Triage in Software Engineering: A Systematic Review of Research and Practice

YONGXIN ZHAO, Nankai University, China
SHENGLIN ZHANG*, Nankai University, China
YUJIA WU, Nankai University, China
YUXIN SUN, Nankai University, China
YONGQIAN SUN, Nankai University, China
DAN PEI, Tsinghua University, China
CHETAN BANSAL, Microsoft, USA
MINGHUA MA*, Microsoft, USA

As modern software systems continue to grow in complexity, triage has become a fundamental process in system operations and maintenance. Triage aims to efficiently prioritize, assign, and assess issues to ensure the reliability of complex environments. The vast amount of heterogeneous data generated by software systems has made effective triage indispensable for maintaining reliability, facilitating maintainability, and enabling rapid issue response. Motivated by these challenges, researchers have devoted extensive effort to advancing triage automation and have achieved significant progress over the past two decades. This survey provides a comprehensive review of 234 papers from 2004 to the present, offering an in-depth examination of the fundamental concepts, system architecture, and problem statement. By comparing the distinct goals of academic and industrial research and by analyzing empirical studies of industrial practices, we identify the major obstacles that limit the practical deployment of triage systems. To assist practitioners in method selection and performance evaluation, we summarize widely adopted open-source datasets and evaluation metrics, providing a unified perspective on the measurement of triage effectiveness. Finally, we outline potential future directions and emerging opportunities to foster a closer integration between academic innovation and industrial application. All reviewed papers and projects are available at https://github.com/AIOps-Lab-NKU/TriageSurvey.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Computer systems organization → Maintainability and maintenance**; • **Computing methodologies → Artificial intelligence**.

Additional Key Words and Phrases: Triage, software engineering, bug reports, incident tickets, alerts

1 Introduction

As modern software systems and large-scale online services continue to evolve in complexity, the volume of software bugs, incidents, and alerts has grown exponentially. If not addressed promptly, such issues can lead to severe operational disruptions and substantial financial losses [160, 162, 191]. Effective issue management has thus become indispensable to ensuring the stability and reliability of large-scale systems. When an issue report is submitted, it initiates a critical, multi-step process known as *triage*. Broadly defined, triage encompasses a sequence of analytical activities aimed at efficiently managing the lifecycle of an issue. As shown in Figure 1, the process involves identifying duplicates, prioritizing the issue's urgency, classifying the issue's type (*e.g.*, bug, feature request, or security vulnerability), and

*Shenglin Zhang and Minghua Ma are the corresponding authors.

Authors' Contact Information: Yongxin Zhao, zyx_nkcs@mail.nankai.edu.cn, Nankai University, China; Shenglin Zhang, zhangsl@nankai.edu.cn, Nankai University, China; Yujia Wu, 2120240813@mail.nankai.edu.cn, Nankai University, China; Yuxin Sun, 2111498@mail.nankai.edu.cn, Nankai University, China; Yongqian Sun, sunyongqian@nankai.edu.cn, Nankai University, China; Dan Pei, peidan@tsinghua.edu.cn, Tsinghua University, China; Chetan Bansal, chetanb@microsoft.com, Microsoft, USA; Minghua Ma, minghuama@microsoft.com, Microsoft, USA.

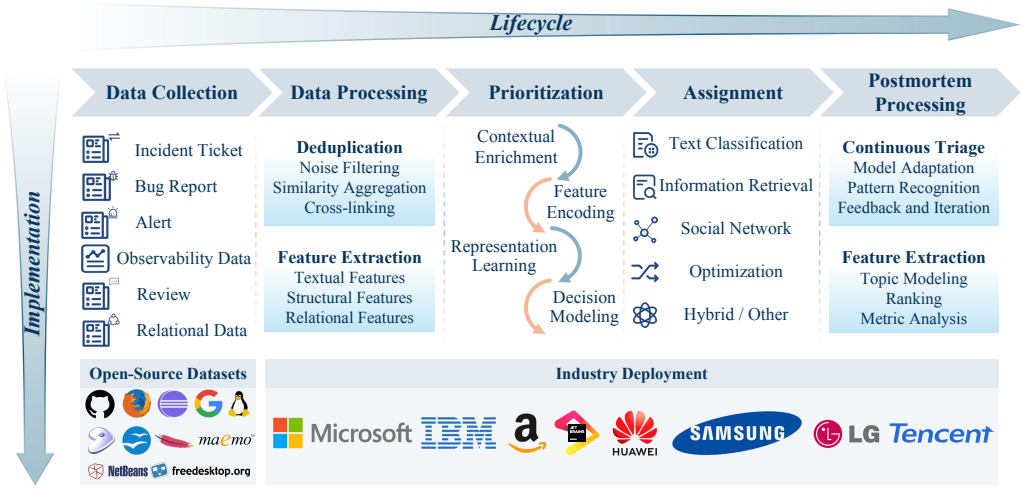


Fig. 1. A conceptual map of automated triage research and practice.

routing the issue to the most appropriate entity for resolution. This entity may be a specific developer, a component team, or an automated analysis pipeline.

Manual triage, however, is both time-consuming and labor-intensive, particularly given the sheer volume of reports in large-scale projects [160]. For example, the Eclipse project receives approximately 91 bug reports per day [153], while Mozilla handles around 300 bug reports daily [46]. Moreover, triagers in such environments must allocate reports across numerous developers [87], yet it is unrealistic to expect them to possess a comprehensive understanding of each developer's technical expertise, domain familiarity, and current workload [40]. Consequently, manual triage often results in repeated report reassignment, commonly referred to as "bug tossing", until a suitable resolver is identified. This process not only delays issue resolution but also diminishes operational efficiency and user satisfaction. Given these challenges, the adoption of automated triage mechanisms has become critical to maintaining the reliability, availability, and scalability of modern software systems. Understanding the role, objectives, and advantages of triage, therefore, provides essential context for developing effective, intelligent solutions that enhance issue resolution efficiency and optimize resource allocation.

1.1 Why Triage in Software Engineering?

In contemporary software engineering (SE), the ever-increasing volume and complexity of incident tickets, bug submissions, and system alerts have rendered triage a cornerstone process for ensuring operational efficiency and system reliability [27, 146, 160, 192]. Effective triage, spanning incident triage, bug triage, and alert aggregation, delivers several key benefits:

- **Accelerated issue resolution.** By systematically prioritizing, classifying, and routing reports, triage reduces manual workload and mitigates the "report tossing" phenomenon, where reports are repeatedly reassigned until they reach an appropriate resolver [78, 146]. This streamlining significantly shortens the time to resolution and minimizes service disruption.

- **Enhanced diagnostic accuracy.** Automated triage mechanisms effectively filter out duplicates, irrelevant, or low-quality reports [25, 98, 114], ensuring that engineers work with concise, actionable information. This improves diagnostic precision and prevents redundant investigations.
- **Facilitated cross-team collaboration.** By providing a consolidated and structured view of issues, triage promotes effective communication and knowledge sharing among development, operations, and support teams. This is particularly critical in large-scale or community-driven projects, where issues are typically assigned to teams or modules rather than individual engineers [13, 79].
- **Foundation for intelligent automation.** The structured outputs generated during triage serve as high-quality input for machine learning and knowledge-based reasoning systems. These outputs facilitate the automation of downstream tasks such as root cause analysis, failure prediction, and recommendation of remediation actions [128, 178], thereby enabling proactive and intelligent system management.

Despite these advantages, practical triage remains a complex and challenging endeavor. The heterogeneity of input data, the coexistence of structured and unstructured information, and the intricate dependencies among triage subtasks impose significant constraints on full automation. As a result, the development of systematic, data-driven, and explainable triage approaches has become a pressing need for modern SE practices.

1.2 Why a Survey of Triage in Software Engineering?

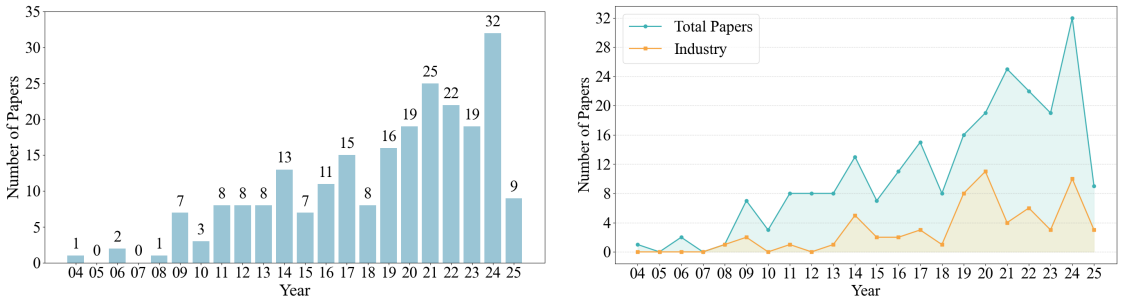
With the escalating complexity of software ecosystems and the growing prevalence of incidents, research on automated triage has witnessed rapid progress and increasing scholarly attention, as illustrated in Figure 2a. A diverse range of studies has explored various subtasks, such as bug classification, duplicate detection, and incident management, reflecting both the practical importance and the technical challenges inherent in triage. Furthermore, Figure 2b highlights the increasing number of triage-related publications produced in collaboration with industry partners, underscoring the real-world applicability and impact of this line of research.

However, existing surveys often adopt a fragmented perspective, typically focusing on individual subtasks rather than viewing triage as an integrated, end-to-end process. For instance, some reviews focus exclusively on developer assignment [112], while others examine incident triage within cloud environments [128, 178]. This fragmented view overlooks the dependencies and interactions among subtasks, for example, how duplicate detection [114] informs component assignment [146], or how bug tossing knowledge [78] enhances developer recommendation [164]. Such isolation hinders a holistic understanding of triage as a cohesive, interdependent process that spans multiple organizational and technical dimensions.

To bridge this gap, this survey provides a unified and comprehensive examination of triage across the full lifecycle. We systematically review techniques encompassing data preprocessing, prioritization, assignment, and postmortem analysis, offering an integrative perspective on how these components interact. This holistic approach not only reveals methodological trends and open research challenges but also provides actionable insights for practitioners seeking to design or enhance triage pipelines in real-world systems. By consolidating fragmented knowledge into a coherent framework, this survey aims to advance both theoretical understanding and practical implementation of triage in modern software engineering.

1.3 Research Questions.

With the growing complexity of modern software systems, triage has become an essential process for effectively managing and prioritizing issues. Over the past two decades, research on triage in software engineering has advanced



(a) Publication distribution over time (2004–Present).

(b) Publication distribution involving industry collaboration (2004–Present).

Fig. 2. Analysis of Publication Trends on Triage in Software Engineering.

considerably, spanning diverse contexts such as bug report assignment, incident management, and alert prioritization. Despite this progress, the existing body of work remains fragmented, characterized by inconsistent definitions, heterogeneous methodologies, and varying scopes of application. In light of the rapid proliferation of data-driven and AI-assisted approaches, a comprehensive and systematic review of triage research in software engineering is both timely and necessary.

To this end, we structure our survey around the following research questions, each targeting a fundamental dimension of triage in software engineering:

- **RQ1:** How have existing triage methods evolved across the different stages of the triage lifecycle?
- **RQ2:** What practical challenges and limitations are encountered during the deployment of triage processes in real-world settings?
- **RQ3:** How is the effectiveness of triage approaches evaluated, and what metrics and benchmarks are commonly adopted?

These research questions form a coherent taxonomy that captures the conceptual and practical evolution of triage research. RQ1 provides a lifecycle-oriented synthesis of existing methods, mapping the progression of triage techniques across various phases. RQ2 examines the operational and organizational challenges that emerge in real-world deployments, offering insights into gaps between research and practice. Building upon these foundations, RQ3 focuses on evaluation methodologies, highlighting how different metrics and benchmarks are used to assess triage effectiveness. Collectively, this structured framework enables a systematic exploration of triage in software engineering, bridging conceptual understanding, methodological development, and empirical evaluation.

1.4 Survey Structure.

The overall structure of this survey is depicted in Figure 3. Section 2 introduces essential background concepts, followed by Section 3, which details the methodology adopted in this work. Section 4 presents related studies organized according to the triage lifecycle. Section 5 discusses practical challenges encountered in real-world triage processes. Section 6 outlines evaluation metrics and publicly available benchmarks to facilitate empirical studies. Finally, Section 7 highlights promising directions for future research.

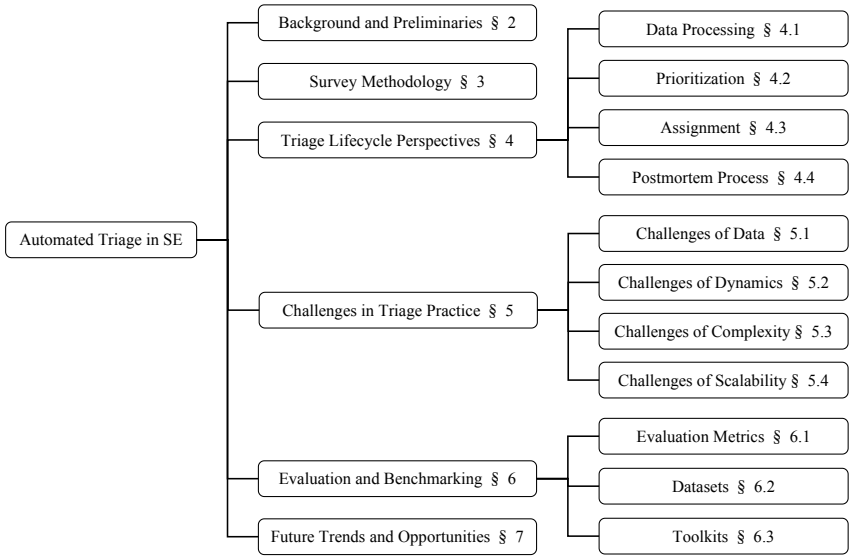


Fig. 3. Structure of this survey.

2 Background and Preliminaries

In this section, we provide the foundational background of triage in software engineering by tracing its historical origins, exploring its adoption, and describing the key data sources that support and inform triage practices. We then establish a conceptual framework for our survey by defining a general triage lifecycle. Finally, we review related surveys to position our work within the broader research landscape.

2.1 Background of Triage in Software Engineering

Historical Origins. The term *triage* originates from the French verb *trier*, meaning "to sort" or "to select". It was initially introduced within the French medical services as a systematic approach to classifying patients [130]. Historically, triage referred to the process of stratifying individuals into categories such as immediate, urgent, and non-urgent during mass casualty incidents, particularly in military contexts. The primary objective was to maximize survival rates under conditions of limited medical resources [35, 130].

Adoption. With the rapid growth in complexity and scale of software systems, the fundamental philosophy of triage, prioritizing interventions under resource constraints, has found significant relevance beyond traditional medical applications. Within software engineering, triage has become a key practice for managing the overwhelming influx of information, such as bug reports, alerts, and user feedback, that arise during software development and operation. The capacity to efficiently identify, prioritize, and allocate resources to the most critical and high-impact issues is essential for maintaining software quality, ensuring service reliability, and enhancing user satisfaction. To meet these demands, a wide range of automated and intelligent triage approaches have been proposed, drawing upon advances in machine learning, natural language processing, and data mining. These systems aim not only to automate but also to improve upon manual triage processes in terms of both efficiency and accuracy.

Data Sources. A wide spectrum of data sources is utilized in the triage of software engineering, encompassing bug reports, incident tickets, alerts, observability data, reviews, and relational data. Each type of data contributes a unique yet complementary perspective on system behavior, operational conditions, and potential issue contexts.

- **Incident Tickets:** Incident tickets encompass a range of textual and semi-structured inputs, such as customer-reported incident (CI) tickets, monitor-reported incident (MI) tickets, and raw incident tickets generated within operational environments. These tickets, typically expressed in natural language, contain essential contextual details that support key triage tasks, including incident linking, team assignment, and prioritization. The narrative descriptions often reveal early indicators of system degradation and provide contextual signals that guide subsequent analysis and resolution.
- **Bug Reports:** Bug reports are typically semi-structured, containing both unstructured narratives and structured metadata, along with historical relational information. The unstructured portion includes summaries and detailed descriptions that often specify reproduction steps, expected and actual outcomes, error messages, stack traces, and embedded code snippets. The structured portion consists of predefined fields such as product, component, operating system, software version, severity, and priority, which provide essential categorical features frequently utilized in triage algorithms. Historical and relational data, including assignment histories and developer activity logs, capture previous decision-making patterns and developer expertise. These data collectively provide valuable context for improving triage accuracy and efficiency.
- **Alerts:** Alerts are automatically generated by monitoring systems through predefined rules or anomaly detection mechanisms. They act as direct triggers for incidents and usually contain metadata such as timestamps, source identifiers, severity levels, and brief diagnostic descriptions. Alerts form the initial layer of triage, serving as early warnings that enable proactive system management and timely response to potential issues.
- **Observability Data:** Observability data, such as Key Performance Indicators (KPIs), metrics, traces and system logs, offer detailed insights into system behavior and performance dynamics. In triage, these data serve as the analytical foundation for tasks including anomaly detection, root cause analysis, and responsible team assignment. Their temporal nature enables real-time monitoring and supports the correlation of performance deviations with potential system issues.
- **Reviews:** Review data consists primarily of textual information from two sources: users and developers. User data includes ratings and text in app reviews, reflecting their experiences with the product. Developer data consists of commit information and code comments detailing functionality and data transfer. Compared to reports and alerts, review data is larger, less structured, and more conversational, requiring advanced natural language processing (NLP) techniques for analysis.
- **Relational Data:** Relational data encapsulates the historical and contextual interactions among bugs, developers, and software components. In contrast to isolated bug reports, relational data captures the dynamic dependencies and collaboration patterns that emerge throughout the bug resolution process. These data are typically represented as heterogeneous networks or bipartite graphs, where nodes correspond to entities and edges encode their relationships. By modeling such interactions, relational data facilitates the inference of latent expertise, developer collaboration behaviors, and structural dependencies among software modules.

2.2 A General Triage Lifecycle

To provide a comprehensive understanding of automated triage, we conceptualize it as a multi-stage lifecycle, as illustrated in Figure 4. This lifecycle models the trajectory of an issue report from its initial submission to its final resolution and the subsequent extraction of actionable knowledge. While not all triage systems implement every stage, this model captures the essential processes that have been extensively studied in the literature. The lifecycle can be broadly divided into four major phases: *Data Processing*, *Prioritization*, *Assignment*, and the *Postmortem Process*.

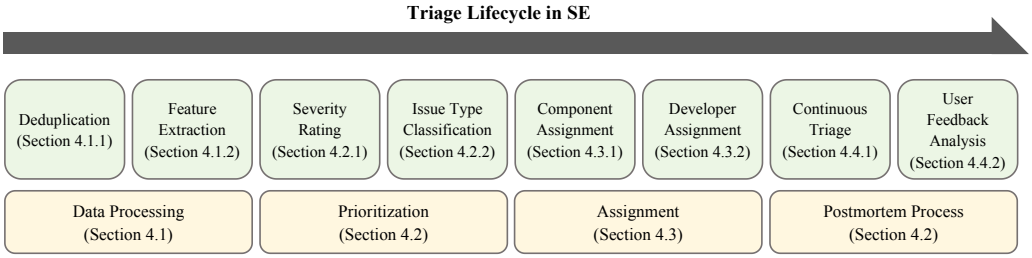


Fig. 4. The general lifecycle of triage in SE.

2.2.1 Data Processing. The first phase of the triage lifecycle focuses on transforming raw, unstructured issue reports into structured, feature-rich data that are suitable for automated analysis. The quality of this phase directly influences the performance of subsequent stages. It typically encompasses two main tasks:

- **Deduplication:** Large-scale projects frequently receive multiple reports describing the same underlying defect. Detecting and merging these duplicates is critical for reducing redundant engineering effort. Deduplication is often formulated as a similarity-matching problem, where the objective is to identify duplicate reports based on textual and metadata similarities. The main challenge lies in detecting duplicates that differ in their wording or structure, which requires sophisticated feature extraction methods and similarity metrics [114].
- **Feature Extraction:** Issue reports usually consist of a mixture of natural language descriptions, code snippets, stack traces, and metadata. The task of feature extraction involves identifying and representing meaningful information from these heterogeneous inputs. A key challenge lies in capturing the semantic relationships between textual and structural elements [46]. Early studies relied on classical text mining techniques such as Term Frequency–Inverse Document Frequency (TF–IDF) and Latent Semantic Indexing (LSI) [2]. More recent methods incorporate structural analysis by parsing code into Abstract Syntax Trees (ASTs) to represent code semantics separately from textual descriptions [12]. The advent of pre-trained language models (PLMs), including BERT and its variants, has further advanced this area by enabling deep contextual embeddings that capture rich semantic and syntactic information from issue descriptions [46, 87].

2.2.2 Prioritization. After data preprocessing, the next phase involves assessing the urgency and nature of an issue to determine its relative importance. This prioritization phase ensures that limited engineering resources are allocated to the most critical problems. It generally includes the following two tasks:

- **Severity Rating:** The goal of this task is to automatically predict the severity level of a bug or incident (*e.g.*, Blocker, Critical, or Minor) [11]. Accurate severity prediction is essential for effective risk management and for ensuring that system-critical issues receive timely attention. Recent studies have explored the use of large language models, sometimes augmented with sentiment analysis, to infer the implicit urgency embedded in textual descriptions and to improve prediction accuracy [11].
- **Issue Type Classification:** In addition to severity, it is often necessary to categorize issues based on their fundamental nature. This classification distinguishes between types such as functional bugs, performance degradations, and security vulnerabilities, thereby supporting more targeted handling. Some studies have proposed taxonomies of issue types and developed automated classifiers to operationalize these distinctions [25]. Other approaches employ multi-task learning frameworks to jointly perform issue classification and developer assignment, enhancing the overall triage efficiency [12].

2.2.3 Assignment. The assignment phase is concerned with routing each issue to the most appropriate entity for resolution. The effectiveness of this process directly affects the average Time-To-Resolution. Depending on organizational structure and issue granularity, assignment may occur at different levels:

- **Component Assignment:** In large projects, software is often divided into multiple components, each maintained by a dedicated team. Component assignment aims to identify which component is likely affected by the issue, thereby narrowing the search space for subsequent developer assignment. Research in this area often focuses on learning the relationships between issue descriptions and component-specific information. Recent work has introduced deep learning models that address challenges such as ambiguous or few-shot components, where traditional feature-based methods perform poorly [145, 171].
- **Developer Assignment:** This task involves assigning the issue directly to the developer or team best equipped to resolve it. Developer assignment has been a long-standing research focus, evolving from early machine learning models based on text categorization [9] to contemporary deep learning approaches that utilize graph neural networks to capture collaboration structures [39, 164]. More recent studies have also leveraged large language models for semantic understanding and context reasoning [87]. Advanced systems additionally incorporate dynamic contextual factors such as workload, expertise, and availability [73]. In industrial settings, this task is often adapted to *team-level assignment*, where issues are routed to the most relevant team instead of a single developer to align with collaborative workflows and organizational practices [79, 137].

2.2.4 Postmortem Process. The triage lifecycle is not a linear or static process. Modern triage systems increasingly incorporate feedback loops and continuous learning mechanisms that allow them to adapt and evolve. This adaptive capability is essential for managing the dynamic nature of software projects and the changing needs of users. Within this ongoing process, two key aspects are particularly significant: continuous triage and the integration of user feedback.

- **Continuous Triage:** Traditional triage models often make a single, static assignment decision when a new report is submitted. In practice, however, the initial information provided by a report may be incomplete, and additional context frequently emerges as engineers discuss and investigate the issue. Continuous triage addresses this limitation by iteratively refining the assignment as new information, such as discussion comments or updated diagnostics, becomes available [27]. This iterative re-evaluation is particularly important in large-scale online service environments, where incident discussions evolve rapidly and timely adjustments can significantly reduce service downtime.
- **User Feedback Analysis:** In addition to data derived from internal development and monitoring processes, feedback from end-users provides valuable, real-world insights into software performance and emerging issues. Analyzing user feedback, commonly expressed through application reviews or problem reports, serves as a proactive input to the triage pipeline [54]. Although the immediate context of user reviews differs from that of internal issue reports, both share common linguistic and semantic characteristics. Consequently, analytical methods developed for processing user reviews can inform and enhance triage practices by offering complementary perspectives on system reliability and user experience.

2.3 Related Surveys

Comprehensive surveys and systematic reviews have extensively examined various subdomains within software maintenance and IT operations. As summarized in Table 1, these related studies can be broadly classified into two major categories. The first category, represented by works such as Remil et al. [128], Yu et al. [178], and Notaro et al. [115],

Table 1. Comparison of existing triage-related surveys based on their focused domain and analytical perspectives. The abbreviations in the "Data Sources" column stand for: **A** = Alerts, **OD** = Observability Data (*e.g.*, KPIs, Metrics, Logs, and Traces), **IT** = Incident Tickets, and **BR** = Bug Reports.

Reference	Year	Focus Domain	Data Sources	Analysis Perspectives
Remil et al. [128]	2024	Incident management	OD, IT	Fundamental abilities of AIOps
Yu et al. [178]	2024	Alert and incident management	A, IT, OD	Workflow of management
Notaro et al. [115]	2021	Failure management	BR, IT, OD	Intervention time window
Zhang et al. [186]	2016	Bug resolution	BR	Lifecycle of bug resolution
Bocu et al. [20]	2023	Bug triage and management	BR	Triage techniques
Uddin et al. [153]	2024	Bug triage	BR	Triage techniques
Nagwani et al. [112]	2023	Bug triage	BR	Triage techniques
Qian et al. [126]	2023	Bug triage	BR	Input data source
Akila et al. [3]	2014	Bug triage	BR	Workflow of triage
Our work	-	Triage in SE	A, BR, IT, OD	Lifecycle of Triage

explores overarching management frameworks. In these studies, triage is considered one component within a broader operational framework, and the primary emphasis lies on the overall system lifecycle or high-level capabilities.

The second category of related surveys focuses specifically on software defects, offering detailed analyses of bug triage techniques. Representative examples include the studies by Zhang et al. [186], Nagwani et al. [112], and Qian et al. [126]. These surveys concentrate almost exclusively on bug reports as the primary data source and investigate various approaches to defect management, classification, and assignment.

This distinction reveals a clear research gap. Although individual subdomains of triage have been studied in depth, no existing survey has systematically synthesized or compared the triage process across different, yet conceptually related, artifacts in software engineering. In particular, there remains a lack of unified examination encompassing alerts, incidents, and bugs. While triage has often been treated as a subordinate activity within larger operational workflows, it in fact constitutes a critical and high-leverage process that warrants focused and systematic investigation.

As modern software systems continue to grow in complexity, the increasing volume, heterogeneity, and urgency of alerts, incidents, and bug reports have made manual triage both inefficient and error-prone [46, 160, 192]. These challenges highlight the necessity for a dedicated and comprehensive examination of automated triage methodologies that transcend specific data types or operational contexts.

In contrast to prior domain-specific reviews, the present survey adopts a holistic perspective by positioning triage itself as the central object of study. The key contribution of this work lies in systematically analyzing software triage as a unified process that spans the entire lifecycle across alerts, incidents, and bugs. This survey offers a comparative analysis of the complete triage pipeline, encompassing data preprocessing, prioritization, assignment, and post-processing, and highlights both the commonalities and distinctive features observed in each domain. To the best of our knowledge, this is the first comprehensive survey that bridges the Artificial Intelligence for IT Operations (AIOps) and IT Service Management (ITSM) community, primarily focused on alert, incident, and operational data, with the software engineering community, which traditionally emphasizes bug data, under a unified triage framework. By integrating insights from these previously separate research areas, this study provides a coherent understanding of software triage and aims to foster future innovation through the convergence of methodologies and perspectives.

3 Survey Methodology

3.1 Survey Scope

The term triage originated in the medical domain, where it refers to the process of prioritizing patients based on the severity of their conditions to ensure that scarce medical resources are allocated efficiently to those in greatest need [130].

Over time, this concept has evolved beyond its clinical roots, emerging as a general strategy for managing large volumes of competing tasks under resource constraints. Today, triage serves as a fundamental mechanism in a broad range of domains, including software engineering [169], cyber security [94], and emergency response systems [50].

Within the context of software development and operations, triage is critical to managing the growing complexity and scale of modern computing systems. It facilitates timely and informed decision-making by prioritizing, classifying, and routing issues, such as bug reports, system alerts, service incidents, and performance anomalies, to the appropriate stakeholders, whether human engineers or automated remediation systems. Effective triage not only reduces the cognitive burden on developers and operators but also enhances system reliability, service responsiveness, and organizational agility.

This survey presents a comprehensive examination of triage practices in the domain of software development and operations. Rather than treating triage as a monolithic or static process, we conceptualize it as a dynamic, multi-phase lifecycle consisting of four interrelated stages: issue intake and preprocessing, prioritization, assignment, and resolution feedback. Each stage introduces distinct technical and organizational challenges, including but not limited to information overload, ambiguity in prioritization criteria, coordination inefficiencies, and the absence of standardized workflows and tools. By systematically analyzing this end-to-end triage lifecycle, we aim to surface common patterns and pain points, critically evaluate existing solutions, and identify opportunities for intelligent automation, human-in-the-loop augmentation, and continuous process improvement.

3.2 Paper Collection

Our paper collection process includes two steps: keyword searching and snowballing.

3.2.1 Keyword Searching. To ensure a comprehensive and rigorous foundation for this survey, we adopted a systematic literature collection methodology centered on the DBLP Computer Science Bibliography [1], a widely recognized and authoritative source in the field of computer science. DBLP provides extensive bibliographic metadata for major conferences and journals, and has been frequently used in prior surveys on software engineering and related domains [28, 183, 184]. Given its broad coverage, we selected DBLP as our primary indexing platform, noting that papers indexed by other scientific databases (*e.g.*, Google Scholar, arXiv) are generally a subset of those available via DBLP [183].

We began by identifying a set of high-impact, peer-reviewed publication venues across several relevant areas, including software engineering, artificial intelligence, data mining, and computer systems. Specifically, our selection comprised 13 conferences (*e.g.*, ICSE, ASE, ESEC/FSE, AAAI, SIGKDD, SIGCOMM, INFOCOM, NDSS, WWW, ISSRE, CSCW, ICDE, and CIKM) and 4 journals (*e.g.*, TSE, TOSEM, TKDE, and JSS). To ensure consistency, three authors independently reviewed the titles, abstracts, and introductions of papers published in these venues, jointly refining a set of search keywords through collaborative discussion. The finalized search query was defined as: *("Triage") AND ("Incident" OR "Bug" OR "Alert")*. During the search process for bug triage studies, we observed that most papers [2, 9, 88, 120] treat "bug triage" as synonymous with "bug assignment", focusing primarily on developer recommendation tasks. To ensure comprehensive coverage of this subdomain, we therefore extended our query to also include the keyword *("Bug Assignment")*. In parallel, as an emerging field, incident triage has an unclear definition, and there are relatively few papers with it as a keyword. Therefore, we also introduced the following search query to incorporate incident management into consideration, which is *("Incident Diagnosis") AND ("Incident Management ")*. In addition, since some triage-related papers [26, 27, 33, 60] regard review-related methods as customer input, we additionally used the search query *("user review") AND ("user comments ")* to search for relevant papers.

We then conducted iterative searches over a 20-year publication window, manually screening all retrieved papers for relevance. This process resulted in an initial corpus of 195 candidate studies deemed pertinent to the scope of triage in software development and operations.

3.2.2 Snowballing Strategy. To augment our initial dataset and reduce the risk of publication bias, we employed a snowballing strategy, as recommended by Wohlin et al. [163]. This involved both backward snowballing (examining references cited by the initial studies) and forward snowballing (identifying studies that cited the initial corpus). In each iteration, we applied a consistent set of criteria to ensure that only papers directly relevant to our research scope were retained.

This iterative snowballing process continued until theoretical saturation was reached, that is, no additional relevant papers were identified in subsequent rounds. Through this method, we identified and incorporated 51 additional studies, resulting in a final corpus of 246 peer-reviewed papers encompassing a broad range of triage-related research.

3.2.3 Quality Assessment. To ensure the validity and reliability of our findings, we conducted a structured quality assessment of all candidate studies, in line with best practices for systematic literature reviews [82]. Each study was evaluated based on a predefined checklist across three critical dimensions [66, 69]:

(1) **Data Transparency:** Whether the paper clearly describes its dataset(s), including collection methods, characteristics, and availability.
(2) **Methodological Rigor:** Whether the proposed model, framework, or approach is described in sufficient technical detail to allow reproducibility.
(3) **Evaluation Clarity:** Whether the evaluation procedure is robust, and whether results are reported in a transparent and interpretable manner.

Two authors independently assessed the 246 studies identified via keyword search and snowballing. In cases of disagreement, a third author facilitated resolution through discussion and consensus. Based on this process, 12 studies were excluded due to insufficient methodological clarity or reporting rigor, and the remaining 234 studies were included in the final corpus for analysis.

3.3 Publication Trend and Distributions

In total, we collected 234 studies related to triage in the context of software engineering and operations. Figure 2a presents the histogram of annual papers. As illustrated, there has been a consistent upward trajectory in research activity, with a particularly marked increase in recent years. This trend underscores the rising importance of triage in managing the complexity, scale, and dynamism of modern software systems, thereby reaffirming the timeliness and relevance of this survey.

To gain further insight into how triage-related research is disseminated across the academic landscape, we analyzed the distribution of publications by venue. As shown in Figure 5, the majority of the selected studies have been published in leading software engineering venues, such as ICSE, ESEC/FSE, ASE, ISSRE, TSE, and the JSS. These venues have served as primary forums for advancing triage techniques, particularly in the areas of bug report classification, fault prioritization, and incident handling workflows. Beyond the domain of software engineering, triage research has also garnered increasing attention in adjacent venues, including data mining (*e.g.*, SIGKDD), artificial intelligence (*e.g.*, AAAI), and computer networks (*e.g.*, SIGCOMM).

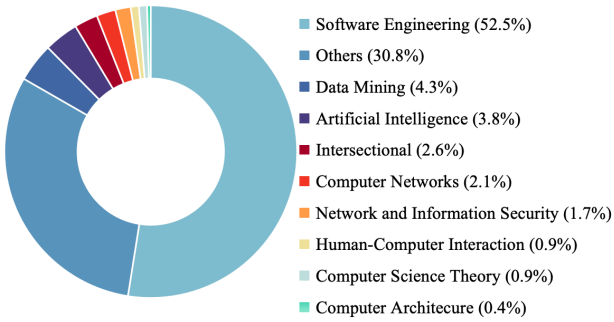


Fig. 5. Publication distribution of distinct venues.

These patterns indicate that triage is inherently multidisciplinary, integrating insights from software engineering, artificial intelligence, data mining, and distributed systems. The broad range of publication venues highlights its wide applicability and cross-domain relevance, spanning both theoretical advancements and practical implementations.

We further investigated the evolution of academic–industrial collaborations in triage research. As depicted in Figure 2b, such collaborations were nearly nonexistent before 2008 and began to emerge gradually in the following years. A noticeable upward trajectory became evident after 2014, with the number of joint studies increasing alongside the overall volume of triage-related publications. The most pronounced surge occurred between 2019 and 2020, marking the peak of industry collaborations within the field. Despite subsequent fluctuations in publication counts, the proportion of industry-affiliated research has remained consistently significant, reflecting the increasing practical orientation and applied relevance of triage research in contemporary software engineering.

This growing trend underscores the strong alignment between academic inquiry and industrial needs, demonstrating that triage is not only a subject of theoretical interest but also a practical necessity for real-world systems. The sustained presence of industry collaborations further highlights the demand for scalable, actionable, and deployable triage solutions capable of supporting complex production environments.

4 Review from Lifecycle Perspectives

In this section, we organize the collected papers from the perspective of the triage lifecycle. Figure 1 presents the common tasks throughout the triage lifecycle.

4.1 Data Processing

Data processing constitutes a critical stage within the triage pipeline, transforming raw system data into a structured representation that facilitates subsequent diagnostic analysis. Core tasks in this stage include data deduplication and feature extraction. This section surveys prior studies on processing methodologies, with a particular focus on these key tasks. Given the heterogeneity of data sources, spanning differences in structure, granularity, and reliability, existing studies adopt diverse data processing pipelines. Accordingly, we organize the review by primary data source and summarize the corresponding processing strategies and methodological choices within each category.

4.1.1 Deduplication. In large-scale enterprises, report processing systems generate thousands of reports daily, creating substantial challenges for triage engineers [148]. A significant proportion of these reports are repetitive and often

refer to the same defect. By enabling collective processing, the aggregation of such duplicate reports can markedly improve the efficiency of classification engineers. The report duplication approach seeks to identify reports that refer to identical bugs, cloud incidents, or system defects, allowing engineers to focus on resolving underlying issues rather than spending effort on redundant inspections.

Incident Tickets. In cloud computing environments, service incidents pose significant risks to both customer satisfaction and business revenue. A major challenge arises from the distributed and uncoordinated nature of incident reporting, often leading to redundant or duplicate tickets [98]. Consequently, the effective aggregation of related reports has become a prerequisite for efficient incident triage. Existing studies address this challenge through three main perspectives: offline correlation and clustering, information extraction for data filtering, and online graph-based aggregation.

(1) *Offline Correlation and Clustering:* Early studies explored the use of statistical and semantic associations to cluster related incidents. Ding et al. [44] established a framework for mining correlated events through concept-based clustering and similarity retrieval. Subsequent work, such as LinkCM [60], introduced transfer learning and semantic encoding to improve incident linkage across large-scale systems. LiDAR [32] further incorporated structural information from component dependency networks, extending beyond purely textual similarity. More recently, iPACK [98] integrated multiple data sources, customer tickets, and system incident logs, to achieve more accurate aggregation of duplicate reports. These studies collectively demonstrate a gradual evolution from static similarity-based clustering to multi-source semantic correlation for improved incident triage.

(2) *Information Extraction for Valid Data:* Another filtering research line focuses on extracting discriminative information from service tickets to filter relevant data before aggregation. Lou et al. [102] introduced a rule-based process for identifying effective attribute combinations from customer work orders. With advances in large language models (LLMs), TixFusion [148] automates this process through iterative extraction of key operational and anomaly-related features from textual tickets, enhancing aggregation precision. Together, these studies highlight a shift from manual feature engineering toward intelligent and adaptive data understanding.

(3) *Online Graph-Based Aggregation:* To overcome the latency of offline approaches, GRLIA [34] proposes an unsupervised graph learning framework for real-time aggregation of correlated incidents. By encoding both topological and temporal dependencies among cascading failures, GRLIA supports scalable and adaptive triage in dynamic environments.

Bug Reports. Within the software development lifecycle, especially during the development and maintenance phases, bug reports serve as a major diagnostic resource. Bug Tracking Systems (BTS) such as Bugzilla and JIRA function as centralized repositories that accumulate vast amounts of historical data [141]. Large-scale open-source projects like Mozilla or Eclipse may contain hundreds of thousands of reports, resulting in considerable redundancy and overlapping information. The unstructured nature of textual descriptions and the variability of reporter expertise further complicate automated analysis.

To address these challenges, researchers have explored various preprocessing strategies to improve the efficiency of downstream triage and duplicate detection tasks [131]. Early studies primarily focused on dataset reduction to eliminate redundant or noisy information. Zou et al. [195] applied a joint feature and instance selection approach to construct smaller yet more effective training sets, demonstrating that the order of these two reduction phases significantly affects classification performance. Building on this idea, Xuan et al. [172] employed a binary classifier to automatically determine the optimal application order of instance and feature selection, thereby enhancing the quality of the reduced datasets used in triage.

More recent work has shifted attention toward feature extraction and representation optimization for duplicate detection. Neysiani et al. [114] proposed an approach that aggregates unigram and bi-gram textual features with term frequency and inverse document frequency, introducing a hybrid metric to evaluate feature effectiveness and construct a compact yet informative feature set.

Collectively, these studies highlight an evolution from dataset reduction to refined feature engineering, emphasizing the critical role of preprocessing in improving the accuracy and scalability of automated bug report analysis.

Alerts. Alerts play a crucial role in maintaining the reliability of modern online service systems by providing early warnings of potential failures. However, due to the intricate interdependencies among system components, a single fault can trigger a large number of cascading alerts, which often overwhelm traditional operational workflows and render manual triage infeasible [29]. Consequently, the automated aggregation and summarization of alerts have become critical to reducing redundancy and enhancing operational efficiency.

(1) *Graph-Based and Statistical Summarization:* Early studies focused on graph-based and statistical summarization to mitigate alert storms. NoDoze [67] constructs causal dependency graphs of alerts, assigns anomaly scores according to historical frequency, and propagates them through network diffusion to identify representative subgraphs for triage. Similarly, AlertRank [192] conducts an empirical study of large-scale alert storms and combines accurate storm detection with summarization to recommend representative alerts for analysts.

(2) *Deep Representation Learning:* Subsequent research introduced deep representation learning to improve correlation detection among alerts. Warden [89] enhances incident management by automatically grouping correlated alerts and enabling proactive failure detection in large-scale cloud environments. Building on this idea, OAS [29] extracts both semantic and behavioral features from raw alerts and fuses them through a deep learning model to determine inter-alert correlations for effective summarization.

(3) *Large Language Models:* More recent approaches leverage LLMs to integrate contextual reasoning and external operational knowledge. COLA [84] exemplifies this trend by combining correlation extraction with LLM-based reasoning, guided by standard operation procedures (SOPs) as auxiliary knowledge for online alert aggregation.

Despite their progress, these methods remain constrained by a heavy reliance on supervised training. The requirement for extensive, manually labeled data imposes substantial operational costs and limits its generalization to dynamic or data-scarce environments.

Reviews. After existing triage methods generate initial results, users or engineers often provide feedback to refine system performance [33]. Such feedback can enhance the accuracy and reliability of incident resolution, yet its practical use is challenged by the noisy and unstructured nature of user comments. In particular, app reviews are typically short and contain a large proportion of irrelevant information; only about 30% of them offer actionable insights for app improvement [31]. Consequently, effective techniques for filtering and prioritizing informative feedback have become an essential research focus.

Early work, such as AR-Miner [31] established a foundational framework for extracting valuable user opinions from large-scale reviews. The approach integrates four key stages: filtering noisy or irrelevant reviews, applying topic modeling to cluster similar feedback, ranking reviews based on informativeness, and visualizing the top-ranked items to support developers' decision-making. Building upon this line, PAID [53] advances the granularity of analysis from reviews to phrases. It extracts and prioritizes key phrases through rule-based selection and maintains a Phrase Bank for developers. To capture the temporal dynamics of user concerns, PAID further applies Dynamic Latent Dirichlet Allocation (dLDA) to model topic evolution across app versions and recommend the most relevant phrases for each identified topic. More recently, IFeedback [194] extends feedback analysis toward automated fault detection. It constructs

Word Combination-based Indicators (WCIs) by pairing words in feedback text and filters them with historical data to retain those most indicative of system faults. These indicators serve as metrics for identifying potential issues in real time.

Overall, the evolution from AR-Miner to PAID and IFeedback reflects a progressive refinement of feedback analysis, from coarse-grained filtering of informative reviews to fine-grained phrase extraction and automated fault detection. These methods collectively aim to enhance the interpretability and utility of user feedback in guiding software maintenance and evolution.

Table 2. Summary of data deduplication methods in triage oprations based on their category, the year, and the data. The abbreviations in the "Data" column stand for: **A** = Alerts, **R** = Reviews, **IT** = Incident Tickets, **L** = Logs, **KPI** = Key Performance Indicators, **DN** = Dependency Networks, and **BR** = Bug Reports.

Category	Technique	Year	Data	Core Method
Incident Tickets	Ding et al. [44]	2014	IT	FCA + GVSM
	LinkCM [60]	2020	IT	Symmetric Model + BERT + Decomposable Attention Mechanism
	LiDAR [32]	2020	IT, DN	TextCNN + Node2Vec
	iPACK [98]	2023	IT	Incident-Aware Framework
	Lou et al. [102]	2017	IT	Extracting Attribute Combinations
	TixFusion [148]	2025	IT	LLM
	GRLIA [34]	2021	IT, DN, KPI	Graph Representation Learning
Bug Reports	Zou et al. [195]	2011	BR	Training Set Reduction Approach
	Xuan et al. [172]	2014	BR	Instance Selection + Feature Selection + Binary Classifier
	Neysiani et al. [114]	2020	BR	Term Frequency + Inverse Document Frequency
Alerts	NoDoze [67]	2019	A	Causal Dependency Graphs + Network Diffusion
	AlertRank [192]	2020	A	Accurate Storm Detection
	Warden [89]	2021	A	Automated Alert Grouping + Proactive Failure Detection
	OAS [29]	2022	A	Deep Learning Model
	COLA [84]	2024	A, SOP	Integrates Correlation Extraction + LLM-Based Reasoning
Reviews	AR-Miner [31]	2014	R	Topic Modeling + Ranking Scheme + Intuitive Visualization Approach
	PAID [53]	2015	R	Rule-Based Filtering Strategy + Grouping-Based Ranking Strategy + dLDA
	IFeedback '[194]	2019	R	WCIs + Rule-Based Filtering Strategy

4.1.2 Feature Extraction. Reports serve as dense, multimodal data sources, amalgamating structured fields like timestamps with unstructured, free-form text such as failure descriptions and operator annotations. The inherent verbosity and heterogeneity of this content pose a significant challenge to automated analysis. Consequently, the ability to accurately and efficiently distill actionable diagnostic information from these reports is paramount for effective triage and subsequent resolution.

Incident Tickets. Incident processing methods have progressively evolved from text-based analysis to multimodal integration, requiring different feature extraction strategies to address the increasing diversity of operational data.

(1) *Textual and Sequential Incident Analysis:* Early approaches focused on mining textual and sequential patterns from historical incidents. Shao et al. [140] employed probabilistic sequence modeling to discover resolution patterns that supported expert recommendation, while the IcM BRAIN framework [33] introduced AI-based processing of incident tickets and customer inputs to improve triage and correlation efficiency in large-scale production environments.

(2) *Semantic and Structural Modeling:* As incident data grew in scale and complexity, textual analysis alone became insufficient to capture the contextual and relational characteristics of service failures. Consequently, later research emphasized semantic and structural modeling to represent incidents in more meaningful and interconnected ways. Triangle [180] addressed this need by enhancing key information extraction through semantic alignment between incident descriptions and domain documents, thereby improving triage accuracy and contextual consistency. Building

on this direction, COT [159] further extended semantic modeling to a structural level by constructing correlation graphs that captured dependencies among incidents and services.

(3) *Multimodal and LLM-Driven Approaches:* With the growing availability of heterogeneous operational data such as metrics, logs, traces, and textual reports, research has increasingly shifted toward multimodal fusion and LLM-based reasoning to achieve comprehensive situational awareness. FaultProfIT [71] initiated this direction by structuring textual tickets into key fields, facilitating more accurate context interpretation by LLMs. Building upon the integration of multiple information sources, DiLink [56] combined textual and structural embeddings within a unified framework to enhance multimodal representation learning. Extending this idea further, Goel et al. [57] applied LLMs in combination with vector-based retrieval to improve incident summarization and historical similarity matching.

(4) *Visual-Text Fusion in Ticket Understanding:* In addition, Mandal et al. [105] explored visual-text fusion by processing screenshots attached to service tickets, combining image and text extraction to enhance entity recognition and context understanding.

Overall, these methods illustrate a clear progression from early sequence-based and textual processing toward semantically enriched, multimodal, and LLM-empowered approaches, reflecting the ongoing shift toward comprehensive and context-aware incident feature extraction.

Bug Reports. In bug triage, feature extraction from both unstructured textual content and structured metadata plays a pivotal role in determining accurate developer assignment. Bug reports typically contain summaries, detailed descriptions, and error traces as unstructured text, together with structured fields such as product, component, severity, and version.

(1) *Text-Based Approaches:* Early studies primarily adopted statistical classifiers, among which the Naive Bayes (NB) model was the most widely used. Murphy et al. [110] applied a Naive Bayes classifier to bug report text for developer prediction, but its performance was constrained by the assumption of word independence and heuristic features. To overcome these limitations, Xuan et al. [173] enhanced NB by integrating Expectation–Maximization (EM) and a Weighted Recommendation List, enabling the exploitation of unlabeled data for improved accuracy. Alenezi et al. [6] further combined NB with Chi-Square–based term selection and load-balancing strategies, although the proposed system lacked empirical validation. With the rise of semantic representation learning, subsequent research shifted toward richer text modeling. Panda et al. [118] transformed unstructured reports into developer–topic relations via LDA topic modeling and applied Intuitionistic Fuzzy Sets to capture uncertainty in developer expertise. Wang et al. [156] treated bug assignment as a text classification task, systematically comparing combinations of word-embedding models (*e.g.*, Word2Vec, GloVe, ELMo, BERT) and deep classifiers (*e.g.*, LSTM, Bi-LSTM with attention, TextCNN) for predicting the most suitable developers.

(2) *Metadata-Driven and Code-Driven Approaches:* Recognizing the limitations of purely textual information, later work explored structured and code-related features. Park et al. [121] enhanced triage for non-textual bug reports by computing import-path similarities, using Jaccard and tree-edit distances, to infer bug types and maintain developer profiles via time-decayed repair history. Sun et al. [147] proposed EDR_SI, which incorporated developer habits and experiences through Collaborative Topic Modeling on historical commits, extracting personalized developer–file associations. Xu et al. [171] designed a Crash Bug Component Prediction (CBCP) model that mapped crash call-stack functions to components, computing IDF-based statistics to train a Random Forest for component-level fault localization.

(3) *Multi-Field and Contextual Feature Fusion:* Beyond text and code, several studies investigated additional report fields to enhance feature diversity. Sarkar et al. [137] empirically demonstrated that alarm logs and crash dumps did not improve triage accuracy on Ericsson's dataset, and introduced a confidence-threshold mechanism for selective

high-confidence predictions. Shokripour et al. [142] proposed a time-aware TF-IDF method that emphasized temporally relevant terms to better match developers' historical expertise. Sajedi et al. [135] refined textual input using a Stack Overflow–based technical vocabulary, retaining only domain-specific terms and weighting developer expertise by recent activity. Nath et al. [113] combined discrete and textual features through PCA and entropy-based keyword filtering, using probabilistic multi-labeling to reflect team-level expertise. Li et al. [91] conducted a large-scale empirical analysis revealing that traditional textual features can degrade classifier performance due to noisy content such as code snippets and stack traces. Building on this, Li et al. [90] found that nominal features, such as reporter, component, and priority, capture developer preferences more effectively than advanced NLP-derived text embeddings.

Observability Data. Logs provide chronological records of system events and serve as essential evidence for fault detection and diagnosis. Early systems such as SAS [101] automated incident analysis by correlating heterogeneous logs and key performance indicators to generate reports and recommendations, supporting incident triage through human-in-the-loop decision-making. Subsequent studies have advanced this line of research by leveraging large language models for semantic understanding and retrieval. COMET [160] exemplifies this trend through an LLM-enhanced pipeline that transforms raw logs into compact representations and retrieves semantically similar historical incidents to assist fault management and team prediction. Moreover, ART [149] fuses metrics, logs, and traces into unified temporal representations, capturing both cross-modal and temporal dependencies.

Overall, these approaches illustrate a progression from early feature-driven systems to semantically enriched, LLM-augmented frameworks that improve automation and interpretability in incident analysis.

Reviews. User reviews provide essential insights into user needs and product evolution, making their systematic extraction and interpretation crucial for informed software maintenance. Research in this field has evolved from explicit linkage construction to semantic understanding and automated analysis.

CRISTAL [117] initiated this direction by linking user reviews with issues and commits, enabling traceability between feedback and development actions. Wang et al. [157] extended this idea by quantifying these relationships through topic-based modeling, capturing how user requests and feature updates co-evolve over time. Building on these foundations, ALLHANDS [182] advances toward intelligent feedback understanding by leveraging large language models to transform unstructured reviews into structured semantic representations.

Collectively, these studies reflect a methodological progression from linking feedback to interpreting its semantic implications, highlighting the growing sophistication of user review analysis in supporting adaptive software evolution.

Relational Data. In addition to individual bug reports, relational data encodes the historical and contextual interactions among bugs, developers, and components, providing a structural foundation for enhancing bug triage accuracy. A range of approaches have leveraged these relations to extract informative features and represent developer expertise more effectively.

(1) *Constructing Relational Networks:* One major line of research focuses on constructing relational networks that explicitly capture the connections among developers, components, and bugs. BugFixer [70] models such relations through a Developer–Component–Bug network, integrating network-derived associations with textual similarity to jointly represent structural and semantic information for developer recommendation. Building upon this idea, KSAP [189] formulates a heterogeneous network and employs meta-path–based relational extraction to model collaboration patterns among developers. By coupling these structured relational features with nearest-neighbor search on historical reports, it strengthens the alignment between textual relevance and collaborative context.

(2) *Representation Learning on Interaction Graphs:* While these methods rely on handcrafted relational structures, subsequent studies emphasize representation learning on interaction graphs, enabling more flexible and expressive

modeling of complex dependencies. PCG [38] embeds bugs and developers into a unified vector space, where prototype clustering and semantic contrastive learning jointly capture explicit and implicit relational cues. NCGBT [47] extends this representation paradigm by introducing a bipartite graph formulation and neighborhood contrastive learning, thereby enhancing local structural consistency while preserving semantic proximity. Complementarily, BPTRM [162] exploits tossing relationships between developers to construct a personalized collaboration graph. By encoding these interaction patterns together with bug content features, it refines developer representations in a way that aligns historical behavior with current bug requirements.

(3) *Temporal Dynamics:* A significant research direction involves incorporating temporal dynamics into the analysis of developer collaboration to model its evolution. For instance, Wu et al. [164] jointly model structural and temporal dependencies through a graph recurrent convolutional network that encodes developers' dynamic interaction preferences. Similarly, GCBT [39] unifies spatial and temporal graph convolutions to extract both static and evolving expertise representations, with bug embeddings initialized via pretrained language models to ensure semantic consistency across tasks.

By combining these perspectives, relational data–driven approaches provide a comprehensive understanding of developer–bug interactions, leading to more accurate and context-aware bug triage.

Table 3. Summary of extracting feature from data in triage operations based on their category, the year, and the data. The abbreviations in the "Data" column stand for: **R** = Reviews, **IT** = Incident Tickets, **L** = Logs, **KPI** = Key Performance Indicators, **DN** = Dependency Networks, **CC** = Code Commits, **BR** = Bug Reports, **SC** = Source Code, **ST** = Stack Traces, and **MD** = Metadata (*e.g.*, historical developer assignments, comments).

Category	Technique	Year	Data	Core Method
Incident Tickets	Shao et al. [140]	2008	IT	Variable-Order Markov + VMS Search
	IcM BRAIN framework [33]	2020	IT, R	AI-Based Techniques
	Triangle [180]	2021	IT, KPI	Semantic Alignment + TF-IDF + Team Information Enrichment Mechanism
	COT [159]	2021	IT	Historical Meta-Incident Construction + Correlation Graph Construction
	FaultProfIT [71]	2024	IT	Regex/Parsers + MacBERT
	DiLink [56]	2024	IT, DN	TF-IDF + LSTM + Node2Vec + Graph Attention Networks + Orthogonal Procrustes Alignment
	Goel [57]	2024	IT, KPI	GPT-3.5-turbo + FAISS
	Mandal et al. [105]	2019	IT	Contour Detection + Canny Edge Detection + ResNet50 + CNN + Tesseract
Bug Reports	Murphy et al. [110]	2004	BR	Naive Bayes Classifier
	Xuan et al. [173]	2010	BR	Naive Bayes Model + Expectation-Maximization + Weighted Recommendation List
	Alenezi et al. [6]	2013	BR	Chi-Square Test + Naive Bayes Classifier
	Panda et al. [118]	2022	BR	LDA Topic Modeling + Intuitionistic Fuzzy Sets
	Wang et al. [156]	2024	BR	Word2Vec + GloVe + NextBug + ELMo + BERT + LSTM + Bi-LSTM + TextCNN
	Park et al. [121]	2016	BR, SC	Import-Path Similarities
	Sun et al. [147]	2017	BR, CC	Collaborative Topic Modeling
	Xu et al. [171]	2023	BR, ST	Crash Bug Component Prediction Model + IDF + Random Forest Model
	Sarkar et al. [137]	2019	BR	Logistic Regression + Confidence-Threshold Mechanism
	Shokripour et al. [142]	2015	BR	Time-Aware TF-IDF + Ranking
	Sajedi et al. [135]	2020	BR	Stack Overflow-Based Vocabulary Filtering
	Nath et al. [113]	2021	BR	PCA + Entropy-Based Keyword Selection + Contribution-Weighted Probability Labeling
	Li et al. [91]	2021	BR	VSM + TF-IDF
	Li et al. [90]	2024	BR	TextCNN + SVM Classifiers

Continued on next page

Table 3 – *Continued from previous page*

Category	Technique	Year	Data	Core Method
Observability Data	SAS [101]	2013	L, KPI	Data-Driven Techniques
	COMET [160]	2024	L	Multi-Stage Processing Pipeline + GPT + FastText
	ART [149]	2024	L, M	Transformer Encoder Self-Attention + GRU + GraphSAGE
Reviews	CRISTAL [117]	2015	CC, R	Issue and Commit Extractor + Information Retrieval Techniques
	Wang et al. [157]	2017	R, L	Biterm Topic Model
	ALLHANDS [182]	2025	IT, R	Regular Expressions + HTML Parsers + LLM-Based Classification + Abstractive Topic Model
Relational Data	BugFixer [70]	2014	BR, MD	Developer-Component-Bug Network + Textual Similarity Search
	KSAP [189]	2016	BR, MD	Developer-Component-Bug Network + K-Nearest-Neighbor Search
	PCG [38]	2024	BR, MD	Bug–Developer Interactions Graph + Semantic Contrastive Learning
	NCGBT [47]	2025	BR, MD	Bug–Developer Interactions Graph + Neighborhood Contrastive Learning
	BPTRM [162]	2025	BR, MD	Bug–Developer Interactions Graph
	Wu et al. [164]	2022	BR, MD	Developer Collaboration Networks + JRWalk + GRCNN
	GCBT [39]	2023	BR, MD	Bug–Developer Interactions Graph + IR-Based Classifier

4.2 Prioritization

Prioritization in triage refers to the systematic process of ranking or ordering issues such as software bugs, system alerts, or performance anomalies according to predefined criteria, including severity, urgency, cost, or relevance. It plays a vital role in enabling efficient resource allocation and ensuring timely issue resolution within software engineering and IT operations. In software development and system maintenance, triage entails analyzing incoming data to identify, classify, and delegate issues to appropriate personnel or automated procedures. Prioritization determines the order in which issues are addressed, allowing critical problems to be resolved promptly, which helps minimize downtime, control operational costs, and enhance user satisfaction.

The effectiveness of prioritization lies in its ability to optimize resource utilization, such as developer time or computing capacity, while improving system reliability and operational efficiency. Its importance becomes particularly evident in large-scale systems, where the volume and heterogeneity of incoming issues can easily exceed manual processing capabilities. Sound prioritization strategies address key challenges such as sparse data, information overload, and diverse issue types. This section synthesizes prior research on prioritization in triage systems, with a focus on categorizing studies based on the output data types.

4.2.1 Severity Rating. In both software development and cloud incident management, severity serves as a critical criterion for classifying and prioritizing bugs and incidents according to their potential impact and the urgency of remediation. Clearly defined severity levels enable development teams, testers, operators, and other stakeholders to systematically assess the significance of addressing specific faults and to allocate resources accordingly.

Incident Tickets. Effective incident prioritization aims to distinguish critical events from incidental or low-impact ones, thereby enabling timely and resource-efficient responses.

Existing studies approach this challenge from complementary perspectives that progressively enhance interpretability, contextual awareness, and automation. DeepIP [30] addresses the problem from a data-driven learning perspective. It employs a neural model to identify incidents that are likely incidental and ranks all reported events according to their predicted relevance, thereby supporting more focused remediation. Building on this concept of data-guided prioritization, Saurabh et al. [104] incorporate multi-criteria features and expert knowledge into the prioritization

process. Their framework integrates statistical and linguistic indicators through a heuristic weighting scheme and refines the prioritization through regression modeling validated by domain experts, thus enhancing the interpretability and reliability of the results. Extending these approaches toward richer contextual reasoning, Sadlek et al. [132] introduce an attack-graph-based framework that evaluates incident severity in relation to potential kill-chain progressions and asset criticality. By embedding prioritization within a structured representation of cyberattack sequences, this method captures interdependencies among alerts and enables severity assessment grounded in operational context.

Bug Reports. A substantial body of research has focused on automating bug severity and priority prediction by leveraging both textual content and metadata from bug reports [5, 37]. These studies can be broadly divided into two complementary lines: (1) severity estimation and prioritization, which infer the relative importance of bugs, and (2) severity-aware triage optimization, which integrates severity signals into task assignment and scheduling.

(1) *Severity Estimation and Prioritization:* Within the first line of work, several studies adopt classification and similarity-based learning to infer severity levels. Kanwal et al. [80] utilized supervised classifiers such as SVM and Naïve Bayes to quantify the contribution of different bug report features and establish evaluation metrics for automated prioritization. Zhang et al. [185] further incorporated historical case similarity, combining feature-based learning with K-nearest retrieval to improve the reliability of severity prediction and fixer recommendation.

(2) *Severity-Aware Triage Optimization:* Building on these predictive foundations, subsequent research integrates severity inference into broader triage and scheduling frameworks. Jahanshahi et al. [75] reconstructed historical bug dependency graphs to analyze how severity interacts with bug interdependencies and developer workloads, providing dynamic evaluation metrics for severity-driven triage performance. Extending this idea, their later work [73] introduced a schedule- and dependency-aware framework that models bug-developer assignment as an optimization problem, incorporating severity, dependency, and workload constraints to improve triage efficiency.

Recent studies have enhanced these severity prediction pipelines through deeper semantic modeling and uncertainty handling. Arshad et al. [11] employed a transformer-based architecture to capture contextual and sentiment information within bug reports, enabling semantic reasoning for real-time severity prediction. Panda et al. [119] further addressed uncertainty and class imbalance by integrating topic modeling with intuitionistic fuzzy representations, allowing the model to express multi-level priority associations with soft, probabilistic interpretations.

Collectively, these studies reveal a coherent methodological landscape in which severity estimation serves as a foundational signal for downstream triage optimization. Traditional classification and similarity learning establish interpretable feature mappings, dependency- and schedule-aware frameworks operationalize severity within realistic coordination constraints, and recent transformer- and fuzzy-based models extend this pipeline toward context-sensitive and uncertainty-resilient prioritization.

Alerts. In the context of alert prioritization, research has progressively advanced from probabilistic modeling to feature-driven learning frameworks. Lin et al. [95] proposed CAR, which captures temporal and content correlations among heterogeneous alerts through a hierarchical probabilistic framework, enabling the ranking of both individual alerts and their underlying patterns. Subsequently, Zhao et al. [192] developed AlertRank, which integrates textual, temporal, and anomaly-related information into a unified feature representation and applies a learning-to-rank model to assign severity scores. Together, these methods reflect a shift from statistical dependency modeling toward interpretable, data-driven approaches for effective alert prioritization.

Table 4. Summary of severity rating based on their category, the year, and the data. The abbreviations in the "Data" column stand for: **A** = Alerts, **R** = Reviews, **IT** = Incident Tickets, **L** = Logs, **KPI** = Key Performance Indicators, and **BR** = Bug Reports.

Category	Technique	Year	Data	Core Method
Incident Tickets	DeepIP [30]	2021	IT	Attention-Based Convolutional Neural Network
	Sadlek et al. [132]	2025	IT, L, KPI	MulVAL Attack Graph Generator + MITRE ATT &CK Techniques
	Saurabh et al. [104]	2022	IT, R	Regression-Based Prioritization Technique + Weighted Heuristic
Bug Reports	Kanwal et al. [80]	2012	BR	SVM + Naive Bayes Classifier
	Zhang et al. [185]	2016	BR	REP Algorithm + K-Nearest Neighbor Classification
	Arshad et al. [11]	2024	BR	Fine-Tuned Transformer Model
	Panda et al. [119]	2024	BR	Intuitionistic Fuzzy Sets + Topic Modeling
	Jahanshahi et al. [75]	2022	BR	Bug Dependency Graph + Developer Load Tracking
	Jahanshahi et al. [73]	2022	BR	Schedule and Dependency-Aware Bug Triage + SVM/LDA + Integer Programming
Alerts	CAR [95]	2018	A	Hierarchical Bayesian + Entity Embedding-Based Approach
	AlertRank [192]	2020	A, KPI	Feature Set + XGBoost Ranking

4.2.2 Issue Type Classification. In triage, accurately distinguishing between recurring issues and previously unseen problems is crucial, as it enables the reuse of established repair procedures for known issues or the initiation of in-depth investigation for unknown ones. Moreover, the ability to identify anomalous reports that are directly associated with actual faults from a large volume of alerts, particularly during alert floods, allows engineers to prioritize the most critical incidents. This targeted focus not only reduces cognitive load but also enhances the efficiency and timeliness of fault response. Furthermore, such prioritization plays a key role in mitigating operational risks, preventing service degradation, and optimizing resource allocation in large-scale systems.

Structure Information. For algorithms with low dependency on historical data, unsupervised and lightweight supervised techniques are commonly employed to classify issue types. These methods can be broadly categorized into clustering-based anomaly grouping and classification-based issue typing, reflecting the evolution from pattern discovery to semantically guided categorization.

(1) *Clustering-Based Methods:* Early efforts emphasized unsupervised grouping of anomalies and updates based on statistical and structural similarity. Lim et al. [92] introduced a Hidden Markov Random Field combined with EM-based clustering (HMRF-kMedoid-EM) to detect performance anomaly categories. The model discretizes performance metrics, identifies salient attributes, and supports systematic triage through cluster-driven anomaly interpretation. Lin et al. [93] improved clustering precision by detecting connected components and recursively refining large clusters via graph cuts, followed by Non-negative Matrix Factorization (NMF) for dimensionality reduction and hierarchical grouping through KD-tree and linkage clustering. Wang et al. [157] applied k-means to cluster update activities (AUs), using Euclidean similarity and average-distance variation to determine cluster numbers. Each centroid represented an Update Pattern reflecting the evolution of feature demand and responsiveness. Later, Zhao et al. [191] extended clustering-based detection to streaming alerts, combining Extreme Value Theory (EVT) with Isolation Forest denoising and clustering to identify alert storms and select representative alerts, thus reducing diagnostic redundancy while preserving key signals.

(2) *Classification-Based Methods:* Subsequent studies shifted toward supervised and multi-label formulations to directly predict issue categories. Catolino et al. [25] established an automated bug classification framework grounded in a nine-class taxonomy (*e.g.*, Configuration, Network, GUI, Security), derived through iterative content analysis of 1,280 bug reports. Building on such taxonomy-driven approaches, Meher et al. [108] developed a deep learning framework that integrates expert-validated keyword sets for eight bug types and heuristically annotated 1.36 M reports using

Word Mover's Distance, enabling attention-based models (Transformer, BERT, CodeBERT, DistilBERT) to perform fine-grained classification. Aung et al. [12] proposed Multi-Triage, a multi-label model jointly predicting issue types and responsible developers. By separating text and code tokens and augmenting contextual data, it improves the robustness of imbalanced classification. Expanding the label scope, Sepahvand et al. [138] distinguished design-related from implementation-related defects via a CNN that fuses textual and code-smell features, effectively linking defect symptoms with design-level anti-patterns such as large classes or high complexity.

Overall, these studies demonstrate a clear methodological evolution, from unsupervised clustering for anomaly grouping toward deep and multi-label classification frameworks that incorporate semantic, contextual, and design-level cues for precise issue typing.

Historical Information The utilization of historical data is crucial for issue type classification, as it enables models to capture recurring patterns, contextual dependencies, and developer behavior over time, thereby improving the accuracy of issue categorization and supporting more consistent, data-driven decision-making in software maintenance processes.

In the early stage of incident and bug report analysis, research primarily focused on structural modeling and knowledge-driven reasoning. Xuan et al. [174] first leveraged developer social networks to rank contribution priorities and enhance bug triage and severity prediction. Building on structural similarity, Park et al. [121] addressed non-textual bug reports such as crash logs by computing code import path similarity using Jaccard or tree-edit distance and classifying reports with k-nearest neighbors. Further extending this direction, Zeng et al. [181] introduced Kilo, which encoded expert domain knowledge as hierarchical vectors within a probabilistic graphical model, enabling multi-label reasoning through a sum–product inference mechanism.

Subsequent work shifted toward representation learning and semantic similarity modeling to overcome the limitations of purely structural features. LinkCM [60] proposed a symmetric multi-instance model to learn effective representations from correlated system incidents, updating only fully connected layers with labeled CI–MI linking data for efficient triage. Haering et al. [65] further enhanced semantic linkage by encoding problem and bug reports into context-sensitive vector embeddings using DistilBERT and ranking their cosine similarity, thereby improving automatic bug report recommendation and linkage accuracy.

Recent studies have advanced toward hierarchical and large-model–based reasoning frameworks for complex failure understanding. ART [149] adopted Extreme Value Theory and cut-tree clustering to detect anomalies and assign failures by modeling system-level deviations, while FaultProfIT [71] utilized hierarchy-guided contrastive learning over MacBERT and Graphormer encoders to capture multi-level fault taxonomy. Building on the progress of large language models, ALLHANDS [182] integrated in-context learning with retrieval-based prompting and human-in-the-loop refinement for feedback categorization and topic modeling. Its LLM-based QA agent further decomposes analytical queries, generates executable Python code, and produces multimodal insights, signifying the transition toward intelligent, knowledge-grounded incident analysis.

Table 5. Summary of issue type classification based on their category, the year, and the core method.

Category	Technique	Year	Core Method
Structure Information	Lim et al. [92]	2014	Hidden Markov Random Field + EM-Based Clustering (HMRF-kMedoid-EM)
	Lin et al. [93]	2014	Non-Negative Matrix Factorization + KD-Tree
	Wang et al.[157]	2017	K-Means + Euclidean Distance
	Zhao et al. [191]	2020	Extreme Value Theory + Isolation Forest-Based Denoising

Continued on next page

Table 5 – *Continued from previous page*

Category	Technique	Year	Core Method
Structure Information (continued)	Catolino et al. [25]	2019	Taxonomy Construction + Classification Model
	Meher et al. [108]	2024	Taxonomy Construction + Attention-Based Classification Model
	Aung et al. [12]	2022	Contextual Data Augmentation + Multi-Label Classifying
	Sepahvand et al. [138]	2023	CNN-Based Model + PMD-Based Analysis
Historical Information	Xuan et al. [174]	2012	Social Network
	Park et al. [121]	2016	SVM + LDA Topic Modeling + Content-Based Recommendation + Content-Boosted Collaborative
	Kilo [181]	2017	Domain Expert Knowledge + Sum-Product Algorithm
	LinkCM [60]	2020	Symmetric Model + Fully-Connected Network
	Haering et al. [65]	2021	DistilBERT
	ART [149]	2024	Extreme Value Theory + Cut-Tree Clustering
	FaultProfIT [71]	2024	MacBERT-Based Incident Encoder + Graphormer-Based Hierarchy Encoder
	ALLHANDS [182]	2025	Multi-Task In-depth Analysis + Hierarchical Agglomerative Clustering + LLM-Based QA Agent

4.3 Assignment

In modern triage systems, the assignment of issues to appropriate components and developers can be partially or fully automated through predefined rules or machine learning–based algorithms. These assignment mechanisms typically leverage a combination of factors, including the issue's associated component, the historical workload distribution among developers, and prior assignment patterns. Effective component assignment ensures that the issue is directed to the correct functional module or subsystem, thereby enabling precise fault localization. Subsequent developer assignment further refines the process by selecting the most suitable individual or team based on domain expertise, prior experience with similar issues, and current availability.

Once an issue has been assigned, the triage system provides the assignee with comprehensive report details to facilitate efficient resolution. This information package generally includes the issue description, steps to reproduce the problem, relevant logs, screenshots, code snippets, and any other contextual data collected during the classification phase. Providing complete and well-structured report details is critical for ensuring that developers or maintenance teams gain an immediate and accurate understanding of the problem, thereby reducing the need for repeated clarification, minimizing handoff delays, and ultimately accelerating defect resolution in large-scale software systems.

4.3.1 Component Assignment. When assigning issues to components, triage teams typically evaluate the functional scope, historical fault patterns, and recent change history of each component. Components with a documented history of similar faults or recent code modifications are often prioritized for investigation, as they present a higher likelihood of being the fault source. In addition, the stability profile and maintenance ownership of a component are critical considerations; assigning issues to components actively maintained by dedicated teams can facilitate faster diagnosis and resolution. By aligning issue assignment with component expertise, change context, and operational responsibility, triagers can improve fault localization accuracy, reduce unnecessary cross-component investigation, and enhance the overall efficiency of defect management in large-scale software systems.

Text Classification. Early studies on text-based component assignment applied traditional machine learning and knowledge-engineering approaches, where statistical models and expert-defined rules were used to map event or report attributes to predefined categories [64]. With the rise of transformer architectures, research gradually shifted toward fine-tuning pre-trained language models, achieving consistent improvements over earlier neural or feature-based baselines while showing that traditional TF-IDF methods remain competitive in certain constrained settings [46]. More recent efforts emphasize industrial applicability. Borg et al. [21] propose a routing framework that assigns reports to

software modules rather than individuals and integrates confidence-based human oversight, balancing automation and reliability in large-scale maintenance workflows.

Information Retrieval. Beyond text classification, another common approach to component assignment is information retrieval, which locates historical cases or components relevant to a new report.

(1) *Similarity-Based Retrieval:* Early approaches primarily relied on textual similarity and statistical correlation to locate related incidents. SAS [101] detects incident beacons through anomaly analysis and probabilistic correlation, and retrieves recurring failures via signature-based matching and log-likelihood evaluation. Ding et al.[44] extend textual retrieval by constructing structured action triples ⟨verb, target, location⟩, combining semantic parsing of descriptions with log-based context extraction. CRISTAL[117] further enriches similarity computation by integrating textual information from commit notes and code identifiers, and computes asymmetric Dice similarity between reviews and issue reports. Together, these studies represent the foundation of retrieval-based triage, relying on handcrafted linguistic or statistical features to quantify incident similarity.

(2) *Representation Learning for Retrieval:* With advances in deep learning, representation learning has become central to retrieval accuracy and scalability. LiDAR [32] learns joint textual and component embeddings to generate unified vector representations, from which linkage confidence between incidents can be efficiently computed. Zhang et al.[188] employ a deep neural model that projects bug descriptions into discriminative latent spaces, balancing component frequency bias through class-weighted loss and enabling near real-time assignment. Beyond textual embeddings, Xu et al.[171] incorporate structured and behavioral features by encoding call-stack frequency and positional information, allowing machine learning models to predict fault-prone components and reduce reassignment delay. Collectively, these methods advance incident retrieval from surface-level similarity toward semantically grounded representation learning.

(3) *Hybrid and LLM-Enhanced Retrieval:* Recent work integrates retrieval-based correlation mining with large language model reasoning to enhance interpretability and adaptability. COLA [84] combines probabilistic and graph-based correlation mining, leveraging conditional probability, node2vec, and Skip-gram, with an LLM reasoning module based on two-round prompting, in-context learning, and parameter-efficient tuning for improved F1 performance. Goel et al. [57] further demonstrate the potential of LLMs by incorporating X-lifecycle data into GPT-4–based reasoning pipelines for root cause recommendation and monitor classification, integrating both event semantics and dependency metadata. These hybrid designs illustrate a shift toward intelligent retrieval systems that unify statistical correlation, learned representation, and natural language reasoning within a single framework.

Overall, the evolution of information retrieval methods reflects a clear trajectory, from lexical similarity to representation learning and finally to LLM-based reasoning, toward more adaptive, semantically rich, and context-aware incident triage.

Social Network Modeling. Following retrieval-based approaches, researchers have explored social and dependency structures to enhance triage effectiveness. Su et al. [146] leveraged a bug-tossing knowledge graph within a learning-to-rank framework, later extended by DEEPTRIAG [145] through deep ensemble modeling. At the service level, COT [159] employed graph-based reasoning to capture interactions among incidents and components. These efforts move triage modeling toward relational and graph-centric representations.

Data Bias Modeling. In parallel, some studies address the data bias and concept drift that emerge in continuously evolving bug report streams. Chrupala et al.[36] applied online learning algorithms to adapt component assignment models in real time, mitigating distributional shifts. Mandal et al.[105] further tackled data imbalance by combining ensemble classification with information retrieval, enabling robust resolution prediction across both frequent and

rare problem categories. These approaches underscore growing attention to temporal dynamics and long-tail effects, establishing data bias modeling as a complement to content- and structure-based triage.

Table 6. Summary of component assignment methods categorized by their problem formulation. The abbreviations in the "Category" column stand for: **TC** = Text Classification, **IR** = Information Retrieval, **SN** = Social Network Modeling, and **DB** = Data Bias Modeling. The abbreviations in the "Data" column stand for: **L** = Logs, **M** = Metrics, **IT** = Incident Tickets, **A** = Alerts, **R** = Reviews, **BR** = Bug Reports, **HD** = Historical Documents, **CC** = Code Commits, and **DN** = Dependency Networks.

Category	Technique	Year	Data	Core Method
TC	Gupta et al. [64]	2009	IT, L, M	Machine-Learning + Naive Bayesian + Knowledge-Engineering Approach
	Dipongkor et al. [46]	2023	BR	Transformer-Based Models + DeBERTa
	Borg et al. [21]	2024	BR, R	Logistic Regression + Confidence-Based Human-in-the-Loop Strategy
IR	SAS [101]	2013	IT, L	FCA + DMI + Signature-Based Retrieval
	Ding et al. [44]	2014	L, HD	Cosine Score + Generating Triple Structures
	CRISTAL [117]	2015	IT, CC	Asymmetric Dice Similarity Coefficient
	LiDAR [32]	2020	IT, DN	Calculate Linkage Confidence Score
	Zhang et al. [188]	2020	BR	Neural Network + Text-Projection Features with Class-Based Weighting
	Xu et al. [171]	2023	BR	Aggregating the Frequency and Position of Component Functions + Trained Machine Learning Model
	COLA [84]	2024	A, SOP	Conditional Probability + Node2Vec + Skip-Gram + Jaccard Similarity Denoising + Two-Round Prompting + ICL
	Goel [57]	2024	IT, M	Multi-Stage Data Processing + LLM Inference
SN	Su et al. [146]	2021	BR	LambdaMART-Based Learning-to-Rank Framework + Bug Tossing Knowledge Graph
	DEEPTRIAG [145]	2023	BR	Deep Ensemble Model
	COT [159]	2021	IT	Text Parsing + Graph Construction + SVM + Decision Tree
DB	Chrupala et al. [36]	2012	BR	Online Learning Algorithms
	Mandal et al. [105]	2019	IT	Linear SVM + MLP Ensemble Classifier + Query Predefined Solutions + CORI Algorithm

4.3.2 Developer Assignment. During issue assignment, triage teams often consider developers' historical performance and prior experience in resolving similar problems. Developers who demonstrate a proven track record of successfully addressing comparable issues or consistently delivering high-quality fault fixes are frequently prioritized for assignment. Beyond individual expertise, effective collaboration and communication are essential for addressing complex, cross-cutting faults. Consequently, triagers may allocate issues to teams or individuals with a documented history of successful collaboration, thereby ensuring that established communication channels and coordination mechanisms are leveraged. Such assignment strategies not only increase the likelihood of timely resolution but also help reduce coordination overhead and improve overall fault management efficiency in large-scale software systems.

From a methodological perspective, prior studies on automated developer assignment can be categorized based on how they model the assignment problem. Text Classification (TC) approaches treat the task as a supervised classification problem, where each developer corresponds to a potential class label and models are trained to predict the most suitable assignee based on the textual content of bug reports [51, 62, 96, 150, 154, 161, 193]. In contrast, Information Retrieval (IR) methods frame assignment as a similarity search, ranking candidate developers according to the relevance of their historical records to the current issue, often using techniques such as TF-IDF, BM25, or topic modeling to compute textual similarity [123]. Social or Collaboration Network (SN) approaches exploit the structural relationships among developers, files, and modules, capturing patterns of past collaboration and expertise propagation through graph-based models or network embeddings [111, 165]. Optimization or Decision-making (OPT) approaches formulate assignment as a constrained optimization or sequential decision problem, balancing multiple objectives such as expertise, workload, and cost, and often employing search algorithms or evolutionary strategies [63, 76, 99, 127]. Finally, Other / Hybrid (OTH) approaches integrate multiple paradigms or techniques, such as combining rule-based, ML,

embedding, and network features, to leverage complementary information sources for more robust and context-aware assignment [41, 42, 72, 152, 176, 179, 187].

Text Classification. Focusing on TC, prior work has progressed from traditional supervised models to deep learning and PLM-based approaches for automated bug assignment.

(1) *Traditional Machine Learning Approaches:* As one of the earliest studies on automated bug assignment, Anvik et al. [8, 9] trained Naïve Bayes and SVM models on labeled bug reports, while Anvik et al. [10] incorporated project metadata and historical reports to produce ranked developer lists. Ahsan et al. [2] combined feature selection with Latent Semantic Indexing to reduce TF-IDF matrices for classifier training. Costriage [120] integrated per-developer SVM classifiers with collaborative filtering and cost estimation for effort-aware ranking.

(2) *Deep Learning Approaches:* Deep learning methods capture semantic and sequential dependencies. Lee et al. [88] used CNNs with Word2Vec embeddings and dynamic retraining. DeepTriage [106] employed bidirectional RNNs with attention, while iTriage [166] integrated sequence-to-sequence textual modeling with metadata features.

(3) *Pre-trained Language Models and Ensemble Strategies:* Following this wave of neural architectures, pre-trained language models (PLMs) and model ensembles have further advanced the state of the art in automated assignment. LBT-P [87] leverages multi-layer PLM embeddings, mitigating catastrophic forgetting, and producing efficient developer rankings. Wang et al.[156] empirically studied combinations of embeddings and classifiers, and Dipongkor[85] demonstrated that ensembles of transformer-based LLMs outperform individual models.

(4) *Industrial Scale and Robust Deployment:* The latest studies focus on practical, large-scale deployment, robustness, and multi-source integration. Park et al. [121] extended Costriage to non-textual reports using LDA and content-boosted collaborative filtering. Jonsson et al. [79] applied ensemble-based stacked classifiers to assign bug reports to teams. Sarkar et al. [137] combined logistic regression with incremental learning and high-confidence prediction for robust industrial triage. More recent work, such as BTAL [190] integrates multiple bug report sources, encodes textual content with BERT and TextCNN, and applies adaptive loss functions to mitigate class imbalance. FLSCL [155] leverages supervised contrastive learning over fixers' reports, producing robust embeddings fed into Bi-LSTM or BERT classifiers.

Taken together, the evolution of TC-based bug triage reflects both methodological innovation and practical impact: advances in representation learning and model ensembling have enhanced predictive performance, while integration of multi-source data and robust industrial strategies ensures applicability in large-scale, real-world software development environments.

Information Retrieval. Early IR-based approaches primarily leveraged textual similarity, while later studies incorporated temporal, contextual, structural, and network information to capture evolving developer expertise.

(1) *Textual Similarity and Fuzzy Expertise Modeling:* Bugzie [151] represents developer expertise using fuzzy sets combined with a cache-based mechanism, continuously updating membership scores from newly fixed bug reports. Incoming bugs are assigned by aggregating relevant fuzzy sets, improving accuracy and efficiency. Panda et al. [118] introduced a framework combining LDA topic modeling with Intuitionistic Fuzzy Sets (IFS) to represent uncertainty in developer expertise. Similarity measures and fuzzy α-cut selection enable the identification of developers likely to resolve new bugs.

(2) *Contextual and Structural Enhancements:* Subsequent work extended textual similarity by modeling temporal and contextual dynamics. VTBA [135] ranks developers based on technical term matching filtered via Stack Overflow and weights historical fixes by recency, representing each developer as a "document" of past bug sub-documents. Goyal [58] presents Visheshagya for time-based assignment, W8Prioritizer for prioritization via AHP, and NRFixer for predicting fixability of non-reproducible bugs, forming a comprehensive recommender system for diverse bug types.

Structural and authorship information has also been leveraged: Linares-Vásquez et al.[97] combine IR with source code authorship, ranking developers by file headers, while Shokripour et al.[141] restrict candidate files to those within the reported component and compute each developer's expertise based on the number of past change activities on those files, weighted by the recency of the changes.

(3) *Latent and Network-Based Representations:* Another line of work explored latent relationships through topic modeling and network-based representations of developer-bug interactions. Dretom [170] models developer expertise and interest using topic models derived from historical bug-resolving records, enabling the ranking of developers for new bug reports. Building on network representations, BugFixer [70] constructs a Developer–Component–Bug network, ranking developers based on both connectivity and similarity to historical bug-fix reports. Furthermore, $TopicMiner^{MTM}$ [167] extends LDA with multi-feature topic modeling that incorporates product and component information, enabling incremental developer assignment and significantly improving triage accuracy.

(4) *Interactive Retrieval Systems:* Complementing automated retrieval, interactive systems have been designed to assist developers in managing and exploring bug reports. PorchLight [22] introduces a specialized query language for tagging bug reports, enabling developers to organize and explore them in meaningful groups and thus mitigating the inefficiencies of one-by-one inspection in large-scale triaging scenarios.

IR-based approaches have evolved from purely textual similarity and fuzzy expertise modeling to incorporate temporal, contextual, structural, and network information. The integration of interactive systems further enhances developer support, collectively enabling more accurate and scalable bug assignment.

Social or Collaboration Network. SN approaches exploit the structural relationships among developers, files, and modules to capture collaboration patterns and expertise propagation, enabling more informed bug assignment decisions.

(1) *Early Graph-Based Methods:* Initial studies focused on modeling developer collaboration and bug flow using static or manually constructed networks. Jeong et al.[78] construct bug tossing graphs to represent developer collaboration and team structure, integrating historical assignment sequences to predict suitable developers and identify shorter paths to the fixer. Building on this idea, Bhattacharya et al.[18] leverage multi-feature tossing graphs to model developer activity and bug flow more comprehensively. They combine these graphs with incremental classifiers to provide textual support, effectively identifying potential developers while reducing tossing path lengths. Similarly, FixerCache [158] introduces an unsupervised strategy that dynamically maintains developer caches for each component, ranking developers by activeness. This approach achieves high prediction accuracy and diversity while avoiding the training overhead associated with supervised models.

(2) *Community-Level Extensions:* Subsequent approaches extended static networks to community-level structures, emphasizing collaborative expertise and developer clusters. DECOBA [13] constructs social networks of developers based on their bug-fixing contributions and detects communities. For a new bug report, it assigns a relevant developer community and ranks developers within that community by experience, ensuring that collaborative expertise is leveraged to identify the most suitable developers for triage and resolution.

(3) *Graph Neural Networks and Contrastive Learning:* Recent advances leverage graph neural networks and contrastive learning to dynamically model bug–developer relationships in an end-to-end manner. Wu et al. [164] propose a spatial–temporal dynamic GNN for automated bug triaging. It models evolving developer collaboration networks using a joint random walk (JRWalk) mechanism, and learns node spatial-temporal features through a graph recurrent convolutional neural network (GRCNN), enabling the prediction of the most suitable bug fixers. PCG[38] and NCGBT [47]

further enhance these models by applying contrastive learning to refine node representations, jointly considering structural and semantic relationships for candidate developer assignment.

Overall, the evolution of SN-based approaches highlights a progression from simple representations of developer interactions to more sophisticated models that integrate collaboration, community structures, and semantic context, enabling more accurate and adaptive bug triage.

Optimization or Decision-making. In OPT approaches, developer assignment is modeled as a constrained optimization problem, aiming to maximize overall project efficiency while minimizing manual triage effort. Early studies laid the theoretical foundation for expertise-based assignment.

(1) *Foundational Optimization Models:* Baysal et al.[15] proposed a theoretical framework that infers developer expertise from historical bug-fixing records, combining preference elicitation and expertise recommendation to allocate bugs optimally. While largely conceptual due to evaluation challenges, it laid the groundwork for optimization-based assignment. T-REC[116] extended this perspective by retrieving similar historical bugs using FastText and BM25F, fusing rankings probabilistically to recommend Top-K technical groups, thereby reducing manual triage and bug tossing. RAPTOR [81] formalized developer assignment as a multi-knapsack problem, integrating bug priority, severity, developer experience, and activity to maximize overall project bug-fixing efficiency under developer time constraints.

(2) *Multi-Objective and Dependency-Aware Optimization:* Building on optimization, subsequent approaches incorporate task dependencies and multi-objective search to better balance developer workload and bug handling order. Etemadi et al.[49] proposed a scheduling-driven assignment method that decomposes bugs into subtasks, models dependencies via a task dependency graph (TDG), and applies a multi-objective evolutionary algorithm to generate Pareto-optimal schedules minimizing fixing time and cost. Almhana et al.[7] leveraged file-level dependencies to define bug report relations, applying NSGA-II search to produce Pareto-optimal sequences that balance bug priority and cognitive load for developers.

(3) *Integration of Historical Context and Scheduling Constraints:* To integrate historical context and scheduling constraints, some methods combine optimization with dependency-aware models and dynamic decision-making. Jahanshahi et al. [75] leverage the Wayback Machine to reconstruct historical bug triage scenarios, dynamically updating bug dependency graphs and developer workloads. It enables integration of custom prioritization or assignment algorithms, including optimization-based methods like S-DABT [73], allowing assignments to be scheduled and balanced while respecting dependencies and historical context. Building on this, they then apply S-DABT [73], which integrates textual bug data, estimated fixing costs, bug dependencies, and developers' schedules into an integer programming framework, predicting suitability scores with SVM and LDA, and optimizing assignments to balance workload and respect dependencies. Finally, they propose ADPTriage [74]

(4) *Online Learning and Multi-Agent Strategies:* Recent works explore online learning and multi-agent systems to adaptively assign developers under uncertainty. Singh et al.[144] propose Enhanced_CMAB_Triage, a contextual multi-armed bandit approach that integrates bug features, developer activity, and similarity-based expert selection to balance exploration and exploitation for cold-start bugs. Triangle [180] adopts a multi-agent architecture, including Analyser, Triage Decider, and Team Manager. The analyzer performs semantic distillation to extract key phrases, Triage Decider selects 5 team candidates by combining TF-IDF and LLM, and Team Manager queries monitor logs to generate enriched information. The negotiation mechanism uses team candidate voting, confirming the team if over half agree, otherwise reselecting with feedback of enriched information, up to 5 rounds.

OPT-based methods have transitioned from static formulations to adaptive, multi-objective, and dependency-aware frameworks, with recent studies incorporating online learning and multi-agent coordination to cope with real-world uncertainty.

Other / Hybrid. OTH approaches integrate multiple paradigms, such as rule-based reasoning, machine learning, feature embedding, and network modeling, to leverage complementary information sources for more robust and context-aware developer assignment.

(1) *Integrating Historical and Log-Based Signals Beyond Textual Reports:* Some studies extend beyond traditional textual bug reports by incorporating historical execution or service data. WHOSEFAULT [139] integrates fault localization, history mining, and expertise mapping to directly assign developers to execution failures without relying on textual bug reports. DeCaf [14] employs machine learning and pattern mining over large-scale service logs to automatically diagnose and triage KPI performance regressions. Lim et al. [92] employ a Hidden Markov Random Field (HMRF)-based clustering model to discretize performance metrics and identify recurrent or previously unseen performance issues.

(2) *Multi-Feature and Knowledge-Enhanced Recommendation:* A second line of work enriches developer assignment by combining heterogeneous features and personalized expertise. Yang et al.[177] extract latent topics from historical bug reports and compute multi-feature similarities (*e.g.*, component, product, severity, priority) to jointly recommend developers and predict bug severity on large-scale datasets. Bhattacharya et al.[17] refine classification and tossing prediction by incorporating additional report attributes, intra-fold updates, and multi-feature tossing graphs. KSAP [189] retrieves similar resolved reports via K-nearest-neighbor search, ranks developers using heterogeneous proximity within a multi-entity collaboration network, and fuses textual and social features to generate a ranked developer list.

(3) *Neural and Graph-Based Hybrid Models:* Recent hybrid approaches adopt neural architectures to capture complex patterns in bug reports and developer behavior. DeepTriage [124] constructs an ensemble of FastTree binary classifiers with gradient boosting, supplemented by an inverted index to mitigate cold-start issues. BRAIN [33] leverages GRU-based sequential modeling, attention masking, and CNN-based language encoding to classify incidents using textual, conversational, and environmental information. GCBT [39] builds a bipartite bug–developer graph where bug nodes are initialized through NLP pre-training and developer nodes via attribute encoding. Spatial–temporal graph convolutions model evolving expertise, and an IR-based classifier matches bugs to developers, enabling correlation-aware triaging.

(4) *Rule-Based and Personalized Tossing Integration:* Hybrid strategies also combine rule-based reasoning with developer-specific behavior modeling. EDR_SI [147] enhances developer recommendation by integrating expertise and coding habits through Collaborative Topic Modeling (CTM), providing not only ranked developers but also personalized contextual information such as code files and developer networks. AutoAnalysis [160] employs a rule-based decision tree to encode engineers' historical experience for root cause identification, generating interpretable incident summaries that guide downstream language models. BPTRM [162] introduces personalized tossing relationships by learning a developer transition matrix via attention over historical tossing paths, refining bug–developer matching through both textual and behavioral signals.

Overall, OTH approaches highlight a shift toward integrating heterogeneous information and intelligent reasoning mechanisms, aiming to enhance contextual awareness, personalization, and interpretability in developer assignment.

Table 7. Summary of developer assignment approaches categorized by problem formulation. The abbreviations in the "Category" column stand for: **TC** = Text Classification, **IR** = Information Retrieval, **SN** = Social Network Modeling, **OPT** = Optimization / Decision-making, and **OTH** = Other / Hybrid. The abbreviations in the "Data" column stand for: **BR** = Bug Reports, **CC** = Code Commits, **MD** = Metadata (developer assignments, users, comments, milestones, tags), **TS** = Tossing Sequences, **L** = Logs, **M** = Metrics, **ST** = Stack Traces, and **IT** = Incident Tickets.

Category	Technique	Year	Data	Core Method
TC	Anvik et al. [9]	2006	BR	Text Categorization + Supervised Learning + Naïve Bayes / SVM / C4.5 + Developer Ranking
	Anvik et al. [10]	2011	BR, MD	Textual Feature Extraction + Supervised Learning Classifier + Developer Ranking
	Ahsan et al. [2]	2009	BR	TF–IDF + LSI + SVM
	Costriage [120]	2011	BR, MD	SVM + Cost-Aware Adjustment + Collaborative Filtering
	Lee et al. [88]	2017	BR, MD	CNN + Word2Vec Embeddings + Developer Classification + Classifier Management
	Deeptriage [106]	2019	BR, ST	Deep Bi-Directional RNN with Attention + LSTM + Text Classification + Probability Scoring
	iTriage [166]	2019	BR, TS	Seq2Seq Feature Learning + Classifier
	LBT-P [87]	2022	BR, MD	Knowledge Distillation + PLM Layer-Wise Embedding + Multi-Layer Classifier + Weighted Output
	Wang et al. [156]	2024	BR, MD	Word2Vec / GloVe / NextBug / ELMo / BERT + TextCNN / LSTM / Bi-LSTM / Attention + MLP / Naive Bayes + Embedding-Based Text Classification
	Dipongkor et al. [85]	2024	BR	Fine-Tuned LLMs + Voting/Stacking Ensemble + Hinge Loss Sequence Classification
	Park et al. [121]	2016	BR, MD, CC	SVM + LDA Topic Modeling + Content-Based Recommendation + Content-Boosted Collaborative Filtering + Dynamic Developer Profiles
	Jonsson et al. [79]	2016	BR, MD	Stacked Generalization + Ensemble Classifiers + Text Feature Encoding + Team-level Classification
	Sarkar et al. [137]	2019	BR, MD, L	Textual Features + Categorical Features + Logistic Regression + Incremental Learning + High-Confidence Filtering
	BTAL [190]	2025	BR, MD	BERT Embeddings + TextCNN Local Features + Multi-Source Metadata Fusion + Adaptive Loss Function
	FLSCL [155]	2025	BR, MD	ELMo Embeddings + BERT Embeddings + Bi-LSTM-Attention Classifier + Fixer-Level Supervised Contrastive Learning + Cross-Entropy Loss
IR	Bugzie [151]	2011	BR	Fuzzy Sets + Cache-Based Dynamic Scoring + Aggregation of Relevant Terms for Developer Ranking
	Panda et al. [118]	2022	BR, MD	LDA Topic Modeling + Intuitionistic Fuzzy Sets + IFSim Similarity Measures + Fuzzy α-Cut Expert Selection
	VTBA [135]	2020	BR, MD	Technical Terms Filtering + Stack Overflow Vocabulary + Developer-as-Document + Sub-Document Modeling + Time-Aware Weighting + IR-based Ranking
	Goyal et al. [58]	2017	BR, MD	Time-based IR Ranking (Visheshagya) + Parameter Prioritization (W8Prioritizer) + NRFixer Model for Non-Reproducible Bugs
	Linares-Vásquez et al. [97]	2012	BR, MD	Latent Semantic Indexing + Code Authorship Analysis for Developer Ranking
	Shokripour et al. [141]	2013	BR, CC	Noun Filtering + Location-Based Developer Recommendation
	Dretom [170]	2012	BR, MD	Topic Modeling + Developer Expertise Modeling + Interest-Aware Ranking
	BugFixer [70]	2014	BR, CC, MD	Vector Space Model + Developer-Component-Bug Network + Ranking
	$TopicMiner^{MTM}$ [167]	2011	BR	Multi-Feature Topic Model + TopicMiner
	PorchLight [22]	2013	BR, MD	Tagging via Bug Tagging Language + Query-Based Bug Set Creation + Interactive UI for Group Triaging
SN	Jeong et al. [78]	2009	BR, TS, MD	Bug Tossing Graphs + Markov Chain Modeling + Weighted Breadth-First Search + Developer Prediction Integration
	Bhattacharya et al. [18]	2012	BR, TS, MD	Multi-Feature Tossing Graphs + Incremental Classification + Developer Activity Labeling
	FixerCache [158]	2014	BR	Unsupervised Developer Cache + Activeness Scoring
	DECOBA [13]	2013	BR, MD	Bug Term Matrix + Developer Collaboration Network + Community Detection + Community-Based Assignment + Expertise Ranking
	Wu et al. [164]	2022	BR, MD, TS	Joint Random Walk + Graph Recurrent Convolutional Neural Network + Developer Prediction

Continued on next page

Table 7 – *Continued from previous page*

Category	Technique	Year	Data	Core Method
SN (continued)	PCG [38]	2024	BR, MD	Embedding Initialization + Prototype Clustering Augmentation + Graph Collaborative Filtering + Semantic Contrastive Learning + Multitask Joint Learning
	NCGBT [47]	2024	BR, MD	Bipartite Graph Modeling + Pre-trained Language Model Initialization + Multi-Layer Graph Neural Network + Neighborhood Contrastive Learning + BPR Loss Optimization
OPT	Baysal et al. [15]	2009	BR, MD	Preference Elicitation + Expertise Recommendation + Task Allocation
	IT-REC [116]	2019	BR, MD	FastText Vector Space Model + Extended BM25F IR + Probabilistic Top-K Ranking
	RAPTOR [81]	2019	BR, MD	Multi-Knapsack Modeling + Suitability Score Calculation + Linear Programming Solver
	Etemadi et al. [49]	2021	BR, MD, CC	NSGA-II + Task Dependency Graph + Developer-Task Assignment Vector + Scheduling Vector + Greedy Local Search + Multi-Objective Evaluation
	Almhana et al. [7]	2021	BR, MD, CC, TS	Bug File Localization + Dependency Calculation + Multi-Objective NSGA-II Search + Pareto-Optimal Bug Sequencing
	Wayback [75]	2022	BR, MD, TS, L	Historical Event Replay + Bug Dependency Graph Update + Developer Load Tracking + Modular Algorithm Integration
	S-DABT [73]	2022	BR, MD, TS	SVM Text Classification + LDA-Based Cost & Dependency Estimation + Schedule-Aware Programming + Multi-Objective Optimization
	ADPTriage [74]	2023	BR, MD, TS, L	LDA Topic Modeling + Markov Decision Process + Approximate Dynamic Programming + Dynamic Bug Assignment
	Singh et al. [144]	2025	BR, MD, TS	Developer Activity Level + Similarity-Based Candidate Filtering + Contextual Multi-Armed Bandits + Exploration-Exploitation Optimization
	Triangle [180]	2021	IT, M	Multi-Agent Architecture + Semantic Distillation + TF-IDF + Negotiation Mechanism
OTH	WhoseFault [139]	2012	CC	Fault Localization + History Mining (line-level) + Expertise Assignment for Ranked Developer List
	DeCaf [14]	2020	L	Random Forests + Pattern Mining + Custom Scoring
	Lim et al. [92]	2014	M	HMRF-Based Clustering + EM Optimization
	Yang et al. [177]	2014	BR, MD	LDA Topic Modeling + Developer Ranking + KNN
	Bhattacharya et al. [17]	2010	BR, MD	Incremental Naïve Bayes / Bayesian Networks Multi-Feature Tossing Graphs + Ranking
	KSAP [189]	2016	BR, MD	K-Nearest-Neighbor Search + Heterogeneous Developer Collaboration Network + Meta-Path Proximity Ranking
	DeepTriage [124]	2020	BR, MD, L, ST	MART + LightGBM + CNN + Inverted Index + Clustering
	BRAIN [33]	2020	IT, MD, M, ST	GRU-Based Model + Attention Masking Strategy + CNN-Based Neural Language Model
	GCBT [39]	2023	BR, MD	Triaging Graph Construction + NLP-Based Node Initialization + Spatial Convolution + Temporal Convolution + Bug Embedding Augmentation + IR-Based Classifier
	EDR_SI [147]	2017	BR, CC, MD	Historical Commits Preprocessing + Collaborative Topic Modeling + Personalized Developer–Code Ranking + Supplementary Information Construction
	AutoAnalysis [160]	2024	IT, L, MD, ST	Log filtering + Keyword Extraction + Incident Embedding + Similarity-Based Retrieval + LLM-Based Triage
	BPTRM [162]	2025	BR, MD, TS	BERT Embedding + Bug Attribute Encoding + SIMIR / SVM / DeepTriage + Attention-Based Ability Matching + Tossing Transition Probability Matrix + Top-K Recommendation

4.4 Postmortem Process

Following the resolution of an incident or bug, the triage process often includes a structured postmortem phase aimed at capturing actionable knowledge and improving future issue management. This phase typically involves labeling and categorizing the resolved issue with metadata such as root cause classification, impacted components, resolution type, and severity level. These labels enable efficient indexing, retrieval, and statistical analysis of historical cases. In addition, linking the issue to related incidents, commits, or configuration changes supports traceability and facilitates the identification of recurring fault patterns.

A well-executed postmortem process also includes documenting lessons learned, decision rationales, and any procedural improvements identified during resolution. Such documentation not only serves as a reference for preventing similar issues in the future but also strengthens automated triage systems by enriching training datasets with high-quality, structured information. Furthermore, tracking post-resolution follow-up actions, such as applying preventive patches, updating monitoring rules, or refining alert thresholds, ensures that the organization benefits from a continuous feedback loop between operational experience and fault management practices. By systematically embedding these postmortem practices into triage workflows, organizations can enhance knowledge reuse, improve fault detection accuracy, and reduce mean time to resolution in large-scale software and service environments.

4.4.1 Continuous Triage. In large-scale online service systems, incident triage aims to assign newly reported incidents to the most appropriate teams. However, due to incomplete information and evolving context, initial assignments are often inaccurate, requiring repeated reassessment and cross-team discussion. This iterative refinement process, referred to as continuous incident triage [27], highlights the need for approaches capable of dynamically updating triage decisions as new information emerges. Existing research on continuous triage can be broadly grouped into three methodological paradigms.

(1) *Learning-Driven Refinement:* Learning-driven methods, such as DeepCT [27], formalize triage as an incremental learning process, where models progressively update assignment predictions based on ongoing interactions or discussion histories. Earlier probabilistic frameworks, such as Shao et al.'s work [140], also reflect this idea, modeling team transitions as evolving state sequences to capture the dependencies among successive assignment decisions.

(2) *Data-Driven Adaptation:* Data-driven approaches extend this paradigm by incorporating model uncertainty and distributional drift into the triage process. For instance, Scouts [55] detects shifts in incident patterns and dynamically adjusts routing decisions, while Ticket-BERT [100] employs an iterative fine-tuning cycle that continuously integrates newly labeled incidents to maintain model adaptability. These techniques emphasize automated model updating as a mechanism for sustaining performance under evolving data conditions.

(3) *Human-in-the-Loop Collaboration:* In contrast, team-driven approaches integrate human expertise into the triage loop. Triangle [180] exemplifies this perspective by coupling semantic enrichment with consensus-based decision refinement, where automated recommendations are iteratively reviewed and adjusted through human feedback until convergence.

Overall, continuous triage methods share a common goal of enabling adaptive and iterative decision-making. They differ primarily in the source of feedback, but all aim to transform triage from a static one-shot prediction into a dynamic, continuously improving process.

4.4.2 User Feedback Analysis. Only a limited number of studies have investigated feedback issues arising after a single distribution failure. Considering the similarity between app user reviews and user submissions following triage failures, several lines of research have explored how user feedback can support post-release issue management, evolving from early integration with code artifacts to more advanced text-driven and learning-based analyses.

Early studies, such as Shokripour et al. [141], incorporated user feedback into software repositories by linking review content with code commits and historical bug data to support defect localization and assignment. Palomba et al. [117] extended this idea by constructing traceability links between user reviews and code changes, enabling release-level monitoring and assessment of quality variations.

Subsequent research shifted toward mining and modeling user feedback as an independent information source. Gao et al. [54] adopted topic modeling to identify emerging issues across app versions, dynamically combining prior and

current topics to capture evolving user concerns. Later, Etaiwi et al. [48] and Malgaonkar et al. [104] advanced the analysis by applying text preprocessing, clustering, and embedding-based similarity measures to prioritize user-reported issues and support developer decision-making.

Besides, Di et al. [43] revisited the connection between user feedback and software quality by correlating feedback metrics, code characteristics, and app ratings. This line of work bridges earlier repository-based and review-based methods, highlighting the feedback loop between user experience and software maintenance activities.

Table 8. Summary of postmortem processing based on their category, the year, and the data. The abbreviations in the "Data" column stand for: **M** = Metrics, **TR** = Triage Results, **IT** = Incident Tickets, **HTS** = Historical Ticket Sequences, **R** = Reviews, **RR** = Release Ranks, and **CC** = Code Commits.

Category	Technique	Year	Data	Core Method
Continuous Triage	DeepCT [27]	2019	D, TR	GRU+Attention-Mask Strategy
	Shao et al. [140]	2008	HTS, TR	First-Order Markov+Variable-Order Markov
	Scouts [55]	2020	IT	Random Forest+Improved Change Point Detection
	Ticket-BERT [100]	2023	IT	Fine-Tune
	Triangle [180]	2021	IT, M	Semantic Distillation+Discussion Group Voting
User Feedback Analysis	Shokripour et al. [141]	2013	R, CC	Noun Extraction Process+Simple Term Weighting Scheme
	CRISTAL [117]	2015	R, RR	Review Coverage+Monitoring Component
	Gao et al. [54]	2018	R	IDEA to Automatically Identify Emerging Issues+AOLDA for Online Review Analysis
	Etaiwi et al. [48]	2020	R	CLAP Tool+ExactAlgorithm+BioConcert+KwikSort
	Malgaonkar et al. [104]	2022	R, IT	DistilBERT+Cosine Similarity
	Di et al. [43]	2021	R, M	Spearman's Rank Correlation Coefficient

Finding 1: Evolution of Triage Methods Across the Lifecycle

Over time, triage methods have evolved to encompass the entire lifecycle, from data preprocessing to postmortem analysis. Early research primarily focused on data deduplication and feature extraction to enhance input quality, followed by advances in prioritization techniques such as severity assessment and issue classification. Subsequent studies emphasized assignment optimization for components and developers, while recent efforts have introduced continuous triage and feedback-driven refinement mechanisms. This evolutionary trajectory signifies a transition from isolated task automation to adaptive, end-to-end triage systems capable of continuous learning and improvement in real-world settings.

5 Challenges in Triage Practice

Triage, including bug triage in software engineering and incident triage in large-scale online or cloud services, is a multifaceted process that assigns reports, tickets, or alerts to the most appropriate resolver, whether an individual developer, a team, or an automated system. As modern software and service infrastructures evolve, triage practices encounter increasingly complex challenges arising from both technical barriers and operational demands. Drawing on recent literature and industrial practice, we summarize the key challenges as follows.

5.1 Data Quality, Diversity, and Representation

5.1.1 Data Noise and Incomplete Information. Bug reports and incident tickets are often poorly structured, ambiguous, or irrelevant. Common issues include missing reproduction steps, vague or noisy descriptions, absent contextual information such as environment details, severity, or affected component, and inconsistent field usage across projects and systems.

- **Multimodal Data Noise.** The integration of multimodal inputs such as text, code, logs, metrics, and configurations increases noise, especially in cloud service environments where heterogeneous data sources coexist [91, 106]. For text data, there is a large volume of logs and discussion texts with significant noise, making direct processing difficult. Existing methods struggle to extract high-quality key points from a large amount of text [160]. Code snippets lack structural information in feature representation, making it difficult to learn effectively [12]. For review of apps, vocabulary mismatches always happen between user reviews and source code or issues reported in issue trackers [117].
- **Human and System-induced Reporting Errors.** Reports are frequently affected by human errors, false alarms, or transient system anomalies, leading to non-reproducible or irrelevant cases such as alert storms or incidental incidents [61]. Besides, there may have been cases where this value was not a person with expertise in that area of the source code. For example, the person who fixed the bug may not have commit rights, and another project member may be required to commit the fix [141].
- **Heterogeneity Across Organizations and Platforms.** Triage systems operate across diverse organizations and platforms, where data structures, naming conventions, and semantic expressions vary widely [88, 100]. Monitoring alerts in production, customer support tickets, and automatically generated system events often exhibit distinct temporal, textual, and structural characteristics. Large-scale online service systems do not share much similarity: they are developed for different purposes by different teams. They possess various code logic, implementation languages, and architectures [194]. Furthermore, update patterns can also differ across distinct features of the same application, as well as for the same feature at different time points. These differences thus require systematic analysis [157].

5.1.2 Class Imbalance and High Dimensionality. Assignments typically involve hundreds or even thousands of potential targets, including developers, teams, or components. The majority of reports concentrate on a small subset of assignees, creating a long-tail distribution. Models tend to overfit frequent classes while underperforming on rare ones, reducing assignment accuracy and leading to overloaded teams or prolonged unassigned cases. At the data collection level, a large number of possible teams not only challenge machine learning models in terms of complexity but also exhibit extreme data imbalance. A wide variety of data formats supported in modern incident management systems requires the routing system to use different approaches to parsing this content and provide useful features information to machine learning [124]. Besides, the disproportionate ratio of class labels in the training data affects the performance of the classification prediction model [12].

5.2 Dynamic Ecosystem and Evolving Context

5.2.1 Concept Drift and Temporal Variability. Frequent organizational and system changes, such as team restructuring, personnel turnover, or component refactoring, rapidly invalidate historical assignment patterns. This results in concept drift, where models must adapt to shifting data distributions in real time [21, 36, 188]. However, understanding the issue and providing appropriate healing action depend heavily on domain knowledge [44]. The dependency structure is neither known nor fixed in large online service systems. Therefore, it is hard to use the dependency graph to identify linked incidents [32]. High-frequency updates in practical environments impose stringent requirements on both adaptability and stability.

5.2.2 Cold-start Problem. New developers, teams, or components lack sufficient historical data, making it difficult to assess their expertise or availability. Cold-start challenges are especially pronounced in system rollouts, team expansion, or platform migration, which are common in cloud monitoring and DevOps contexts [144].

5.2.3 Knowledge and Expertise Decay. Developer expertise diminishes over time, and inactive or departing experts leave behind outdated assignment traces [107]. The system components are mostly developed by different teams, and expert knowledge is limited. There is a lack of a systematic mechanism for sharing knowledge of historical events, resulting in incident resolution relying on a few experts and a longer MTTR [102]. Effective triage systems must detect these changes to avoid misdirected assignments. Evolving code ownership and product lifecycles further require dynamic tracking of responsibility boundaries.

5.3 Multi-factor and Multi-step Assignment Complexity

5.3.1 Multi-dimensional Decision Factors. Accurate triage requires integrating multiple factors beyond text classification. These include current workload and availability of developers or teams, skills, and historical contributions such as authorship or activity records, individual or team preferences where pull-based assignment is adopted, and task-related constraints such as deadlines, severity, priority, or cost [49, 121, 196]. It may be the case that the developer who actually fixed a bug according to the "Line 10 Rule" is not the optimal choice for that report, whereas we treat it as such [141]. Besides, the problem space is complex, with diverse causes for events involving hardware, network, resource competition, and other aspects, requiring a comprehensive analysis of various types of monitoring data [101]. Consequently, the feature vectors tend to be extremely large, which renders it highly challenging for a single event router to integrate monitoring data from all teams. Furthermore, the events themselves are rare occurrences, making it difficult to address this challenge by increasing the number of training samples [55].

5.3.2 Complex Priorities and Dependency Structures. Reports may contain explicit or implicit dependencies such as blocker, related, or duplicate relationships [74]. Failures often cascade across teams or components, requiring algorithms to jointly handle interdependent cases and update dependency queues dynamically. The large-scale online service system has numerous components developed by different teams, leading to a lack of comprehensive understanding among engineers about the system, as well as a lack of knowledge-sharing mechanisms, resulting in slow event resolution and long average recovery times [101]. The virtualization and dynamic allocation of resources in cloud environments make it difficult to associate physical and virtual resources. The integration of scattered event tickets, CMDB, and monitoring data is needed to eliminate data silos [64].

5.3.3 Reassignment Cost. Incorrect assignments frequently result in issue tossing, where reports are repeatedly reassigned among teams or individuals. This significantly increases resolution time and reduces overall efficiency [141]. Optimization must therefore consider not only predictive accuracy but also tossing path length and average time to resolution [78, 146].

5.4 Scalability, Real-world Integration, and Explainability

5.4.1 Scalability and Efficiency. Deployed triage systems must scale to millions of reports or alerts, often in real time, as seen in Microsoft Azure or global banking alert systems [87, 191]. Besides, the agile development model in cloud services requires the development and deployment process to be flexible and meet certain production requirements in terms of execution time, or error tolerance to maintain high service availability [124]. This demands models with

high inference speed and efficient resource utilization. Today, ticket routing is usually driven by expert decisions. It is not uncommon that, due to human error or inexperience, a ticket is mistakenly transferred to a group that cannot solve the problem, which might lead to a long and inefficient routing sequence. In such cases, not only are resources wasted, but it would also take a longer time to close tickets, causing customer dissatisfaction [140]. Some existing methods train developer recommendation and issue type assignment as independent tasks, resulting in task repetition and overlooking the correlating information between tasks [12].

5.4.2 Integration with Real-world Workflows. Triage systems must integrate seamlessly into DevOps, agile, and team-specific processes without disrupting established practices [134]. Outputs should be interpretable, allowing engineers to understand assignment rationales and confidence levels. Large-scale organizations further require mechanisms for cross-team permissions and inter-departmental communication. However, traditional hierarchical multi-label classification algorithms, when applied to ticket classification, fail to fully incorporate the domain experts' prior knowledge, so that it is unable to quickly adapt to the new system environment. The existing loss functions are ineffective in distinguishing between different types of misclassifications, making it difficult to accurately assess classification performance [181].

5.4.3 Lack of Standardized Evaluation and Reproducibility. Research on triage often lacks standardized task definitions, evaluation metrics, dataset preprocessing pipelines, and gold standards, making cross-study comparison difficult [134]. Industrial data remains largely proprietary, limiting reproducibility and slowing technology transfer. The black-box models (such as DNNs) perform well in event classification, but their lack of interpretability makes it difficult for engineers to understand and trust the prediction results [129].

Finding 2: Practical Challenges in Triage Practice

In practice, triage processes face considerable challenges arising from data quality issues, dynamic project ecosystems, and operational complexity. Real-world datasets are often noisy, incomplete, and heterogeneous, while the continual evolution of project contexts results in concept drift and knowledge obsolescence. Furthermore, multifactor decision dependencies and the absence of standardized evaluation frameworks impede scalability and seamless integration into development workflows. Collectively, these limitations underscore a persistent gap between research-oriented prototypes and deployable triage solutions.

6 Evaluation and Benchmarking

6.1 Evaluation Metrics

The evaluation of triage techniques has relied on a wide range of metrics that capture not only predictive accuracy but also ranking effectiveness, computational efficiency, and practical impact on software engineering workflows. For clarity, these metrics can be categorized into four groups.

6.1.1 Accuracy Metrics. For binary triage tasks, conventional classification metrics are commonly employed to evaluate whether the model correctly distinguishes between positive and negative cases. These metrics provide a foundational assessment of model discrimination and general predictive reliability. Representative metrics include:

- **Accuracy:** The proportion of correctly predicted instances among all predictions. Accuracy has been employed to evaluate classifiers for team assignment in industrial contexts [79] and to compare deep learning models against human triagers [88].

- **Precision, Recall, and F-score:** Precision quantifies the fraction of correctly predicted positive instances among all positive predictions, whereas recall measures the fraction of true positives correctly identified. The F-score provides a harmonic balance between the two, with variants such as $F1$ and $F2$ used depending on whether precision or recall is prioritized. These metrics have been foundational since early bug triage research [2, 9], and remain widely adopted, with recent studies employing $F2$-scores to emphasize recall-oriented evaluation [12].
- **AUC (Area Under the ROC Curve) and Area Under the Precision–Recall Curve:** These metrics assess the model's overall ranking capability across different classification thresholds. They are particularly useful in imbalanced datasets, such as those involving rare component assignments or low-severity bugs [25, 91].
- **Cohen's Kappa:** Occasionally adopted to measure the degree of agreement between predicted and actual classifications beyond random chance [2].

6.1.2 Ranking and Top-k Metrics. While binary classification metrics focus on the correctness of a single predicted label, multi-class or recommendation-based triage tasks typically aim to rank multiple candidate labels (*e.g.*, developers, priorities, or solutions). In such cases, ranking-based and top-k metrics are more appropriate to capture the quality of ordered recommendations. Key evaluation measures include:

- **Top-k Accuracy (Hit@k, Acc@k):** Determines whether the correct label appears within the top-k ranked predictions, commonly with $k \in \{1, 3, 5, 10, 20\}$ [38, 39, 87, 106, 146].
- **Mean Reciprocal Rank (MRR):** Evaluates the ranking position of the correct label by averaging the reciprocal rank across all instances. MRR has been widely used to assess the quality of ranked developer recommendations and other triage-related retrieval tasks, including those employing transformer-based or time-sensitive features [46, 142, 145].
- **Mean Average Precision (MAP):** The metric measures the mean of the Average Precision (AP) values computed across all queries, such as bug reports. It not only accounts for ranking quality but also integrates recall considerations, providing a comprehensive assessment of retrieval performance. Compared with Mean Reciprocal Rank (MRR), MAP offers a more holistic reflection of the overall ranking effectiveness, as it rewards systems that maintain high precision across varying recall levels. [118, 189].
- **Normalized Discounted Cumulative Gain (NDCG@k):** Measures ranking quality by assigning higher importance to correct predictions appearing earlier in the ranked list, thereby capturing both relevance and position sensitivity [146].
- **Precision@k, Recall@k, and $F1$@k:** Extensions of the traditional classification metrics adapted to top-k recommendation scenarios. These are often employed to evaluate developer recommendation models by measuring the proportion of correct labels among the top-k candidates [118, 168].
- **Diversity Metrics:** Used to assess the variety of recommended items, ensuring that suggestions (*e.g.*, developers or solutions) are not redundant or overly concentrated. For instance, diversity metrics have been used in caching-based triage systems to promote balanced and diverse developer recommendations [158].
- **Nearest False Negatives (NFN) and Nearest False Positives (NFP):** These two metrics are used to evaluate the performance of priority classification tasks by quantifying the proximity between the predicted and actual priority levels of misclassified bug reports. Specifically, NFN measures how close an incorrectly predicted lower-priority report is to its true higher-priority label, while NFP assesses the reverse case. Together, they capture the severity of misclassification and provide a nuanced understanding of how prediction errors deviate from true priority rankings [80].

6.1.3 Efficiency Metrics. Since triage is often performed in time-critical operational environments, efficiency-oriented evaluation plays an essential role in assessing both the responsiveness and the practical utility of triage systems. Commonly used measures include:

- **Assignment Accuracy and Fixing Time:** These metrics jointly evaluate predictive correctness and temporal efficiency. The total time saved by an automated triage system can be quantified using two complementary measures: *gain-in*, representing the time saved through correct routing, and *gain-out*, denoting the time saved by preventing incorrect assignments [55].
- **Recommendation and Query Response Time:** Captures the latency required to generate triage recommendations or respond to queries. This metric directly reflects the system's computational efficiency and its suitability for deployment in real-time settings [151].
- **Time to Mitigation (TTM), First Triage Time (FTT), and Mean Steps to Resolve (MSTR):** Measure the timeliness and process efficiency of triage workflows, reflecting how quickly a reported issue progresses from identification to actionable resolution [140, 160].
- **Reduction in Tossing Path Lengths:** Evaluates the degree to which a triage approach reduces unnecessary reassignments before reaching the correct developer or team. Shorter tossing paths indicate improved routing precision and decreased communication overhead [78].
- **Distribution of Due Dates and Convergence Analysis:** Analyze the temporal dynamics of triage decisions, including how assignment timing aligns with project deadlines or optimization objectives. These advanced metrics are used to evaluate the balance between assignment delay and developer suitability, as well as to assess model convergence and stability over time [74].

6.1.4 User-Centered and Qualitative Evaluation. Beyond quantitative performance metrics, several studies emphasize user-centered and qualitative assessments to evaluate the perceived usability, interpretability, and adoption potential of triage systems.

- **Likert-Scale Surveys:** Often implemented as five-point questionnaires designed to assess dimensions such as ease of use, perceived usefulness, and the likelihood of adoption by practitioners [22].
- **Qualitative Feedback and Interviews:** Provide insights into practitioners' perceptions regarding the interpretability, reliability, and practical value of triage systems. For example, structured interviews have been conducted to capture developers' experiences with new triage tools [22], while broader surveys have explored how practitioners perceive automated bug management techniques more generally [196].

6.1.5 Structural Classification Metrics.

- **Hierarchical Classification Accuracy:** In hierarchical or multi-label triage scenarios, such as categorizing issue types or ticket hierarchies, accuracy is often assessed using specialized indicators including Hamming Loss, HMC-Loss, H-Loss, and Parent–Child Error. These metrics evaluate not only label correctness but also the consistency of predictions within hierarchical structures [181].

6.2 Datasets

High-quality, large-scale datasets provide indispensable experimental foundations and evaluation benchmarks for algorithmic innovation and technological advancement. They play a pivotal role in fostering knowledge integration and accelerating innovation across academia and industry. Similar to other data-driven fields, datasets serve as a

cornerstone in the triage domain. However, while industrial-grade software systems naturally generate heterogeneous and large-scale data, academic research often lacks access to these real-world datasets. Consequently, many empirical studies rely on incomplete, small-scale, or simulated data, limiting the industrial applicability of their findings. This challenge is particularly acute in the context of software fault diagnosis, where the absence of high-quality datasets poses a significant barrier to validation and reproducibility.

Advancing triage research requires joint efforts from both industry and academia. Industrial stakeholders are uniquely positioned to provide production-level event logs, bug reports, and observability data, whereas academia can contribute by curating labeled datasets, developing advanced triage solutions, and facilitating open sharing of resources. The construction of standardized datasets and the establishment of unified evaluation metrics demand collaborative engagement across multiple parties. Based on a survey of existing efforts, we summarize the current landscape of publicly available triage-related datasets as follows.

6.2.1 Bug Triage Datasets. In the domain of bug triage, the open culture of large-scale open-source projects has significantly facilitated research by providing extensive repositories of publicly accessible defect data. Such datasets not only enable reproducibility of experimental results but also constitute valuable resources for advancing subsequent studies.

Data Sources and Characteristics. The most widely adopted datasets originate from public defect tracking systems (*e.g.*, Bugzilla, JIRA) maintained by prominent open-source communities such as Mozilla (Firefox), Eclipse, Apache, and GCC. These datasets primarily consist of structured bug reports, which typically include metadata such as titles, detailed descriptions, submitters, associated products and components, operating systems, priorities, and severity levels. Additionally, the "tossing history" of bug reports, records of developer reassignments, has emerged as a critical feature for modeling collaboration networks and analyzing assignment processes. Based on a comprehensive literature and repository review, we assemble publicly accessible triage datasets, summarized in Table 8.

Table 9. Summary of publicly available datasets.

Name	Data	Details
MultiTriage [12]	Bug Reports from Eclipse & Github OSS projects	https://github.com/thazin31086/MultiTriage/tree/master/Project/Data
	Original Issue Report of the Open-Source ASP.NET Core Project	https://github.com/dotnet/aspnetcore
	The Preprocessed Partial Dataset Used for Multi-Triage	http://dx.doi.org/10.5281/zenodo.5532458
Wu et al. [164]	Fixed bug reports from two OSS projects, namely Eclipse and Mozilla.	https://github.com/ssea-lab/BugTriage/tree/master/raw%20data
ADPTriage [74]	Bug reports from EclipseJDT & GCC & Mozilla OSS projects	https://github.com/HadiJahanshahi/ADPTriage/tree/main/Toy_Example
Zhang et al. [185]	Bug reports from Eclipse & GCC & Mozilla & Netbeans & OpenOffice OSS projects	https://github.com/ProgrammerCJC/SPFR
VTBA [135]	Bug reports from 13 popular GitHub projects (*e.g.*, Angular.js, Rails, Elasticsearch)	https://github.com/TaskAssignment/VTBA
Li et al. [91]	Bug reports from 6 OSS projects (*e.g.*, Cassandra, Flex, Hbase)	https://github.com/lizx2017/textMyth
S-DABT [73]	Bug reports from EclipseJDT & GCC & Mozilla & OpenOffice OSS projects	https://github.com/HadiJahanshahi/SDABT/tree/main/dat
Wang et al. [156]	Bug reports from EclipseJDT & GCC & Firefox OSS projects	https://github.com/AI4BA/dl4ba
Gao et al. [54]	User reviews of 6 popular apps	https://github.com/ReMine-Lab/IDEA
Di et al. [43]	Reviews and versions of Android apps& code quality metrics	https://github.com/sealuzh/user-satisfaction
ART [149]	Microservice datasets & failure cases	https://github.com/bbyldebb/ART

Continued on next page

Table 9 – *Continued from previous page*

Name	Data	Details
Sadlek et al. [132]	Testbed data with Windows/Ubuntu hosts & alerts & logs from 5 attack scenarios	http://dx.doi.org/10.5281/zenodo.14547668
Gao et al. [52]	Reviews from 5 apps (*e.g.*, eBay, Viber)	https://github.com/monsterLee599/SOLAR
Haering et al. [65]	Problem reviews and bug reports from 4 open-source apps (*e.g.*, Firefox, VLC)	https://mast.informatik.uni-hamburg.de/replication-packages/
Phetrungnapha et al. [125]	Mobile app user review dataset with titles, descriptions, ratings, labeled as feature requests, problem discoveries, etc	https://mast.informatik.uni-hamburg.de/app-review-analysis
Linares et al. [97]	Issue & Commit Comment from ArgoUML, JEdit, MuCommander	http://www.cs.wm.edu/semeru/data/icsm2012-authorship/
Lim et al. [92]	Performance Metric Data Records	http://www.dropbox.com/s/pj1miqu00ryoj9a/HMRF-kMedoid-EM.zip
Nagwani et al. [112]	GitHub bug repository	https://github.com/orgs/github/repositories
Bushehrian et al. [24]	One-phase and Two-phase method from Camel, CloudStack, Geode and Hbase	https://github.com/Ziba-Ghane/CQC.git
Dipongkor [45]	Sun Firefox, JDT, Netbeans, GUO Firefox, GCC datasets	https://github.com/farhan-93/bugtriage
$Triage_{Expert-Recency}$ [59]	Mozilla Firefox project bug reports	https://bugzilla.mozilla.org/describecomponents.cgi?product=Firefox
	SeaMonkey project bug reports	https://bugzilla.mozilla.org/describecomponents.cgi?product=SeaMonkey
Jalal [4]	Eclipse project's Bugzilla bug report dataset with 48 features	https://data.mendeley.com/datasets/t6d9y7yt54/1
Sajedi et al. [133]	Bug reports from 20 large GitHub projects	https://github.com/alisajedi/BugTriaging
Bernardi et al. [16]	Bug and communication data from Apache httpd, GNU GCC, Mozilla Firefox, and Xorg Xserver projects	https://github.com/mlbresearch/talking-data
GitBugs [122]	Bug report dataset from 9 open-source projects	https://github.com/av9ash/gitbugs/
Krasniqi [83]	Bug reports from six OSS projects (Bugzilla & Jira)	https://zenodo.org/record/6412840
Bugayenko et al. [23]	Software development tasks ("puzzles") extracted from industrial code repositories	https://github.com/cqfn/pdd-data-analysis
James et al. [77]	crash reports from large-scale OSS bug repositories	https://github.com/kedjames/crashsearch-triage
MSR2013 [86, 109]	Reported bugs extracted from the Eclipse and Mozilla projects.	https://github.com/ansymo/msr2013-bug_dataset
Bug Triaging [109]	Bugs tagged in the Eclipse dataset.	https://www.kaggle.com/datasets/monika11/bug-triagingbug-assignment/data
Bugzilla-Mozilla [143] [103, 126, 175]	System Bug report	https://bugzilla.mozilla.org/home
Eclipse [126, 136, 143] [110, 120, 151, 166] [103, 112, 175, 177]	Bugs reported & Bugs changed	https://bugs.eclipse.org/bugs/
NetBeans [112, 126, 143] [103, 175]	NetBeans bug repository	https://issues.apache.org/jira/projects/NETBEANS
Apache [120, 151]	Bugs reported & Bugs changed	https://issues.apache.org/jira/
GCC [112, 151]	Bugs reported & Bugs changed	http://gcc.gnu.org/bugzilla/
Linux kernel [120]	Bugs reported & Bugs changed	https://bugzilla.kernel.org/
Gentoo [166]	Bugs reported & Bugs changed	https://bugs.gentoo.org)

Data Accessibility. Most datasets used in defect classification research are fully open. Numerous studies explicitly acknowledge that their data was collected from publicly available platforms, with some works providing processed datasets or reproducible packages [12, 146].

Data Scale and Type. These datasets are generally large-scale, ranging from tens of thousands to several hundred thousand reports. For instance, Mozilla and Eclipse datasets often include between 100,000 and 500,000 defect reports. While the core data is textual (titles and descriptions), structured metadata provides supplementary classification features.

Industrial Proprietary Data. Beyond open-source repositories, certain studies have leveraged proprietary defect datasets from companies such as LG Electronics, IBM (Jazz), and Microsoft (Windows Vista). These datasets offer

insights into defect characteristics within real industrial environments. However, due to commercial confidentiality, they are rarely made publicly available, limiting reproducibility and wider academic adoption.

6.2.2 Incident Triage Datasets. In contrast to defect classification, acquiring datasets for incident management, classification, and prioritization in large-scale online service systems remains significantly more challenging.

Data Sources and Characteristics. Research in this area largely depends on proprietary datasets collected from production environments of technology companies such as Microsoft (particularly Azure), IBM, and Google. Unlike defect reports, incident datasets are inherently multimodal, encompassing user-submitted natural language incident tickets alongside diverse machine-generated data, including system logs, performance metrics, service dependency graphs, configuration change logs, and alerts. The heterogeneity and complexity of this data underscore the core challenges of incident triage.

Data Accessibility. Due to concerns over trade secrets, system security, and user privacy, nearly all incident triage datasets remain proprietary. Consequently, most published studies describe only high-level data characteristics (*e.g.*, "six months of incident data totaling 90 GB" or "terabytes of daily service logs"), without releasing raw or anonymized datasets. This lack of accessibility severely impedes reproducibility and prevents fair cross-study comparisons.

Research Challenges and Outlook. The field of incident triage is heavily constrained by its reliance on proprietary industrial data. To overcome this impasse and facilitate technological transfer from research to practice, collaborative initiatives are urgently needed to build and release benchmark datasets that faithfully represent real-world industrial scenarios. Establishing standardized datasets and unified evaluation metrics would be a critical enabler for accelerating innovation, enhancing comparability of research outcomes, and fostering the development of advanced and practically deployable triage solutions.

6.3 Toolkits

In this section, we present a curated list of open-source code repositories and toolkits derived from the surveyed papers on bug triage, incident triage, and related tasks. These resources are based on works that explicitly indicate code availability in the provided document. We focus on those with accessible links or confirmed releases, as they enable researchers, developers, and practitioners to replicate experiments, extend models, or integrate them into real-world systems. Table 10 summarizes publicly available toolkits, which serve as essential baselines for benchmarking new approaches.

Table 10. Summary of publicly available triage toolkits.

Name	Year	Data	Details
MultiTriage [12]	2022	Bug Reports from Eclipse & Github OSS projects	A neural network based bug triage learning model to recommend the list of developers and issue types most relevant to a new issue report. https://github.com/thazin31086/MultiTriage
LR-BKG [146]	2021	Bug Reports from Mozilla OSS projects	A learning-to-rank framework that learns to distinguish correct, erroneous and irrelevant bugcomponent assignments, based on a rich set of features derived from bug tossing knowledge graph. https://github.com/SuYanqi/LR-BKG
Wu et al. [164]	2022	Fixed bug reports from two OSS projects, namely Eclipse and Mozilla.	A spatial–temporal dynamic graph neural network (ST-DGNN) framework to improve automated bug triaging by modeling developer collaboration networks over time and predicting the most suitable bug fixers. https://github.com/ssea-lab/BugTriage/tree/master/GRCNN

Continued on next page

Table 10 – *Continued from previous page*

Name	Year	Data	Details
ADPTriage [74]	2023	Bug reports from EclipseJDT & GCC & Mozilla OSS projects	A triage model for ITS accounts for the uncertainties, which not only assigns the bugs to the most appropriate developers or postpones them to the future but also determines the assignment timing according to the likelihood of having a particular bug type in the system and possible changes in developers' schedules in the future. https://github.com/HadiJahanshahi/ADPTriage/tree/main/Toy_Example
Zhang et al. [185]	2016	Bug reports from Eclipse & GCC & Mozilla & Netbeans & OpenOffice OSS projects	An automatic approach to perform severity prediction and fixer recommendation Based on the features (*e.g.*, textural similarity and developers' experience) extracted from top-K nearest neighbours of the new bug report. https://github.com/ProgrammerCJC/SPFR
VTBA [135]	2020	Bug reports from 13 popular GitHub projects (*e.g.*, Angular.js, Rails, Elasticsearch)	An vocabulary and time-aware bug-assignment approach by matching technical terms filtered via Stack Overflow and weighting historical fixes based on recency. https://github.com/TaskAssignment/VTBA
Wayback [75]	2022	Bug reports from EclipseJDT & Mozilla & OpenOffice OSS projects	An event-replay-based approach to reconstructing historical bug triage scenarios, enabling dependency-aware and workload-balanced assignment through dynamic bug dependency graph updates. https://github.com/HadiJahanshahi/WaybackMachine
S-DABT [73]	2022	Bug reports from EclipseJDT & Mozilla & OpenOffice OSS projects	An schedule and dependency-aware bug triage approach, which utilizes integer programming and machine learning techniques to assign bugs to suitable developers. https://github.com/HadiJahanshahi/SDABT
SevPredict [11]	2024	Bug reports from 8 OSS projects (*e.g.*, Mozilla Core, Mozilla Firefox)	A GPT–2-based framework for automated bug severity prediction, which preprocesses bug report text, extracts sentiment features, and inputs these into a fine-tuned transformer model, capturing semantic and contextual patterns to generate real-time severity predictions for integration with bug tracking systems. https://huggingface.co/spaces/AliArshad/SeverityPrediction
Wang et al. [156]	2024	Bug reports from EclipseJDT & GCC & Firefox OSS projects	An empirical approach to evaluating word embedding and deep learning combinations for automated bug assignment. https://github.com/AI4BA/dl4ba
AutoAnalysis [160]	2024	ERP Incident tickets	SplitSD4X groups incidents via subgroup discovery to summarize black box explanations.https://github.com/RemilYoucef/split-sd4x
Gao et al. [54]	2018	User reviews of 6 popular apps	Proposes IDEA framework, uses AOLDA to track version-sensitive topic distribution, detects emerging app issues, and labels topics with semantics and sentiment.https://github.com/ReMine-Lab/IDEA
Di et al. [43]	2021	Reviews and versions of Android apps& code quality metrics	Classifies user reviews into PD/FR via URM, correlates app ratings with code quality metrics to identify critical user feedback.https://github.com/sealuzh/user-satisfaction
ART [149]	2024	Microservice datasets & failure cases	Proposes ART unsupervised framework, uses Transformer/GRU/GraphSAGE to model multi-dependencies, unifying AD, FT, and RCL.https://github.com/bbyldebb/ART
Sadlek et al. [132]	2025	Testbed data with Windows/Ubuntu hosts & alerts & logs from 5 attack scenarios	Proposes severity-based cyber incident triage via kill chain attack graphs, uses MulVAL to generate graphs and match alert sequences.http://dx.doi.org/10.5281/zenodo.14547668
Gao et al. [52]	2022	Reviews from 5 apps (*e.g.*, eBay, Viber)	Proposes SOLAR framework with review helpfulness prediction, topic-sentiment modeling, and multi-factor ranking to summarize useful app reviews.https://github.com/monsterLee599/SOLAR
Haering et al. [65]	2021	Problem reviews and bug reports from 4 open-source apps (*e.g.*, Firefox, VLC)	Proposes DeepMatcher, uses DistilBERT for text embedding and cosine similarity to match app reviews with bug reports.https://mast.informatik.uni-hamburg.de/replication-packages/
Sajedi [133]	2016	Bug reports from 20 large GitHub projects	An expertise-aware bug triaging approach leveraging developers' Stack Overflow activities to identify suitable assignees. https://github.com/alisajedi/BugTriaging
He et al. [68]	2021	Bug reports from Chromium, Mozilla Core & Firefox, NetBeans, and Eclipse OSS projects	An end-to-end hierarchical attention network approach for automatic bug triage. https://github.com/username1017/Bug-triage-with-HAN
Bhuyan et al. [19]	2021	Bug reports from Mozilla projects	A web browser plug-in for Google Chrome which recommends developer expertise based on the bug report. https://chromewebstore.google.com/detail/recommend-expertise/clpcpddhohohhfcnkiknfopaeikbngid?pli=1
Krasniqi [83]	2022	Bug reports from six OSS projects (Bugzilla & Jira)	An automated quality-based bug classification approach leveraging feature selection and machine learning algorithms. https://zenodo.org/record/6412840
James et al. [77]	2022	Crash reports from large-scale OSS bug repositories	An LSH- and sequential-pattern-mining–based approach for fingerprinting and clustering crash reports to identify duplicate and related bugs. https://github.com/kedjames/crashsearch-triage

Finding 3: Evaluation and Benchmarking of Triage Approaches

In evaluation, triage approaches are examined using a range of accuracy, ranking, efficiency, and user-centered metrics. Traditional metrics such as accuracy, precision, recall, and Top-k performance remain predominant, while efficiency metrics and qualitative user feedback offer complementary insights into real-world applicability. Most existing studies utilize open-source bug repositories such as Mozilla and Eclipse; however, research on incident triage remains constrained by the scarcity of industrial datasets and the lack of standardized benchmarks. These limitations underscore the necessity for unified datasets, consistent evaluation protocols, and reproducible toolkits to facilitate fair and practical comparisons across triage methods.

7 Future Trends and Opportunities

7.1 Fusion of Heterogeneous Data Sources

Triage in modern systems is inherently multimodal, drawing on diverse data sources such as logs, alerts, engineer discussions, postmortem analyses, and system documentation. Each of these sources provides a distinct perspective: logs capture fine-grained system behaviors, discussions reflect human reasoning and contextualization [160], while documentation encodes structured definitions and domain-specific knowledge. Existing approaches, however, often treat these modalities in isolation or perform naive concatenation at the representation level, leading to information loss and reduced interpretability.

Beyond textual and structured data, triage increasingly involves visual artifacts such as defect screenshots [148]. These visual elements are often indispensable for root cause analysis, as they encode operational signals that are difficult to fully convey in text. For instance, error screenshots and short video captures uploaded by beta users provide direct visual insights into the state of the operating system, offering contextual cues that are otherwise lost in textual descriptions.

However, the reliance on such visual information places a heavy burden on human triage engineers. Each submitted ticket must be carefully examined, including both text and visual attachments, before being assigned to the appropriate development team. This process is both time-intensive and expertise-dependent: experienced engineers with domain-specific knowledge of mobile operating systems and related components can typically process only around forty tickets per day [148]. The scarcity of such skilled engineers, coupled with their central role in the pipeline, often results in significant delays and bottlenecks, ultimately reducing the overall efficiency of incident management. This tension highlights a critical gap: while visual artifacts greatly enrich interpretability, they simultaneously exacerbate the challenges of scalability and automation in current triage practices.

A promising research direction lies in the systematic fusion of heterogeneous data, where the complementary strengths of different modalities are jointly leveraged. For instance, temporal correlations between log bursts and discussion timelines may yield stronger signals for incident categorization, while aligning system documentation with log templates could enable automatic mapping of raw anomalies to meaningful fault categories. Designing architectures that dynamically weight diverse inputs, adapt to noisy signals, and resolve conflicts across modalities will be essential for developing robust and interpretable triage systems.

7.2 Integration of Domain Knowledge

Despite the growing success of data-driven techniques, effective triage in real-world settings continues to rely heavily on domain expertise. Practitioners regularly employ service-specific terminology, heuristics, and taxonomies to make

timely and accurate decisions. However, most existing triage models fail to explicitly encode such structured knowledge, resulting in limited interpretability and poor generalization across projects or services.

Recent studies have begun to bridge this gap by integrating knowledge graphs, ontologies, and rule-based constraints to enhance data-driven triage. For example, Jahanshahi et al. [74] propose policy learning constrained by project-specific knowledge, while Wu et al. [164] incorporate developer collaboration graphs to capture implicit expertise and team dynamics.

Future research should focus on knowledge-enhanced triage frameworks, where structured representations, such as failure taxonomies, component hierarchies, and domain ontologies, are integrated with representation learning and LLM-based reasoning. This trend aligns with the broader movement in AIOps toward knowledge-grounded automation, where decisions are both data-driven and semantically interpretable. Knowledge-guided models inherently promote explainability, as their reasoning aligns with concepts familiar to engineers, thereby improving trust, transparency, and adoption in industrial environments.

7.3 Human-in-the-Loop and Continuous Learning

While automation continues to advance, human expertise remains indispensable in high-stakes triage scenarios. Prior research has consistently shown that practitioners' feedback is critical for refining model predictions, resolving ambiguous cases, and maintaining operational trust [27]. However, existing pipelines often treat human input as a post hoc verification step rather than a core learning signal.

Future triage frameworks should embrace human-in-the-loop (HITL) paradigms, where engineers' corrections, rationales, and annotations are continuously incorporated into model retraining processes, enabling mutual adaptation between humans and AI systems. Semi-supervised and reinforcement learning techniques could exploit weak supervision signals, such as corrective labels, ranking feedback, or textual annotations, to update models online and improve performance over time.

Developing transparent feedback interfaces and explainable reasoning mechanisms will be crucial to facilitate effective collaboration between AI and human experts. Such mechanisms not only enhance model accountability and interpretability but also foster long-term trust and continuous learning in production triage environments.

7.4 Generalizability and Model Adaptability in Software Evolution

A persistent limitation of conventional triage systems lies in their dependence on static rules or narrowly trained models, which struggle to cope with the diversity and dynamism of large-scale service environments [46]. As incident patterns evolve and architectures change, these static approaches degrade in accuracy, scalability, and adaptability [12]. While heuristic and rule-based automation has reduced manual effort to some extent, it remains inadequate for capturing the complex and multifaceted nature of real-world failure contexts.

Recent studies advocate for continuous model adaptation and cross-project generalization, leveraging techniques such as domain adaptation, transfer learning, and incremental fine-tuning [146]. Future research should emphasize rigorous benchmarking across heterogeneous environments and issue types, accompanied by standardized evaluation frameworks that reflect evolving operational realities. Moreover, integrating adaptive feedback loops and hybrid designs, combining LLM reasoning with domain-specific knowledge graphs or causal inference models, holds promise for creating triage systems that are intelligent, resilient, and scalable.

8 Conclusion

With the increasing scale and complexity of software and service systems, as well as the continuous advancement of deployment technologies, triage has emerged as a critical research frontier in software engineering. This survey provides a comprehensive review of 234 studies on software triage published between 2004 and 2025, summarizing existing system architectures and technical approaches. Through an extensive analysis of prior research, we outlined the major progress achieved in triage automation and examined the empirical differences between academic research and industrial practice. To promote further research and practical adoption, we consolidated widely used datasets and evaluation metrics that establish a consistent framework for assessing triage effectiveness. The survey also identified several promising directions for future work, emphasizing the importance of closer collaboration between academia and industry. Our goal is to contribute to the advancement of modern software systems by offering researchers a comprehensive reference and valuable insights that may inspire future exploration in triage technologies.

References

[1] 2025. DBLP. https://dblp.org/

[2] Syed Nadeem Ahsan, Javed Ferzund, and Franz Wotawa. 2009. Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In *2009 Fourth International Conference on Software Engineering Advances*. IEEE, 216–221.

[3] V Akila, G Zayaraz, and V Govindasamy. 2014. Bug triage in open source systems: a review. *International Journal of Collaborative Enterprise* 4, 4 (2014), 299–319.

[4] Jalal Sadoon Hameed Al-Bayati, Mohammed Al-Shamma, and Furat Nidhal Tawfeeq. 2024. Enhancement of Recommendation Engine Technique for Bug System Fixes. *Journal of Advances in Information Technology* 15, 4 (2024).

[5] Iyad Alazzam, Ahmed Aleroud, Zainab Al Latifah, and George Karabatis. 2020. Automatic bug triage in software systems using graph neighborhood relations for feature augmentation. *IEEE Transactions on Computational Social Systems* 7, 5 (2020), 1288–1303.

[6] Mamdouh Alenezi, Kenneth Magel, and Shadi Banitaan. 2013. Efficient Bug Triaging Using Text Mining. *J. Softw.* 8, 9 (2013), 2185–2190.

[7] Rafi Almhana and Marouane Kessentini. 2021. Considering dependencies between bug reports to improve bugs triage. *Automated Software Engineering* 28, 1 (2021), 1.

[8] John Anvik. 2006. Automating bug report assignment. In *Proceedings of the 28th international conference on Software engineering*. 937–940.

[9] John Anvik, Lyndon Hiew, and Gail C Murphy. 2006. Who should fix this bug?. In *Proceedings of the 28th international conference on Software engineering*. 361–370.

[10] John Anvik and Gail C Murphy. 2011. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20, 3 (2011), 1–35.

[11] Muhammad Ali Arshad, Adnan Riaz, Rubia Fatima, and Affan Yasin. 2024. SevPredict: Exploring the Potential of Large Language Models in Software Maintenance. *AI* 5, 4 (2024), 2739–2760.

[12] Thazin Win Win Aung, Yao Wan, Huan Huo, and Yulei Sui. 2022. Multi-triage: A multi-task learning framework for bug triage. *Journal of Systems and Software* 184 (2022), 111133.

[13] Shadi Banitaan and Mamdouh Alenezi. 2013. Decoba: Utilizing developers communities in bug assignment. In *2013 12th International Conference on Machine Learning and Applications*, Vol. 2. IEEE, 66–71.

[14] Chetan Bansal, Sundararajan Renganathan, Ashima Asudani, Olivier Midy, and Mathru Janakiraman. 2020. DeCaf: diagnosing and triaging performance issues in large-scale cloud services. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice* (Seoul, South Korea) *(ICSE-SEIP '20)*. Association for Computing Machinery, New York, NY, USA, 201–210. doi:10.1145/3377813.3381353

[15] Olga Baysal, Michael W Godfrey, and Robin Cohen. 2009. A bug you like: A framework for automated assignment of bugs. In *2009 IEEE 17th International Conference on Program Comprehension*. IEEE, 297–298.

[16] Mario Luca Bernardi, Gerardo Canfora, Giuseppe A. Di Lucca, Massimiliano Di Penta, and Damiano Distante. 2018. The relation between developers' communication and fix-Inducing changes: An empirical study. *Journal of Systems and Software* 140 (2018), 111–125. doi:10.1016/j.jss.2018.02.065

[17] Pamela Bhattacharya and Iulian Neamtiu. 2010. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *2010 IEEE International Conference on Software Maintenance*. 1–10. doi:10.1109/ICSM.2010.5609736

[18] Pamela Bhattacharya, Iulian Neamtiu, and Christian R Shelton. 2012. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software* 85, 10 (2012), 2275–2292.

[19] Shayla Azad Bhuyan and John Anvik. 2021. Evaluating Visual Explanation of Bug Report Assignment Recommendations (S).. In *SEKE*. 334–339.

[20] Razvan Bocu, Alexandra Baicoianu, and Arpad Kerestely. 2023. An Extended Survey Concerning the Significance of Artificial Intelligence and Machine Learning Techniques for Bug Triage and Management. *IEEE Access* 11 (2023), 123924–123937. doi:10.1109/ACCESS.2023.3329732

[21] Markus Borg, Leif Jonsson, Emelie Engström, Béla Bartalos, and Attila Szabó. 2024. Adopting automated bug assignment in practice—a longitudinal case study at Ericsson. *Empirical Software Engineering* 29, 5 (2024), 126.

[22] Gerald Bortis and André van der Hoek. 2013. PorchLight: A tag-based approach to bug triaging. In *2013 35th International Conference on Software Engineering (ICSE)*. 342–351. doi:10.1109/ICSE.2013.6606580

[23] Yegor Bugayenko, Ayomide Bakare, Arina Cheverda, Mirko Farina, Artem Kruglov, Yaroslav Plaksin, Giancarlo Succi, and Witold Pedrycz. 2022. Automatically Prioritizing and Assigning Tasks from Code Repositories in Puzzle Driven Development. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*. 722–723. doi:10.1145/3524842.3528512

[24] Omid Bushehrian and Ziba Ghane. 2023. Code quality control by bug report classification. *Software Quality Journal* 31, 3 (2023), 991–1007.

[25] Gemma Catolino, Fabio Palomba, Andy Zaidman, and Filomena Ferrucci. 2019. Not all bugs are the same: Understanding, characterizing, and classifying bug types. *Journal of Systems and Software* 152 (2019), 165–181.

[26] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 111–120.

[27] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous Incident Triage for Large-Scale Online Service Systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 364–375. doi:10.1109/ASE.2019.00042

[28] Junjie Chen, Jibesh Patra, Michael Pradel, Yingfei Xiong, Hongyu Zhang, Dan Hao, and Lu Zhang. 2020. A survey of compiler testing. *ACM Computing Surveys (CSUR)* 53, 1 (2020), 1–36.

[29] Jia Chen, Peng Wang, and Wei Wang. 2022. Online summarizing alerts through semantic and behavior information. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1646–1657. doi:10.1145/3510003.3510055

[30] Junjie Chen, Shu Zhang, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Yu Kang, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2021. How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (Virtual Event, Australia) *(ASE '20)*. Association for Computing Machinery, New York, NY, USA, 373–384. doi:10.1145/3324884.3416624

[31] Ning Chen, Jialiu Lin, Steven CH Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th international conference on software engineering*. 767–778.

[32] Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, Zhangwei Xu, and Dongmei Zhang. 2020. Identifying linked incidents in large-scale online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 304–314. doi:10.1145/3368089.3409768

[33] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, Yingnong Dang, Feng Gao, Pu Zhao, Bo Qiao, Qingwei Lin, Dongmei Zhang, and Michael R. Lyu. 2020. 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* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1487–1497. doi:10.1145/3368089.3417055

[34] Zhuangbin Chen, Jinyang Liu, Yuxin Su, Hongyu Zhang, Xuemin Wen, Xiao Ling, Yongqiang Yang, and Michael R. Lyu. 2021. Graph-based Incident Aggregation for Large-Scale Online Service Systems. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 430–442. doi:10.1109/ASE51524.2021.9678746

[35] Michael Christ, Florian Grossmann, Daniela Winter, Roland Bingisser, and Elke Platz. 2010. Modern triage in the emergency department. *Deutsches Ärzteblatt International* 107, 50 (2010), 892.

[36] Grzegorz Chrupała. 2012. Learning from evolving data streams: online triage of bug reports. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. 613–622.

[37] Byron Cook, Björn Döbel, Daniel Kroening, Norbert Manthey, Martin Pohlack, Elizabeth Polgreen, Michael Tautschnig, and Pawel Wieczorkiewicz. 2020. Using model checking tools to triage the severity of security bugs in the Xen hypervisor. In *# PLACEHOLDER_PARENT_METADATA_VALUE#*, Vol. 1. TU Wien Academic Press, 185–193.

[38] Jie Dai, Qingshan Li, Shenglong Xie, Daizhen Li, and Hua Chu. 2024. PCG: A joint framework of graph collaborative filtering for bug triaging. *Journal of Software: Evolution and Process* 36, 9 (2024), e2673.

[39] Jie Dai, Qingshan Li, Hui Xue, Zhao Luo, Yinglin Wang, and Siyuan Zhan. 2023. Graph collaborative filtering-based bug triaging. *Journal of Systems and Software* 200 (2023), 111667.

[40] Anh-Hien Dao and Cheng-Zen Yang. 2023. Automated Priority Prediction for Bug Reports Using Comment Intensiveness Features and SMOTE Data Balancing. *International Journal of Software Engineering and Knowledge Engineering* 33, 03 (2023), 415–433. doi:10.1142/S021819402350002X

[41] Disha Devaiya, John Anvik, Meher Bheree, and Farjana Yeasmin Omee. 2021. Evaluating a Tool for Creating Bug Report Assignment Recommenders (S).. In *SEKE*. 271–274.

[42] Disha Thakarshibhai Devaiya, John Anvik, Farjana Yeasmin Omee, and Meher Bheree. 2021. CASTR: Assisting Bug Report Assignment Recommender Creation.. In *SEKE*. 639.

[43] 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. *Journal of Software: Evolution and Process* 33, 3 (2021), e2316.

[44] Rui Ding, Qiang Fu, Jian Guang Lou, Qingwei Lin, Dongmei Zhang, and Tao Xie. 2014. Mining historical issue repositories to heal large-scale online service systems. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 311–322.

[45] Atish Kumar Dipongkor, Md Saiful Islam, Ishtiaque Hussain, Sira Yongchareon, and Sajib Mistry. 2023. On fusing artificial and convolutional neural network features for automatic bug assignments. *IEEE Access* 11 (2023), 49493–49508.

[46] Atish Kumar Dipongkor and Kevin Moran. 2023. A comparative study of transformer-based neural text representation techniques on bug triaging. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1012–1023.

[47] Haozhen Dong, Hongmin Ren, Jialiang Shi, Yichen Xie, and Xudong Hu. 2024. Neighborhood contrastive learning-based graph neural network for bug triaging. *Science of Computer Programming* 235 (2024), 103093.

[48] Layan Etaiwi, Sylvie Hamel, Yann-Gaël Guéhéneuc, William Flageol, and Rodrigo Morales. 2020. Order in chaos: prioritizing mobile app reviews using consensus algorithms. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 912–920.

[49] Vahid Etemadi, Omid Bushehrian, Reza Akbari, and Gregorio Robles. 2021. A scheduling-driven approach to efficiently assign bug fixing tasks to developers. *Journal of Systems and Software* 178 (2021), 110967.

[50] Gerard FitzGerald, George A Jelinek, Deborah Scott, and Marie Frances Gerdtz. 2010. Emergency department triage revisited. *Emergency Medicine Journal* 27, 2 (2010), 86–92.

[51] Adrian-Cătălin Florea, John Anvik, and Răzvan Andonie. 2017. Parallel implementation of a bug report assignment recommender using deep learning. In *International Conference on artificial neural networks*. Springer, 64–71.

[52] Cuiyun Gao, Yaoxian Li, Shuhan Qi, Yang Liu, Xuan Wang, Zibin Zheng, and Qing Liao. 2022. Listening to users' voice: Automatic summarization of helpful app reviews. *IEEE Transactions on Reliability* 72, 4 (2022), 1619–1631.

[53] Cuiyun Gao, Baoxiang Wang, Pinjia He, Jieming Zhu, Yangfan Zhou, and Michael R Lyu. 2015. PAID: Prioritizing app issues for developers by tracking user reviews over versions. In *2015 IEEE 26th international symposium on software reliability engineering (ISSRE)*. IEEE, 35–45.

[54] Cuiyun Gao, Jichuan Zeng, Michael R Lyu, and Irwin King. 2018. Online app review analysis for identifying emerging issues. In *Proceedings of the 40th international conference on software engineering*. 48–58.

[55] Jiaqi Gao, Nofel Yaseen, Robert MacDavid, Felipe Vieira Frujeri, Vincent Liu, Ricardo Bianchini, Ramaswamy Aditya, Xiaohang Wang, Henry Lee, David Maltz, Yu Minlan, and Arzani Behnaz. 2020. Scouts: Improving the diagnosis process through domain-customized incident routing. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 253–269.

[56] Supriyo Ghosh, Karish Grover, Jimmy Wong, Chetan Bansal, Rakesh Namineni, Mohit Verma, and Saravan Rajmohan. 2024. Dependency Aware Incident Linking in Large Cloud Systems. In *Companion Proceedings of the ACM Web Conference 2024* (Singapore, Singapore) *(WWW '24)*. Association for Computing Machinery, New York, NY, USA, 141–150. doi:10.1145/3589335.3648311

[57] Drishti Goel, Fiza Husain, Aditya Singh, Supriyo Ghosh, Anjaly Parayil, Chetan Bansal, Xuchao Zhang, and Saravan Rajmohan. 2024. X-lifecycle learning for cloud incident management using llms. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 417–428.

[58] Anjali Goyal. 2017. Effective Bug Triage for Non-Reproducible Bugs. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 487–488. doi:10.1109/ICSE-C.2017.41

[59] Anjali Goyal and Neetu Sardana. 2024. An Integrated Approach using Developer Profiles with Temporal Dynamics for Assignee Recommendation in Non-Reproducible Bugs. *Procedia Computer Science* 235 (2024), 2833–2842.

[60] Jiazhen Gu, Jiaqi Wen, Zijian Wang, Pu Zhao, Chuan Luo, Yu Kang, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, Bo Qiao, Liqun Li, Qingwei Lin, and Dongmei Zhang. 2020. Efficient customer incident triage via linking with system incidents. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1296–1307. doi:10.1145/3368089.3417061

[61] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2011. "Not my bug!" and other reasons for software bug report reassignments. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work* (Hangzhou, China) *(CSCW '11)*. Association for Computing Machinery, New York, NY, USA, 395–404. doi:10.1145/1958824.1958887

[62] Shikai Guo, Xinyi Zhang, Xi Yang, Rong Chen, Chen Guo, Hui Li, and Tingting Li. 2020. Developer activity motivated bug triaging: via convolutional neural network. *Neural Processing Letters* 51, 3 (2020), 2589–2606.

[63] Chetna Gupta and Mário M Freire. 2021. A decentralized blockchain oriented framework for automated bug assignment. *Information and Software Technology* 134 (2021), 106540.

[64] Rajeev Gupta, K Hima Prasad, Laura Luan, Daniela Rosu, and Chris Ward. 2009. Multi-dimensional knowledge integration for efficient incident management in a services cloud. In *2009 IEEE International Conference on Services Computing*. IEEE, 57–64.

[65] Marlo Haering, Christoph Stanik, and Walid Maalej. 2021. Automatically matching bug reports with related app reviews. In *2021 IEEE/ACM 43rd international conference on software engineering (ICSE)*. IEEE, 970–981.

[66] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2011. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering* 38, 6 (2011), 1276–1304.

[67] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. *Network and Distributed Systems Security Symposium* (2019). https://par.nsf.gov/biblio/10085663

[68] Huoliang He and ShunKun Yang. 2021. Automatic Bug Triage Using Hierarchical Attention Networks. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 1043–1049. doi:10.1109/QRS-C55045.2021.00158

[69] Seyedrebvar Hosseini, Burak Turhan, and Dimuthu Gunarathna. 2017. A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering* 45, 2 (2017), 111–147.

[70] Hao Hu, Hongyu Zhang, Jifeng Xuan, and Weigang Sun. 2014. Effective bug triage based on historical bug-fix information. In *2014 IEEE 25th international symposium on software reliability engineering*. IEEE, 122–132.

[71] Junjie Huang, Jinyang Liu, Zhuangbin Chen, Zhihan Jiang, Yichen Li, Jiazhen Gu, Cong Feng, Zengyin Yang, Yongqiang Yang, and Michael R Lyu. 2024. Faultprofit: Hierarchical fault profiling of incident tickets in large-scale cloud systems. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. 392–404.

[72] Jinxiao Huang and Yutao Ma. 2019. Predicting the fixer of software bugs via a collaborative multiplex network: Two case studies. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 469–488.

[73] Hadi Jahanshahi and Mucahit Cevik. 2022. S-DABT: Schedule and dependency-aware bug triage in open-source bug tracking systems. *Information and Software Technology* 151 (2022), 107025.

[74] Hadi Jahanshahi, Mucahit Cevik, Kianoush Mousavi, and Ayşe Başar. 2023. ADPTriage: Approximate dynamic programming for bug triage. *IEEE Transactions on Software Engineering* 49, 10 (2023), 4594–4609.

[75] Hadi Jahanshahi, Mucahit Cevik, José Navas-Sú, Ayşe Başar, and Antonio González-Torres. 2022. Wayback Machine: A tool to capture the evolutionary behavior of the bug reports and their triage process in open-source software systems. *Journal of Systems and Software* 189 (2022), 111308.

[76] Hadi Jahanshahi, Kritika Chhabra, Mucahit Cevik, and Ayþe Baþar. 2021. DABT: A dependency-aware bug triaging method. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*. 221–230.

[77] Kedrian James, Yufei Du, Sanjeev Das, and Fabian Monrose. 2022. Separating the Wheat from the Chaff: Using Indexing and Sub-Sequence Mining Techniques to Identify Related Crashes During Bug Triage. In *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*. 31–42. doi:10.1109/QRS57517.2022.00014

[78] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 111–120.

[79] Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson. 2016. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering* 21, 4 (2016), 1533–1578.

[80] Jaweria Kanwal and Onaiza Maqbool. 2012. Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology* 27, 2 (2012), 397–412.

[81] Yutaro Kashiwa. 2019. RAPTOR: Release-aware and prioritized bug-fixing task assignment optimization. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 629–633.

[82] Barbara Kitchenham. 2004. Procedures for performing systematic reviews. *Keele, UK, Keele University* 33, 2004 (2004), 1–26.

[83] Rrezarta Krasniqi and Hyunsook Do. 2022. Automatically Capturing Quality-Related Concerns in Bug Report Descriptions for Efficient Bug Triaging. In *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering* (Gothenburg, Sweden) *(EASE '22)*. Association for Computing Machinery, New York, NY, USA, 10–19. doi:10.1145/3530019.3530021

[84] Jinxi Kuang, Jinyang Liu, Junjie Huang, Renyi Zhong, Jiazhen Gu, Lan Yu, Rui Tan, Zengyin Yang, and Michael R. Lyu. 2024. Knowledge-aware Alert Aggregation in Large-scale Cloud Systems: a Hybrid Approach. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice* (Lisbon, Portugal) *(ICSE-SEIP '24)*. Association for Computing Machinery, New York, NY, USA, 369–380. doi:10.1145/3639477.3639745

[85] Atish Kumar Dipongkor. 2024. An Ensemble Method for Bug Triaging using Large Language Models. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings* (Lisbon, Portugal) *(ICSE-Companion '24)*. Association for Computing Machinery, New York, NY, USA, 438–440. doi:10.1145/3639478.3641228

[86] Ahmed Lamkanfi, Javier Perez, and Serge Demeyer. 2013. The Eclipse and Mozilla Defect Tracking Dataset: a Genuine Dataset for Mining Bug Information. In *MSR '13: Proceedings of the 10th Working Conference on Mining Software Repositories, May 18–-19, 2013. San Francisco, California, USA*.

[87] Jaehyung Lee, Kisun Han, and Hwanjo Yu. 2022. A light bug triage framework for applying large pre-trained language model. In *Proceedings of the 37th IEEE/ACM international conference on automated software engineering*. 1–11.

[88] Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. 2017. Applying deep learning based automatic bug triager to industrial projects. In *Proceedings of the 2017 11th Joint Meeting on foundations of software engineering*. 926–931.

[89] Liqun Li, Xu Zhang, Xin Zhao, Hongyu Zhang, Yu Kang, Pu Zhao, Bo Qiao, Shilin He, Pochian Lee, Jeffrey Sun, Feng Gao, Li Yang, Qingwei Lin, Saravanakumar Rajmohan, Zhangwei Xu, and Dongmei Zhang. 2021. Fighting the Fog of War: Automated Incident Detection for Cloud Systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 131–146. https://www.usenix.org/conference/atc21/presentation/li-liqun

[90] Zexuan Li and Kaixin Huang. 2024. Automatic bug assignments without texts: a study. *Frontiers of Computer Science* 18, 4 (2024), 184210.

[91] Zexuan Li and Hao Zhong. 2021. Revisiting textual feature of bug-triage approach. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1183–1185.

[92] Meng-Hui Lim, Jian-Guang Lou, Hongyu Zhang, Qiang Fu, Andrew Beng Jin Teoh, Qingwei Lin, Rui Ding, and Dongmei Zhang. 2014. Identifying Recurrent and Unknown Performance Issues. In *2014 IEEE International Conference on Data Mining*. 320–329. doi:10.1109/ICDM.2014.96

[93] Derek Lin, Rashmi Raghu, Vivek Ramamurthy, Jin Yu, Regunathan Radhakrishnan, and Joseph Fernandez. 2014. Unveiling clusters of events for alert and incident management in large-scale enterprise it. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, New York, USA) *(KDD '14)*. Association for Computing Machinery, New York, NY, USA, 1630–1639. doi:10.1145/2623330.2623360

[94] Tao Lin. 2018. A data triage retrieval system for cyber security operations center. (2018).

[95] Ying Lin, Zhengzhang Chen, Cheng Cao, Lu-An Tang, Kai Zhang, Wei Cheng, and Zhichun Li. 2018. Collaborative Alert Ranking for Anomaly Detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (Torino, Italy) *(CIKM '18)*. Association for Computing Machinery, New York, NY, USA, 1987–1995. doi:10.1145/3269206.3272013

[96] Zhongpeng Lin, Fengdi Shu, Ye Yang, Chenyong Hu, and Qing Wang. 2009. An empirical study on bug assignment automation using chinese bug data. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 451–455.

[97] Mario Linares-Vásquez, Kamal Hossen, Hoang Dang, Huzefa Kagdi, Malcom Gethers, and Denys Poshyvanyk. 2012. Triaging incoming change requests: Bug or commit history, or code authorship?. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. 451–460. doi:10.1109/ICSM.2012.6405306

[98] Jinyang Liu, Shilin He, Zhuangbin Chen, Liqun Li, Yu Kang, Xu Zhang, Pinjia He, Hongyu Zhang, Qingwei Lin, Zhangwei Xu, Saravan Rajmohan, Dongmei Zhang, and Michael R. Lyu. 2023. Incident-aware Duplicate Ticket Aggregation for Cloud Systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 2299–2311. doi:10.1109/ICSE48619.2023.00193

[99] Jin Liu, Yiqiuzi Tian, Xiao Yu, Zhijiang Yang, Xiangyang Jia, Chuanxiang Ma, and Zheng Xu. 2016. A multi-source approach for bug triage. *International Journal of Software Engineering and Knowledge Engineering* 26, 09n10 (2016), 1593–1604.

[100] Zhexiong Liu, Cris Benge, and Siduo Jiang. 2023. Ticket-bert: Labeling incident management tickets with language models. *arXiv preprint arXiv:2307.00108* (2023).

[101] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2013. Software analytics for incident management of online services: An experience report. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 475–485. doi:10.1109/ASE.2013.6693105

[102] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2017. Experience report on applying software analytics in incident management of online service. *automated software engineering* 24, 4 (2017), 905–941.

[103] Shiva Theja Lyakajigari, Sindhu Lakkavtri, Sai Mehar Ankathi, Ramu Kuchipudi, and Ramakrishna Kolikipogu. 2024. review on Automated software Bug Triage system. In *2024 International Conference on Healthcare Innovations, Software and Engineering Technologies (HISET)*. IEEE, 346–349.

[104] Saurabh Malgaonkar, Sherlock A Licorish, and Bastin Tony Roy Savarimuthu. 2022. Prioritizing user concerns in app reviews–A study of requests for new features, enhancements and bug fixes. *Information and Software Technology* 144 (2022), 106798.

[105] Atri Mandal, Shivali Agarwal, Nikhil Malhotra, Giriprasad Sridhara, Anupama Ray, and Daivik Swarup. 2019. Improving it support by enhancing incident management process with multi-modal analysis. In *International Conference on Service-Oriented Computing*. Springer, 431–446.

[106] Senthil Mani, Anush Sankaran, and Rahul Aralikatte. 2019. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In *Proceedings of the ACM India joint international conference on data science and management of data*. 171–179.

[107] Dominique Matter, Adrian Kuhn, and Oscar Nierstrasz. 2009. Assigning bug reports using a vocabulary-based expertise model of developers. In *2009 6th IEEE International Working Conference on Mining Software Repositories*. 131–140. doi:10.1109/MSR.2009.5069491

[108] Jyoti Prakash Meher, Sourav Biswas, and Rajib Mall. 2024. Deep learning-based software bug classification. *Information and Software Technology* 166 (2024), 107350.

[109] Hufsa Mohsin, Chongyang Shi, Shufeng Hao, and He Jiang. 2022. SPAN: A self-paced association augmentation and node embedding-based model for software bug classification and assignment. *Knowledge-Based Systems* 236 (2022), 107711. doi:10.1016/j.knosys.2021.107711

[110] G Murphy and Davor Cubranic. 2004. Automatic bug triage using text categorization. In *Proceedings of the sixteenth international conference on software engineering & knowledge engineering*. Citeseer, 1–6.

[111] Hoda Naguib, Nitesh Narayan, Bernd Brügge, and Dina Helal. 2013. Bug report assignee recommendation using activity profiles. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 22–30.

[112] Naresh Kumar Nagwani and Jasjit S. Suri. 2023. An artificial intelligence framework on software bug triaging, technological evolution, and future challenges: A review. *International Journal of Information Management Data Insights* 3, 1 (2023), 100153. doi:10.1016/j.jjimei.2022.100153

[113] Vaskar Nath, David Sheldon, and John Alphonso-Gibbs. 2021. Principal component analysis and entropy-based selection for the improvement of bug triage. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 541–546.

[114] Behzad Soleimani Neysiani, Seyed Morteza Babamir, and Masayoshi Aritsugi. 2020. Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems. *Information and Software Technology* 126 (2020), 106344.

[115] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. 2021. A Survey of AIOps Methods for Failure Management. *ACM Trans. Intell. Syst. Technol.* 12, 6, Article 81 (Nov. 2021), 45 pages. doi:10.1145/3483424

[116] Cícero Augusto De Lara Pahins, Fabrício D'Morison, Thiago M Rocha, Larissa M Almeida, Arthur F Batista, and Diego F Souza. 2019. T-REC: Towards accurate bug triage for technical groups. In *2019 18th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 889–895.

[117] Fabio Palomba, Mario Linares-Vásquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2015. User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 291–300.

[118] Rama Ranjan Panda and Naresh Kumar Nagwani. 2022. Topic modeling and intuitionistic fuzzy set-based approach for efficient software bug triaging. *Knowledge and Information Systems* 64, 11 (2022), 3081–3111.

[119] Rama Ranjan Panda and Naresh Kumar Nagwani. 2024. Software bug priority prediction technique based on intuitionistic fuzzy representation and class imbalance learning. *Knowledge and Information Systems* 66, 3 (2024), 2135–2164.

[120] Jin-woo Park, Mu-Woong Lee, Jinhan Kim, Seung-won Hwang, and Sunghun Kim. 2011. Costriage: A cost-aware triage algorithm for bug reporting systems. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 25. 139–144.

[121] Jin-woo Park, Mu-Woong Lee, Jinhan Kim, Seung-won Hwang, and Sunghun Kim. 2016. Cost-aware triage ranking algorithms for bug reporting systems. *Knowledge and Information Systems* 48, 3 (2016), 679–705.

[122] Avinash Patil. 2025. GitBugs: Bug Reports for Duplicate Detection, Retrieval Augmented Generation, Triage, and More. *arXiv preprint arXiv:2504.09651* (2025).

[123] Xinyu Peng, Pingyi Zhou, Jin Liu, and Xu Chen. 2017. Improving Bug Triage with Relevant Search.. In *SEKE*. 123–128.

[124] Phuong Pham, Vivek Jain, Lukas Dauterman, Justin Ormont, and Navendu Jain. 2020. Deeptriage: Automated transfer assistance for incidents in cloud services. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3281–3289.

[125] Kittisak Phetrungnapha and Twittie Senivongse. 2019. Classification of mobile application user reviews for generating tickets on issue tracking system. In *2019 12th International Conference on Information & Communication Technology and System (ICTS)*. IEEE, 229–234.

[126] Cheng Qian, Ming Zhang, Yuanping Nie, Shuaibing Lu, and Huayang Cao. 2023. A survey on bug deduplication and triage methods from multiple points of view. *Applied Sciences* 13, 15 (2023), 8788.

[127] Md Mainur Rahman, Guenther Ruhe, and Thomas Zimmermann. 2009. Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 439–442.

[128] Youcef Remil, Anes Bendimerad, Romain Mathonat, and Mehdi Kaytoue. 2024. Aiops solutions for incident management: Technical guidelines and a comprehensive literature review. *arXiv preprint arXiv:2404.01363* (2024).

[129] Youcef Remil, Anes Bendimerad, Marc Plantevit, Céline Robardet, and Mehdi Kaytoue. 2021. Interpretable summaries of black box incident triaging with subgroup discovery. In *2021 IEEE 8TH international Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 1–10.

[130] Iain Robertson-Steel. 2006. Evolution of triage systems. *Emergency medicine journal* 23, 2 (2006), 154–155.

[131] Korosh Koochekian Sabor, Abdelwahab Hamou-Lhadj, and Alf Larsson. 2017. Durfex: a feature extraction technique for efficient detection of duplicate bug reports. In *2017 IEEE international conference on software quality, reliability and security (QRS)*. IEEE, 240–250.

[132] Lukáš Sadlek, Muhammad Mudassar Yamin, Pavel Čeleda, and Basel Katt. 2025. Severity-based triage of cybersecurity incidents using kill chain attack graphs. *Journal of Information Security and Applications* 89 (2025), 103956.

[133] Ali Sajedi Badashian, Abram Hindle, and Eleni Stroulia. 2016. Crowdsourced Bug Triaging: Leveraging Q&A Platforms for Bug Assignment. In *Fundamental Approaches to Software Engineering*, Perdita Stevens and Andrzej Wąsowski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 231–248.

[134] Ali Sajedi-Badashian and Eleni Stroulia. 2020. Guidelines for evaluating bug-assignment research. *Journal of Software: Evolution and Process* 32, 9 (2020), e2250.

[135] Ali Sajedi-Badashian and Eleni Stroulia. 2020. Vocabulary and time based bug-assignment: A recommender system for open-source projects. *Software: Practice and Experience* 50, 8 (2020), 1539–1564.

[136] Mina Samir, Nada Sherief, and Walid Abdelmoez. 2023. Improving bug assignment and developer allocation in software engineering through interpretable machine learning models. *Computers* 12, 7 (2023), 128.

[137] Aindrila Sarkar, Peter C Rigby, and Béla Bartalos. 2019. Improving bug triaging with high confidence predictions at ericsson. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 81–91.

[138] Reza Sepahvand, Reza Akbari, Behnaz Jamasb, Sattar Hashemi, and Omid Boushehrian. 2023. Using word embedding and convolution neural network for bug triaging by considering design flaws. *Science of Computer Programming* 228 (2023), 102945.

[139] Francisco Servant and James A. Jones. 2012. WhoseFault: Automatic developer-to-fault assignment through fault localization. In *2012 34th International Conference on Software Engineering (ICSE)*. 36–46. doi:10.1109/ICSE.2012.6227208

[140] Qihong Shao, Yi Chen, Shu Tao, Xifeng Yan, and Nikos Anerousis. 2008. Efficient ticket routing by resolution sequence mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Las Vegas, Nevada, USA) *(KDD '08)*. Association for Computing Machinery, New York, NY, USA, 605–613. doi:10.1145/1401890.1401964

[141] Ramin Shokripour, John Anvik, Zarinah M Kasirun, and Sima Zamani. 2013. Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. In *2013 10th working conference on mining software repositories (MSR)*. IEEE, 2–11.

[142] Ramin Shokripour, John Anvik, Zarinah M Kasirun, and Sima Zamani. 2015. A time-based approach to automatic bug report assignment. *Journal of Systems and Software* 102 (2015), 109–122.

[143] Neetu Singh and Sandeep Kumar Singh. 2023. An Empirical Assessment of the Performance of Multi-Armed Bandits and Contextual Multi-Armed Bandits in Handling Cold-Start Bugs. In *Proceedings of the 2023 Fifteenth International Conference on Contemporary Computing*. 750–758.

[144] Neetu Singh and Sandeep Kumar Singh. 2025. Navigating bug cold start with contextual multi-armed bandits: an enhanced approach to developer assignment in software bug repositories. *Automated Software Engineering* 32, 2 (2025), 39.

[145] Yanqi Su, Zheming Han, Zhipeng Gao, Zhenchang Xing, Qinghua Lu, and Xiwei Xu. 2023. Still confusing for bug-component triaging? Deep feature learning and ensemble setting to rescue. In *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*. IEEE, 316–327.

[146] Yanqi Su, Zhenchang Xing, Xin Peng, Xin Xia, Chong Wang, Xiwei Xu, and Liming Zhu. 2021. Reducing bug triaging confusion by learning from mistakes with a bug tossing knowledge graph. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 191–202.

[147] Xiaobing Sun, Hui Yang, Xin Xia, and Bin Li. 2017. Enhancing developer recommendation with supplementary information via mining historical commits. *Journal of Systems and Software* 134 (2017), 355–368.

[148] Yongqian Sun, Bowen Hao, Xiaotian Wang, Chenyu Zhao, Yongxin Zhao, Binpeng Shi, Shenglin Zhang, Qiao Ge, Wenhu Li, Hua Wei, and Dan Pei. 2025. LLM-Augmented Ticket Aggregation for Low-cost Mobile OS Defect Resolution. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering* (Clarion Hotel Trondheim, Trondheim, Norway) *(FSE Companion '25)*. Association for Computing Machinery, New York, NY, USA, 215–226. doi:10.1145/3696630.3728547

[149] Yongqian Sun, Binpeng Shi, Mingyu Mao, Minghua Ma, Sibo Xia, Shenglin Zhang, and Dan Pei. 2024. Art: A unified unsupervised framework for incident management in microservice systems. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1183–1194.

[150] Ashish Sureka, Himanshu Singh, Manjunat Bagewadi, Abhishek Mitra, and Rohit Karanth. 2015. A Decision Support Platform for Guiding a Bug Triage for Resolver Recommendation Using Textual and Non-Textual Features.. In *QuASoQ/WAWSE/CMCE@ APSEC*. 23–30.

[151] Ahmed Tamrawi, Tung Thanh Nguyen, Jafar M. Al-Kofahi, and Tien N. Nguyen. 2011. Fuzzy set and cache-based approach for bug triaging. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (Szeged, Hungary) *(ESEC/FSE '11)*. Association for Computing Machinery, New York, NY, USA, 365–375. doi:10.1145/2025113.2025163

[152] Yuan Tian, Dinusha Wijedasa, David Lo, and Claire Le Goues. 2016. Learning to rank for bug report assignee recommendation. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. IEEE, 1–10.

[153] KM Uddin. 2024. *A NOVEL BUG TRIAGING STRATEGY USING DEVELOPER RECOMMENDATION AND LOAD BALANCING MODEL*. Ph. D. Dissertation. © University of Dhaka.

[154] Hongbing Wang and Qi Li. 2021. Effective bug triage based on a hybrid neural network. In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 82–91.

[155] Rongcun Wang, Xingyu Ji, Yuan Tian, Senlei Xu, Xiaobing Sun, and Shujuan Jiang. 2025. Fixer-level supervised contrastive learning for bug assignment. *Empirical Software Engineering* 30, 3 (2025), 76.

[156] Rongcun Wang, Xingyu Ji, Senlei Xu, Yuan Tian, Shujuan Jiang, and Rubing Huang. 2024. An empirical assessment of different word embedding and deep learning models for bug assignment. *Journal of Systems and Software* 210 (2024), 111961.

[157] Shance Wang, Zhongjie Wang, Xiaofei Xu, and Quan Z Sheng. 2017. App update patterns: How developers act on user reviews in mobile app stores. In *International Conference on Service-Oriented Computing*. Springer, 125–141.

[158] Song Wang, Wen Zhang, and Qing Wang. 2014. FixerCache: unsupervised caching active developers for diverse bug triage. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Torino, Italy) *(ESEM '14)*. Association for Computing Machinery, New York, NY, USA, Article 25, 10 pages. doi:10.1145/2652524.2652536

[159] Yaohui Wang, Guozheng Li, Zijian Wang, Yu Kang, Yangfan Zhou, Hongyu Zhang, Feng Gao, Jeffrey Sun, Li Yang, Pochian Lee, et al. 2021. Fast outage analysis of large-scale production clouds with service correlation mining. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 885–896.

[160] Zexin Wang, Jianhui Li, Minghua Ma, Ze Li, Yu Kang, Chaoyun Zhang, Chetan Bansal, Murali Chintalapati, Saravan Rajmohan, Qingwei Lin, Dongmei Zhang, Changhua Pei, and Gaogang Xie. 2024. Large Language Models Can Provide Accurate and Interpretable Incident Triage. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. 523–534. doi:10.1109/ISSRE62328.2024.00056

[161] Miaomiao Wei, Shikai Guo, Rong Chen, and Jian Gao. 2018. Enhancing bug report assignment with an optimized reduction of training set. In *International Conference on Knowledge Science, Engineering and Management*. Springer, 36–47.

[162] Wei Wei, Haojie Li, Xinshuang Ren, Feng Jiang, Xu Yu, Xingyu Gao, and Junwei Du. 2025. Improving bug triage with the bug personalized tossing relationship. *Information and Software Technology* 179 (2025), 107642.

[163] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 1–10.

[164] Hongrun Wu, Yutao Ma, Zhenglong Xiang, Chen Yang, and Keqing He. 2022. A spatial-temporal graph neural network framework for automated software bug triaging. *Knowledge-Based Systems* 241 (2022), 108308.

[165] Wenjin Wu, Wen Zhang, Ye Yang, and Qing Wang. 2011. Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In *2011 18th Asia-Pacific Software Engineering Conference*. IEEE, 389–396.

[166] Sheng-Qu Xi, Yuan Yao, Xu-Sheng Xiao, Feng Xu, and Jian Lv. 2019. Bug triaging based on tossing sequence modeling. *Journal of Computer Science and Technology* 34, 5 (2019), 942–956. doi:10.1007/s11390-019-1953-5

[167] Xin Xia, David Lo, Ying Ding, Jafar M. Al-Kofahi, Tien N. Nguyen, and Xinyu Wang. 2017. Improving Automated Bug Triaging with Specialized Topic Model. *IEEE Transactions on Software Engineering* 43, 3 (2017), 272–297. doi:10.1109/TSE.2016.2576454

[168] Xin Xia, David Lo, Xinyu Wang, and Bo Zhou. 2013. Accurate developer recommendation for bug resolution. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. 72–81. doi:10.1109/WCRE.2013.6671282

[169] Jialiang Xie, Minghui Zhou, and Audris Mockus. 2013. Impact of triage: a study of mozilla and gnome. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 247–250.

[170] Xihao Xie, Wen Zhang, Ye Yang, and Qing Wang. 2012. Dretom: Developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th international conference on predictive models in software engineering*. 19–28.

[171] Yang Xu, Chao Liu, Yong Li, Qiaoluan Xie, and Hyun-Deok Choi. 2023. A Method of Component Prediction for Crash Bug Reports Using Component-Based Features and Machine Learning. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 773–777.

[172] Jifeng Xuan, He Jiang, Yan Hu, Zhilei Ren, Weiqin Zou, Zhongxuan Luo, and Xindong Wu. 2014. Towards effective bug triage with software data reduction techniques. *IEEE transactions on knowledge and data engineering* 27, 1 (2014), 264–280.

[173] Jifeng Xuan, He Jiang, Zhilei Ren, Jun Yan, and Zhongxuan Luo. 2010. Automatic Bug Triage using Semi-Supervised Text Classification.. In *SEKE*. 209–214.

[174] Jifeng Xuan, He Jiang, Zhilei Ren, and Weiqin Zou. 2012. Developer prioritization in bug repositories. In *2012 34th International Conference on Software Engineering (ICSE)*. 25–35. doi:10.1109/ICSE.2012.6227209

[175] Asmita Yadav, Mohammed Baljon, Shailendra Mishra, Sandeep Kumar Singh, Sharad Saxena, and Sunil Kumar Sharma. 2024. Developer load balancing bug triage: Developed load balance. *Expert Systems* 41, 6 (2024), e13006.

[176] Asmita Yadav, Sandeep Kumar Singh, and Jasjit S Suri. 2019. Ranking of software developers based on expertise score for bug triaging. *Information and Software Technology* 112 (2019), 1–17.

[177] Geunseok Yang, Tao Zhang, and Byungjeong Lee. 2014. Towards Semi-automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-feature of Bug Reports. In *2014 IEEE 38th Annual Computer Software and Applications Conference*. 97–106. doi:10.1109/COMPSAC.2014.16

[178] Qingyang Yu, Nengwen Zhao, Mingjie Li, Zeyan Li, Honglin Wang, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2024. A survey on intelligent management of alerts and incidents in IT services. *Journal of Network and Computer Applications* 224 (2024), 103842.

[179] Xu Yu, Fayang Wan, Junwei Du, Feng Jiang, Lantian Guo, and Junyu Lin. 2021. Bug triage model considering cooperative and sequential relationship. In *International Conference on Wireless Algorithms, Systems, and Applications*. Springer, 160–172.

[180] Zhaoyang Yu, Minghua Ma, Xiaoyu Feng, Ruomeng Ding, Chaoyun Zhang, Ze Li, Merali Chintalapati, Xuchao Zhang, Rujia Wang, Chetan Bansal, Sarvan Rajmohan, Qingwei Lin, Shenglin Zhang, Changhua Pei, and Dan Pei. 2025. Triangle: Empowering Incident Triage with Multi-LLM-Agents. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. ACM.

[181] Chunqiu Zeng, Wubai Zhou, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. 2017. Knowledge guided hierarchical multi-label classification over ticket data. *IEEE Transactions on Network and Service Management* 14, 2 (2017), 246–260.

[182] Chaoyun Zhang, Zicheng Ma, Yuhao Wu, Shilin He, Si Qin, Minghua Ma, Xiaoting Qin, Yu Kang, Yuyi Liang, Xiaoyu Gou, et al. 2025. Allhands: Ask me anything on large-scale verbatim feedback via large language models. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 43–57.

[183] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* 48, 1 (2020), 1–36.

[184] Li Zhang, Jia-Hao Tian, Jing Jiang, Yi-Jun Liu, Meng-Yuan Pu, and Tao Yue. 2018. Empirical research in software engineering—a literature survey. *Journal of Computer Science and Technology* 33 (2018), 876–899.

[185] Tao Zhang, Jiachi Chen, Geunseok Yang, Byungjeong Lee, and Xiapu Luo. 2016. Towards more accurate severity prediction and fixer recommendation of software bugs. *Journal of Systems and Software* 117 (2016), 166–184.

[186] Tao Zhang, He Jiang, Xiapu Luo, and Alvin T.S. Chan. 2016. A Literature Review of Research in Bug Resolution: Tasks, Challenges and Future Directions. *Comput. J.* 59, 5 (05 2016), 741–773. doi:10.1093/comjnl/bxv114

[187] Tao Zhang, Geunseok Yang, Byungjeong Lee, and Eng Keong Lua. 2014. A novel developer ranking algorithm for automatic bug triage using topic model and developer relations. In *2014 21st Asia-Pacific Software Engineering Conference*, Vol. 1. IEEE, 223–230.

[188] Wei Zhang. 2020. Efficient bug triage for industrial environments. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 727–735.

[189] Wen Zhang, Song Wang, and Qing Wang. 2016. KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity. *Information and software technology* 70 (2016), 68–84.

[190] Yanmei Zhang, Yuhang Sun, Yi Shi, Shujuan Jiang, and Guan Yuan. 2025. BTAL: An imbalance software bug report triage approach based on BERT-TextCNN. *Information and Software Technology* 183 (2025), 107731.

[191] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, Yong Wu, Fang Zhou, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Understanding and handling alert storm for online service systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice* (Seoul, South Korea) *(ICSE-SEIP '20)*. Association for Computing Machinery, New York, NY, USA, 162–171. doi:10.1145/3377813.3381363

[192] Nengwen Zhao, Panshi Jin, Lixin Wang, Xiaoqin Yang, Rong Liu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Automatically and Adaptively Identifying Severe Alerts for Online Service Systems. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 2420–2429. doi:10.1109/INFOCOM41043.2020.9155219

[193] Yuan Zhao, Tieke He, and Zhenyu Chen. 2019. A unified framework for bug report assignment. *International Journal of Software Engineering and Knowledge Engineering* 29, 04 (2019), 607–628.

[194] Wujie Zheng, Haochuan Lu, Yangfan Zhou, Jianming Liang, Haibing Zheng, and Yuetang Deng. 2019. iFeedback: Exploiting user feedback for real-time issue detection in large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 352–363.

[195] Weiqin Zou, Yan Hu, Jifeng Xuan, and He Jiang. 2011. Towards training set reduction for bug triage. In *2011 IEEE 35th annual computer software and applications conference*. IEEE, 576–581.

[196] Weiqin Zou, David Lo, Zhenyu Chen, Xin Xia, Yang Feng, and Baowen Xu. 2020. How Practitioners Perceive Automated Bug Report Management Techniques. *IEEE Transactions on Software Engineering* 46, 8 (2020), 836–862. doi:10.1109/TSE.2018.2870414