that this may result particularly useful for the issue management process as it would (i) allow to automatically label the issue as safety-related, and (ii) highlight the safety-related text, so that it would be possible to properly analyze it during the issue prioritization process.
- *RQ*$_2$***: What are the main hazards and accidents emerging from safety issues reported in UAV software platforms?*** For achieving a more in-depth understanding of the specific types of safety issues occurring in drone systems, we analyze the hazards and related accidents emerging from safety-related issues. Specifically, by performing an open coding on 273 safety-related issues extracted from three different GitHub drone projects, containing at least one sentence describing a safety concern, we propose a categorization of the main hazards and accidents that occur in such issues. In the first stage, our goal is to analyze the relationship between hazards and accidents. At the same time, this would pave the way for automated tools aimed at supporting hazard classification and resolution prioritization (*e.g.*, based on the possible consequences).

Fig. 2 depicts the research approach we followed to answer our research questions. Our research approach consists of four phases. The *Data Collection* phase aims to build the dataset to be used for answering the given research questions and it is described in Section 3.2. During the phase of *Gold Standard Definition*, sentences of the dataset are manually annotated with safety and non-safety labels through a process of individual analysis, review, and open discussion. This process is detailed in Section 3.3. The labeled sentences are then used to train several ML techniques discussed in Section 3.4. Finally, in the *Hazards & Accidents* phase, the hazards and the related accidents reported in the safety-related sentences are identified through an approach explained in Section 3.5.

Table 1. Dataset characterization

Project	# Closed Issues	# Safety-related Issues	# Closed Pull Requests	# Safety-related 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%)**

3.2 Data Collection

The study *context* consists of three open-source UAVs hosted on GitHub, from which we extracted 304 issues labeled as safety-related. To identify UAV-related projects, we queried GitHub to retrieve all projects having as topic *drone* and/or *drones*, and as main programming language *C* or *C++*, the ones predominant for CPS development [154]. From the list, we removed fork projects, and projects not being forked, and ordered them based on the number of forks, *i.e.,* used as a proxy of the project's popularity. As an outcome, we ended up with a list of 150 candidate UAV-related projects.

Since our goal is to identify and analyze the safety-related issues occurring in UAV platforms, for each project, we downloaded the set of labels used for tagging the user-reported issues, and filtered out all the projects that do not have safety-related labels. After the filtering, we ended up with 3 UAV-related projects (as reported in Table 1), for which we downloaded the whole set of closed issues and pull requests by relying on Perceval [46]. Finally, we filtered out all the issues and pull requests not being labeled with a safety-related label, ending up with a total of 304 closed issues tagged with at least one safety label.

Table 1 reports, for each project included in our study, the total number of closed issues and pull requests, as well as, among them how many are labeled as reporting a safety concern. Note that, two out of three projects included in our study, *i.e.,* ARDUPILOT and PX4-AUTOPILOT, have been considered as references in other studies dealing with UAVs [170]. Furthermore, by looking at data reporting in Table 1, even if all the considered projects use both issues and pull requests for supporting software development and evolution, only a small percentage of them (1.67% of issues and 0.51% of pull requests) are labeled as discussing safety properties.

For each issue and pull request having a safety-related label, we used the BEE tool [153] to structure its title and description into sentences. While other bug trackers (*e.g.,* JIRA or Bugzilla) provide explicit fields for issue descriptions, the GitHub issue tracker does not, and the issue/pull request description is usually provided in the issue's first comment. Therefore, similar to previous work [7, 24, 108, 109, 138], here and in the remainder of the paper, we refer to the first comment of an issue/pull request as the *issue description.*

As a result, the 304 safety-related issues have been decomposed into 1,916 sentences.

3.3 Gold Standard Definition

To check whether it is possible to classify issues (not) reporting safety-related concerns, we need to properly identify, among the 1,916 sentences, the ones that refer to safety properties. To do this, all authors (henceforth referred to as coders), performed a manual classification of the sentences, considering two categories, *i.e.,* safety-related and not.

Identifying the presence of safety concerns in a piece of text is a subjective task, *e.g.,* interpretation of the text might be different for different individuals or might depend on their cultural background. For this reason, we conducted a pilot labeling study. Specifically, after having randomly sampled 10 safety-related issues in our dataset, accounting for 49 sentences, we relied on a plenary meeting (*i.e.,* pilot study), where, upon discussion, each sentence was properly classified as (not) describing safety-related concerns. Once the coders reached a common understanding of the labeling procedure, we assigned the remaining 1,867 sentences to two coders each, *i.e.,* an average of $\simeq$ 747 sentences for each coder.

At the end of the labeling procedure, an open discussion was performed by adding a third coder, to check all the sentences for which there were disagreements among the original coders (284 out of 1,867). Even if the coders agreed in 84.8% of the cases on whether a sentence discusses a safety-related property, it is still possible that they agreed by chance. For this reason, we computed the Cohen's k [29] which resulted to be 0.68 (substantial). After disagreements' resolution, among the 1,916 sentences in our dataset we identified 837 (43.7%) discussing safety concerns, while the remaining 1,079 did not. In other words, among the 304 safety-related issues, 274 contain at least one sentence in the title or description describing a safety-related concern. As regards the 30 safety-labeled issues in the original dataset that are not being classified as safety-related in our study, we found cases where:

- the description was empty and the title was not clearly reporting a safety concern, such as PR # 13926 [94] from PX4-Autopilot entitled *"navio_sysfs_rc_in: only publish input_rc if connected and all channels are non-zero"*;
- the safety concerns were discussed in a related issue/pull request pointed out in the description, such as PR # 1129 [78] from dRonin where the description simply reports the observed behavior, *i.e., "battery is also very low, and stack analysis confirms usage is increased"* while the referred issue highlights the in-flight reset occurring due to the low battery value; and
- the safety concerns are raised later on during the reviewing process of the issue/pull request, such as issue # 2011 [72] from ardupilot where the first comment simply limited to describe the problem: *"We probably want to keep logging after a parachute release to capture the impact in the logs"*, while only reading the follow-up comments it is possible to acknowledge that the current behavior might lead to the arming status being inadvertently changed.

The manual inspection of the 274 safety-related issues also revealed that one of these issues [59] aimed at summarizing multiple and too generic hazardous circumstances when using a tablet ground station as a user interface. For this reason, we decided not to consider such an issue when conducting the subsequent analysis (i.e., the identification of hazards and accidents). The labeled dataset is used as input for addressing both RQ_1 and RQ_2. Specifically, the whole labeled dataset (304 issues) is used to experiment with ML models, while the 273 issues with at least one safety-related sentence in either the title or the description are further classified to determine a set of hazards and accidents occurring in UAV systems.

3.4 RQ_1: To what extent can machine learning models automatically identify safety-related concerns in issue reports of UAV software platforms?

After the definition of the golden standard, we benchmark different techniques (proposed for issue report classification purposes) when used to automatically identify safety-related sentences in the titles and descriptions of UAV-reported issues.

3.4.1 ML Techniques. The experimented techniques have been used in related literature to work at the document level (*i.e.,* they consider the whole text in the issue report to automatically label it). In contrast, since we aim at specifically detecting the safety-related sentences appearing in issue reports to help developers timely identify safety-related UAV issues and ease the inspection of such issues when performing hazard analysis, we leveraged the aforementioned approaches to operate at the sentence level. For achieving this goal, we considered all safety-critical (837) and non-safety-critical (1,079) sentences in our dataset, collected through the metadata analysis explained in Section 3.3, and leveraged the different ML techniques, by applying the steps described below.

Textual corpus. All the manually-labeled sentences in our dataset are used as an information base to build a textual corpus. that is used in the next steps to extract different types of textual features relevant to the ML-based classification.

Feature extraction. From the aforementioned textual corpus we extracted the following sets of features:

- *Bag of Words (BOW) features*: All terms contained in the sentences are indexed and pre-processed applying stop-word removal (using the English Standard Stop-word list) and stemming (*i.e.,* English Snowball Stemmer), to reduce the number of textual features to consider as input to the ML techniques [9]. The outcome of this step is a Term-by-Document matrix, *M*, where each column represents a sentence of our dataset and each row represents a term contained in a generic safety-critical or non-safety-critical sentence belonging to the 304 issues/pull requests object of the study. Each entry $\mathbf{M}_{[i,j]}$ of the matrix represents the weight (or relevance) of the i$-th$ term contained in the j$-th$ sentence. The weights in the matrix are determined by using the *tf-idf* score [9], which allows for identifying the most important terms occurring in the sentences. The weighted matrix, $\mathbf{M}_{tf-idf}$, represents the outcome of this step.
- *Latent Semantic Indexing (LSI) features*: LSI is an indexing and retrieval method that uses singular value decomposition (SVD) to identify relationships between the terms and concepts contained in the unstructured text [23, 123, 145]. LSI is typically used for its analysis of latent (hidden) semantics in a text, addressing the problems of polysemy (*i.e.,* a term having a different meaning in different contexts) and synonymy (*i.e.,* two or more terms denoting the same concept). Using the R package *lsa*, we applied LSI (with the standard share of 0.5) to transform the Tf-idf matrix, $\mathbf{M}_{tf-idf}$, obtained as an outcome from the previous step, in a different space, called $\mathbf{M}_{LSI}$, so that terms having the same meaning are mapped to the same concept.
- *N-gram features*: We computed n-grams [127, 175] of size 2, 3, and 4 from each sentence in our dataset. The weights in the matrix are determined by using the *tf-idf* score [9], which allows for identifying the most important n-grams occurring in the sentences. The weighted matrix, $\mathbf{M}_{N-grams}$, represents the outcome of this step.
- *OB, EB, and S2R features*: The BEE tool [153] verifies, given a bug report, whether a sentence describes the current and unexpected behavior (OB), the expected behavior (EB), and the steps needed to reproduce the problem (S2R). Hence, we extracted, for each sentence in our dataset, such additional features since it could happen, for example, that safety-related concerns may be expressed while describing the current and undesired behavior. As a result, among the 1,916 sentences, 1,106 describe the current behavior, 186 report the expected behavior, and no sentences are reporting the steps to reproduce the problem. Note that, the distribution of OB, EB, and S2R in our sample is in line with previous literature showing that bug reports very often describe the current behavior, while only 35.2% and 51.4% of the bug reports detail EB and S2R, respectively [24]. The matrix $\mathbf{M}_{OB-EB}$ (reporting the presence/absence of OB and EB in each sentence) represents the outcome of this step.

Classification. We automatically identify safety-critical sentences by adopting various ML models and a 10-fold cross-validation strategy. To check whether the results are biased due to the dataset imbalance (*i.e.,* the majority of issues and pull requests considered in our dataset originates from ARDUPILOT), we also report results of a cross-project analysis, in which we use the best performing ML configurations (identified using 10-fold cross-validation strategy) to train it on the ARDUPILOT sentences and evaluate the classification performance on the sentences belonging to the other two projects, *i.e.,* PX4-AUTOPILOT and DRONIN.

We experiment with different shallow ML techniques (using Weka[1]), namely standard probabilistic Naive Bayes classifier, J48 decision tree, Sequential Minimal Optimization (SMO) model for Support Vector Machines (SVM), Logistic Regression, and Random Forest.

[1]http://waikato.github.io/weka/

The rationale for the choice is to include different types of classifiers, namely, a probabilistic one (Naive Bayes), a tree-based one (J48), a regression-based one (Logistic Regression) a black-box model that creates a boundary between data points (SVM), and, finally, an ensemble classifier (Random Forest). These ML techniques have been previously used for the classification of bug reports [4, 182], user review [38, 39, 139], and other artifacts [41, 42, 146] textual content.

The ML models are fed with the following combinations of features: $\mathbf{M}_{BOW}$, $\mathbf{M}_{LSI}$, $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$, and $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$ + $\mathbf{M}_{N-grams}$. It is important to mention that we did not report other configurations of features (e.g., $\mathbf{M}_{OB-EB}$ only), since for them we achieved poor experimental results (or no difference in results compared to other combinations of features). We decided to benchmark shallow ML techniques since, in cases of limited training data availability, they can still produce comparable or superior results [107] that allow an easier explainability than those generated by deep neural networks [150].

It is important to highlight that we initially used the *out-of-the-box* implementation of all the experimented ML algorithms (*i.e.,* no hyperparameters were tuned), as we wanted to first estimate the potential lower bound in the classification performance for the different algorithms, this by experimenting with different combinations of features (i.e., $\mathbf{M}_{BOW}$, $\mathbf{M}_{LSI}$, $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$, and $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$ + $\mathbf{M}_{N-grams}$).

A range of different optimization algorithms can be used to achieve potentially better results with respect to the default configuration of parameters of the ML models, with two of the most common methods represented by Random Search and Grid Search [2, 14, 15]. Grid search is great for spot-checking combinations that are known to perform well. Then, we experimented with Grid search as hyperparameter optimization approach [2, 14, 15] to investigate potential optimal combinations of parameters for the selected shallow ML models. Specifically, with Grid search we experimented with several parameters combinations of the various ML models (for a total of around 600 experimented combinations) using a 10-fold validation setting, as summarized below:

- For the **Decision Tree (J48)** we covered all possible combinations of the following parameters:
  - **C (confidenceFactor)**: Is the confidence factor, and we experimented with values $[0.001, 0.01, 0.05, 0.1, 0.5]$
  - **M (minNumObj)**: Is the minimum number of instances in a leaf, and we experimented with values $[1, 10, 20, 50, 100]$
  - **R (reducedErrorPruning)**: Reduced error pruning is an alternative algorithm for pruning that focuses on minimizing the statistical error of the tree. We experimented with the following values $[yes, no]$
  - **S (subtreeRaising)**: This is a specific method of pruning whereby a whole set of branches further down the tree are moved up to replace branches that were grown above it. We experimented with the following values of it $[yes, no]$
- For the **SMO** model we covered all possible combinations of the following parameters:
  - **C**: Represents the complexity parameter, and we experimented with the values $[0.1, 0.2, 0.5, 0.8, 1, 10, 50, 100, 500, 1000]$
- For the **Naive-Bayes** we covered all possible combinations of the following parameters:
  - **K**: Use Kernel estimator, and we experimented with values $[yes, no]$
  - **D**: use Supervised discretization, and we experimented with values $[yes, no]$
- For the **Logistic Regression** model we covered all possible combinations of the following parameters:
  - **R (ridge)**: Is the Ridge in the log-likelihood, and we experimented with values $logspace(2, -9, num = 25)$
- For the **Random Forest** model we covered all possible combinations of the following parameters:
  - **I (numIterations)**: Is the number of trees in the forest, and we experimented with values $[5, 10, 100, 1000, 2000]$
  - **K (numFeatures)**: Is the max number of features considered for splitting a node, and we experimented with values $[0, 10, 100, 500, 1000]$
  - **depth**: Is the maximum depth of the tree (0 unlimited), and we experimented with values $[0, 5, 10, 20]$

- **M (minNumObj)**: Is the minimum number of instances in a leaf , and we experimented with values [1, 10, 20, 50, 100]

It is important to note that, the aforementioned Grid search setting results in the application of a 10-fold cross-validation strategy over all experiments, with the usage of the configuration of features that achieves the best classification performance when using the default hyperparameter configurations of the ML models (as it is possible to see in Section 4.1). This means that the first round of experiments with default parameters had the aim of establishing the best features' set to train the various ML models (which is $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$), while the hyperparameter tuning computed with Grid search focuses on investigating if it is possible to improve the performance using only the best features' set.

3.4.2 FastText. Complementary to the aforementioned shallow ML models, we experimented with fastText (by using default values of parameters[2]), a library leveraging word embeddings as internal representation for text classification created by Facebook's AI Research lab. FastText is an extension of Word2Vec: while Word2Vec, under the hood, uses words to predict words, fastText operates at a more granular level and breaks words into several n-grams (sub-words) [19], benefiting from this information for better representing rare and out-of-vocabulary words. FastText was successfully used for identifying the correct labels to assign to GitHub issues [108, 109]. Recent research demonstrated that fastText is among the best performing classifiers in issue type prediction tasks [100].

3.4.3 Evaluation Metrics. To evaluate the performance of the selected ML models, we adopted well-known metrics used to assess the performance of ML classifiers, namely precision, recall, and F-measure [8]. For this kind of problem, the choice of such evaluation metrics is dictated by having an indication of:

- the extent to which the experimented classifiers avoid false alarms in terms of safety issues (precision);
- the extent to which the classifiers do not miss safety-related sentences (recall); and
- an aggregation of the above (F-measure).

Furthermore, as a binary classifier may perform very well on a given label and poorly on the other (*e.g.,* performing worse than a constant classifier), we report the metrics for all labels (*i.e., Safety* and *Non-Safety*, referring to the safety relevance of issues). As we perform cross-validation, precision, recall, and F-measure are averaged over the 10 folds.

We statistically compared the performance of different classifiers using the McNemar's exact test [128] from the R *exact2x2* package [3]. The McNemar's test is a pairwise proportion tests. The test pairwise compares two classifiers by analyzing proportion differences on a confusion matrix counting the number of cases where both classifiers succeed, both fail, the first succeeds and the second fails, and vice versa. Due to multiple comparisons, the resulting p-values are adjusted using the Benjamini-Hochberg correction [13]. The McNemar's test is complemented by the Odds Ratio (OR) effect size measure, which is the ratio between the odds of the experimental and control group (OR> 1 indicates that the experimental group has higher odds to succeed).

To identify the features that contribute more towards the safety sentence classification, we report feature importance indications. We do so for the ML technique that performs better. As discussed in Section 4.1, this is the case of Random Forest. Therefore, we compute and report the Mean Decrease Gini (also called Mean Decrease in Impurity), which is the standard technique to analyze feature importance for Random Forest [20, 53, 125, 164].

[2]For further details, see https://fasttext.cc/docs/en/options.html
[3]https://cran.r-project.org/web/packages/exact2x2/index.html

3.5 RQ_2: What are the main hazards and accidents emerging from safety issues reported in UAV software platforms?

To identify hazards and related accidents, *i.e.,* events or conditions resulting in unexpected behavior being able to damage people, the environment and/or the equipment, we manually analyzed the 273 issues containing at least one sentence describing a safety concern, through a card sorting strategy [155]. Specifically, the manual analysis has been performed by four coders (all of them are authors of the paper), guaranteeing that each bug was classified by two different coders. Since the labeling procedure did not start with a predefined set of categories related to hazards and accidents, even in this case, we performed a pilot study, on the same 10 issues used in the previous pilot study, to determine a set of coding guidelines to follow during the subsequent independent labeling procedure.

To perform the issue labeling process, we used online spreadsheets (a separate spreadsheet for each coder), where coders could use drop-down menus to select previously identified categories or add new ones when those did not fit. Note that, during the labeling process, the categories identified by one coder were not shared with the others, finally resulting in 465 different categories of hazards and 43 categories of accidents. Going over the identified categories, we found that, in several cases, different coders used different wordings for pointing out the same hazards and accidents. Further discrepancies between some of the identified categories depended on the different granularity levels used by the coders while reporting them. For these reasons, before the resolution of disagreements, two coders went through the whole set of the originally identified categories and clustered them. In the end, they obtained 20 categories describing hazards, and eight categories related to accidents. The categories that emerged from the clustering were used to refine the original labeling results. Specifically, all four coders leveraged such categories to properly discuss, through online meetings, the cases in which a disagreement between them in terms of hazards or accidents occurred. Furthermore, to reduce the agreement by chance that could have been introduced as a consequence of the clustering procedure on the original lists, we proceeded to discuss also the cases where the original coders agreed.

To sum up, out of the 273 safety-related issues analyzed, 258 clearly mentioned the hazard, while only 115 indicated the accident occurred or which might have likely occurred. The final categorization we provide is made up of 19 hazards and 7 accidents.

4 RESULTS

In this section, we report and discuss the results for RQ1 and RQ2.

4.1 RQ_1: To what extent can machine learning models automatically identify safety-related concerns in issue reports of UAV software platforms?

As detailed in Section 3.4, we experimented with different ML classifiers, namely Naive Bayes, J48, SMO, Logistic Regression, Random Forest, and fastText, leveraging different combinations of features. In the following, we report and discuss our results. Note that we only report statistical comparisons where appropriate, because we computed statistical comparisons among all possible comparisons of treatments. Detailed statistical results can be found in the replication package.

Table 2 summarizes the results obtained when considering different ML classifiers with different combinations of features. As shown in Table 2, Random Forest, together with fastText are the classifiers showing the best performance in terms of precision, recall, and f-measure.

Based on the statistical comparison, Random Forest with BOW outperforms all other techniques (adjusted p-values< 0.05) with ORs ranging from 1.3 for the comparison with BOW+SMO and 2.95 for the comparison with BOW+Logistic regression. There is no statistically significant difference when augmenting BOW with OB/EB/S2R and applying Random Forest (p-value=0.6). Indeed, the weighted precision, recall, and f-measure only slightly

Table 2. RQ1 Performance of the ML classifiers for classifying safety-related sentences

ML Model	Processing	Safety Pr	Safety Rc	Safety F_1	Non-Safety Pr	Non-Safety Rc	Non-Safety F_1	Weighted Avg. Pr	Weighted Avg. Rc	Weighted Avg. F_1
Naive Bayes	BOW	0.71	0.73	0.72	0.79	0.77	0.78	0.76	0.75	0.75
	LSI	0.62	0.79	0.70	0.80	0.63	0.70	0.72	0.70	0.70
	BOW+OB/EB/S2R	0.71	0.74	0.72	0.79	0.77	0.78	0.76	0.75	0.76
	BOW+n-grams+OB/EB/S2R	0.48	0.96	0.64	0.86	0.19	0.31	0.69	0.53	0.45
J48	BOW	0.72	0.66	0.69	0.75	0.80	0.78	0.74	0.74	0.74
	LSI	0.65	0.64	0.65	0.72	0.73	0.73	0.69	0.69	0.69
	BOW+OB/EB/S2R	0.74	0.67	0.70	0.76	0.81	0.79	0.75	0.75	0.75
	BOW+n-grams+OB/EB/S2R	0.74	0.67	0.70	0.76	0.82	0.79	0.75	0.75	0.75
SMO	BOW	0.76	0.72	0.74	0.79	0.83	0.81	0.78	0.78	0.78
	LSI	0.78	0.72	0.75	0.80	0.84	0.82	0.79	0.79	0.79
	BOW+OB/EB/S2R	0.77	0.74	0.75	0.80	0.83	0.82	0.79	0.79	0.79
	BOW+n-grams+OB/EB/S2R	0.77	0.72	0.74	0.79	0.84	0.81	0.78	0.78	0.78
Logistic Regr.	BOW	0.58	0.59	0.58	0.68	0.66	0.67	0.63	0.63	0.63
	LSI	0.77	0.74	0.76	0.81	0.83	0.82	0.79	0.79	0.79
	BOW+OB/EB/S2R	0.59	0.59	0.59	0.68	0.68	0.68	0.64	0.64	0.64
	BOW+n-grams+OB/EB/S2R	0.74	0.72	0.73	0.78	0.80	0.79	0.76	0.77	0.76
Random Forest	BOW	0.80	0.74	0.77	0.81	0.85	0.83	**0.80**	**0.80**	**0.80**
	LSI	0.68	0.75	0.71	0.79	0.72	0.75	0.74	0.74	0.74
	BOW+OB/EB/S2R	0.80	0.75	0.78	0.82	0.86	0.84	**0.81**	**0.81**	**0.81**
	BOW+n-grams+OB/EB/S2R	0.83	0.65	0.73	0.77	0.90	0.83	0.79	0.79	0.79
fastText	–	0.78	0.80	0.79	0.85	0.81	0.83	**0.83**	**0.81**	**0.81**

increase from 0.80 to 0.81. We can also notice that the introduction of n-grams in the model slightly lowers the performance to 0.79 for all metrics, yet such a difference is not statistically significant (p-value=0.31). This may happen because terms contributing toward the classification of a safety-related sentence are unlikely to be sequences of adjacent words.

When training the classifiers using the features extracted with LSI, Logistic Regression outperforms Random Forest: 0.79 versus 0.74 on all the evaluation metrics being considered. The difference is statistically significant (p-value$<$ 0.01, with an OR=1.5). However, there is no statistically significant difference with BOW+Random Forest (p-value= 0.28). Hence the application of LSI in this context may not result as particularly convenient, given that building a LSI space is typically more expensive than just creating a simple BOW model.

As regards the classifiers showing the worst performance, we found that Naive Bayes trained with all features shows an average f-measure of 0.45, while Logistic Regression trained with BOW features has an average F-measure of 0.63. Based on the results reported in Table 2 it is possible to conclude that, for most of the ML classifiers, the performance tends to be already very positive when considering the BOW features, with marginal improvements when adding OB/EB/S2R. Only Naive Bayes does not show an improvement in terms of F-Measure results when combining different types of features. The latter might happen because the Naive Bayes classifiers show limited improvements when increasing the number of features [138, 139].

It is important to highlight that, in general, for almost all the ML models trained with multiple features, the performance tends to be better in identifying true negatives (i.e., issues not containing safety-related concerns).

This may happen because the dataset used for training the classifiers is imbalanced, *i.e.,* $\simeq 56\%$ of the sentences do not discuss any safety-related concerns.

When comparing the results obtained by Random Forest and fastText, we must mention that, as shown in Table 2, fastText and Random Forest both achieve an average f-measure value of 0.81. It is important to highlight that fastText achieves good performance by using a standard configuration (as detailed in Section 3.4.2), while Random Forest, achieves similar results only when applying several pre-processing steps and experimenting with specific sets of features. Interestingly, fastText achieves a higher recall on the *Safety* class, which is important to identify safety-related issues timely during the issue management process. It is worth noticing that fastText does not outperform the other experimented techniques by a large margin (as it happens in the context of assigning labels to issues by considering the whole text reported [109]) achieving comparable results to the ones achieved by the Random Forest classifier trained with $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$ features.

Our dataset has two levels of imbalance: (i) the imbalance of the projects (*i.e.,* a project has many more safety issues than the others) and (ii) the imbalance of the number of samples in the *Safety* and *Non-Safety* classes. These two issues are addressed in this Section (see "Imbalance handling techniques" and "Cross-project analysis" paragraphs).

Imbalance handling techniques. To check whether the results are biased due to the imbalance of the number of samples in the *Safety* and *Non-Safety* classes, we also report the results of imbalance handling techniques applied to our dataset, considering the best performing shallow ML model in all our experiments, *i.e.,* Random Forest trained with $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$ features. We then compare the results of Random Forest when using the default configuration and its variant using imbalance handling techniques. Specifically, the imbalance handling approach applied was conducted by performing the following steps:

(1) 242 *Non-Safety* sentences were randomly discarded from the dataset (*i.e.,* undersampling), to obtain a balanced dataset (*i.e.,* 837 *Safety* sentences and 837 *Non-Safety* sentences).
(2) We leveraged the balanced dataset obtained from the previous step to experiment with 10-fold cross-validation using the default configuration of Random Forest trained with $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$ features.

Hence, we decided to use undersampling (and not oversampling) as the imbalance handling strategy since, as reported in previous work [159], oversampling methods tend to generate false examples, causing classifiers to perform well in labs but more likely to fail in practice. We follow the recommendation of such previous work [159] that suggests avoiding oversampling methods when dealing with sensitive applications such as security, autonomous driving, aviation safety, and medical applications. Given the projects investigated in our study fall in the domain of aviation safety, we decided to experiment with the undersampling strategy. Table 3 reports the results achieved by (i) Random Forest when using its default configuration (trained with $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$ features) and 10-fold cross-validation on the unbalanced original dataset (described in Section 3.2), and (ii) the default configuration of Random Forest (trained with $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$ features) in a 10-fold cross-validation setting by leveraging the balanced dataset obtained when applying the undersampling strategy. As we can observe from Table 3, the performance obtained by the two variants of Random Forest are almost identical, so no major improvements are achieved by balancing the dataset. We conjecture that this happens because our dataset does not present a heavy imbalance in terms of the number of samples in the *Safety* and *Non-Safety* classes. Indeed, as reported in Section 3.3, we have a fairly balanced set of sentences: *i.e.,* 837 *Safety* sentences (representing 43.7% of the data), and 1,079 *Non-Safety* sentences (representing 56.31% of data).

Cross-project analysis. As detailed in Section 3.4, to check whether the results are biased due to the dataset imbalance (*i.e.,* the majority of issues and pull requests considered in our dataset belong to ARDUPILOT), we also report results of a cross-project analysis, in which we use the best performing configurations, *i.e.,* Random Forest trained with $\mathbf{M}_{BOW}$ + $\mathbf{M}_{OB-EB}$ features, and fastText. As it is shown in Table 4, for Random Forest, the average precision and recall values decreased compared to a 10-fold setting. Consequently, the overall F-measure

Table 3. Comparison of results of Random Forest when using M_{BOW} + M_{OB-EB} features and 10-fold cross-validation on imbalanced and balanced datasets.

ML Technique	Precision	Recall	F-Measure
Random Forest without imbalance handling techniques	0.81	0.81	0.81
Random Forest with undersampling	0.81	0.81	0.81

Table 4. Cross-project Evaluation based on Random Forest using BOW and OB-EB combination of features and fastText

ML Technique	Class	Precision	Recall	F-Measure
Random Forest	Safety	0.63	0.37	0.46
	Non-Safety	0.69	0.87	0.77
	Weighted Avg.	0.67	0.68	0.65
FastText	Safety	0.79	0.62	0.70
	Non-Safety	0.79	0.90	0.84
	Weighted Avg.	**0.79**	**0.76**	**0.77**

decreases of 0.16. In particular, for the Random Forest algorithm, we observe a quite significant degradation in the recall concerning the identification of the sentences labeled as *Safety*. This performance degradation could happen because the terminology used to describe safety issues can be slightly different in ARDUPILOT, compared to the other projects.

When comparing the performance of fastText in the cross-project setting and the results achieved by this model in a 10-fold setting (see Table 2), we can observe that such performance only slightly decreased, *i.e.,* the weighted average f-measure in a cross-project setting is 0.77 while it is equal to 0.81 in a 10-fold setting. Specifically, in a cross-project setting, the default implementation of fastText achieves a recall of 0.62 for the sentences in the *Safety* category, suggesting that the usage of word embeddings (*i.e.,* the features' representation strategy leveraged by fastText) allows to partially mitigate the problem of different vocabularies used in heterogeneous projects to describe safety-related problems.

Feature Importance. To better understand the features that contribute more to the classification, we computed the Mean Decrease Gini (also called Mean Decrease in Impurity) [53, 125, 164] for Random Forest, considering BOW and OB/EB/S2R features. As shown in Fig. 3, we can find intuitive (top 20) features considered important for the classification of safety-related sentences. Specifically, hardware-related features (*e.g., motor* and *vehicle*) and aspects concerning concrete states (*e.g., failsafe*) or actions (*e.g., arm* and *disarm*) of the UAVs tend to be considered as important features by the model. Although the keyword *safety* is among the top 20 relevant features, it only appears in 48 out of 837 (less than 6%) safety-related sentences in our dataset.

Hyperparameter Optimization. As detailed in Section 3.4.1, we experimented with Grid search as hyperparameter optimization approach [2, 14, 15] to investigate potential optimal combinations of parameters for the selected shallow ML models. Specifically, with Grid search, we experimented with several parameter combinations for the various ML models. In total, we experimented with around 600 combinations using a 10-fold validation setting for the selected shallow ML models (all detailed results are shared in our replication package along with the Python code used to run the Grid search experiments). As is shown in Table 5, for Random Forest, the average Precision, Recall, and F-Measure values slightly increased (by about +1%) compared to its default configuration. For most of the other models, the obtained improvements are slightly more evident in terms of

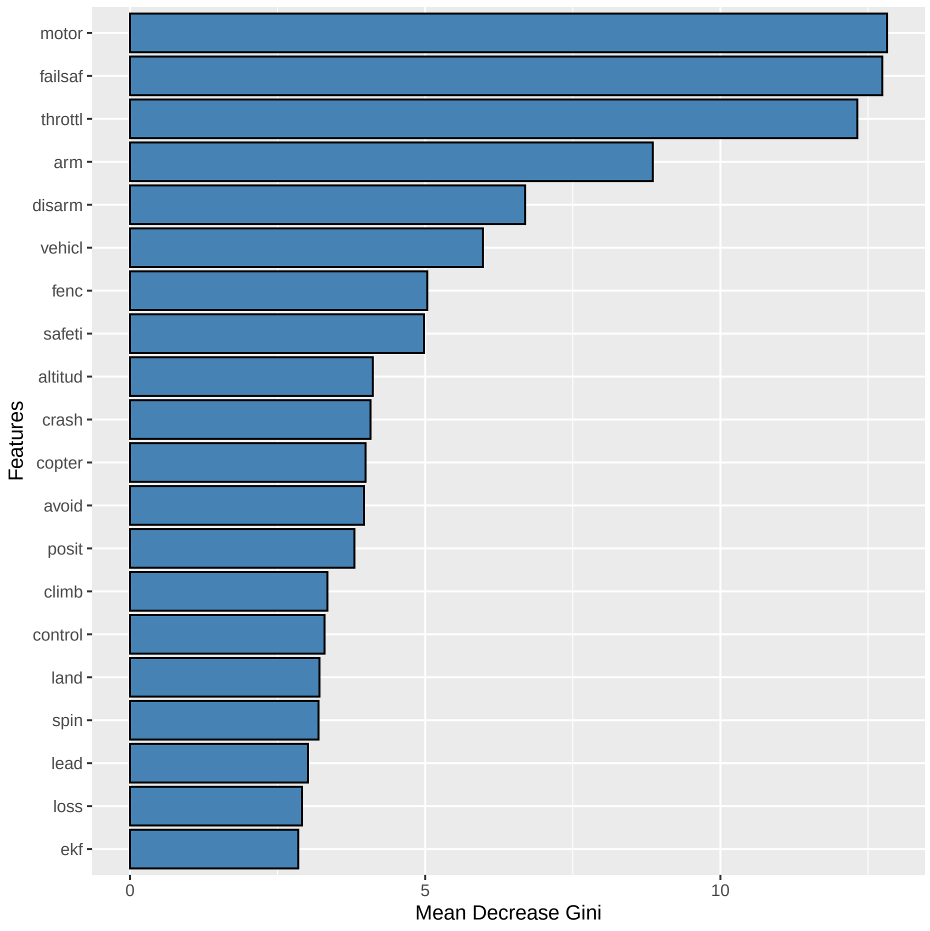

Fig. 3. Mean Decrease Gini when using BOW and OB/EB/S2R combination of features. The top 20 features are visualized.

Precision, Recall, and F-Measure. In particular, for the J48 algorithm, we observe a +2% of improvements for all metrics. Similarly, for Naive-Bayes we observe small improvements in Recall (+1%), Precision (+2%), and F-Measure (+1%) values. For Logistic Regression we observe more substantial improvements in Recall (+16%), Precision (+16%), and F-Measure (+16%) values. SMO is the only model that did not achieve any observable improvement in terms of F-Measure. Overall, considering mainly the model having the higher values of both Precision and Recall (*i.e.,* values > 0.80), Random Forest and fastText are the best performing models, as observed before, with Random Forest, slightly better in terms of F-Measure (+1%). Interestingly, after a Grid search optimization step, all experimented ML strategies are not able to achieve more than 0.82 of F-Measure, which represents—within the considered hyperparameter values—the upper bound of our experiments.

Table 5. Grid search on shallow ML models using BOW and OB-EB combination of features

ML Technique	Best Parameters Config.	Precision	Recall	F-Measure
Random Forest	I=1000, K=1 depth=0, M=10	0.82	**0.82**	**0.82**
J48	C=0.1, M=50, R=no, S=yes	0.77	0.77	0.77
SMO	C=0.8	0.79	0.79	0.79
Logistic	R=100	0.80	0.80	0.80
Naive-Bayes	K=yes	0.77	0.77	0.77

RQ_1 Summary: *The Random Forest (RF) algorithm obtains precision, recall, and F-measure values of about 81-82%, with the model considering hardware-related keywords and text features about states (or actions) of the UAVs as the most important ones. Considering additional features to the BOW ones or experimenting with grid search strategies do not substantially impact the classification performance of RF. The RF results are similar to the ones obtained by using the default implementation of fastText in a 10-fold validation setting. FastText does not outperform the other experimented techniques by a large margin, as it happens in the context of assigning labels to issues. Nevertheless, fastText showed more promising results in a cross-project setting compared to RF.*

4.2 RQ_2: What are the main hazards and accidents emerging from safety issues reported in UAV software platforms?

Table 6. Hazard categories and corresponding occurrences in our dataset of 273 safety-related issues and pull requests.

Category	Description	Occ.
event/action allowed in undesired states/configurations	Scenarios where some events, *e.g.*, in-flight reboot, or actions, *e.g.*, disable safety checks, are allowed while they should not	40 (14.65%)
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	34 (12.45%)
undesired failsafe behavior	The behavior of the system to deal with a failsafe operation does not reflect the expected one	32 (11.72%)
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	25 (9.16%)
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	20 (7.33%)

Continued on next page

Table 6 – *Continued from previous page*

Category	Description	Occ.
misleading or missing data/measurements/reports	Data coming from external devices and sensors being noisy or missing, *e.g.,* wrong values in transmission channels	18 (6.59%)
missing or misleading communication with pilot	Conditions and/or events not notified to the pilot, *e.g.,* no alert in presence of unusually high throttle output	16 (5.86%)
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	12 (4.40%)
action not allowed in specific states/configurations	Scenarios where some events or actions, *e.g.,* disarm speed controller, are not allowed while they should	11 (4.03%)
inappropriate mode changes/ handling	Incorrect switches between modes, or inappropriate handling of rapid mode switching	9 (3.30%)
onboard instrumentation issue	Unexpected behavior in the instrumentation being available onboard, *e.g.,* GPS, GCS, or RC	7 (2.56%)
inappropriate safety switch handling	The safety switch used to enable/disable the outputs to motors and servos is not handled based on what reported in the requirements specification	7 (2.56%)
undetected failure	The system is not able to properly detect the presence of failures in both the hardware and software components, *e.g.,* undetected and unhandled airspeed blockage	7 (2.56%)
timing/timeout/ synchronization issue	Inappropriate handling of synchronization across different functionalities and timing issues when various components, both hardware and software, must communicate	5 (1.83%)
communication failure	Two or more hardware components/devices fail to properly communicate, *e.g.,* data are sent when not needed/requested	4 (1.47%)
lack of control	Scenarios where it is not possible to retake control of the system after specific conditions	4 (1.47%)
inappropriate handling for high vibrations	The occurrence of high vibrations introducing instabilities are not properly handled, *e.g.,* no failsafe actions for high vibration	3 (1.10%)
memory/stack issue	The system lacks a proper handling mechanism of the memory/stack needed to guarantee that the system behaves as expected, *e.g.,* to store new data coming from sensors	2 (0.73%)
battery issue	Battery voltage insufficient for arming or battery insufficient to guarantee a full flight	2 (0.73%)

To explore the recurrent hazards and accidents reported in UAV software platforms, and answer RQ_2, we performed a manual analysis of the 273 issues and pull-requests of our dataset containing at least one sentence describing a safety-related concern (as described in Section 3.5). Specifically, based on the title and the description content, each issue is assigned to at most one hazard and/or accident category. Upon completion of the manual inspection and clustering procedure (see Section 3.5), we found 19 different categories of hazards and seven categories of accidents reported in Tables 6 and 7, respectively, along with their occurrences.

As reported in Table 6, undesired hardware behavior [75] (4.40%), onboard instrumentation issues, such as GPS, GCS, or RC connection lost during flight [57] (2.56%), communication failures, such as improper communication with motors and between motors [76] (1.47%), inappropriate handling (due to either hardware or software defects) for high vibrations [89] (1.10%), and battery-related issues [77] (0.73%) are signaled as sources of hazards in only about 10% of the cases, meaning that, in the great majority of the considered reports, the hazardous circumstances depend on software-related defects. Specifically, in about 40% of the analyzed issues, the reported hazards depend on misbehaviors occurring when the system enters failsafe mode (*undesired behavior on failsafe or error condition/configuration* + *undesired failsafe behavior*, 24.18%), or the missing inhibition of certain actions (*e.g.,* reboot while in-flight [87], motor lockdown during takeoff [95], etc.) when the system is in specific states (14.65%).

Concerning hazards happening in failsafe mode, we distinguish misbehaviors depending on improper failsafe settings [54] (*i.e., undesired failsafe behavior*) from unsafe actions occurring when a failsafe (or error condition) is triggered [60] (*i.e., undesired behavior on failsafe or error condition/configuration*). As regards the former, Issue # 292 [54] from ARDUPILOT having as title *"Battery Monitoring RTL Bug"* states a case where there is a bug inside the functionality handling the failsafe condition due to low battery. Specifically, *"a low battery warning can cause a permanent RTL that can[not] be overridden"*. As regards the latter, instead, Issue # 697 [60] from ARDUPILOT clearly mentions the case where the propellers *"go to 100% Full Throttle"* when a radio signal loss occurs (*i.e.,* the failsafe condition is triggered), resulting in the vehicle flipping.

On the one hand, in ≃ 9% of the inspected reports, the risky situations are due to insufficient checks such as Issue # 6649 [79] from ARDUPILOT where there is a need to *"check that the battery voltage is above the ARMING_MIN_VOLT and ARMING_MIN_VOLT2 parameter values"* before arming the vehicle. On the other hand, there are 20 (7.33%) issues/pull requests discussing hazardous situations due to an improper parameter setting, such as PR # 1516 [81] from DRONIN where there is the need for changing the output calibration, *i.e., "offset the minimum of the calibration range up a little bit from what we send"* during disarming.

Furthermore, in ≃ 13% of the manually analyzed pull requests and issues, misleading data [65, 83] (6.59%), or missing warnings when the system is under anomalous conditions [62, 69] (5.86%) are highlighted as risky situations by both developers and/or end-users.

In the remaining cases, dangerous events are related to the inability to perform certain actions when the system is in specific states [68] (4.03%), improper mode switches (*inappropriate mode changes/handling* [86] + *inappropriate safety switch handling* [84], 5.86%), the missing detection of failures occurred [85] (2.56%), race [90] or timing [71] conditions (1.83%), impossibility to take control [63] (1.47%), and memory size issues [70] (0.73%).

Moving the attention to the accidents generated by risky situations, as expected, crashes/collisions [64, 66] or abnormal behaviors while in-flight [74] are the most recurrent ones. Specifically, (potential) accidents in these categories have been indicated in more than 20% of the manually inspected issues (see Table 7). In a further 8% of the analyzed reports, users signaled (potential) mishaps connected with problematic landing/return-to-location operations. For instance, users experienced the automated triggering of blind [93] or hard [91] landings after the occurrence of unexpected events, confirming that landing is among the most critical and accident-prone phases of a UAV flight [44]. Flyaways [161] (*i.e.,* the devices fly off from their users) are signaled in nearly 7% of investigated issues, while stabilization/positioning accidents [88] are disclosed in less than 4% of the cases. Finally, in less than 3% of the inspected reports, users indicated (potential) operator injuries [73] (1.83%) or takeoff issues [58] (0.73%).

Table 7. Accident categories and corresponding occurrences in our dataset of 273 safety-related issues and pull requests.

Category	**Description**	**Occ.**
crashing/collision issue	The drone crashes on the ground or else collides with a an object while in-flight	35 (12.82%)
anomalous flight conduct	The drone manifests anomalous behavior while in-flight, *e.g.,* in-flight resets	23 (8.42%)
landing/rtl issue	The drone has anomalous behavior while landing or returning-to-location	21 (7.69%)
aircraft lost/flown away	The drone stops taking actions from the pilot and starts flying autonomously	19 (6.96%)
stabilization/orientation/ positioning issue	The drone experiences issues with positional and localization data, *e.g., "you can create congestion on the serial port to the GPS and the position data gets delayed"*	10 (3.66%)
operator injury	The unexpected behavior of the drone leads to damage to the operator, *e.g.,* motors spin when connecting battery	5 (1.83%)
takeoff issue	The drone experiences issues during takeoff	2 (0.73%)

To better understand the cause-effect relationships between the different hazard and accident categories, Fig. 4 reports the occurrences with which specific hazard categories (detailed in Table 6) co-occur with the different accident categories (see Table 7). To avoid discussing relations occurring only once in our dataset, Fig. 4 only shows the co-occurrences that took place more than once. Note that, the thickness of the lines is proportional to the number of issues reporting, at the same time, a specific hazard (on the left) and a specific accident (on the right).

Out of the overall 34 issues describing undesired behaviors when the system enters the failsafe mode or encounters an error condition (see Table 6), (i) 8 (23.53%) led to landing/rtl problems, (ii) 6 (17.65%) caused crashes or collisions, and (iii) 3 (8.82%) resulted in flyaways, *e.g.,* PR # 8039 [82] from ARDUPILOT stating *"Prevent DCM fallback from triggering a flyaway"*. As regards the former, Issue # 1164 [61] from ARDUPILOT reports a situation where the undesired behavior of the drone when a battery failsafe is triggered results in an immediate undesirable landing of the vehicles (*i.e., "give the operator the possibility to calculate and configure a battery level at which it is still safe to RTL/fly to a failsafe destination before landing"*). Issue #1234 [67] from DRONIN, instead, discusses a scenario where the drone crashes as a consequence of *"motors keep running at neutral throttle until the arming timeout has expired"* once a TX failsafe is triggered — *"this is unsafe, as the motors will likely burn up in a crash."*

Similarly, out of the overall 32 signaled undesired failsafe behaviors (see Table 6), (i) 4 (12.50%) led to crashes or collisions, (ii) 3 (9.38%) caused landing/rtl problems, (iii) 3 (9.38%) provoked stabilization/positioning accidents, (iv) 3 (9.38%) induced abnormal in-flight behaviors, and (v) 3 (9.38%) co-occurred with flyaways. For instance, Issue #374 [55] from ARDUPILOT clearly discussed a scenario where the *"copter lost around 30m of height"* due to a bug in the handling of the failsafe operation triggered by a RC-signal loss. Differently, when talking about anomalous behavior of the UAV while in flight, PR # 16594 [96] from ARDUPILOT reports about a situation where it is mandatory to have *"the EKF failsafe [being] trigger[ed] soon after the vehicle is armed in stabilizing (or any other non-GPS mode) if it does [not] have a good position estimate."*

Crashes or collisions might also be due to (i) misleading data (11.43% of reported crashes/collisions) such as *"there is no filtration in the current sensor drivers, and any noise gets passed down as obstacles; this gives a sudden jerky response by the vehicle, which may even lead to crash"* [97], (ii) inadequate checks (11.43% of reported crashes/collisions), (iii) missing inhibition of certain actions in specific states (11.43% of reported crashes/collisions) like *"allowing the operator to inhibit ADS-B avoidance below the specified altitude"* so that it is possible to avoid *"crashing into trees or buildings"* as described in PR # 7074 [80] from ARDUPILOT, (iv) undesired hardware behaviors

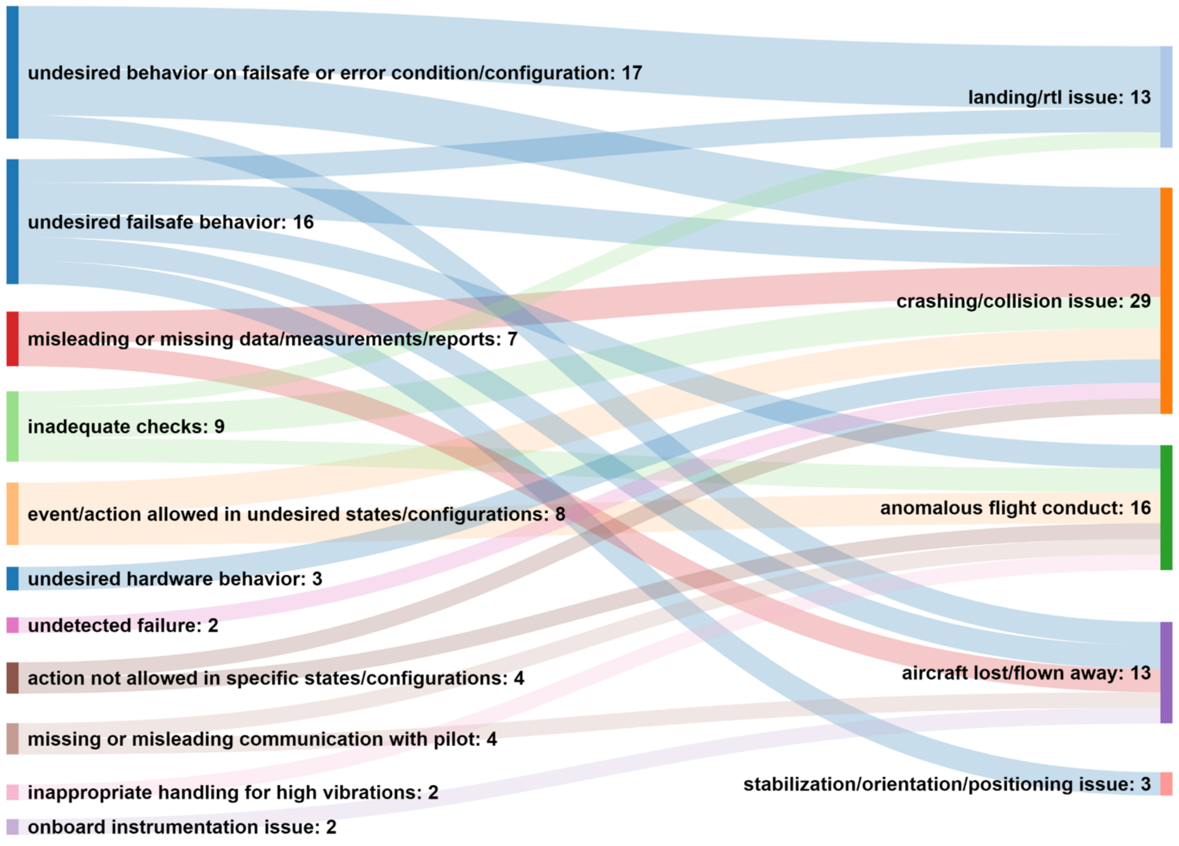

Fig. 4. Co-occurrences of hazard categories and accident categories.

(8.57% of reported crashes/collisions), (v) undetected failures (5.71% of reported crashes/collisions), or (vi) inability to perform certain actions (5.71% of reported crashes/collisions).

Besides, out of the 16 issues reported as an hazard the missing or misleading communication with the pilot, 2 (12.50%) caused flyaways, and 2 (12.50%) led to abnormal in-flight behaviors, such as *"add warning to user from vehicle and ground station when user is approaching the fence [so that] it's more obvious for the user to see the vehicle distance from the fences"* [62]. Abnormal in-flight behaviors might also arise when specific actions are not inhibited (17.39% of the reported abnormal in-flight behaviors), or disabled (8.70% of the reported abnormal in-flight behaviors), and in case of inadequate checks (13.04% of the reported abnormal in-flight behaviors) or high vibrations were not properly handled (8.70% of the reported abnormal in-flight behaviors). As regards the latter, PR # 12349 [89] from ARDUPILOT describes a case where the inappropriate detection of high vibrations and loss of altitude results in a vehicle climbing very quickly while it is not expected to climb at all.

Flyaways, instead, might also depend on missing, misleading, noisy data or measurements coming from the sensors (15.79% of reported flyaways) such as *"catch fly-aways caused by bad compass heading"* [56] and onboard instrumentation issues (10.53% of reported flyaways), such as *"sudden bad GPS position leads to Loiter flying off (GPSGlitch)"* [57].

Finally, landing/rtl problems might also be caused by inadequate checks (9.52% of reported landing/rtl problems). For instance, PR # 15092 [92] from ARDUPILOT reports a dangerous situation where *"the vehicle continuously climbs or descends at its maximum rate which at best leads to a hard landing"* that is caused by a missing pre-arm check of the EKF's altitude estimate.

The results presented above, on the one hand provide an empirical characterization of safety issues in UAV-related software. On the other hand, such results pave the way to research approaches aimed at supporting a better analysis and testing of UAV software:

- *Have a clear mapping of actions and events enabled or inhibited in each specific configuration or state.* The obtained results showed that, if the system performs an action in a given state, or handles an event that should be ignored, or fails to handle an event that should be handled, then accidents can occur. Therefore, the studied bugs urge the need for clearly specifying—*e.g.,* through state machines—the system behavior in different states. Moreover, given such state models, it is desirable to leverage suitable hazard analysis approaches [10] or testing approaches [135] not only to check whether events/inputs are correctly captured in each state, but also, through the testing of *sneaky paths* (*i.e.,* unspecified transitions in a state machine) that unexpected events do not have unintended consequences.
- *Failsafe and error modes.* We have seen how the improper handle of the failsafe mode may be the cause of accidents, for example when a rotor is not stopped after an emergency landing. On the one hand, this highlights design problems, and the need to clearly specify how the UAV should behave in such failsafe or error mode. On the other hand, this also triggers the need for suitable state-based testing, as explained above. More in general, such results also suggest the need to adapt or customize failsafe and error modes for specific UAVs.
- *Test of misleading data or improper parameter settings.* Misleading data from communication channels or from sensors (*e.g.,* malfunctioning ones) may also cause unintended consequences and, ultimately, accidents. This raises the need to use fault injection [169] and mutation testing techniques with the aim of simulating such errors, and test whether they are correctly handled. Furthermore, this trigger the need for customized sets of high-level mutant operators specialized for such a domain, similar to what has been done in domains such as deep learning-based systems and embedded software [102, 106]. Last, but not least, this triggers the development of approaches aimed at supporting the developer in the definition of input data or state consistency check, where appropriate.
- *Cause-effect analysis.* The analysis of the co-occurrences between hazard and accident categories (see Fig. 4), as well as the frequencies of reported accidents (see Table 7) can be leveraged not only to develop approaches aimed at supporting issue triaging by predicting the likely root causes based on the accidents and other observable elements, but, also, to prioritize test and analysis activities [17, 18], by determining what kinds of root causes can be responsible for the most dangerous accidents.

RQ$_2$ Summary: *In the great majority of the considered reports, the hazardous circumstances depend on software-related defects. The most recurrent signaled accidents are related to crashes/collisions, while, in about 40% of the cases, hazards are due to undesired behaviors occurring when the system enters failsafe mode or to the missing inhibition of certain actions when the system is in specific states.*

5 THREATS TO VALIDITY

Threats to *construct validity* concern the relationship between theory and observation. The first type of threat could be related to imprecision and subjectiveness in our classifications, and specifically to (i) distinguish safety-related sentences from sentences that were not related to safety and (ii) identify hazards and accidents appearing in safety issues. As far as the former is concerned, we first established the classification criteria to drive the analysis carried out by the authors individually, as explained in Section 3.3. To reduce the mistakes due to subjective interpretation, each sentence was classified by two annotators, inter-rater agreement was computed, and disagreements were resolved through a discussion involving a third rater. The most critical disagreements were examined by all the annotators through an open debate. For the identification of hazards and accidents, multiple annotators were involved and a process similar to the previous one was followed. In this case, we applied an open coding strategy. The codes defined by the annotators were then clustered and through a joint discussion discrepancies were resolved. In this case, given the exploratory nature of the coding, achieving an inter-rater agreement was less meaningful. Instead, we discussed every single issue to avoid agreement by chance.

Threats to *internal validity* concern factors internal to the study that could influence our results. First, the splitting of sentences and identification of OB, EB, and S2R may be affected by the imprecision of the used tool *i.e.,* BEE [153]. We mitigated this choice by manually reviewing all sentences and fixing the tool's mistakes at least for the splitting (while on the OB, EB, and S2R the reliability of these features depends on *BEE*'s accuracy).

Performances of the ML techniques used in RQ_1 may depend on the setting of their hyper-parameters. We initially leveraged default settings of *Weka*, knowing that the obtained results can represent a lower bound for the classification performances. Then, we experimented with Grid search as hyperparameter optimization approach to investigate potential optimal combinations of parameters for the selected shallow ML models.

Threats to *conclusion validity* concern the relationship between experimentation and outcome. In RQ_1, we used appropriate statistical procedures (*i.e.,* McNemar's tests and Odds Ratio effect size) to support the comparison of different classifiers.

Threats to *external validity* concern the generalizability of our findings. For RQ_1, the applicability of the classifiers to further projects may require retraining, even though we have shown the performance achieved by the best performing models in a cross-project setting. Another concern could be the ability of the classifiers to properly work on sentences coming from other bugs. To address this issue, we plan for future work to experiment with the investigated approaches in the context of further CPS projects and domains. Also, while we compared five shallow machine learning techniques of different type and fastText, it is entirely possible that other techniques we did not consider could perform better.

For RQ_2, our results—within the UAV domain—in terms of categorization of hazards and accidents may not generalize beyond the studied projects, therefore it is entirely possible (and likely) that studying further projects may enlarge the set of considered categories.

Threats to *reliability validity* concern the ability to reproduce our study. We have created a replication package, available on Zenodo [43], containing data from the analyzed issues, as well as results from the different steps of the manual classifications for both RQ_1 and RQ_2, and raw results of the ML classifiers.

6 RELATED WORK

Our work falls at the intersection between the research fields of (i) issue report classification and (ii) software engineering for UAVs. In the following, we provide an overview of the recent literature and highlight the contributions of our paper concerning each of the two research fields.

6.1 Issue Classification

As reported in previous research [31], for popular projects the number of pending issues constantly grows [110] and, despite the use of labels has a positive impact on the issue management [22], they are rarely used by GitHub developers. For this reason, researchers proposed automated approaches supporting the issue management process, with techniques leveraging machine learning [4, 16, 30], textual analysis [174], natural language parsing [41], and summarization approaches [132, 137, 147].

Other, complementary research directions concern the automated classification and analysis of the textual content of issues [181] for detecting duplicated bugs [174] and predicting reopened issues [177]. In recent years, tools have been designed to automatically estimate the issue lifetime [111, 112].

In the context of issue textual analysis, recent work [138, 172] empirically investigates the combination of ML and textual analysis techniques to automatically predict whether issues will be not fixed, by analyzing (only) the titles and descriptions of reported issues. Similarly, Cabot *et al.* [22] proposed labels to classify issues in open source projects, and (ii) Guo *et al.* [98] presented an approach to determine the bugs that will be actually fixed. Finally, Kallis *et al.* [108, 109] proposed *Ticket Tagger*, a GitHub app analyzing the issue title and description through machine learning techniques to automatically assign labels to each issue submitted on GitHub, accordingly. Based on a default implementation of fastText, *Ticket Tagger* can predict three categories of default labels: *bug*, *enhancement*, and *question*. Herbold *et al.* [100] demonstrated that *Ticket Tagger* represents one of the most effective approaches to identify issues of the *bug* type.

Recently, Wang *et al.* [171] proposed PLPI, a framework able to identify labels with similar meanings and predict project-specific labels to address the problem deriving from the usage of custom label schemes.

Similar to previous research [138], to build the ground truth we perform a manual inspection of UAV issues. However, we specifically focus on safety-related issues, comparing the performance achieved by different ML approaches when used to predict safety-related sentences from user-reported issues of UAV platforms. Our evaluation is useful to help developers timely identify the UAV issues that are likely to be safety-related. Besides, to ease the inspection of such issues, the experimented techniques can also help highlight the specific safety-related sentences occurring in the issue titles and descriptions.

6.2 Software Engineering for UAVs

UAVs are often mentioned as fascinating examples of self-adaptive CPSs. Software engineering for these types of systems is an increasingly explored research field. In particular, recent efforts mainly focused on bug characterization [52], testing [1, 37, 180], and verification [27] of self-adaptive CPSs.

Another emerging area of research is related to the automated generation of oracles for testing and localizing faults in CPSs. For instance, Menghi *et al.* [131] proposed SOCRaTes, an approach to automatically generate online test oracles in Simulink able to handle CPS Simulink models featuring continuous behaviors and involving uncertainties. The oracles are generated from requirements specified in a signal logic-based language. In a similar effort, He *et al.* [99] proposed a system identification-based oracle for fault localization in CPS software.

Cleland-Huang and Vierhauser proposed SafetyScrum [28], an agile software development methodology that augments the Scrum life-cycle with safety-related activities to incrementally track the safety status of UAV systems. Previous research also proposed programming languages for supporting UAV-specific aspects, such as energy awareness [124] and context adaptation [21], while Liang *et al.* [120] focused on how Bounding Functions (*i.e.,* dynamic checks inserted by developers to ensure that specific variables stay within a prescribed range) are used in the Paparazzi autopilot software.

Different researchers proposed methods to perform safety analysis of UAV systems. For instance, Denney and Pai [33–36] focused on different aspects of UAV safety and proposed automated approaches for creating and maintaining safety assurance cases. Recently, Vierhauser *et al.* [168] presented an approach for deriving human-interaction hazards (hazards related to human-system interactions). In particular, the authors provided a set of domain-level hazard trees designed for the safety analysis of UAV systems.

Prior studies that most relate to ours are the ones concerning the analysis of UAV-specific software issues reported by developers and/or end-users. In this context, Wang *et al.* [170] conducted an empirical study to (i) characterize UAV-specific bugs, (ii) identify their root causes, and (iii) propose repairing strategies. They further identified five challenges associated with detecting and fixing such UAV-specific bugs. As in our work, to perform the analysis, they collected issues from PX4 and ArduPilot GitHub projects. Similarly, Taylor *et al.*. [160] analyzed the root causes, severity, and position in the firmware architecture of bugs occurring in these two open-source projects. However, while Wang *et al.* and Taylor *et al.* collected the issues with the *bug* label assigned, we specifically focused on safety-related issues (*i.e.,* having safety-related labels assigned), beyond collecting issues from a further project (*i.e.,* dRonin). Unlike Wang *et al.* and Taylor *et al.*, who focused on the root causes of UAV bugs, we analyzed the safety-related UAV issues to present two sets of categories modeling the main hazards and accidents occurring in the analyzed projects, beyond assessing automated strategies to support the identification of safety-related concerns occurring in such issues. Besides, we also analyzed the frequencies with which the specific hazard categories co-occur with the various accident categories, to better understand recurrent cause-effect relationships.

7 CONCLUSIONS AND FUTURE WORK

Unmanned Aerial Vehicles (UAV) software systems are highly-dependable. The presence of defects in their software can cause harmful accidents, resulting in damage to the vehicle, but also potentially harmful events for people and the environment.

This paper contributes to the state-of-the-art by exploring ways to aid developers when triaging of safety-critical issues for UAVs. Specifically, the paper provides the following contributions:

(1) an evaluation of various Machine Learning (ML)-based classifiers, trained with different text representations and algorithms, by experimenting on a dataset consisting of 1,916 manually-labeled sentences extracted from safety-related issues and pull requests of three popular open-source UAV platforms; and
(2) a categorization of hazards and accidents, along with their relations, obtained by performing an open coding over 273 safety-related issues.

For what concerns the sentence classification, we found that a simple bag-of-words model with a Random Forest classifier already achieves good classification performance, with precision, recall, and f-measure reaching $\simeq 0.8$. More complex models do not achieve additional benefits. Besides, we obtained a set of 19 hazard categories and 7 accident categories. Such categories have been further analyzed and discussed (by also providing qualitative examples) to better understand the most likely occurring accidents for the different hazard categories.

Our study can have relevant implications for both *developers* and *researchers*. For what concerns UAV *developers*, our study triggers the possibility of leveraging automated approaches to support them in identifying safety-related concerns in issue reports. Besides, the elicited hazard and accident categories highlight safety-related specific problems that need to be carefully monitored in UAV development. For instance, to avoid hazardous circumstances, our results underline the need for developers to have a more clear mapping of inhibited and enabled actions/events in each specific configuration/state, beyond the need for implementing redundant checks to detect the presence of misleading data or improper parameter settings.

For what concerns *researchers*, our work fosters research toward better testing and analysis of safety-related issues in UAV development. Indeed, our study highlights the need for envisioning testing strategies that allow more comprehensive audits of the system behaviors in failsafe or error modes. Other researchers could also build on our study and the provided manually-labeled data to probe further automated approaches for labeling safety-related sentences, thus finding the best-performing strategies and settings. Our work also encourages the development of specific tools to help UAV developers avoid safety-related recurring problems. Finally, complementary empirical research could focus on investigating the difficulty (*e.g.,* in terms of duration and effort) of fixing safety-related specific UAV issues and developing tools to help developers allocate the proper resources to various types of UAV-specific safety-related issues.

Future work aims at replicating the work in other contexts, *e.g.,* closed-source ones, but also to experiment the safety issue classification in domains different from UAV, also experimenting the extent to which models can be (partially) transferred between domains or whether they require a complete retrain because of very different safety-related features. Also, we plan to develop further support for safety issue triaging, *e.g.,* by leveraging natural language processing techniques for the automated identification of hazards to better support fault localization and fixing tasks.

ACKNOWLEDGMENT

We gratefully acknowledge the Horizon 2020 (EU Commission) support for the project *COSMOS* (DevOps for Complex Cyber-physical Systems), Project No. 957254-COSMOS.

REFERENCES

[1] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing vision-based control systems using learnable evolutionary algorithms. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. 1016–1026. https://doi.org/10.1145/3180155.3180160

[2] Muhammad Adnan, Alaa Abdulsalm Alarood, Muhammad Irfan Uddin, and Izaz ur Rehman. 2022. Utilizing grid search cross-validation with adaptive boosting for augmenting performance of machine learning models. *PeerJ Comput. Sci.* 8 (2022), e803. https://doi.org/10.7717/peerj-cs.803

[3] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2018. Is it a bug or an enhancement?: a text-based approach to classify change requests. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering, CASCON 2018, Markham, Ontario, Canada, October 29-31, 2018*. 2–16. https://dl.acm.org/citation.cfm?id=3291293

[4] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2008. Is It a Bug or an Enhancement?: A Text-based Approach to Classify Change Requests. In *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds* (Ontario, Canada) *(CASCON '08)*. ACM, New York, NY, USA, Article 23, 15 pages.

[5] Ardupilot.org. 2021. Open Source Drone Software. Versatile, Trusted, Open. ArduPilot. https://ardupilot.org/

[6] Fredrik Asplund, John Alexander McDermid, Robert Oates, and Jonathan Roberts. 2019. Rapid Integration of CPS Security and Safety. *IEEE Embed. Syst. Lett.* 11, 4 (2019), 111–114. https://doi.org/10.1109/LES.2018.2879631

[7] Muhammad Ilyas Azeem, Sebastiano Panichella, Andrea Di Sorbo, Alexander Serebrenik, and Qing Wang. 2020. Action-based Recommendation in Pull-request Development. In *ICSSP '20: International Conference on Software and System Processes, Seoul, Republic of Korea, 26-28 June, 2020*. 115–124. https://doi.org/10.1145/3379177.3388904

[8] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison-Wesley. http://sunsite.dcc.uchile.cl/irbook

[9] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[10] Ahmed Mohamed Bakr, May Salama, and Abdelwahab K. Alsammak. 2021. A Hierarchical State Machine Model for Hazard Analysis of Real-time Safety Critical Systems. *Comput. Sci.* 22, 1 (2021). https://doi.org/10.7494/csci.2021.22.1.3547

[11] Eulalia Balestrieri, Pasquale Daponte, Luca De Vito, and Francesco Lamonaca. 2021. Sensors and Measurements for Unmanned Systems: An Overview. *Sensors* 21, 4 (2021), 1518. https://doi.org/10.3390/s21041518

[12] Eulalia Balestrieri, Pasquale Daponte, Luca De Vito, Francesco Picariello, and Ioan Tudosa. 2021. Sensors and Measurements for UAV Safety: An Overview. *Sensors* 21, 24 (2021), 8253. https://doi.org/10.3390/s21248253

[13] Yoav Benjamini and Yosef Hochberg. 1995. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)* 57, 1 (1995), 289–300.

[14] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems* (Granada, Spain) *(NIPS'11)*. Curran Associates Inc., Red Hook, NY, USA, 2546–2554.

[15] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13 (2012), 281–305. https://doi.org/10.5555/2503308.2188395

[16] Pamela Bhattacharya and Iulian Neamtiu. 2011. Bug-fix Time Prediction Models: Can We Do Better?. In *Proceedings of the 8th Working Conference on Mining Software Repositories* (Waikiki, Honolulu, HI, USA) *(MSR '11)*. ACM, New York, NY, USA, 207–210.

[17] Christian Birchler, Nicolas Ganz, Sajad Khatiri, Alessio Gambi, and Sebastiano Panichella. 2022. Cost-effective Simulation-based Test Selection in Self-driving Cars Software with SDC-Scissor. In *2022 IEEE 29th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. https://doi.org/toappear

[18] Christian Birchler, Sajad Khatiri, Pouria Derakhshanfar, Sebastiano Panichella, and Annibale Panichella. 2022. Single and Multi-Objective Test Cases Prioritization for Self-Driving Cars in Virtual Environments. *ACM Trans. Softw. Eng. Methodol.* (apr 2022). https://doi.org/10.1145/3533818

[19] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2017. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguistics* 5 (2017), 135–146. https://doi.org/10.1162/tacl_a_00051

[20] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (2001), 5–32.

[21] John Henry Burns, Xiaozhou Liang, and Yu David Liu. 2020. Adaptive Variables for Declarative UAV Planning. In *COP '20: Proceedings of the 12th International Workshop on Context-Oriented Programming and Advanced Modularity, COP@ECOOP 2020, Virtual Event, 21 July 2020.* 1:1–1:7. https://doi.org/10.1145/3422584.3422763

[22] Jordi Cabot, Javier Luis Cánovas Izquierdo, Valerio Cosentino, and Belen Rolandi. 2015. Exploring the use of labels to categorize issues in Open-Source Software projects. In *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015.* 550–554.

[23] Giovanni Capobianco, Andrea De Lucia, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. 2009. Traceability Recovery Using Numerical Analysis. In *2009 16th Working Conference on Reverse Engineering.* 195–204. https://doi.org/10.1109/WCRE.2009.14

[24] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering.* 396–407.

[25] Ying Chen and Néstor O. Pérez-Arancibia. 2020. Controller Synthesis and Performance Optimization for Aerobatic Quadrotor Flight. *IEEE Transactions on Control Systems Technology* 28, 6 (2020), 2204–2219. https://doi.org/10.1109/TCST.2019.2919819

[26] Sudipta Chowdhury, Omid Shahvari, Mohammad Marufuzzaman, Xiaopeng Li, and Linkan Bian. 2021. Drone routing and optimization for post-disaster inspection. *Comput. Ind. Eng.* 159 (2021), 107495. https://doi.org/10.1016/j.cie.2021.107495

[27] Shafiul Azam Chowdhury, Sohil Lal Shrestha, Taylor T. Johnson, and Christoph Csallner. 2020. SLEMI: equivalence modulo input (EMI) based mutation of CPS models for finding compiler bugs in Simulink. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020.* 335–346. https://doi.org/10.1145/3377811.3380381

[28] Jane Cleland-Huang and Michael Vierhauser. 2018. Discovering, Analyzing, and Managing Safety Stories in Agile Projects. In *26th IEEE International Requirements Engineering Conference, RE 2018, Banff, AB, Canada, August 20-24, 2018.* 262–273. https://doi.org/10.1109/RE.2018.00034

[29] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[30] William W. Cohen, Vitor R. Carvalho, and Tom M. Mitchell. 2004. Learning to Classify Email into "Speech Acts". In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing , EMNLP 2004, A meeting of SIGDAT, a Special Interest Group of the ACL, held in conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain.* 309–316.

[31] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2017. A Systematic Mapping Study of Software Development With GitHub. *IEEE Access* 5 (2017), 7173–7192.

[32] Raffaello D'Andrea. 2014. Guest Editorial Can Drones Deliver? *IEEE Trans Autom. Sci. Eng.* 11, 3 (2014), 647–648. https://doi.org/10.1109/TASE.2014.2326952

[33] Ewen Denney, Ganesh Pai, and Iain Whiteside. 2017. Modeling the Safety Architecture of UAS Flight Operations. In *Computer Safety, Reliability, and Security - 36th International Conference, SAFECOMP 2017, Trento, Italy, September 13-15, 2017, Proceedings.* 162–178. https://doi.org/10.1007/978-3-319-66266-4_11

[34] Ewen Denney and Ganesh J. Pai. 2014. Automating the Assembly of Aviation Safety Cases. *IEEE Trans. Reliab.* 63, 4 (2014), 830–849. https://doi.org/10.1109/TR.2014.2335995

[35] Ewen Denney and Ganesh J. Pai. 2016. Composition of Safety Argument Patterns. In *Computer Safety, Reliability, and Security - 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings.* 51–63. https://doi.org/10.1007/978-3-319-45477-1_5

[36] Ewen Denney, Ganesh J. Pai, and Ibrahim Habli. 2015. Dynamic Safety Cases for Through-Life Safety Assurance. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2.* 587–590. https://doi.org/10.1109/ICSE.2015.199

[37] Jyotirmoy V. Deshmukh, Marko Horvat, Xiaoqing Jin, Rupak Majumdar, and Vinayak S. Prabhu. 2017. Testing Cyber-Physical Systems through Bayesian Optimization. *ACM Trans. Embed. Comput. Syst.* 16, 5s (2017), 170:1–170:18. https://doi.org/10.1145/3126521

[38] Andrea Di Sorbo, Giovanni Grano, Corrado Aaron Visaggio, and Sebastiano Panichella. 2021. Investigating the criticality of user-reported issues through their relations with app rating. *J. Softw. Evol. Process.* 33, 3 (2021).

[39] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. What would users change in my app? Summarizing app reviews for recommending software changes. In *Proc. Int'l Symposium on Foundations of Software Engineering (FSE).* 499–510.

[40] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Corrado Aaron Visaggio, and Gerardo Canfora. 2017. SURF: summarizer of user reviews feedback. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume.* 55–58. https://doi.org/10.1109/ICSE-C.2017.5

[41] Andrea Di Sorbo, Sebastiano Panichella, Corrado Aaron Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Harald C. Gall. 2015. Development Emails Content Analyzer: Intention Mining in Developer Discussions (T). In *30th IEEE/ACM International Conference on*

Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015. 12–23.

[42] Andrea Di Sorbo, Sebastiano Panichella, Corrado Aaron Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Harald C. Gall. 2016. DECA: development emails content analyzer. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, Laura K. Dillon, Willem Visser, and Laurie A. Williams (Eds.). ACM, 641–644. https://doi.org/10.1145/2889160.2889170

[43] Andrea Di Sorbo, Fiorella Zampetti, Corrado A. Visaggio, Massimiliano Di Penta, and Sebastiano Panichella. 2021. *Dataset of the paper "Automated Identification and Qualitative Characterization of Safety Concerns Reported in UAV Software Platforms"*. https://doi.org/10.5281/zenodo.6207783

[44] Ahmad Din, Basilio Bona, Joel Morrissette, Moazzam Hussain, Massimo Violante, and M. Fawad Naseem. 2012. Embedded Low Power Controller for Autonomous Landing of UAV Using Artificial Neural Network. In *10th International Conference on Frontiers of Information Technology, FIT 2012, Islamabad, Pakistan, December 17-19, 2012*. 196–203. https://doi.org/10.1109/FIT.2012.42

[45] Osama Doss and Tim P. Kelly. 2016. Challenges and Opportunities in Agile Development in Safety Critical Systems: A Survey. *ACM SIGSOFT Softw. Eng. Notes* 41, 2 (2016), 30–31. https://doi.org/10.1145/2894784.2894798

[46] Santiago Dueñas, Valerio Cosentino, Gregorio Robles, and Jesus M Gonzalez-Barahona. 2018. Perceval: Software project data at your will. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 1–4.

[47] Jakob Engel, Vladlen Koltun, and Daniel Cremers. 2018. Direct Sparse Odometry. *IEEE Trans. Pattern Anal. Mach. Intell.* 40, 3 (2018), 611–625. https://doi.org/10.1109/TPAMI.2017.2658577

[48] Matthias Faessler, Davide Falanga, and Davide Scaramuzza. 2017. Thrust Mixing, Saturation, and Body-Rate Control for Accurate Aggressive Quadrotor Flight. *IEEE Robotics Autom. Lett.* 2, 2 (2017), 476–482. https://doi.org/10.1109/LRA.2016.2640362

[49] Philipp Foehn, Angel Romero, and Davide Scaramuzza. 2021. Time-optimal planning for quadrotor waypoint flight. *Sci. Robotics* 6, 56 (2021), 1221. https://doi.org/10.1126/scirobotics.abh1221

[50] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. 2014. SVO: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 15–22. https://doi.org/10.1109/ICRA.2014.6906584

[51] Fortune Business Insights. 2020. Market Research Report: Unmanned Aerial Vehicle (UAV) Market Size, Share & COVID-19 Impact Analysis, By Class (Small UAVs, Tactical UAVs, and Strategic UAVs), By Technology (Remotely Operated, Semi-autonomous, and Fully-autonomous), By System (UAV Airframe, UAV Payloads, UAV Avionics, UAV Propulsion, and UAV Software), By Application (Military, Commercial and Recreational), and Regional Forecast, 2020-2027. https://www.fortunebusinessinsights.com/industry-reports/unmanned-aerial-vehicle-uav-market-101603

[52] Joshua Garcia, Yang Feng, Junjie Shen, Sumaya Almanee, Yuan Xia, and Qi Alfred Chen. 2020. A comprehensive study of autonomous vehicle bugs. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*. 385–396. https://doi.org/10.1145/3377811.3380397

[53] Carina Gerstenberger and Daniel Vogel. 2015. On the efficiency of Gini's mean difference. *Stat. Methods Appl.* 24, 4 (2015), 569–596. https://doi.org/10.1007/s10260-015-0315-x

[54] Github.com. 2013. ArduPilot/ardupilot Issue #292. https://github.com/ArduPilot/ardupilot/issues/292. (Last access: 18/02/2022).

[55] Github.com. 2013. ArduPilot/ardupilot Issue #374. https://github.com/ArduPilot/ardupilot/issues/374. (Last access: 18/02/2022).

[56] Github.com. 2013. ArduPilot/ardupilot Issue #572. https://github.com/ArduPilot/ardupilot/issues/572. (Last access: 18/02/2022).

[57] Github.com. 2013. ArduPilot/ardupilot Issue #616. https://github.com/ArduPilot/ardupilot/issues/616. (Last access: 19/02/2022).

[58] Github.com. 2013. ArduPilot/ardupilot Issue #647. https://github.com/ArduPilot/ardupilot/issues/647. (Last access: 28/01/2022).

[59] Github.com. 2013. ArduPilot/ardupilot Issue #656. https://github.com/ArduPilot/ardupilot/issues/656. (Last access: 18/02/2022).

[60] Github.com. 2013. ArduPilot/ardupilot Issue #697. https://github.com/ArduPilot/ardupilot/issues/697. (Last access: 18/02/2022).

[61] Github.com. 2014. ArduPilot/ardupilot Issue #1164. https://github.com/ArduPilot/ardupilot/issues/1164. (Last access: 18/02/2022).

[62] Github.com. 2014. ArduPilot/ardupilot Issue #1254. https://github.com/ArduPilot/ardupilot/issues/1254. (Last access: 18/02/2022).

[63] Github.com. 2014. ArduPilot/ardupilot Issue #1372. https://github.com/ArduPilot/ardupilot/issues/1372. (Last access: 19/02/2022).

[64] Github.com. 2014. ArduPilot/ardupilot Issue #1516. https://github.com/ArduPilot/ardupilot/issues/1516. (Last access: 30/01/2022).

[65] Github.com. 2014. ArduPilot/ardupilot Issue #1648. https://github.com/ArduPilot/ardupilot/issues/1648. (Last access: 18/02/2022).

[66] Github.com. 2014. ArduPilot/ardupilot Issue #841. https://github.com/ArduPilot/ardupilot/issues/841. (Last access: 30/01/2022).

[67] Github.com. 2014. d-ronin/dRonin Issue #1234. https://github.com/d-ronin/dRonin/issues/1234. (Last access: 18/02/2022).

[68] Github.com. 2015. ArduPilot/ardupilot Issue #2190. https://github.com/ArduPilot/ardupilot/issues/2190. (Last access: 18/02/2022).

[69] Github.com. 2015. ArduPilot/ardupilot Issue #2666. https://github.com/ArduPilot/ardupilot/issues/2666. (Last access: 28/01/2022).

[70] Github.com. 2015. d-ronin/dRonin Pull Request #275. https://github.com/d-ronin/dRonin/pull/275. (Last access: 19/02/2022).

[71] Github.com. 2015. d-ronin/dRonin Pull Request #342. https://github.com/d-ronin/dRonin/pull/342. (Last access: 18/02/2022).

[72] Github.com. 2016. ArduPilot/ardupilot Issue #2011. https://github.com/ArduPilot/ardupilot/issues/2011. (Last access: 18/02/2022).

[73] Github.com. 2016. ArduPilot/ardupilot Pull Request #3669. https://github.com/ArduPilot/ardupilot/pull/3669. (Last access: 31/01/2022).

[74] Github.com. 2016. d-ronin/dRonin Issue #1123. https://github.com/d-ronin/dRonin/issues/1123. (Last access: 28/01/2022).

[75] Github.com. 2016. d-ronin/dRonin Issue #406. https://github.com/d-ronin/dRonin/issues/406. (Last access: 19/02/2022).

[76] Github.com. 2016. d-ronin/dRonin Issue #587. https://github.com/d-ronin/dRonin/issues/587. (Last access: 19/02/2022).

[77] Github.com. 2016. d-ronin/dRonin Issue #642. https://github.com/d-ronin/dRonin/issues/642. (Last access: 19/02/2022).

[78] Github.com. 2016. d-ronin/dRonin Pull request #1129. https://github.com/d-ronin/dRonin/issues/1129. (Last access: 18/02/2022).

[79] Github.com. 2017. ArduPilot/ardupilot Issue #6649. https://github.com/ArduPilot/ardupilot/issues/6649. (Last access: 18/02/2022).

[80] Github.com. 2017. ArduPilot/ardupilot Pull Request #7074. https://github.com/ArduPilot/ardupilot/pull/7074. (Last access: 18/02/2022).

[81] Github.com. 2017. d-ronin/dRonin Pull Request #1516. https://github.com/d-ronin/dRonin/pull/1516. (Last access: 18/02/2022).

[82] Github.com. 2018. ArduPilot/ardupilot Pull Request # 8039. https://github.com/ArduPilot/ardupilot/pull/8039. (Last access: 18/02/2022).

[83] Github.com. 2018. ArduPilot/ardupilot Pull Request #7440. https://github.com/ArduPilot/ardupilot/pull/7440. (Last access: 18/02/2022).

[84] Github.com. 2018. ArduPilot/ardupilot Pull Request #8180. https://github.com/ArduPilot/ardupilot/pull/8180. (Last access: 18/02/2022).

[85] Github.com. 2018. ArduPilot/ardupilot Pull Request #9381. https://github.com/ArduPilot/ardupilot/pull/9381. (Last access: 18/02/2022).

[86] Github.com. 2018. ArduPilot/ardupilot Pull Request #9858. https://github.com/ArduPilot/ardupilot/pull/9858. (Last access: 18/02/2022).

[87] Github.com. 2018. ArduPilot/ardupilot Pull Request #9941. https://github.com/ArduPilot/ardupilot/pull/9941. (Last access: 18/02/2022).

[88] Github.com. 2019. ArduPilot/ardupilot Pull Request #11025. https://github.com/ArduPilot/ardupilot/pull/11025. (Last access: 30/01/2022).

[89] Github.com. 2019. ArduPilot/ardupilot Pull Request #12439. https://github.com/ArduPilot/ardupilot/pull/12439. (Last access: 19/02/2022).

[90] Github.com. 2019. PX4/PX4-Autopilot Pull Request #13560. https://github.com/PX4/PX4-Autopilot/pull/13560. (Last access: 18/02/2022).

[91] Github.com. 2020. ArduPilot/ardupilot Issue #15608. https://github.com/ArduPilot/ardupilot/issues/15608. (Last access: 30/01/2022).

[92] Github.com. 2020. ArduPilot/ardupilot Pull Request #15092. https://github.com/ArduPilot/ardupilot/pull/15092. (Last access: 18/02/2022).

[93] Github.com. 2020. PX4/PX4-Autopilot Issue #15065. https://github.com/PX4/PX4-Autopilot/issues/15065. (Last access: 28/01/2022).

[94] Github.com. 2020. PX4/PX4-Autopilot Pull Request #13926. https://github.com/PX4/PX4-Autopilot/pull/13926. (Last access: 18/02/2022).

[95] Github.com. 2020. PX4/PX4-Autopilot Pull Request #14428. https://github.com/PX4/PX4-Autopilot/pull/14428. (Last access: 18/02/2022).

[96] Github.com. 2021. ArduPilot/ardupilot Pull Request #16594. https://github.com/ArduPilot/ardupilot/pull/16594. (Last access: 18/02/2022).

[97] Github.com. 2021. ArduPilot/ardupilot Pull Request #16604. https://github.com/ArduPilot/ardupilot/pull/16604. (Last access: 18/02/2022).

[98] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2010. Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1* (Cape Town, South Africa) *(ICSE '10)*. ACM, New York, NY, USA, 495–504.

[99] Zhijian He, Yao Chen, Enyan Huang, Qixin Wang, Yu Pei, and Haidong Yuan. 2019. A system identification based Oracle for control-CPS software fault localization. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. 116–127. https://doi.org/10.1109/ICSE.2019.00029

[100] Steffen Herbold, Alexander Trautsch, and Fabian Trautsch. 2020. On the feasibility of automated prediction of bug and non-bug issues. *Empir. Softw. Eng.* 25, 6 (2020), 5333–5369. https://doi.org/10.1007/s10664-020-09885-w

[101] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. 392–401. https://doi.org/10.1109/ICSE.2013.6606585

[102] Robert M. Hierons and Tao Xie. 2021. Adaptive or embedded software testing and mutation testing. *Softw. Test. Verification Reliab.* 31, 7 (2021). https://doi.org/10.1002/stvr.1798

[103] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. 2013. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Auton. Robots* 34, 3 (2013), 189–206. https://doi.org/10.1007/s10514-012-9321-0

[104] Janna Huuskonen and Timo Oksanen. 2018. Soil sampling with drones and augmented reality in precision agriculture. *Comput. Electron. Agric.* 154 (2018), 25–35. https://doi.org/10.1016/j.compag.2018.08.039

[105] International Civil Aviation Organization (ICAO). 2013. Safety management manual; 3rd ed.; Doc 9859.

[106] Gunel Jahangirova and Paolo Tonella. 2020. An Empirical Evaluation of Mutation Operators for Deep Learning Systems. In *13th IEEE International Conference on Software Testing, Validation and Verification, ICST 2020, Porto, Portugal, October 24-28, 2020*. 74–84.

[107] Christian Janiesch, Patrick Zschech, and Kai Heinrich. 2021. Machine learning and deep learning. *Electron. Mark.* 31, 3 (2021), 685–695. https://doi.org/10.1007/s12525-021-00475-2

[108] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019*. IEEE, 406–409. https://doi.org/10.1109/ICSME.2019.00070

[109] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2021. Predicting issue types on GitHub. *Sci. Comput. Program.* 205 (2021), 102598. https://doi.org/10.1016/j.scico.2020.102598

[110] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. 2015. Issue Dynamics in Github Projects. In *Product-Focused Software Process Improvement - 16th International Conference, PROFES 2015, Bolzano, Italy, December 2-4, 2015, Proceedings*. 295–310.

[111] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. 2016. Using Dynamic and Contextual Features to Predict Issue Lifetime in GitHub Projects. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 291–302.

[112] Sunghun Kim and E. James Whitehead, Jr. 2006. How Long Did It Take to Fix Bugs?. In *Proceedings of the 2006 International Workshop on Mining Software Repositories* (Shanghai, China) *(MSR '06)*. ACM, New York, NY, USA, 173–174.

[113] Nancy Leveson. 2020. Are you sure your software will not kill anyone? *Commun. ACM* 63, 2 (2020), 25–28. https://doi.org/10.1145/3376127

[114] Nancy G. Leveson. 1984. Software Safety in Computer-Controlled Systems. *Computer* 17, 2 (1984), 48–55. https://doi.org/10.1109/MC.1984.1659054

[115] Nancy G. Leveson. 1986. Software Safety: Why, What, and How. *ACM Comput. Surv.* 18, 2 (1986), 125–163. https://doi.org/10.1145/7474.7528

[116] Nancy G. Leveson. 1991. Software Safety: In Embedded Computer Systems. *Commun. ACM* 34, 2 (1991), 34–46. https://doi.org/10.1145/102792.102799

[117] Nancy G. Leveson. 1995. Safety as a System Property. *Commun. ACM* 38, 11 (1995), 146. https://doi.org/10.1145/219717.219816

[118] Nancy G. Leveson and Peter R. Harvey. 1983. Analyzing Software Safety. *IEEE Trans. Software Eng.* 9, 5 (1983), 569–579. https://doi.org/10.1109/TSE.1983.235116

[119] Nancy G. Leveson and Peter R. Harvey. 1983. Software fault tree analysis. *J. Syst. Softw.* 3, 2 (1983), 173–181. https://doi.org/10.1016/0164-1212(83)90030-4

[120] Xiaozhou Liang, John Henry Burns, Joseph Sanchez, Karthik Dantu, Lukasz Ziarek, and Yu David Liu. 2021. Understanding Bounding Functions in Safety-Critical UAV Software. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. 1311–1322. https://doi.org/10.1109/ICSE43902.2021.00119

[121] Mikael Lindvall, Adam Porter, Gudjon Magnusson, and Christoph Schulze. 2017. Metamorphic Model-Based Testing of Autonomous Systems. In *2nd IEEE/ACM International Workshop on Metamorphic Testing, MET@ICSE 2017, Buenos Aires, Argentina, May 22, 2017*. IEEE Computer Society, 35–41. https://doi.org/10.1109/MET.2017.6

[122] Shuhang Liu, Zhiqing Wei, Zijun Guo, Xin Yuan, and Zhiyong Feng. 2018. Performance Analysis of UAVs Assisted Data Collection in Wireless Sensor Network. In *87th IEEE Vehicular Technology Conference, VTC Spring 2018, Porto, Portugal, June 3-6, 2018*. IEEE, 1–5. https://doi.org/10.1109/VTCSpring.2018.8417673

[123] Tao Liu, Zheng Chen, Benyu Zhang, Wei-Ying Ma, and Gongyi Wu. 2004. Improving Text Classification using Local Latent Semantic Indexing. In *Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004), 1-4 November 2004, Brighton, UK*. IEEE Computer Society, 162–169. https://doi.org/10.1109/ICDM.2004.10096

[124] Yu David Liu and Lukasz Ziarek. 2017. Toward Energy-Aware Programming for Unmanned Aerial Vehicles. In *3rd IEEE/ACM International Workshop on Software Engineering for Smart Cyber-Physical Systems, SEsCPS@ICSE 2017, Buenos Aires, Argentina, May 21, 2017*. 30–33. https://doi.org/10.1109/SEsCPS.2017.8

[125] Fernando Martinez-Taboada and Jose Ignacio Redondo. 2020. Induction of decision trees. *PLOS ONE* (2020).

[126] Luiz Eduardo G. Martins and Tony Gorschek. 2017. Requirements Engineering for Safety-Critical Systems: Overview and Challenges. *IEEE Software* 34, 4 (2017), 49–57. https://doi.org/10.1109/MS.2017.94

[127] Graham McDonald, Craig Macdonald, and Iadh Ounis. 2015. Using Part-of-Speech N-grams for Sensitive-Text Classification. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval, ICTIR 2015, Northampton, Massachusetts, USA, September 27-30, 2015*, James Allan, W. Bruce Croft, Arjen P. de Vries, and Chengxiang Zhai (Eds.). ACM, 381–384. https://doi.org/10.1145/2808194.2809496

[128] Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* 12, 2 (01 Jun 1947), 153–157.

[129] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. 2015. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*. IEEE, 6235–6240. https://doi.org/10.1109/ICRA.2015.7140074

[130] Daniel Mellinger and Vijay Kumar. 2011. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*. 2520–2525. https://doi.org/10.1109/ICRA.2011.5980409

[131] Claudio Menghi, Shiva Nejati, Khouloud Gaaloul, and Lionel C. Briand. 2019. Generating automated and online test oracles for Simulink models with continuous and uncertain behaviors. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. 27–38. https://doi.org/10.1145/3338906.3338920

[132] Laura Moreno and Andrian Marcus. 2018. Automatic software summarization: the state of the art. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. 530–531.

[133] Mark W. Müller, Markus Hehn, and Raffaello D'Andrea. 2015. A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation. *IEEE Trans. Robotics* 31, 6 (2015), 1294–1310. https://doi.org/10.1109/TRO.2015.2479878

[134] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. 2015. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics* 31, 5 (2015), 1147–1163. https://doi.org/10.1109/TRO.2015.2463671

[135] A. Jefferson Offutt and Aynur Abdurazik. 1999. Generating Tests from UML Specifications. In *UML (Lecture Notes in Computer Science, Vol. 1723)*. Springer, 416–429.

[136] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan I. Nieto. 2017. Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*. IEEE, 1366–1373. https://doi.org/10.1109/IROS.2017.8202315

[137] Sebastiano Panichella. 2018. Summarization techniques for code, change, testing, and user feedback (Invited paper). In *2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests, VST@SANER 2018, Campobasso, Italy, March 20, 2018*. 1–5.

[138] Sebastiano Panichella, Gerardo Canfora, and Andrea Di Sorbo. 2021. "*Won't We Fix this Issue?*" Qualitative characterization and automated identification of wontfix issues on GitHub. *Inf. Softw. Technol.* 139 (2021), 106665. https://doi.org/10.1016/j.infsof.2021.106665

[139] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2015. How can i improve my app? Classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, Rainer Koschke, Jens Krinke, and Martin P. Robillard (Eds.). IEEE Computer Society, 281–290. https://doi.org/10.1109/ICSM.2015.7332474

[140] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H. Ang. 2017. Perception, Planning, Control, and Coordination for Autonomous Vehicles. *Machines* 5, 1 (2017). https://doi.org/10.3390/machines5010006

[141] Ana Pereira and Carsten Thomas. 2020. Challenges of Machine Learning Applied to Safety-Critical Cyber-Physical Systems. *Mach. Learn. Knowl. Extr.* 2, 4 (2020), 579–602. https://doi.org/10.3390/make2040031

[142] Pixhawk.org. 2021. Pixhawk | The hardware standard for open-source autopilots. https://pixhawk.org/

[143] Tong Qin, Peiliang Li, and Shaojie Shen. 2018. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Trans. Robotics* 34, 4 (2018), 1004–1020. https://doi.org/10.1109/TRO.2018.2853729

[144] Yi Qin, Tao Xie, Chang Xu, Angello Astorga, and Jian Lu. 2020. CoMID: Context-Based Multiinvariant Detection for Monitoring Cyber-Physical Software. *IEEE Trans. Reliab.* 69, 1 (2020), 106–123. https://doi.org/10.1109/TR.2019.2933324

[145] Oscar Quispe, Alexander Ocsa, and Ricardo Coronado. 2017. Latent semantic indexing and convolutional neural network for multi-label and multi-class text classification. In *IEEE Latin American Conference on Computational Intelligence, LA-CCI 2017, Arequipa, Peru, November 8-10, 2017*. IEEE, 1–6. https://doi.org/10.1109/LA-CCI.2017.8285711

[146] Pooja Rani, Sebastiano Panichella, Manuel Leuenberger, Andrea Di Sorbo, and Oscar Nierstrasz. 2021. How to identify class comment types? A multi-language approach for class comment classification. *J. Syst. Softw.* 181 (2021), 111047. https://doi.org/10.1016/j.jss.2021.111047

[147] Sarah Rastkar, Gail C. Murphy, and Gabriel Murray. 2010. Summarizing software artifacts: a case study of bug reports. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*. 505–514.

[148] Charles Richter, Adam Bry, and Nicholas Roy. 2013. Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments. In *Robotics Research - The 16th International Symposium ISRR, 16-19 December 2013, Singapore (Springer Tracts in Advanced Robotics, Vol. 114)*, Masayuki Inaba and Peter Corke (Eds.). Springer, 649–666. https://doi.org/10.1007/978-3-319-28872-7_37

[149] Johanna Roiha, Einari Heinaro, and Markus Holopainen. 2021. The Hidden Cairns - A Case Study of Drone-Based ALS as an Archaeological Site Survey Method. *Remote. Sens.* 13, 10 (2021), 2010. https://doi.org/10.3390/rs13102010

[150] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* 1, 5 (2019), 206–215. https://doi.org/10.1038/s42256-019-0048-x

[151] Andrew Singletary, Shishir Kolathaya, and Aaron D. Ames. 2022. Safety-Critical Kinematic Control of Robotic Systems. *IEEE Control Systems Letters* 6 (2022), 139–144. https://doi.org/10.1109/LCSYS.2021.3050609

[152] Ian Sommerville. 2010. *Software Engineering* (9 ed.). Addison-Wesley, Harlow, England.

[153] Yang Song and Oscar Chaparro. 2020. BEE: a tool for structuring and analyzing bug reports. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*. 1551–1555. https://doi.org/10.1145/3368089.3417928

[154] Paul Soulier, Depeng Li, and John R Williams. 2015. A survey of language-based approaches to cyber-physical and embedded system development. *Tsinghua Science and Technology* 20, 2 (2015), 130–141.

[155] Donna Spencer. 2009. *Card sorting: Designing usable categories*. Rosenfeld Media.

[156] Jan-Philipp Steghöfer, Eric Knauss, Jennifer Horkoff, and Rebekka Wohlrab. 2019. Challenges of Scaled Agile for Safety-Critical Systems. In *Product-Focused Software Process Improvement - 20th International Conference, PROFES 2019, Barcelona, Spain, November 27-29, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11915)*, Xavier Franch, Tomi Männistö, and Silverio Martínez-Fernández (Eds.). Springer, 350–366. https://doi.org/10.1007/978-3-030-35333-9_26

[157] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour prediction for autonomous driving systems. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 359–371. https://doi.org/10.1145/3377811.3380353

[158] Sihao Sun, Giovanni Cioffi, Coen de Visser, and Davide Scaramuzza. 2021. Autonomous Quadrotor Flight Despite Rotor Failure With Onboard Vision Sensors: Frames vs. Events. *IEEE Robotics Autom. Lett.* 6, 2 (2021), 580–587. https://doi.org/10.1109/LRA.2020.3048875

[159] Ahmad S. Tarawneh, Ahmad B. Hassanat, Ghada Awad Altarawneh, and Abdullah Almuhaimeed. 2022. Stop Oversampling for Class Imbalance Learning: A Review. *IEEE Access* 10 (2022), 47643–47660. https://doi.org/10.1109/ACCESS.2022.3169512

[160] Max Taylor, Jayson Boubin, Haicheng Chen, Christopher Stewart, and Feng Qin. 2021. A Study on Software Bugs in Unmanned Aircraft Systems. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS).* 1439–1448. https://doi.org/10.1109/ICUAS51884.2021.9476844

[161] Max Taylor, Haicheng Chen, Feng Qin, and Christopher Stewart. 2021. Avis: In-Situ Model Checking for Unmanned Aerial Vehicles. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2021, Taipei, Taiwan, June 21-24, 2021.* 471–483. https://doi.org/10.1109/DSN48987.2021.00057

[162] Yuan Tian, David Lo, and Chengnian Sun. 2013. DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis. In *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013.* 200–209. https://doi.org/10.1109/ICSM.2013.31

[163] Christopher Steven Timperley, Afsoon Afzal, Deborah S. Katz, Jam Marcos Hernandez, and Claire Le Goues. 2018. Crashing Simulated Planes is Cheap: Can Simulation Detect Robotics Bugs Early?. In *11th IEEE International Conference on Software Testing, Verification and Validation, ICST 2018, Västerås, Sweden, April 9-13, 2018.* IEEE Computer Society, 331–342. https://doi.org/10.1109/ICST.2018.00040

[164] Alexander Trautsch, Steffen Herbold, and Jens Grabowski. 2020. Static source code metrics and static analysis warnings for fine-grained just-in-time defect prediction. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME).* 127–138. https://doi.org/10.1109/ICSME46990.2020.00022

[165] Jamal Uddin, Rozaida Ghazali, Mustafa Mat Deris, Rashid Naseem, and Habib Shah. 2017. A survey on bug prioritization. *Artif. Intell. Rev.* 47, 2 (2017), 145–180. https://doi.org/10.1007/s10462-016-9478-6

[166] Chris Cole Drone Wars UK. 2019. ACCIDENTS WILL HAPPEN - A review of military drone crash data as the UK considers allowing large military drone flights in its airspace. https://dronewars.net/wp-content/uploads/2019/06/DW-Accidents-WEB.pdf

[167] U.S. Dept. Of Defense. 1984. MIL-STD-882B:System Safety Program Requirements.

[168] Michael Vierhauser, Md Nafee Al Islam, Ankit Agrawal, Jane Cleland-Huang, and James Mason. 2021. Hazard analysis for human-on-the-loop interactions in sUAS systems. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021.* 8–19. https://doi.org/10.1145/3468264.3468534

[169] Jeffrey M. Voas and Gary McGraw. 1997. *Software Fault Injection: Inoculating Programs against Errors.* John Wiley & Sons, Inc., USA.

[170] Dinghua Wang, Shuqing Li, Guanping Xiao, Yepang Liu, and Yulei Sui. 2021. An exploratory study of autopilot software bugs in unmanned aerial vehicles. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021.* 20–31. https://doi.org/10.1145/3468264.3468559

[171] Jun Wang, Xiaofang Zhang, Lin Chen, and Xiaoyuan Xie. 2022. Personalizing label prediction for GitHub issues. *Information and Software Technology* 145 (2022), 106845. https://doi.org/10.1016/j.infsof.2022.106845

[172] Qingye Wang. 2020. Why Is My Bug Wontfix?. In *2020 IEEE 2nd International Workshop on Intelligent Bug Fixing (IBF).* IEEE, 45–54.

[173] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008.* 461–470. https://doi.org/10.1145/1368088.1368151

[174] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008.* 461–470.

[175] Zhihua Wei, Duoqian Miao, Jean-Hugues Chauchat, and Caiming Zhong. 2008. Feature Selection on Chinese Text Classification Using Character N-Grams. In *Rough Sets and Knowledge Technology, Third International Conference, RSKT 2008, Chengdu, China, May 17-19, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 5009)*, Guoyin Wang, Tianrui Li, Jerzy W. Grzymala-Busse, Duoqian Miao, Andrzej Skowron, and Yiyu Yao (Eds.). Springer, 500–507. https://doi.org/10.1007/978-3-540-79721-0_68

[176] Anna Wojciechowska, Jérémy Frey, Sarit Sass, Roy Shafir, and Jessica R. Cauchard. 2019. Collocated Human-Drone Interaction: Methodology and Approach Strategy. In *14th ACM/IEEE International Conference on Human-Robot Interaction, HRI 2019, Daegu, South Korea, March 11-14, 2019.* IEEE, 172–181. https://doi.org/10.1109/HRI.2019.8673127

[177] Xin Xia, David Lo, Xinyu Wang, Xiaohu Yang, Shanping Li, and Jianling Sun. 2013. A Comparative Study of Supervised Learning Algorithms for Re-opened Bug Prediction. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering (CSMR '13).* 331–334.

[178] Wei Xiao, Christos G. Cassandras, and Calin Belta. 2021. Safety-Critical Optimal Control for Autonomous Systems. *J. Syst. Sci. Complex.* 34, 5 (2021), 1723–1742. https://doi.org/10.1007/s11424-021-1230-x

[179] Fiorella Zampetti, Ritu Kapur, Massimiliano Di Penta, and Sebastiano Panichella. 2022. An empirical characterization of software bugs in open-source Cyber–Physical Systems. *Journal of Systems and Software* 192 (2022), 111425. https://doi.org/10.1016/j.jss.2022.111425

[180] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. DeepBillboard: systematic physical-world testing of autonomous driving systems. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*. 347–358. https://doi.org/10.1145/3377811.3380422

[181] Yu Zhou, Yanxiang Tong, Ruihang Gu, and Harald Gall. 2014. Combining Text Mining and Data Mining for Bug Report Classification. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME '14)*. IEEE Computer Society, Washington, DC, USA, 311–320.

[182] Yu Zhou, Yanxiang Tong, Ruihang Gu, and Harald C. Gall. 2016. Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process* 28, 3 (2016), 150–176.

APPENDIX - EXAMPLES OF HAZARD AND ACCIDENT CATEGORIES

Table 8. Examples of Hazard Categories

Category	**Examples**
event/action allowed in undesired states/configurations	*"I tried arming when not in a transition state and when in a transition state to ensure it was rejecting the arming at the correct times"* *"manualcontrol: prevent spurious disarms in altitude hold mode"* *"Prevent motor movement while programming"* *"Copter: jump in altitude when EKF becomes available"*
undesired behavior on failsafe or error condition/configuration	*"If a radio, GCS, or battery failsafe event happens during that landing, the vehicle should continue with the landing rather than carrying out the failsafe action"* *"Motors keep running on Failsafe if Motors spin at neutral is enabled"* *"For the RC/Transmitter failsafe we need special handling while in Auto mode. This is not the case for the battery failsafe however"* *"While in AUTO flight, our quadplane suddenly decided to go into SAFETY_DISARMED, which immediately resulted in all outputs to motors being shut off. This brought the plane down"*
undesired failsafe behavior	*"Hangtime stays engaged in failsafe,* i.e., *the hangtime is correct for low throttle while valid signal but failsafe kicks in LPS for around 25 seconds when hangtime is set to 2 seconds"* *"Copter: battery failsafe shouldn't switch to RTL mode if we're already in RTL"* *"Copter-3.4-dev: GPS failsafe triggers too quickly"* *"Rover: shorten radio and GCS failsafe timeouts"*
inadequate checks	*"Add servo voltage check. Needed to prevent arming with extremely bad/questionable servo power supply."* *"We should add an attitude consistency pre-arm check to all vehicles that checks that all active AHRS cores have consistent attitudes and if not, display a pre-arm check."* *"We should add a pre-arm check if the terrain database allocation fails so that the user is aware of the problem before flight"* *"pre-arm check if compass1 is disabled but 2 or 3 are enabled"*
improper parameter setting/initialization/configuration	*"If someone uses a 3 position switch, could end up with a badly configured neutral value"*

Continued on next page

Table 8 – *Continued from previous page*

Category	**Examples**
	"Because the MIN had been set to zero, the WPNAV_SPEED was reduced to zero so while the copter switched into RTL it would not return because of the zero WPNAV speed" *"[Fix] Fence Altitude Limit"* *"Plane: TECS: NAVL1_LIM_BANK does not have an appropriate minimum angle forced"*
misleading or missing data/measurements/reports	*"as reported by several logs on the forums, there is no filtration in the current sensor drivers, and any noise gets passed down as obstacles"* *"we could read from the sensor too soon after requesting data and the result coming back would be garbage"* *"Plane: add protection against bad GPS positions"* *"The air pressure can change quite rapidly result[ing] in an altitude error of > 40m for a one hour mission. The resulting problems are obvious, from inadvertent entry into forbidden airspace to crashes during Auto-Land/RTL"*
missing or misleading communication with pilot	*"It would be great if we had a status_text message indicating that the copter was arming. A gcs could then give an audible and visual announcement that the arm command was accepted"* *"The copter falls to the ground with no sounds, beeps, or indications"* *"Copter: warn when single motors reaches throttle limits as it's a sign the motor is failing"* *"When EKF failsafe was triggered, some users were unaware that it was switched to LAND. ToneAlarm makes the user aware [of] the failsafe"*
undesired hardware behavior	*"Plane: Parachute should turn off the motor only and not disarm the AP. Today we have triggered the parachute, the door has opened but the parachute didn't pull out"* *"We have discovered about 38 cases where users have flown with MPU6k accelerometer failures during flight. Sometimes this leads to a crash"*
action not allowed in specific states/configurations	*"Plane 3.2 does not allow disarming with rudder like copter does. I found the arming switch totally pointless when I have ARMING_DIS_RUD=0 which is the real safety for my motor, and to be more precise, I found Arduplane less safe than it should be by default since it will still crash my plane if it reboots in flight because all servos are frozen waiting for me to push the arming switch"* *"GCS should strongly urge user to use switch arming with hangtime"*
inappropriate mode changes/ handling	*"When using Follow-me mode via the tablet, it continually sets Guided mode waypoints [making] impossible to recover control of the craft if the tablet remains in Follow-me"*

Continued on next page

Table 8 – *Continued from previous page*

Category	**Examples**
	"flight: ensure thrust mode is driven in all paths"
onboard instrumentation issue	*"This must be acknowledge and taken note of it in software (esp. on what action to take 'immediately' when the GPS count diminishes suddenly in a matter of seconds - as this is not normal and not seen by all but when it happens it is ALL GONE)"* *"A flight log with two active GPS modules shows 23 GPS glitches in 5 minutes of flight, while GPS conditions were OK"*
inappropriate safety switch handling	*"There is an edge case where forcing the safety switch state can fail for an extended period of time, and then succeed forcing the arming check that users can not opt out of to prevent it"* *"Some controllers may wish to set the safety switch remotely without the user having to physically get close to the machine and push it"*
undetected failure	*"Copter: landing detector false-positive when slowing down in PosHold mode. The cause was a slight motor imbalance combined with a slow-down from fast-forward flight in PosHold mode"* *"add detection of airspeed blockage and switch to non-airspeed control: cope with water droplet blocking pitot"*
timing/timeout/synchronization issue	*"Possible race conditions (they all happen between the check of existence of a topic and trying to create the node): - single instance, with multiple advertisers during the first advertise: both advertisers see the topic as non-existent and try to advertise it"* *"This seems bad– while this timing is probably considerably helpful in ensuring synchronization with genuine Futaba stuff and rejecting illegitimate data, it may work against us in other cases"*
communication failure	*"Just before the touch down, the plane lost RC link and the RTL mode kicked in. I was no longer in control. The ardupilot software executed an immediate right turn towards the Take off position without any climb"*
lack of control	*"Copter: allow pilot to forcibly retake control from GCS with TX. When something goes wrong, it would probably be best for the pilot to be able to reach for the TX, click a switch and be back in control"*
inappropriate handling for high vibrations	*"Copter: prevent bad Z-vibrations causing rapid climb"*
memory/stack issue	*"attitude: increase stack size to 2504 from 2200. Function will use 2440 at peak. This is a nasty one, because most attempts to dynamically measure stack won't cause home position update"*
battery issue	*"After setup for my fixed wing craft I observe a problem,* i.e., *a glitching of the power rails, when I connect my flight battery without connecting USB"*

Table 9. Accident categories' examples

Category	**Example**
crashing/collision issue	*"I just had a crash due to a burnt motor"*
anomalous flight conduct	*"My son and I have experienced in-flight resets of the BrainFPV FC"*
landing/rtl issue	*"GPS lost in position control (or RTL) lands without pilot having control"*
aircraft lost/flown away	*"Stuck baro can lead to flyaway"*
stabilization/orientation/ positioning issue	*"Stabilization: Failsafe provisions don't work properly after apparent code refactor"*
operator injury	*"Safety-Related Motors spin when connecting battery"*
takeoff issue	*"After arming the vehicle just took off and was extremely difficult to control"*