Understanding Software Failures Through Incident Report Analysis

An Empirical Study of 348 Incident Reports from the VOID

Iulia-Maria Aldea[1]

Supervisor(s): Prof. Dr. Ir. Diomidis Spinellis[1], **Eileen Kapel**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Iulia-Maria Aldea
Final project course: CSE3000 Research Project
Thesis committee: Prof. Dr. Ir. Diomidis Spinellis, Eileen Kapel, Assist. Prof. Dr. Benedikt Ahrens

ABSTRACT

Software changes are a leading cause of operational failures in complex production systems. Despite the increasing use of Artificial Intelligence for Development Operations and the availability of postmortem data, research on software incidents remains fragmented and narrowly scoped. This study aims to provide a generalizable understanding of software and change-induced incidents through structured analysis of 348 real-world incident reports from the Verica Open Incident Database. Using few-shot prompting with the GPT-4.1 Mini model, we extract key incident characteristics (root cause, triggering change, impact, severity, and remediation) and apply clustering to identify recurring incident archetypes. Our method achieves over 80% annotation accuracy on a manually labeled subset. We find that over half of incidents stem from software changes, with deployments and configuration updates disproportionately associated with high severity and manual remediation. Capacity issues and code defects are leading root causes. Clustering uncovers several prominent archetypes, including capacity-driven outages, defect-induced degradations, and hybrid failures involving improper changes. These findings support scalable incident analysis and can inform more context-aware operational strategies.

Keywords – AIOps; Incident Characterization; Change-Induced Failures; Incident Archetypes; Software Reliability; Text Mining.

1 INTRODUCTION

Software systems are integral across diverse domains, which is why efforts are continuously expanded to ensure that systems operate normally: from the introduction of Development Operations (DevOps) and change management tools, to automated code review using large language models (LLMs) [1–3]. Incidents still happen and the need to learn from them has been highlighted [4] especially considering the increasing popularity of Artificial Intelligence for Development Operations (AIOps). As such information of past incidents, or more precisely knowing how and why an incident came to be and what it impacted, is important for establishing preventative measures [5].

However, there is currently no standardized way of reporting the incidents. Reports vary considerably across organizations in structure and detail [6], which presents challenges for large-scale comparative analysis. As such, current research focuses either on small-scale, manually analyzed incident samples such as the qualitative study by Sillito *et al.* [7] or is confined to incidents within a single organization. For example, Kapel *et al.* [8] and Wu *et al.* [9] examine change-related failures within ING and ANT Group, respectively. Meanwhile, work on automated diagnosis tends to emphasize root cause analysis (RCA) over broader incident characterization or pattern identification, and is typically domain-specific, such as RCA for cloud outages [10, 11]. As Remil *et al.* [12] note, incident correlation and contextual analysis remain underexplored.

As a result, while prior studies offer valuable insights into causes and remediation of specific incident types, there is still no generalizable, empirical understanding of how software changes contribute to production failures across diverse systems. This is a significant gap, particularly given findings such as those by Zhao *et al.* [13], who report that 70% and 54% of incidents at Google and Baidu, respectively, stem from software changes. Furthermore, AIOps systems rely on structured, high-quality incident data to function effectively [12, 14], reinforcing the need for scalable, automated approaches to incident analysis.

To address this gap in current understanding, we present an exploratory study based on **348** incident reports sourced from the VOID, a publicly accessible community contributed database of operational failure reports. The central research question guiding this study is: ***"What are the characteristics of incident reports caused by software changes?"***. To investigate this, we examine the following research questions:

- **RQ1**: What general characteristics of incidents are evident across the collected incident reports and how can they be automatically extracted?
- **RQ2**: What is the relationship between an incident's cause, impact and remediation that follows from the established characterization?
- **RQ3**: What types of software changes associated with incident occurrence are observed in the dataset?
- **RQ4**: What recurring patterns can be identified, and what are the three most prominent clusters based on incident similarity?

In answering these questions, this study makes three key contributions: (1) we propose a pipeline for automated analysis of incident reports using small-scale LLMs, demonstrating high accuracy in extracting structured incident characteristics; (2) we uncover actionable insights into the relationships between causes, impacts, and remediation strategies - showing, for instance, the heightened severity and recovery complexity of change-induced incidents; and (3) we identify a set of recurring incident archetypes using unsupervised clustering, capturing common patterns such as capacity-driven outages, code defect–related degradations, and complex hybrid failures. Both the data used and the implemented analysis pipeline have been made available in the reproducibility package [15]. These contributions provide both methodological and practical value by enabling large-scale incident analysis and supporting the development of more targeted risk assessment processes.

The remainder of the paper is organized as follows: we begin with an overview of incident management practices and introduce VOID as a valuable collection of incident reports (2), followed by a review of related work within the field (3). We then describe the dataset and introduce the analytical pipelines used to address the research questions (4), before detailing the implementation specifics and evaluation setup (5). The results section reports findings on annotation performance, incident distributions, and clustering outcomes (6). These findings are discussed in the broader spectrum of incident management (7). We conclude by outlining limitations and future directions (8), addressing ethical considerations (9), and summarizing the key contributions of the study (10).

2 BACKGROUND

The study of software system incidents spans cause and response, both immediate as well as long term. To provide necessary context, we begin by outlining general incident management practices

(2.1), including how incidents are detected, escalated, and resolved. We then introduce the Verica Open Incident Database (VOID), a community-curated collection of public incident reports that offers insight into how organizations document and reflect on operational failures (2.2).

2.1 Incident Management

Incidents typically refer to unplanned events that disrupt the normal operation of a system. As such, to be classified as an incident, an event has to be both unplanned as well as require a solution with a high degree of urgency [16].

When an incident occurs, anomalies are detected by engineers or automated monitors, prompting the creation of a ticket and assignment to a resolution team [17]. Once the system is fully restored, a post-incident analysis is conducted, resulting in a report detailing the impact, root cause, resolution, and any follow-up actions taken [18]. The degree of detail present in the report depends on the organization's policies and incident disclosure agreements.

2.2 The VOID Incident Report Collection

The VOID is an open-source, community-contributed collection that aggregates publicly available software incident reports from various organizations. Each entry in the database typically includes the incident narrative (written, depending on the organization, in the format of blog posts, or media articles) along with structured metadata such as incident date, duration, severity (if available), and affected services.

VOID was developed to facilitate cross-organizational learning and to support research into incident patterns, failure modes, and post-incident practices in modern software systems. Its relevance to site reliability engineering (SRE) practices has been highlighted in the annual VOID reports, which summarize high-level trends across collected incidents [19].

3 RELATED WORK

To contextualize our contributions, we first examine how variations in incident documentation affect the comparability of incident data across organizations (3.1). We then survey existing taxonomies that define how incidents and their characteristics are classified (3.2). Finally, we review empirical studies that investigate correlations between incident characteristics such as cause, duration, and mitigation strategy (3.3).

3.1 Incident Reporting

The analysis of incident characteristics is strongly influenced by the way incidents are reported. Several studies have highlighted that incident documentation practices vary considerably between organizations, resulting in inconsistent terminology, report structure, and depth of detail. Menges *et al.* [6] conducted a comparative study of postmortems from large technology companies and concluded that the absence of standardized reporting formats hinders the transfer of knowledge and limits the generalizability of findings.

Further, organizational reporting cultures have been shown to affect what is recorded in incident reports. Wu *et al.* [9] observed that incident documentation practices at Ant Group were influenced by internal policies and domain-specific expectations. Kapel *et al.* [8] reported challenges in tracing incidents back to their triggering software changes, often due to vague narrative descriptions and lack of formal traceability.

3.2 Incident Taxonomies

To support structured reasoning about incidents, various taxonomies have been developed to categorize causes and contributing factors. Li *et al.* [20] proposed a taxonomy of software failure modes that distinguishes between product faults (e.g. logic errors, interface mismatches) and process faults (e.g. design oversights, miscommunication). This classification aims to support risk assessment and fault isolation during incident analysis.

Agrawal *et al.* [21] conducted a systematic literature review of software design errors and proposed a taxonomy based on recurring patterns found across empirical studies. Their taxonomy includes categories such as violation of design principles, incomplete requirements, incorrect data handling, and flawed control structures. The study emphasizes that design-level faults often propagate downstream and may not be immediately detectable, making their classification essential for preventative mitigation strategies.

In addition to academic proposals, industry-driven taxonomies have emerged. The Azure Reliability Tagging System (ARTS) [22] is an internal classification framework used by Microsoft to tag incident records with standardized attributes such as impact scope, triggering conditions, and detection mechanism. ARTS exemplifies how taxonomy-based structuring can enable large-scale trend analysis within a single organization.

3.3 Incident Characteristic Correlation

A number of studies have examined how incident characteristics relate to one another, particularly with respect to cause, duration, mitigation strategies, and severity. Both Davidovic [23] and the VOID report [19] found that incident duration does not reliably correlate with the severity of an incident. They conclude that duration itself does not convey anything meaningful about the impact and resolution of an incident.

Research into RCA has explored both manual and automated approaches to identifying causal factors. Saha *et al.* [5] proposed a method combining time-series anomaly detection with causal inference to attribute failures more accurately. Roy *et al.* [10] demonstrated that the integration of alert metadata with system telemetry improves diagnostic precision. However, several studies have noted that most incidents involve multiple contributing factors rather than a single root cause [19], suggesting that mitigation planning should consider the effects of fault interaction rather than relying on the singular main fault assumption of RCA.

Efforts to link incident causes with remediation strategies have also been explored. Sillito *et al.* [7] analyzed postmortems from large-scale systems and categorized responses into direct fixes, preventative improvements, and organizational changes. While direct mitigations were commonly reported, they find that broader structural or preventative responses were less frequently documented.

Incident impact and severity have been assessed using both qualitative and quantitative frameworks. Cichonski *et al.* [18] provides a structured model incorporating service degradation, data loss, and reputational damage. While widely used, such frameworks are

typically applied manually and are not integrated into automated incident management systems.

4 METHODOLOGY

This study adopts a *repository mining* to quantitatively analyze incident reports. In this section outlining the employed framework, we begin by describing the publicly documented incidents collected (4.1). Following this, we detail the investigation pipelines designed to address each research question (4.2).

4.1 Data Familiarization

Incident reports were sourced from the VOID. To ensure the relevance of our findings as well as reproducibility, we include in our analysis only incident reports that:

- are dated between 2022 and 2024, and
- describe the incident in detail

This time frame restricts the dataset to incidents occurring in a post-pandemic operational context, reducing variability introduced by large-scale organizational shifts (e.g. remote work) [24].

Collecting all incidents within this time frame amounts to a total of **1301** reported incidents out of which only **1268** reports had a textual description available. A distribution of the reports considered for analysis over the years can be seen in Table 1.

However, during this time period, the mean length of an incident report was **1094** characters, and generally had the following structure: "Service X is experiencing issues when performing action Y. Incident has been resolved. The service is being monitored for further issues." While this structure encapsulates all information of interest for the user of the service, it contains little information of use to an engineer working on solving the problem. Only the severity of the incident can potentially be inferred from the impact it has on the customer; however, due to missing information, there is little possibility to argue about the correctness of the assigned severity level. As such, we exclude reports under 750 characters. This amounts to a total of **348** included incidents as shown in Table 1. Following this exclusion, the mean length of a report increases to **3117** characters.

Table 1: Summary of reported incidents and included reports per year (2022–2024)

Year	Reported Incidents	Unavailable Description	Included Reports
2022	1232	32	313
2023	23	0	22
2024	13	1	13
Total	**1268**	**33**	**348**

4.2 Investigation Pipeline

Each research question is addressed through a dedicated pipeline. As such, in this section we reason about text-mining as a characteristics extraction method (4.2.1), then we introduce the Chi-Square independence test, as a method of investigating statistically predictive relationships within the data (4.2.2). Following this, we argue about the use of frequency and correlation analysis methods when determining types of software changes (4.2.3). Lastly, we introduce embedding models and clustering as a means of pattern identification (4.2.4).

4.2.1 RQ1: General Characteristics of Incidents. To identify general characteristics present within our data, we first establish a structured taxonomy of incident attributes. This taxonomy is developed through an iterative process combining insights from prior academic taxonomies on software failures [7, 9, 21] with exploratory analysis of our dataset. Specifically, we review established classifications of faults, errors, and recovery strategies, and adapt these to the specific vocabulary and structure encountered in public incident reports. Our goal is to define a set of attributes that are both expressive and consistently extractable from the available incident reports.

Using this taxonomy, we perform text mining on the entire dataset to extract incident attributes. Given the lack of strict universal reporting guidelines, rule-based and classical information extraction approaches (e.g. Named Entity Recognition, pattern matching) are insufficiently expressive or robust [25]. Instead, we adopt a Question Answering (QA) approach using LLMs.

To balance performance and scalability, we rely on the GPT-4 series of Mini models. These models are significantly more cost-effective than their full-scale counterparts and demonstrate competitive performance across a wide range of Natural Language Processing (NLP) tasks. On the Massive Multitask Language Understanding benchmark, designed to test general knowledge and reasoning, for instance GPT-4.1 Mini achieves a score of 80.5%, outperforming earlier models such as GPT-3.5 Turbo [26]. This highlights its ability to generalize across domains while maintaining efficiency, making it well-suited for the structured extraction of information from unstructured incident reports at scale.

4.2.2 RQ2: Cause-Impact-Remediation Relationship. To examine how incident causes relate to impact and remediation, we perform conditional analysis across structured labels obtained in RQ1. We make use of contingency tables and perform Chi-Square tests of independence to identify statistically significant associations between categorical variables such as root cause category, impact, and mitigation strategy. These tests assess whether, for example, incidents attributed to code regressions are more likely to trigger process-level changes than configuration-induced failures.

This approach allows us to test hypotheses about dependencies between incident properties without assuming linear or continuous relationships. We explode arrays of categorical values into independent binary outcomes, running multiple Chi-Square tests to reason about significance. All significance testing is performed using `scipy.stats` and `statsmodels`, with Bonferroni correction applied to adjust for multiple comparisons.

4.2.3 RQ3: Types of Software Changes. With software changes being one of the most prevalent incident causes [13], we focus on the subset of incidents labeled as change-induced and not explicitly categorized as *maintenance* events. We analyze this subset of incidents by estimating the frequency of recurring change categories and identifying statistically meaningful differences between their characteristics and those of non-change-induced incidents.

Text classification is again performed using few-shot prompting with GPT-4.1 Mini to extract the specific change-type, supported by manual validation to ensure consistency. Once labeled, we apply univariate frequency analysis and compute pairwise correlations between change type, severity, and remediation action taken. Chi-Square tests are used to assess whether incidents induced by change differ significantly in their characteristics from those that are not.

4.2.4 RQ4: Pattern Discovery. To uncover recurring archetypes in incident characterization, we apply unsupervised clustering to the incident descriptions. First, we generate dense vector embeddings for each report using the *all-MiniLM-L6-v2* variant of Sentence-BERT [27], which provides a compact and efficient representation of sentence-level semantics. This model was chosen for its balance between computational efficiency and clustering performance, as demonstrated in the Massive Text Embedding Benchmark [28].

Using the resulting embeddings, we apply KMeans and HDBSCAN to cluster incident reports based on semantic similarity. Clusters are then manually interpreted by examining the top-ranked incidents and common tokens per cluster. This approach reveals latent structure in the dataset and helps identify common failure patterns, especially those not explicitly labeled in prior steps. Visualizations are produced using PCA for dimensionality reduction, enabling inspection of cluster cohesion and separation.

5 STUDY DESIGN

In this section we detail how we instantiated and evaluated the characterization agent (5.1), how Chi-square tests were applied to explore statistically significant relationships (5.2). Lastly we describe the use of unsupervised clustering to group semantically similar incidents (5.3).

5.1 Characterization Agent

To extract structured insights from free-text incident reports, we implemented a QA-style extraction agent using the GPT-4o Mini and GPT-4.1 Mini models. These models were selected for their strong performance on complex NLP tasks, offering a balance between inference quality and scalability [26]. The agent converts narrative descriptions into structured representations aligned with our characterization schema.

Each model was evaluated on a set of **34** manually annotated incidents. The outputs of each model were compared against the established ground truth and selected the model with superior overall performance was selected for all subsequent experiments.

To mitigate hallucination risks, we introduced an "Other" fallback category for each attribute, allowing the model to opt out when the evidence was ambiguous or missing. All used prompts are available in the `agent` module of the replication package [15].

We quantitatively assess model performance by employing standard metrics that compare the predicted characteristics against the manually established ground truth. For evaluating the overlap between model outputs and ground truth in a multi-label, multi-output characterization task, we use the **Jaccard Similarity** [29], defined as: $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$ where A and B represent the sets of predicted and true labels, respectively. This metric captures the ratio of correctly predicted labels to the total number of unique labels in either set.

To account for both observed agreement and agreement occurring by chance in categorical prediction tasks, especially important in the presence of imbalanced and sparse label distributions, we use **Cohen's Kappa** [30], defined as: $\kappa = \frac{p_o - p_e}{1 - p_e}$ where p_o is the observed agreement between model predictions and ground truth, and p_e is the expected agreement by chance. A value of $\kappa = 1$ indicates perfect agreement, while $\kappa = 0$ corresponds to chance-level agreement.

Additionally, to assess the representativeness and completeness of our randomly sampled test set, we apply the Chao1 species richness estimator, adapted from ecological studies to measure category diversity in labeled data [31, 32]. While the full taxonomy of labels is known, Chao1 provides a statistical perspective on whether the observed sample likely captures the true diversity, particularly under limited sample conditions: $\hat{S}_{\text{Chao1}} = S_{\text{obs}} + \frac{f_1^2}{2f_2}$ where S_{obs} is the number of observed categories, f_1 is the number of singletons (categories observed exactly once), and f_2 is the number of doubletons (categories observed exactly twice).

5.2 Incident Characteristics Correlations

Understanding how different incident characteristics relate to one another is crucial for uncovering underlying patterns in the incident data.

For each pair of categorical fields, we create a contingency table of label co-occurrences in the test set and apply the Chi-Square test of independence to assess statistical association. This test suits our categorical data and sample size, comparing observed (O_{ij}) and expected (E_{ij}) frequencies under the null hypothesis of independence: $\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$

Because multiple field-pair comparisons are conducted in parallel, we apply the Bonferroni correction to adjust p-values and control the family-wise error rate. This conservative correction reduces the likelihood of false positives that can arise from conducting multiple hypothesis tests.

In cases where fields are multi-labeled, we binarize the label assignments into a one-hot encoded matrix and compute the contingency tables accordingly. This allows us to evaluate associations between fields even when multiple labels are present per instance.

5.3 Clustering Model

To discover latent patterns in the dataset beyond predefined categories, we performed clustering on vector embeddings of the extracted characterizations.

Each report's characterization, formulated as a short description outlining the characteristics of the incident was encoded using the `all-MiniLM-L6-v2` SBERT model [27], chosen for its efficiency and robustness on clustering benchmarks [28]. We tested multiple clustering algorithms, focusing on HDBSCAN for its ability to identify clusters of arbitrary shape and density. To identify the optimal clustering configuration, we conducted a grid search over key parameters (e.g. distance metric, min cluster size, min samples) as outlined in the `models` module in the replication package [15].

To validate clustering quality, we manually assigned validation incidents to clusters and compared these against model outputs, guiding selection of the optimal configuration. Due to the unsupervised nature of clustering, we have also employed three widely used internal validation metrics: the Silhouette Score (S), the Calinski-Harabasz Index (CH), and the Davies-Bouldin Index (DB) [33–35]. We report on the ratio of noise points (NR) as well. These metrics assess clustering quality without reference to ground truth labels, offering complementary perspectives on intra-cluster cohesion and inter-cluster separation. These metrics provide a quantitative basis for selecting clustering configurations that yield well-separated and coherent groups in the embedded space.

Cluster characteristics were subsequently analyzed to identify prevalent incident patterns within each group. Clustering results were then visualized in two and three dimensions using PCA. These visualizations helped validate cluster cohesion and separability.

6 RESULTS

This section presents the main empirical findings of our study. We begin by introducing the taxonomy of incident characteristics derived from manual annotation (6.1). We then evaluate the performance of our LLM-based characterization agent (6.2) before using it to explore the distribution (6.3) and relationships of key incident attributes (6.4). Finally, we examine the types of software changes associated with incidents (6.5), and identify recurring patterns through clustering analysis (6.6).

6.1 Incident Characteristics Taxonomy

Based on the analysis of incident reports related literature on incident management, we developed a classification system that reflects the most consistently identifiable characteristics across the dataset. This classification system (Table 2) reflects both the descriptive and prioritization-relevant aspects of incidents.

Table 2: Incident Characteristics Taxonomy

Category	Values
Severity Levels	Critical, Major, Minor, Trivial, Maintenance
Impact Tags	Full Outage, Partial Outage, Non-Production Outage, Degraded Performance, Connection Issues, Increased Latency, Data Leakage, Data Corruption, Data Loss, Incomplete Recovery, Manual Recovery Required
Triggering Events (Causes)	Code Defect, Configuration Error, Improper Change, Change Constraint Violation, Capacity Issue
Software Change Type	Dependency Update, Code Deployment, Infrastructure Change, Configuration Change, Security Update, Database Change
Mitigation Strategy	Transference, Avoidance, Reduction, Acceptance
Remediation Actions	Rollback, Hot Fix, Infrastructure Change, Traffic Switch, Fallback, No Operation

We distinguish between two high-level categories: characteristics used for prioritization and general descriptors that describe the nature and resolution of the incident.

Incident Prioritization Characteristics. These attributes influence how incidents are triaged and addressed by engineering teams. Severity levels indicate the urgency of response, while impact tags offer a nuanced description of the incident's operational and user-facing effects. Impact tags are not mutually exclusive and may co-occur in a single incident.

General Incident Descriptors. This set includes characteristics that describe the incident's origin and resolution. The *triggering event* identifies the underlying cause, classified into categories such as configuration errors or code defects. Looking further into the cause, allows us to establish whether an incident was caused by a software change. If this is the case, the incident is flagged and annotated with a *software change type* such as configuration changes, code deployments, or third-party updates. The *mitigation strategy* reflects risk management intent, structured around the TARA framework (Transference, Avoidance, Reduction, Acceptance) [36]. The *remediation action* specifies concrete recovery steps, such as applying a hot fix or executing a rollback.

Although duration is commonly reported, we excluded it from this taxonomy. Prior research [19, 23] indicates that duration has limited explanatory value for engineering teams and shows little correlation with severity.

While the definition of incident does not included maintenance events (due to their nature as planned disruptions to regular service), such events are present within our dataset. To limit model hallucination and mislabeling, we include *maintenance* as both a *cause* and a *severity* tag. Incidents classified as such will be analyzed separately.

6.2 Performance of the Incident Characterization Agent

We have investigated the performance of the chosen models (GPT 4o Mini, GPT 4.1 Mini) by comparing their output to manually established ground truth for **34** randomly sampled reports. The testing set, though small and with some missing known labels, shows no statistically significant evidence of many unseen categories based on `Chao1` index. This indicates the sample is reasonably representative for model comparison, more specifically to determine the feasibility of individual model use for characteristic extraction. Figure 1 presents Jaccard similarity metrics across remediation and impact categories, evaluated using micro, macro, and sample-based averaging. Across both categories, the GPT-4.1 Mini few-shot model consistently achieved the highest scores, while the GPT-4o Mini zero-shot and few-shot models demonstrated comparatively lower performance.

The Cohen's Kappa metric for categorical characteristics (Figure 2) shows an increased performance of the GPT 4.1 Mini Model (average $\kappa = 0.82$) under few-shot prompting, outscoring the 4o Mini variant and zero-shot configurations.

Observation 1. *GPT-4.1 Mini model under few-shot prompting reaches an overall accuracy of above 80% when predicting incident characteristics in accordance to the defined taxonomy.*

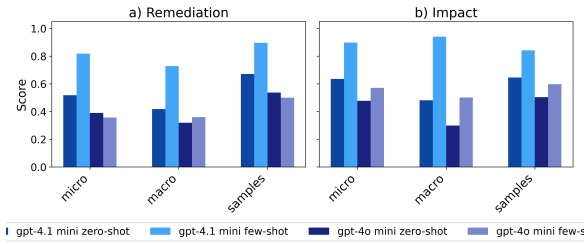


Figure 1: Jaccard Similarity Comparison between Models

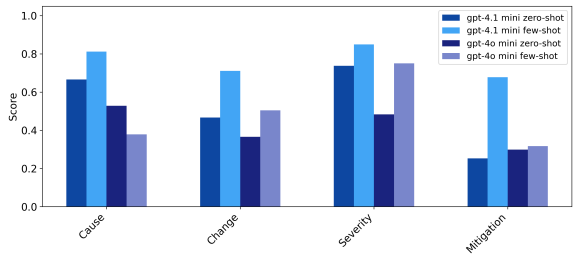


Figure 2: Cohen's Kappa for Cause, Severity, Mitigation

6.3 Characteristics Exploration

We analyzed the distribution of extracted incident characteristics across the dataset, focusing on five key dimensions: cause, impact, mitigation, remediation, and severity. Table 3 presents the frequency distributions for each category based on the agent's responses.

From the analysis, we exclude maintenance events (incidents being attributed the tag "maintenance" for either their cause or severity), amounting to a total of **301** reports considered for frequency analysis. Table 3 presents the frequency distributions for each category based on the agent-generated responses.

Table 3: Top 3 Values per Category (Count and Percentage)

Category	Value	Count	Percent
Cause	Code Defect	85	28.24%
	Capacity Issue	79	26.25%
	Other	78	25.91%
Severity	Major	233	77.41%
	Minor	35	11.63%
	Critical	33	10.96%
Mitigation	Reduction	274	91.03%
	Acceptance	26	8.64%
	Transference	1	0.33%
Impact	Partial Production Outage	228	75.75%
	Degraded Service/Performance	225	74.75%
	Connection Issues	151	50.17%
Remediation	Hot Fix	118	39.20%
	Infrastructure Change	75	24.92%
	No Operation	66	21.93%

We find that *Code Defects* and *Capacity Issues* are the most frequently appearing causes, cumulatively accounting for more than half of the incidents. 77% of incidents are attributed the severity level "major" and lead to *Partial Production Outages* and/or *Degraded Service/Performance*. Further, almost all incidents (91%) apply *Reduction* as a mitigation tactic, typical remediation actions including *Hotfixes* and *Infrastructure Changes*.

Observation 2. *Code Defects and Capacity Issues account for approximately 50% of all reported incidents. Notably, 75% of incidents result in Partial Production Outages and/or Degraded Service.*

6.4 Predictive Relationships

The Chi-squared analysis demonstrated significant (corrected $p < 0.05$) associations between incident causes and several impact types and remediation strategies. Notably, *Capacity Issues* were disproportionately associated with *Connection Issues* ($\chi^2 = 30.74$), Increased Latency ($\chi^2 = 49.36$), and incidents requiring *Manual Recovery* ($\chi^2 = 23.47$), with standardized residuals of 2.61, 5.01, and 2.27 respectively. *Improper Change Operations* were significantly linked to *Full Production Outages* ($\chi^2 = 22.86$) and *Rollback* remediation ($\chi^2 = 98.29$), with residuals of 4.23 and 7.34, highlighting change management as a pivotal area for risk mitigation. Additionally, *Code Defects* were positively associated with *Hot Fix* remediation actions ($\chi^2 = 90.78$, residual=6.37).

Observation 3. *Capacity Issues and Improper Change Operations are the most critical contributors to high-impact incidents, particularly those involving Latency, Connection Failures, and Full Production Outages.*

6.5 Types of Software Changes

Based on the established taxonomy, we report on the frequency of change-induced incidents. We find that a total of **162** (53.82%) incidents are caused by change operations, the most common incident inducing-change operation being *Code Deployments* (50.00%). Other notable incident-inducing changes include *Configuration Changes* (19.75%) and *3rd Party Updates* (11.72%).

In addition to their frequency, change-induced incidents showed a notable severity distribution. Among these incidents, *critical* severity was assigned in **13.58%** of cases, compared to only **7.91%** for non-change-induced incidents. In terms of impact, change-induced incidents exhibited higher proportions of *Manual Recovery Required* (25.93% vs. 10.79%). Conducted Chi-square tests confirm the statistical relevance of these differences ($p < 0.05$).

Observation 4. *Over half (53.82%) of incidents are caused by software changes, predominantly Code Deployments. Change-induced incidents exhibit a higher proportion of critical severity and are more likely to require manual recovery.*

6.6 Pattern Identification

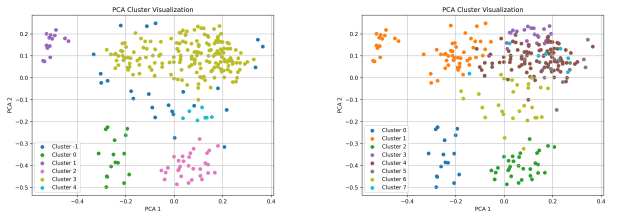
To assess clustering quality across methods, we compared selected configurations of DBSCAN, Gaussian Mixture Models (GMM), and HDBSCAN using standard evaluation metrics (S, CH, DB, NR). Table 4 summarizes the results.

While DBSCAN with $\varepsilon = 0.2$ achieves the best numerical performance, its utility is limited due to an extremely high noise ratio, leaving fewer than 3% of incidents assigned to clusters. Gaussian Mixture Models (GMM), particularly with eight components, provide a more interpretable clustering with full assignment and clean visual separation, despite lower quantitative scores.

Table 4: Clustering performance metrics for selected models.

Model	S	CH	DB	NR
DBSCAN ($\varepsilon = 0.2$)	0.86	134.46	0.16	0.97
GMM (8 components)	0.06	10.73	2.89	0.00
HDBSCAN (min size = 4, dim = 2)	0.02	10.84	2.32	0.08
HDBSCAN (min size = 9, dim = 2)	0.03	12.02	2.22	0.10

The HDBSCAN model with `min_cluster_size=4` and two dimensional input (via PCA dimensionality reduction) demonstrates strong visual coherence and effective noise handling, though quantitative metrics were not computable due to the high-dimensionality and nature of soft cluster boundaries.



(a) HDBSCAN (minsize=4, PCA) **(b) GMM (8 components, PCA)**

Figure 3: Clustering visualization.

Based on both quantitative evaluation and qualitative visual inspection via PCA projection 3, we selected the HDBSCAN (minsize=4, dim=2) and GMM (8 components) configurations as the most appropriate clustering solutions for downstream analysis.

Following the distribution of characteristics within individual GMM clusters, we formulate archetypes, as can be seen in Table 5. To do this, we rely on the co-occurrence of individual characteristics. We find that the top three most prevalent clusters are: Cluster 1, with 55 capacity-related incidents characterized by major outages and performance degradation, Cluster 2, dominated by 39 code defect–related incidents causing major partial outages, and Cluster 5, comprising 14 improper change operation incidents and 13 code defects linked to critical outages requiring hot fixes and rollbacks.

Observation 5. *Clustering identified seven distinct incident archetypes. The top three most common involve capacity-driven major outages, code defect–related major partial outages, and critical outages driven by improper change operations and code defects.*

7 DISCUSSION

In software incident characteristic extraction, LLMs show reliable performance demonstrating their applicability for incident investigation. This contributes to the efforts in understanding failure mechanisms, essential for the improvement of operational reliance. In this section, we synthesize our key findings to provide actionable insights grounded in incident characteristics (7.1), the significant role of change-induced failures (7.2), and the identification of recurring incident archetypes through clustering analysis (7.3).

Table 5: Cluster Archetype Identification

Id	Representative Archetype Description
0	Small-scale, varied issues (improper change, config error, code defect), minor-to-major severity, each with distinct impacts and one-off remediations.
1	Capacity-driven major incidents characterized by degraded performance, increased latency, and partial or full outages. These are primarily resolved through infrastructure changes, traffic rerouting, or hotfixes, with a clear focus on scaling and system limits.
2	Code defects dominate, causing major partial outages and degraded performance; hotfix is the primary remediation.
4	Scheduled maintenance and capacity operations cause minor connection issues, addressed by infrastructure change; low-severity archetype.
5	Critical outages driven by improper change operations and code defects; hotfix and rollback are frequent remediations.
6	Major issues of unknown or other causes result in connection and service degradation, often left unresolved or handled with minimal action.
7	Code/configuration issues cause major partial outages and degraded services, commonly mitigated by rollback or hotfix.

7.1 General Incident Characteristics

The results from our characteristic frequency analysis yield several actionable insights for incident management and align with broader industry findings on incident patterns and resolution strategies.

First, the predominance of code defects (28%) and capacity issues (26%) as root causes highlights the importance of robust testing and scalable infrastructure planning in production environments. This supports earlier findings by Wu *et al.* [9], who identify configuration and code changes as prevalent causes of incidents. Importantly, our predictive analysis reveals that these causes are not only common but also statistically associated with specific high-impact consequences and remediation paths. Capacity issues were significantly associated with connection failures, increased latency, and manual recovery interventions. These relationships suggest that investments in load testing and proactive scaling are not just preventive but potentially high-leverage in reducing downstream impact.

Similarly, improper change operations, though less frequent, were linked to full production outages and rollback remediation. This further underlines the importance of controlled deployment practices and automated rollback mechanisms, to mitigate the functional impact of such system malfunctions.

In terms of impact, Partial Production Outages (76%) and Degraded Performance (75%) dominate, revealing that most incidents do not result in full system failure but still degrade customer experience and reliability. This observation aligns with the notion of gray failures noted in recent site reliability literature [37], showing that recognizing and responding to such incidents quickly is important, especially as they are often precursors to more severe disruptions.

From a mitigation and remediation perspective, the overwhelming preference for reduction strategies (91%) reflects an operational model oriented toward damage containment. This is further supported by hot fixes (39%) and infrastructure changes (25%) as the most common remediation tactics. However, our findings suggest these strategies are tightly coupled to the nature of the incident: capacity issues tend to prompt infrastructure modifications, while

code defects are more often patched reactively. These patterns highlight the potential of cause-aware response playbooks, where detection of a specific root cause could automatically prioritize a remediation action.

7.2 Change-Induced Incidents

Our findings reinforce the central role of software changes as a leading cause of production incidents. More than half (56.48%) of all incidents in our dataset were triggered by change operations, with *Code Deployments* alone accounting for nearly half of these. This is consistent with the observations of Wu *et al.* [9] and Kapel *et al.* [8], both of whom identify code and configuration changes as dominant vectors for operational failure in modern software systems.

The overrepresentation of change-induced incidents in the *critical* severity tier (13.58% vs. 7.91% for other causes) further suggests that changes are not only frequent sources of failure but also disproportionately associated with high-stakes outages. The statistically significant link between change-induced incidents and the need for *manual recovery* aligns with prior evidence from Kapel *et al.* [8], who emphasize that changes often introduce failure modes that automation cannot anticipate or resolve. This finding implies that environments with continuous or high-frequency deployment models may face higher risks unless rapid rollback and recovery strategies are in-place.

Similarly, the non-trivial proportion of incidents caused by *Configuration Changes* and *3rd Party Updates* suggests that managing operational state and external dependencies remains a weak point, suggesting that testing pipelines focused primarily on code correctness are not enough to reliably decrease the risk of incidents.

7.3 Incident Archetypes

Our clustering analysis surfaced seven distinct incident archetypes, offering a compact representation of recurring failure patterns in production systems. The derived archetypes agree with incident taxonomies described by Kapel *et al.* [8] and Wu *et al.* [9], who both emphasize the importance of latent structure to inform prevention and mitigation strategies. For example, our identification of a large, cohesive cluster of capacity-driven outages (Cluster 1), as well as defect-induced service degradations (Cluster 2), reinforces the idea that a relatively small number of root-cause constellations that underlie the majority of high-severity incidents.

Importantly, our method also revealed hybrid failure types, such as Cluster 5, where both Improper Change Operations and Code Defects, resulting in critical outages requiring hotfixes and rollbacks were represented. Such clusters highlight the fact that incident data is not clearly linearly separable and demonstrate the value of unsupervised approaches in detecting complex failures patterns.

8 THREATS TO VALIDITY

Despite the promising findings, several limitations should be acknowledged. In this section, we discuss potential threats to the internal, construct, external, and conclusion validity of our study.

Internal Validity. The manual labeling process used to generate ground truth data was carried out by a single student annotator. This introduces the possibility of subjective interpretation and inconsistency in category assignment. Although a two-step validation process was implemented to improve consistency, the absence of multiple expert annotators and inter-annotator agreement metrics may affect the reliability of the labels.

Construct Validity. The study focuses solely on textual incident reports, omitting multimodal sources such as logs or metrics, which could limit the scope of incident characterization and reduce construct coverage.

External Validity. The dataset used in this study was sourced from the VOID, which is inherently subject to reporting bias due to varying organizational practices, and cultural factors influencing whether incidents are documented and shared. Consequently, the findings may not fully reflect the broader landscape of operational incidents. Furthermore, the comparative evaluation was restricted to a single model family. A broader benchmarking effort involving a wider range of models would provide a more comprehensive and reproducible understanding of model performance in this domain.

Conclusion Validity. The relatively small number of manually annotated incidents limits the robustness of performance comparisons. A larger annotated corpus of incident data would support stronger empirical claims regarding model effectiveness.

9 RESPONSIBLE RESEARCH

This study adheres to the core principles of transparency, reproducibility, and ethical research practices as outlined in the TU Delft Integrity Code[1] and the Netherlands Code of Conduct for Research Integrity[2]. In this section, we outline how we complied with the principles of FAIR (Findable, Accessible, Interoperable, and Reusable) Data (9.1) and acknowledge possible biases present in the data (9.2). Following this, we introduce the broader implications of our work in terms of environmental impact (9.3) and explain how we ensured the replicability and reproducibility of the research (9.4). Finally, we acknowledge the use of AI during the duration of the project and explain how we complied with ethical AI usage in research guidelines (9.5).

9.1 FAIR Data

All incident reports analyzed in this work were sourced from the VOID database, a publicly accessible repository containing post-mortem analyses that have been shared openly. No proprietary or sensitive information is made available through VOID, and no such information was collected and/or utilized in this study.

In order to collect the plain-text representations of the analyses, we employed (and implemented) a *website crawler*. To ensure compliance with ethical web-scraping practices and to avoid placing undue strain on the websites hosting the reports, we followed the following guidelines:

- Wait five seconds between requests to minimize server load.
- Avoid making requests for files already available locally.
- Limit the number of retries in case of failed requests.

It is important to note that before initiating any scraping activities, we consulted the website's `robots.txt` file [38] to verify that such actions were permitted. During the course of our investigation, no restrictions were imposed on our access.

[1] https://www.tudelft.nl/en/about-tu-delft/strategy/integrity-policy/tu-delft-vision-on-integrity-2018-2024

[2] https://www.nwo.nl/en/netherlands-code-conduct-research-integrity

In our reproducibility package [15], we publish both the collected data and data pre-processing steps taken in order to ensure that the data can be easily integrated in our analysis workflow.

9.2 Bias in the Data

As outlined in Section 8, our dataset contains inherent biases. It represents only the subset of incidents that have been voluntarily and publicly reported, which does not encompass the full landscape of operational failures in the industry. Therefore, conclusions drawn from our analysis pertain only to publicly reported incidents and may not generalize to unreported or proprietary events.

Furthermore, the level of detail and quality of these incident reports is influenced by each organization's internal reporting practices. Some companies provide comprehensive postmortem analyzes, while others offer minimal descriptions, leading to variability in data richness and consistency.

Only incident reports published between 2022 and 2024 were considered, and within this range, several entries were excluded due to insufficient detail or lack of actionable content. This exclusion was performed to reduce the likelihood of LLM hallucinations and, more critically, to prevent the inadvertent amplification of biases introduced by prompt engineering. Although we utilized a neutral prompting strategy and included fallback options for ambiguous cases, the LLM was primarily used to simulate the reasoning process of a site reliability engineer, not as an oracle of truth.

9.3 Models and Energy Consumption

The potential environmental impact of this work warrants some consideration. Although our analysis pipeline is computationally efficient, it relies on the use of LLMs for textual classification, which introduces non-negligible inference costs in terms of both energy consumption and financial resources.

To balance performance and sustainability, we opted for LLMs from the GPT-Mini family, which offer a favorable trade-off between inference speed, accuracy, and resource use. These models were used through the OpenAI API and were used in their current form, incurring no additional training costs.

9.4 Reproducible Research

To ensure reproducibility, the complete dataset used in this study - along with any filtering criteria and pre-processing steps - has been made available in the project's public repository and reproducibility package [15]. The repository also includes the source code for all tools developed and utilized throughout the study, including web scraping utilities, characteristics extraction modules, classification scripts, and clustering pipelines.

Additionally, to further promote replicability and reduce variability in experimental outcomes, we explicitly set random seeds for all processes involving stochastic elements, such as data splitting, and clustering initialization.

By making both the data and code publicly available under open licenses, this study promotes open science practices and encourages further research in the areas of operational incident analysis and AIOps systems.

Two reproducibility packages are provided on Zenodo as well:

- the web crawling tool package
- the analysis tool package

9.5 The use of AI

AI tools (ChatGPT[3], Claude[4]) were used throughout this project to support various research and presentation tasks, in accordance with responsible use guidelines. We distinguish between the use of AI for coding (9.5.1) and for paper writing (9.5.2).

9.5.1 AI Assisted Coding. All code used to support this paper is written by a human. However, AI has been used in the process of:

- establishing which Python packages to use for a task
- generating examples for simplified tasks
- understanding error messages

The following prompts were used:

- "*What Python packages are best suited to perform task* X?; where X ranges from performing statistical analysis tests, to web crawling activities.
- "*How can package* X *be used to perform* Y? *Please provide a Python script as an example.*"
- "*I get this error:* X. *Why?*"

The model was treated strictly as a potential knowledge resource, with output in need of manual validation and intervention. Output was not used verbatim.

9.5.2 AI Assisted Paper Writing. During the process of writing the paper, we used AI to:

- Format and refactor LaTeXdocuments, particularly for aligning tables and inserting complex figures
- Generate tabular representations of data provided in its textual format in LaTeX
- Propose alternative data visualizations when plots and/or tables exceeded standard page dimensions
- Refine the writing by suggesting synonymous phrasing or by acting as a critical reviewer

The following prompts were used:

- *What is the Latex code for inserting a figure with 2 subfigures?*
- *Could you provide the Latex code for a table containing data* X?
- *My table does not fit in one column in my two-column paper layout. Could you please suggest how to make it fit? Would another method work better?*
- *What are synonymous phrases for* X?
- *You are a critical reviewer of an academic paper. You need to ensure that section* X *follows the following rules*: Ys. *What are the strengths and weaknesses of this section?*

All AI-generated outputs were carefully reviewed and edited. At no point were generative models allowed to produce final results or analyses without human validation. When assisting in refining of the writing, model generated sentences were not inserted verbatim. Additionally, when alternative data visualizations were proposed, model output was a script to generate the visualization and not the visualization itself. Even in this case, the script was not considered

[3] https://chatgpt.com/
[4] https://claude.ai/

correct by default, but modified and adjusted to fit our analysis pipeline.

We explicitly disclose the use of AI to maintain transparency and to align with emerging norms in the responsible integration of AI in scientific workflows.

10 CONCLUSION

This study advances the understanding of software incidents in production environments by systematically characterizing incident attributes, exploring cause-impact-remediation relationships, and uncovering recurring failure patterns through clustering analysis.

We characterize incidents by their cause, impact, severity, mitigation strategy employed and remediation actions taken. The defined taxonomy also distinguishes between change-induced and other incidents, further characterizing those incidents by the type of software change that triggered them. Our findings confirm that code defects and capacity issues are the most frequent root causes, usually leading to partial outages or degraded performance rather than complete failures. Remediation efforts tend to focus on damage containment, with fixes tailored to specific causes, highlighting the importance of targeted playbooks to improve operational resilience.

We also found clear links between causes, impacts, and responses: capacity problems often result in connection failures and require manual recovery, while improper change operations are more likely to cause full outages that demand rollbacks. Change-related incidents, particularly code deployments, make up over half of all cases and tend to be more severe and challenging to recover from. Configuration changes and third-party updates also contribute, revealing gaps in testing beyond code correctness.

Clustering analysis identified seven distinct incident archetypes, with key groups centered on capacity-driven outages, code defects, and failures tied to improper changes. The presence of mixed clusters underscores the complexity of incident causality and supports the value of unsupervised methods for capturing failure modes.

Overall, these insights offer practical guidance for improving incident prevention, detection, and response to build more resilient software systems. Future work should address current data limitations and explore richer sources to understand incidents better and more reliably.

REFERENCES

[1] W. Tsai, R. Mojdehbakhsh, and F. Zhu, "Ensuring system and software reliability in safety-critical systems", in *Proceedings. 1998 IEEE Workshop on Application-Specific Software Engineering and Technology. ASSET-98 (Cat. No.98EX183)*, 1998, pp. 48–53. DOI: 10.1109/ASSET.1998.688232.

[2] J. Faustino, D. Adriano, R. Amaro, R. Pereira, and M. M. da Silva, "DevOps benefits: A systematic literature review", *Software: Practice and Experience*, vol. 52, no. 9, pp. 1905–1926, 2022. DOI: https://doi.org/10.1002/spe.3096. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3096. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3096.

[3] J. Lu, L. Yu, X. Li, L. Yang, and C. Zuo, "LLaMA-Reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning", in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, 2023, pp. 647–658. DOI: 10.1109/ISSRE59848.2023.00026.

[4] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. 2016. [Online]. Available: http://landing.google.com/sre/book.html.

[5] A. Saha and S. C. H. Hoi, "Mining root cause knowledge from cloud service incident investigations for AIOps", in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 197–206, ISBN: 9781450392266. DOI: 10.1145/3510457.3513030. [Online]. Available: https://doi.org/10.1145/3510457.3513030.

[6] F. Menges and G. Pernul, "A comparative analysis of incident reporting formats", *Computers & Security*, vol. 73, pp. 87–101, 2018, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2017.10.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404817302250.

[7] J. Sillito and E. Kutomi, "Failures and fixes: A study of software system incident response", in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2020, pp. 185–195.

[8] E. Kapel, L. Cruz, D. Spinellis, and A. Van Deursen, "On the difficulty of identifying incident-inducing changes", in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '24, Lisbon, Portugal: Association for Computing Machinery, 2024, pp. 36–46, ISBN: 9798400705014. DOI: 10.1145/3639477.3639755. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3639477.3639755.

[9] Y. Wu *et al.*, "An empirical study on change-induced incidents of online service systems", in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2023, pp. 234–245. DOI: 10.1109/ICSE-SEIP58684.2023.00027.

[10] D. Roy *et al.*, "Exploring LLM-based agents for root cause analysis", in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 208–219, ISBN: 9798400706585. DOI: 10.1145/3663529.3663841. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3663529.3663841.

[11] X. Zhang *et al.*, "Automated root causing of cloud incidents using in-context learning with GPT-4", in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 266–277, ISBN: 9798400706585. DOI: 10.1145/3663529.3663846. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3663529.3663846.

[12] Y. Remil, A. Bendimerad, R. Mathonat, and M. Kaytoue, *AIOps solutions for incident management: Technical guidelines and a comprehensive literature review*, 2024. arXiv: 2404.01363 [cs.OS]. [Online]. Available: https://arxiv.org/abs/2404.01363.

[13] N. Zhao *et al.*, "Identifying bad software changes via multimodal anomaly detection for online service systems", in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021, Athens, Greece: Association for Computing Machinery, 2021, pp. 527–539, ISBN: 9781450385626. DOI: 10.1145/3468264.3468543. [Online]. Available: https://doi.org/10.1145/3468264.3468543.

[14] Y. Dang, Q. Lin, and P. Huang, "AIOps: Real-world challenges and research innovations", in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 4–5. DOI: 10.1109/ICSE-Companion.2019.00023.

[15] I. M. Aldea, *Understanding software failures through incident report analysis - reproducibility package*, Jun. 2025. DOI: 10.5281/zenodo.15712312. [Online]. Available: https://doi.org/10.5281/zenodo.15712312.

[16] I. ISO, "ISO/IEC/IEEE international standard-systems 820 and software engineering–vocabulary", Tech. rep.(Aug 2017). doi: 10.1109/IEEESTD, Tech. Rep., 2017.

[17] Z. Chen *et al.*, "Towards intelligent incident management: Why we need it and how we make it", in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1487–1497, ISBN: 9781450370431. DOI: 10.1145/3368089.3417055. [Online]. Available: https://doi.org/10.1145/3368089.3417055.

[18] P. Cichonski, T. Millar, T. Grance, K. Scarfone, *et al.*, "Computer security incident handling guide", *NIST Special Publication*, vol. 800, no. 61, pp. 1–147, 2012.

[19] Verica, "2022 VOID report", Verica, Tech. Rep., 2022, Accessed: 2025-06-02. [Online]. Available: https://mcusercontent.com/ad9b3a31c069c03a14227a79c/files/29e138f4-05b7-f560-a7d5-c337538a93eb/2022_Void_Report.pdf.

[20] B. Li, M. Li, K. Chen, and C. Smidts, "Integrating software into PRA: A software-related failure mode taxonomy", *Risk Analysis*, vol. 26, no. 4, pp. 997–1012, 2006. DOI: https://doi.org/10.1111/j.1539-6924.2006.00795.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1539-6924.2006.00795.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1539-6924.2006.00795.x.

[21] T. Agrawal, G. S. Walia, and V. K. Anu, "Development of a software design error taxonomy: A systematic literature review", *SN Computer Science*, vol. 5, no. 5, p. 467, 2024.

[22] P. Dogga, C. Bansal, R. Costleigh, G. Jayagopal, S. Nath, and X. Zhang, "AutoARTS: Taxonomy, insights and tools for root cause labelling of incidents in Microsoft Azure", in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, Boston, MA: USENIX Association, Jul. 2023, pp. 359–372, ISBN: 978-1-939133-35-9. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/dogga.

[23] S. Davidovic, *Incident Metrics in SRE*. O'Reilly Media, Inc, 2021.

[24] T. Koutroukis, D. Chatzinikolaou, C. Vlados, and V. Pistikou, "The post-COVID-19 era, fourth industrial revolution, and new globalization: Restructured labor relations and organizational adaptation", *Societies*, vol. 12, no. 6, p. 187, 2022.

[25] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition", *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 50–70, 2022. DOI: 10.1109/TKDE.2020.2981314.

[26] LLMCompare, *GPT-4o Mini vs GPT-4o vs GPT-3.5 Turbo*, https://llmcompare.net/blog/gpt-4o-mini, Accessed: 2025-06-02, 2024.

[27] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks", in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds., Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410. [Online]. Available: https://aclanthology.org/D19-1410/.

[28] N. Muennighoff *et al.*, "MTEB: Massive text embedding benchmark", *arXiv preprint arXiv:2210.07316*, 2022. [Online]. Available: https://arxiv.org/abs/2210.07316.

[29] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning", *Pattern Recognition*, vol. 45, no. 9, pp. 3084–3104, 2012, Best Papers of Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA'2011), ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2012.03.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320312001203.

[30] A. J. Viera and J. M. Garrett, "Understanding interobserver agreement: The kappa statistic", en, *Fam Med*, vol. 37, no. 5, pp. 360–363, May 2005.

[31] A. Chao, "Nonparametric estimation of the number of classes in a population", *Scandinavian Journal of statistics*, pp. 265–270, 1984.

[32] A. Chao, C.-H. Chiu, *et al.*, "Species richness: Estimation and comparison", *Wiley StatsRef: statistics reference online*, vol. 1, p. 26, 2016.

[33] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis", *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987, ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377042787901257.

[34] T. Calinski and J. Harabasz, "A dendrite method for cluster analysis", *Communications in Statistics*, vol. 3, no. 1, pp. 1–27, 1974. DOI: 10.1080/03610927408827101. eprint: https://www.tandfonline.com/doi/pdf/10.1080/03610927408827101. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/03610927408827101.

[35] D. L. Davies and D. W. Bouldin, "A cluster separation measure", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, 1979. DOI: 10.1109/TPAMI.1979.4766909.

[36] J. Wynn *et al.*, "Threat assessment and remediation analysis (TARA)", *MITRE Corporation: Bedford, MA, USA*, 2014.

[37] P. Huang *et al.*, "Gray failure: The Achilles' heel of cloud-scale systems", in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, ser. HotOS '17, Whistler, BC, Canada: Association for Computing Machinery, 2017, pp. 150–155, ISBN: 9781450350686. DOI: 10.1145/3102980.3103005. [Online]. Available: https://doi.org/10.1145/3102980.3103005.

[38] *The web robots pages — robotstxt.org*, https://www.robotstxt.org/, [Accessed 09-05-2025].