AutoARTS: Taxonomy, Insights and Tools for Root Cause Labelling of Incidents in Microsoft Azure

Pradeep Dogga, *UCLA;* Chetan Bansal, Richard Costleigh, Gopinath Jayagopal, Suman Nath, and Xuchao Zhang, *Microsoft*

https://www.usenix.org/conference/atc23/presentation/dogga

This paper is included in the Proceedings of the 2023 USENIX Annual Technical Conference.

July 10–12, 2023 • Boston, MA, USA

978-1-939133-35-9

Open access to the Proceedings of the 2023 USENIX Annual Technical Conference is sponsored by

AutoARTS: Taxonomy, Insights and Tools for Root Cause Labelling of Incidents in Microsoft Azure

Pradeep Dogga
UCLA[*]

Chetan Bansal
Microsoft

Richie Costleigh
Microsoft

Gopinath Jayagopal
Microsoft

Suman Nath
Microsoft

Xuchao Zhang[†]
Microsoft

Abstract

Labelling incident postmortems with the root causes is essential for aggregate analysis, which can reveal common problem areas, trends, patterns, and risks that may cause future incidents. A common practice is to manually label postmortems with a single root cause based on an ad hoc taxonomy of root cause tags. However, this manual process is error-prone, a single root cause is inadequate to capture all contributing factors behind an incident, and ad hoc taxonomies do not reflect the diverse categories of root causes.

In this paper, we address this problem with a three-pronged approach. First, we conduct an extensive multi-year analysis of over 2000 incidents from more than 450 services in Microsoft Azure to understand *all* the factors that contributed to the incidents. Second, based on the empirical study, we propose a novel hierarchical and comprehensive taxonomy of potential contributing factors for production incidents. Lastly, we develop an automated tool that can assist humans in the labelling process. We present empirical evaluation and a user study that show the effectiveness of our approach. To the best of our knowledge, this is the largest and most comprehensive study of production incident postmortem reports yet. We also make our taxonomy publicly available.

1 Introduction

Cloud platforms and services, despite the best efforts of cloud providers, still suffer from production incidents and outages [13,15]. To improve reliability, cloud providers must first discover and resolve existing reliability risks [14]. Aggregating root causes of past incidents based on their post-incident reports (PIRs) is one effective approach to uncover common problem areas, trends, patterns, and risks (e.g., the most common root causes in the last year). Likewise, bucketing past incidents by their root causes can help on-call engineers (OCEs) to quickly retrieve and learn common mitigation strategies for a given root cause category. Such tasks are crucial for large-scale cloud platforms like Microsoft Azure that continuously strive to improve reliability by learning from past incidents caused by diverse factors.

PIRs are commonly written in natural language, with little structure. This makes the tasks of aggregating or bucketing past reports challenging, especially at a large-scale. One common practice to address this is to label each PIR with a *root cause tag* representing the category of its root cause. For example, a PIR for an incident caused by a code bug in the network driver can be tagged as "*Network.Driver.Code.Bug*". This enables quick and accurate aggregation of and retrieval from a large collection of PIRs simply based on their root cause tags instead of expensive and potentially inaccurate natural language processing of the PIR contents. For example, the tags can enable quickly answering questions such as: "how frequently did network driver code bugs cause incidents in the past year?" and "how were such bugs mitigated?"

Challenges and limitations. There are two key challenges in effectively labelling PIRs with root cause tags (more in §2).

The first challenge is to decide *what root cause tags to use to label the PIRs*. A well-defined taxonomy of root cause tags is essential for labelling PIRs consistently, otherwise different teams may tag the same root cause differently, hindering the cross-team aggregation analysis. A well-designed taxonomy for a large-scale cloud system such as Microsoft Azure should balance two competing objectives: it should be *comprehensive* enough to cover the myriad of potential root causes, yet *compact* enough for the OCEs to navigate and use easily. Moreover, it should be *fine-grained* enough to surface actionable insights from across many services.

Designing such a taxonomy is nontrivial. Several recent works analyzed production incidents and proposed taxonomies to capture their root causes. However, most of these works focus on specific root cause categories, such as software bugs [4,5,11,19,21,40], and thus their taxonomies are not comprehensive enough to cover other types of failures (e.g., hardware failures). Some other works consider multiple types of root causes, but they target specific services or

[*]Work done during internship at Microsoft

[†]Except for the first author, all authors are in order by their last names.

systems, such as big-data systems [38], a business data processing platform [9], or a particular cloud service [13, 14], rather than a large-scale cloud system. Moreover, existing taxonomies are not fine-grained enough to represent all the root causes we observe in Azure incidents.

Prior to our work, individual service teams in Microsoft designed their own root cause taxonomies based on domain knowledge. However, due to the lack of a comprehensive root cause labelling, many potential root causes were not anticipated when the taxonomies were designed. Consequently, a large portion of PIRs, whose root causes were not covered by existing taxonomies, were labelled with "Other" instead of more informative root cause tags (see §2 for more details).

The second challenge concerns *how to select root cause tags for incidents*. Currently, this is done manually—an individual (OCE) reads *lengthy* incident and post-incident reports, identifies the root cause, and chooses a suitable tag from a taxonomy that is often a long flat list of tags. This manual process is error-prone and can result in inconsistent tags across incidents—at Microsoft, tens of thousands of OCEs with varying levels of expertise and different interpretations of root cause tags conduct root cause analysis. We manually examined a small sample of 1241 PIRs and found that 29% of the OCE-assigned tags are incorrect. This problem might be mitigated if root cause tagging is done by a small group of experts through a stringent procedure, but this is infeasible at large scale systems like Microsoft Azure.

Contributions. In this work, we make the following three contributions that address the challenges described above.

First, we manually analyze over 2000 high-impact production incidents from 468 services in Microsoft Azure to identify a comprehensive set of root cause categories behind incidents in a large-scale cloud system. We carefully read the incident and post-incident reports and, if needed, consult the engineers from the affected service teams. Unlike previous empirical analyses of production incidents [4,5,9–11,14,21,38,40], we aim to identify not only a single root cause, but *all* factors contributing to the incidents. This analysis took more than four person-years and identifies 346 distinct root cause categories spanning all aspects of a production service, such as hardware and software, infrastructure and application, code and configuration, and so on. To the best of our knowledge, this is the most comprehensive empirical analysis of production incidents in cloud systems, considering the scale of incidents and affected services, the depth of analysis, and the diversity of root causes.

Our empirical analysis (§3) reveals several novel and interesting findings. For example, we show that most production incidents in real-world cloud systems involve multiple contributing factors; hence, preventing such incidents does not always require addressing all the causal factors, but only one (or a small subset) of them. This contrasts with existing research that focuses on a single root cause of an incident [10, 13, 21]. This finding implies that tagging a PIR with a single root cause does not capture the full picture of what caused the incident. Our analysis also shows that incidents result from many diverse factors spanning hardware and software, infrastructure and application, code and configuration, and so on. This, again, contrasts with existing research that focuses on a limited set of factors [4, 5, 9–11, 14, 19, 21, 38, 40]. We also show that the set of root causes evolves over time: new root causes emerge as new services or hardware are deployed, suggesting the need for a continuous root cause labelling effort. While our findings stem from analysis of incidents at Microsoft Azure, we believe that they generalize to similar large-scale systems and impact root cause analysis procedures at large.

Second, we propose a comprehensive taxonomy, the Azure Reliability Tagging System (ARTS), that organizes the root cause categories derived from our analysis. For ease-of-use, our taxonomy is organized hierarchically, with each leaf node representing a *root cause tag* describing a factor contributing to an incident. We expect that the root causes generalize to other cloud services and we open-source our taxonomy for the use of other researchers and practitioners.[1]

Finally, to reduce manual errors and inconsistencies in tagging PIRs, we developed AutoARTS, a tool that leverages machine-learning-based algorithms to assist humans. AutoARTS performs two key tasks: (1) it applies a multi-label classification technique to automatically analyze a PIR (written in natural language) and to extract multiple contributing factors and their corresponding tags from our proposed taxonomy; and (2) it generates a concise text snippet (from the PIR) that summarizes the relevant context for the factors. The snippet allows a human to quickly review the suggested tags without reading lengthy incident reports or PIRs. We describe how we adapt existing ML techniques for this purpose. Our empirical evaluation with real PIRs and a user study demonstrates the effectiveness of our approach. Specifically, our multi-label classification achieves an F1 score of 0.89. In the user study, our text snippets (contexts) received a score of 4.6 out of 5 (5 being 'very useful'), contained *no* unnecessary details, and helped an expert identify two additional contributing factors (in a set of ten incidents) that they had originally missed.

Deployment status. Most of the 468 services whose production incidents we analyze have been deployed as part of Microsoft Azure for several years. High-impact incidents in these services have been analyzed and labelled with ARTS tags since November 2020. The labels, available to all services in Azure, are regularly aggregated to identify key problem areas and to devise actionable insights (examples in §4).

2 Background and motivation

2.1 Background

Incident Reports. Incidents are unplanned outages that impact production services and their users. The severity of an in-

[1] ARTS Taxonomy: https://autoarts-rca-taxonomy.github.io

cident may vary based on aspects such as the criticality of the affected services and the number of impacted users. As described in [31], an incident's life-cycle is a complex process involving several steps such as detection, triaging, diagnosis, root cause analysis, mitigation, and resolution. An *incident report* documents important information related to these various steps. At Microsoft, an incident report can be created by impacted users as well as by automated monitors and it usually contains the following: (1) a concise title, (2) a summary of the incident, highlighting some events in the timeline from detection to mitigation, (3) engineers' discussion thread to share relevant information corresponding to the incident's resolution, and (4) several other fields such as severity, owning team, time to mitigation, mitigation steps, etc.

Post-Incident Reports (PIR). After an incident is resolved, a best practice is to conduct a retrospective or postmortem analysis of the incident to produce a *post-incident report* (PIR). In a PIR, the service team reflects on what went wrong, why it went wrong, what they learned from it, and how to avoid similar incidents in the future. At Microsoft, a PIR is a natural language document and it contains sections such as (1) root cause summary that describes all root causes, (2) five-whys [34] that iteratively drills-down the cause-and-effect relationships of various contributing factors, (3) preventive measures for similar future incidents, and so on. Generating a PIR requires significant effort, and hence, only the incidents with high severity are required to have them at Microsoft.

Root cause labelling. A critical component of a PIR is its *root cause tag* that represents the root cause categories of the incident as determined by a postmortem analysis. For example, a root cause tag *"Authoring.Code.Bug.RaceCondition"* indicates that the incident is caused by a race condition in software code. Such concise labelling allows one to efficiently and accurately aggregate and summarize a large number of historical PIRs solely based on their tags, without expensive and potentially error-prone natural language processing of the PIR contents. These aggregate results are regularly reviewed by the engineering leadership to find global trends (e.g., frequent root cause categories), which guide business decisions such as prioritizing engineering investments. These tags also simplify information retrieval and knowledge sharing: an engineer seeking to actively mitigate an ongoing incident caused by a network driver can quickly retrieve and consult past PIRs with root cause tag *Network.Driver*. For effectiveness of aggregation and retrieval, it is important that the root cause analysis and labelling process is accurate.

At Microsoft, authors of incident reports or PIRs can label their reports with root cause tags selected from taxonomies of root cause categories that are predefined by domain experts.

2.2 Challenges in root cause labelling

We analyze a sample of $\approx 1.7M$ root cause analyses in Microsoft, across all its services, to understand the challenges in root cause labelling. We now summarize the key findings.

Finding 1. *Existing taxonomies, although designed by domain experts, are not comprehensive enough.*

This is due to the lack of a comprehensive study of root causes, many potential root cause categories are missed or not anticipated when a taxonomy is designed. As an implication, a PIR author may not find a suitable predefined root cause tag to describe the current incident. In our sample of root cause analyses, $\approx 20\%$ incidents are labelled as 'Other' and $\approx 58\%$ are labelled with categories containing 'Other' (e.g., 'Network - Other'), implying that their root causes are not covered or only partially covered by the existing taxonomies. Such 'Other' tags are not useful in the aforementioned root cause aggregation and retrieval tasks.

Finding 2. *Existing manual root cause labelling process is expensive and error-prone.*

Root cause label of an incident is often determined based on its PIR and incident report. These documents are usually *long* (4542 words per incident in our sample) and *complex* (on average, ≈ 9 engineers engaged in discussion exchanging 20 comments). Thoroughly understanding these long documents to identify all contributing factors behind an incident, and then selecting from predefined root cause labels that represent the factors, is a nontrivial task.

Even when the root cause is understood, PIR authors may make mistakes in choosing the correct tag. This can happen due to multiple factors. Existing taxonomies at Microsoft are flat long lists, making it difficult to navigate through them and to pick the right tags. Moreover, many individuals are involved in the rootcausing efforts. For example, we observe $34K$ distinct individuals involved in a sample of $600K$ PIRs in Microsoft. This large number of individuals are likely to have varying degrees of expertise and different interpretations of root cause tags. This is further exacerbated by ambiguous or confusing tags in the taxonomy (e.g., 'Network' and 'Datacenter - Network'). All these factors can contribute to inconsistent and/or inaccurate labels. We manually examined a small sample of 1241 PIRs and found that 29% of the assigned tags are incorrect.

Our goals. In this paper we address the challenges above with a three-pronged approach. First, to address the first challenge above, we conduct an extensive multi-year analysis to identify a wide variety of fine-grained root causes of 2000+ production incidents from across 450+ services in Microsoft Azure. Second, based on this analysis, we propose a novel hierarchical and comprehensive taxonomy of potential contributing factors behind the incidents (§4). Third, to address the second challenge above, we develop an automated tool that can assist a human by presenting necessary context from PIRs that identify contributing factors to reduce cognitive load and improving accuracy in the root cause labelling process by suggesting tags from the taxonomy (§5).

3 Empirical Analysis of Production Incidents

3.1 Goals and Methodology

There exists several empirical studies of production incidents in large-scale cloud systems [4, 5, 9–11, 14, 21, 38, 40]. We have two goals that differentiate our study from them. First, we do not restrict our analysis to a limited set of root cause categories (e.g., software bugs [4, 5, 11, 19, 21, 40]) or specific services/platforms (e.g., big-data systems [38]). Second, for each incident, we try to identify not a single root cause, but *all* factors contributing to the incidents. These two goals enable us to identify a wide-range of contributing factors behind incidents happening in large number of services/platforms.

We analyze 2051 high-impact incidents in 468 Microsoft Azure services. We carefully analyze each incident by carefully reading and understanding its incident report and PIR, the discussion comments, and even the work items (e.g., bug fix, system upgrade) that are created due to the incident. When something is not clear, we reach out to the incident owners to clarify. As a part of the analysis, we not only identify the *contributing factors* causing the incident but also extract text snippets or *context* from the incident and PIR which helps explain and justify the identified root cause tag for future reference and validation. Every week, we peer review a randomly selected subset of incidents to help us refine our collective understanding of tag usage, promote learning and improve accuracy. If we identify a new category of root causes, which is not covered by existing tags, we then propose new tags which are internally reviewed before getting introduced to the taxonomy. For any tag in the taxonomy, we also provide it's description in natural language for future reference. This data lives in an internal database which can be easily joined with incident databases and visualization reports are created for easy data analysis based on various pivots such as contributing factors, services, incident impact, etc. We also meet with the engineering teams on a weekly basis, and review our data both for accuracy and to share insights that result in reliability improvements.

Our root cause analysis effort is guided by several principles: (1) Our analysis is intellectually honest so that individual teams that conduct their own postmortem analysis (to identify a single root cause) feel psychologically safe, valued, and included; (2) Our analyses have meaningful depth because we start by capturing customer pain and go down to "work as done" by our engineers; (3) We focus on both depth and breadth of incident analysis enabling us to highlight thematic learning that broadly improves Azure services and the underlying cloud platform; (4) Our findings are turned into new standards or updates to existing standards and are actionable and useful because they address customer pain; and (5) While learning is important; continuous learning is necessary and crucial.

The above process is required to ensure high quality of root cause analysis. However, it needs significant manual effort. Since 2020, we have analyzed 2051 incidents in 468 Azure services with a team of 2-4 members.

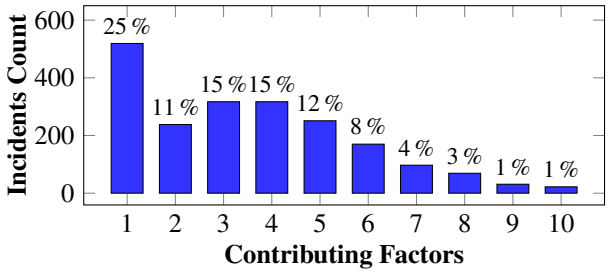

Figure 1: **Distribution of incidents across number of distinct contributing factors (shown until 10 factors).**

3.2 Analysis Results

We here present several interesting findings from our analysis.

Finding 3. *Incidents are often caused by multiple contributing factors working together instead of an isolated root cause.*

This is contrary to prior work [13, 15, 16, 21] that focus on identifying a single root cause per incident. Consider, for example, a real incident where a service became unavailable after a single customer continuously pushed a load that was 60x greater than what the service was scaled to handle. The original PIR author chose the root cause label "*Service – Load Threshold*." This itself is not an inaccurate root cause when forced to pick only one cause. However, there are many more factors involved in this incident: (1) there was an inrush of load from a single customer, (2) there was a lack of throttling on the customer end as well as the service end, (3) increased load significantly increased CPU and heap usage and thread count at the server, which lead to failed requests with exceptions, (4) the exception handling didn't release some resources that were allocated by the failed requests, leading to resource leaks, (5) there were no automated watchdogs to detect early symptoms of the outage (or resource leaks), and (6) the team was unable to access their own metrics during the outage since the metrics were collocated with the service. In contrast, our analysis of the incident identifies all these factors and the corresponding tags in our taxonomy.

Figure 1 shows the distribution of the number of contributing factors behind each incident. As shown, over 75% of incidents have been caused by more than one contributing factors. And, more than 50% of the incidents have 4 or more contributing factors. On average, each incident has ≈ 3.6 factors. This reaffirms the need for holistically analyzing the incidents to understand all the contributing factors.

The presence of multiple contributing factors per incident has important implications. On one hand, identifying the possibility of such incidents before deployment to production with integration and end-to-end tests is challenging since test-

Category	Description	Frequency	TTM (Hrs)
Detection	Issues related to detecting problems before they affect production	**61%**	50
Authoring	Issues in authoring artifacts like code, config, troubleshooting guides, etc.	**50%**	**58**
Dependency	Issues in a dependency the service has, most typically another service but can also be some things where a boundary between teams is present	**37%**	16
Architecture	Issues in how the service is architected and likely where any work to fix would require changes to the architecture of the service	20%	33
Deployment	Issues related to deployment of code or config	20%	27
Process	Any issue caused by human errors, a flawed process or the lack of a process	18%	**123**
Load	Any issue caused by the service not being able to handle changes in load	14%	13
Auth	An authentication or authorization related issue	7%	21
Performance	An issue that caused excess latency	6%	16
Datacenter	Events (hardware, installations, power interruption, etc.) in the datacenter	4%	**70**

Table 1: **High-level root cause categories from ARTS taxonomy with their descriptions, frequency of occurrence in our analysis and mean Time-To-Mitigate (TTM) for incidents caused by their sub-categories.**

ing needs to be performed in the presence of multiple potential contributing factors (e.g., high load *and* no throttling *and* no monitoring of early symptoms). On the other hand, *preventing such an incident does not always require addressing all the causal factors, but only one (or a small subset) of them.* For example, the aforementioned incident could have been prevented by using proper throttling mechanism, *or* by fixing the resource leak bug, *or* by having monitors that can restart the service on early symptoms of resource leaks. This insight presents a unique opportunity to fix the incidents (by addressing the easiest causal factor); but it requires identification of all the causal factors (as we do) instead of identifying a single root cause.

Finding 4. *A wide-variety of factors contribute to production incidents.*

Our analysis identified a wide range of factors, including hardware, software, code bugs, underlying infrastructure to external dependency issues, configuration errors, deployment issues, and so on. Specifically, we have identified 346 root cause categories (i.e., contributing factors) for the 2051 incidents we analyzed. Table 1 shows the high-level root cause categories, each of which contains many finer-grained sub-categories. The full list of categories and their respective frequencies observed in our analysis can be found in https://autoarts-rca-taxonomy.github.io. This contrasts our study with prior works that focus on a small set of root causes such as code bugs [4, 5, 11, 19, 21, 40].

Finding 5. *New root-cause categories keep appearing over time.*

As software and hardware systems evolve, novel root causes appear to contribute to their incidents. For example, when a service migrates to a containerized environment, its incidents may be caused by container-related factors. Similarly, when a service takes a new external dependency, it may start experiencing incidents caused by factors related to the failures of the new dependency. We analyze incidents in the same timeline as they appear and we create root cause categories incrementally—we create a new category only if none of the existing ones can precisely represent a new root cause. We observe that even though many common root cause categories (e.g., code bugs) appear in early incidents that we analyze, a few categories appear only in much later incidents (e.g., those happening two years after the first incident we analyzed). Figure 2 shows how often such new categories appear in our analysis. As shown, even after 1.5 years, new root cause categories appear, albeit with a smaller rate (i.e., we can tag higher number of incidents successfully before we need to introduce a new root cause category). The fact that novel root cause categories keep appearing implies that root cause labelling needs to be a continuous process to identify (and take actions on) emerging root cause categories. This calls for an automated solution.

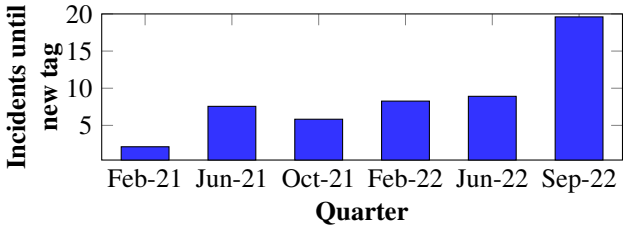

Figure 2: **Average number of incidents successfully tagged until a new root cause tag is introduced across quarters.**

Finding 6. *Lack of monitoring (i.e., observability) is the most common factor behind incidents.*

Table 1 shows the distribution of various contributing factors behind the incidents we analyzed (only high-level factors are shown). As shown, Detection is the most common contributing factor leading to outages. Detection related issues represent missing observability signals that prevent us from detecting early symptoms of problems, many of which could have been avoided, e.g., by rebooting the service, if their early

Contributing Factor	Frequency
Detection.Monitoring.MissingAlert	34%
Authoring.Code.Bug.Change	25%
Detection.Monitoring.InsufficientTelemetry	18%
Detection.Validation.MissingTestCoverage	17%
Detection.Monitoring.CodeDeployment. InsufficientHealthSignal	9%
Authoring.Documentation. NoOrInsufficientTSG	9%
Architecture.SinglePointOfFailure	8%
Authoring.Code.Bug.Latent	7%
Detection.Monitoring.Synthetic	6%
Deployment.Mitigation.ManualTouch	5%

Table 2: **Distribution of top 10 most frequent contributing factors in our analysis from the ARTS taxonomy.**

symptoms were detected. We also analyzed finer-grained contributing factors from ARTS taxonomy. Table 2 shows distributions of the top ten contributing factors (a contributing factor X.Y.Z means factor Z is a specific case of factor Y, which is a specific case of factor X). As shown, Missing Alerts, which is a specific case of Monitoring, which is a specific case of Detection, is the most common contributing factor. Insufficient telemetry captured from services is also a major contributing factor which also prevents from deploying automated alerts. An organizational policy on collecting key telemetry and defining automated watchdogs informed by this aggregate analysis can mitigate incidents (or severity) in the future.

We also analyze the most frequently *co-occurring* root causes to identify the pairs that jointly cause incidents. The two most frequent pairs are "*Authoring.Code.Bug.Change*" & "*Detection.Monitoring.MissingAlert*" (15%) and "*Authoring.Code.Bug.Change*" & "*Detection.Validation.MissingTestCoverage*" (11%). This aligns with our experience that many production incidents are caused by buggy code changes that are deployed without proper monitoring and testing.

Finding 7. *Incidents caused by deployment and datacenter related issues are the most time consuming to mitigate.*

In incident management, TTM is defined as the time elapsed between the start of the incident and when its customer impact was completely resolved. The higher the TTM, the more the customer impact and dissatisfaction. From Table 1, we can see that incidents caused by Process and Datacenter related root causes have the highest mean TTM. Process related incidents have a high TTM because these incidents are caused by human errors and lack of standard operating procedures which result in non-trivial hard-to-resolve issues (e.g., accidental deletion of a database). Datacenter related incidents are caused primarily due to hardware failures which are quite complex given that there are multiple layers of capacity buffers all of which need to fail before an incident is caused by hardware issues.

4 Root Cause Taxonomy

We organize the root cause categories identified in our empirical study as a taxonomy of reliability tags that can be used to label PIRs of incidents.

Design goals. We have the following design goals in designing the taxonomy. First, the taxonomy should be *comprehensive* enough to capture not only the primary root causes of past incidents in Azure, but also other (secondary) contributing factors. Second, in order to avoid having a taxonomy too large to be easily used in practice, the taxonomy should be *sufficient* and it should include only the root causes found in past incidents. This implies that the taxonomy is continuously and organically grown to include new categories as they are discovered. Third, the tags should be *unambiguous*, to enable high-quality annotations. Finally, the taxonomy is organized *hierarchically*, for ease of labelling and updates.

The ARTS taxonomy. We achieve the goals with a novel taxonomy called ARTS (Azure Reliability Tagging System) taxonomy. The taxonomy is built on top of the root cause categories identified by our empirical analysis described before. We start with a small number of tags representing orthogonal categories of themes (such as datacenter issues and authentication issues) and grow it horizontally to include new themes and vertically to include more specific sub-themes as new incidents are analyzed and existing themes/sub-themes deem inadequate. We have established a continuous feedback loop based process for building the ARTS taxonomy and tagging of new incidents on an ongoing basis.

For ease-of-use, we organize the ARTS taxonomy hierarchically, by grouping related sub-themes under one common theme. Currently it consists of four levels and contains 346 root cause categories identified from our empirical analysis. The top level consists of ten broad themes (shown in Table 1), each of which consists of multiple sub-themes. There are 346 leaf nodes, each representing a root cause tag with the name obtained by concatenating the names of the path from the root to the leaf node. For example, the root cause of "a gap in preproduction detection due to missing integration tests" is represented with the tag "*Detection.IntegrationTest.Missing*" in which "*Missing*" is the most precise leaf-level tag. The hierarchical taxonomy naturally distinguishes between problem spaces at different granularities. In this example, if the root cause is that the integration tests existed but were skipped somehow, that representative tag would have the leaf-level tag "*NotRun*" instead of "*Missing*".

As mentioned, the taxonomy is grown as new root causes are identified in newly analyzed incidents. Figure 2 shows how the taxonomy has been growing over time, with the y-axis showing the average number of incidents analyzed until a new tag needed to be introduced in ARTS. A larger value indicates better stability of the taxonomy: many incidents can be analyzed with existing tags. As shown, over time, the taxonomy can be seen becoming stable. Specifically, in the

most recent quarter, only one new tag needed to be introduced after analyzing ≈ 20 incidents (i.e., after using ≈ 70 existing tags) on average. We hope to see significantly more stability in coming months.

For lack of space, we omit further discussion about 346 leaf nodes in the taxonomy. However, we open source the taxonomy, with all tags and their descriptions, at https://autoarts-rca-taxonomy.github.io. We believe our open-source effort will foster future research and allow practitioners use our taxonomy. Even though the ARTS taxonomy is developed based on incidents in Azure, we believe that its categories are general enough to be used in any large-scale cloud system.

Deployment Status. The ARTS taxonomy and PIRs labelled with ARTS tags have been available to Microsoft engineers since when we started building it (November 2020). It enables various service teams within Microsoft Azure to systematically look at past incidents. The learnings have been valuable, especially for service teams without the required expertise or resources to dig deep into their service reliability. They have enabled engineering teams prioritize work items and often times this also results in creation of new engineering standards and tools. For example, lack of unit testing was a common factor contributing to many incidents in a large organization in Azure—incidents were caused by bugs in code that was not tested (or poorly tested). This caused the organization to enforce the policy that all checked-in code must have sufficient tests to achieve a certain code coverage. Lack of throttling was another factor highlighted by aggregating ARTS labels; this started a new engineering group with the goal of building a common throttling service that all Azure services can easily use. Last but not the least, engineering teams emphasize on using more extensive monitoring/observability tools, since as ARTS labelling showed, adequate monitoring could prevent many incidents.

Overall, Microsoft engineers found the ARTS labels in their PIRs useful. Here we show samples of the verbatim feedback from the service owners at Microsoft: *"Tracking incidents against a known set of root causes is extremely useful. Your effort has enabled us to make data-driven decisions, and already produced several benefits in a short time." "Your data established clearly that services that have high maturity in certificate management had fewer outages. This validated that our investment across Azure is in the right direction." "The ARTS report is an important resource for us for data driven planning related to Azure Quality."*

5 AutoARTS for Automated Labelling

As mentioned before, identifying and labelling PIRs with their root causes is expensive and error-prone. To reduce the cost and errors, we have developed an automated tool called AutoARTS that can assist a human in the labelling process with two important tasks. First, it uses a multi-label classification technique to automatically analyze an incident's PIR (written in natural language) and to identify multiple contributing factors and their representative ARTS tags. Second, it can produce a short text snippet (from the PIR) that captures important context explaining the factors. The snippet enables a human to easily review the selected tags without reading lengthy incident reports or PIRs. We now describe how AutoARTS performs these two tasks by using ML techniques. Figure 3 shows the architecture and components of AutoARTS.

5.1 Automatic identification of ARTS tags

AutoARTS uses multi-label text classification to classify a PIR into a set of ARTS tags. One key challenge we face is that conventional multi-label text classification algorithms that treat each class as opaque and independent, require sufficient labelled data for each class to achieve good accuracy. However, even though we have a reasonable collection of labelled data, many of the fine-grained classes (i.e., ARTS contributing factors) contain very few labelled samples (i.e., PIRs). Specifically, 68% classes have fewer than 10 labelled samples in our dataset, which can adversely affect classification accuracy.

To address this, we leverage the hierarchical relationship between root cause labels by encoding the taxonomy structure using Graph Convolutional Networks [18]. Exploiting the structure of the taxonomy enables transfer of knowledge from the categories with adequate labels to categories with few labels (§6.2). In particular, we apply a hierarchical text classification model called HiAGM [42]. Contrary to the conventional multi-label text classification methods that disregards the holistic label structure for label correlation features, this model attempts to fully utilize the mutual interactions between the text feature space and label space, as well as label dependencies. As an illustration, consider the root cause taxonomy in Figure 3. The "*Authoring.Code.Change*" category only contains 13 samples, making training difficult owing to the small amount of labelled samples. However, by modeling the root cause taxonomy as a graph, we can transfer knowledge from "*Authoring.Code.Bug*", which has 733 labels, to "*Authoring.Code.Change*" since they share the features of their same parent root cause category, "*Authoring.Code*".

Given a Post-Incident report $x = (w_1, w_2, \ldots, w_n)$ with n tokens, the sequence of token embedding is initially fed into a bidirectional GRU neural network [6] to extract text contextual features. Following the GRU model, multiple CNN layers are employed to generate the n-gram features. The top-k max pooling layer is then applied to obtain the overall text representation $S \in \mathbb{R}^{n \times d_c}$ that highlights the key information, where n is the top-k output dimension of CNN layers and d_c represents the embedding dimension.

To model the ARTS taxonomy, we formulate the taxonomic hierarchy as a directed acyclic graph $G = \{V, E_t, E_b\}$, where V refers to the set of label nodes. E_t and E_b represent the top-

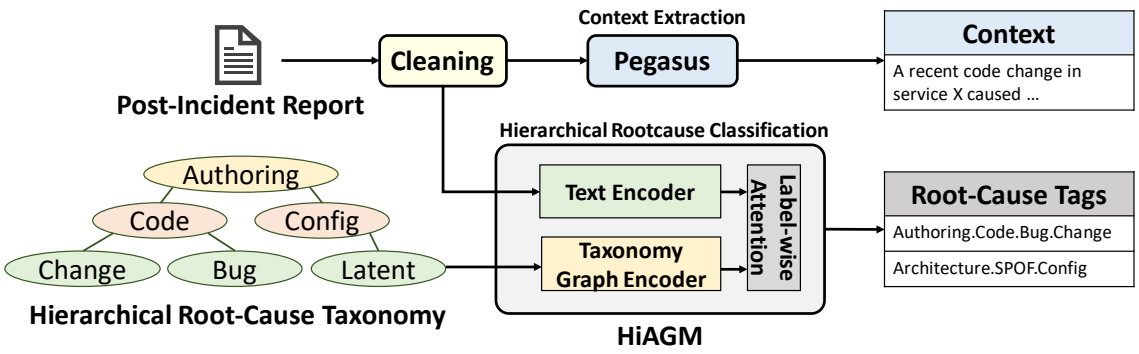

Figure 3: **Overview of our Context Extraction and Hierarchical root cause Classification using Post-Incident reports.**

down and bottom-up hierarchy paths respectively. To encode the hierarchy graph, a GCN-based hierarchy encoder [18], Hierarchy-GCN, is used to aggregate data flows within the top-down, bottom-up and self-loop edges based on the associated neighborhood of each node. The GCN-based graph encoder adapts the convolution concept from images to graphs, in which the graph convolutional operator can effectively convolve the multi-order neighborhood information by forming multiple propagation steps during the forward pass. For each node, the feature information is aggregated by the node feature from all the neighbors, including the node itself, to leverage the graph structure of label taxonomy.

Next, we aggregate the features of texts and labels together using a label-wise attention mechanism [37]. Specifically, the attention α_{kj}, which indicates how informative the j-th text feature is for the k-th label, is calculated as follows:

$$\alpha_{kj} = \frac{e^{\mathbf{s}_j \mathbf{h}_k^T}}{\sum_{j=1}^{n} e^{\mathbf{s}_j \mathbf{h}_k^T}}, \tag{1}$$

where $\mathbf{s}_j$ is the j-th text feature of the root cause input and $\mathbf{h}_k$ represents the k-th node in the label hierarchy. The label-aligned text feature $\mathbf{v}_k = \sum_{i=1}^{n} \alpha_{ki} \mathbf{s}_i$ for the k-th label is then obtained and fed into a classifier for hierarchical label prediction.

Finally, we flatten the label hierarchy by treating all nodes as leaf nodes for multi-label classification, regardless of whether a node is a leaf or an internal node. A binary cross-entropy loss function is employed to train the model using the ground truth and predicted sigmoid score for each label. In additional, a recursive regularization for the parameters of the final fully connected layer is used to encourage classes nearby in the hierarchy to share similar model parameters.

$$L_r = \sum_{i \in C} \sum_{j \in child(i)} \frac{1}{2} \|\mathbf{w}_i - \mathbf{w}_j\|^2, \tag{2}$$

where the node j is the child of node i in the label hierarchy.

5.2 Context Extraction from PIR

The objective of context extraction is to extract key text snippets from a given Post-Incident Report (PIR) for on-call engineers to comprehend the contributing factors of an incident without reading the complete lengthy report. Many text summarization techniques exist. *Abstractive summarization*, where summaries may contain generated sentences, is not a good fit for us since our goal is to select and highlight existing texts in the PIR, as is done by *extractive summarization*. However, existing language models such as BERT [8] and XLNet [36] are trained on large corpuses such as Wikipedia articles, etc., where the syntax and semantics of the language used is quite different from what is observed in PIRs due to domain-specific usage of words (e.g., 'Fabric' in networking terminology vs clothing) and different vocabulary. Moreover, existing extractive summarization models are trained on and their traditional usage in *summarizing* text documents, which is different from context extraction from PIRs. Our experimental evaluation in §6.3 shows that they perform poorly. We therefore finetune an existing model called Pegasus [39] with our labelled data (from our empirical study described before).

In Pegasus, important sentences are removed or masked from an input text and are generated together as one output sequence from the other remaining sentences, similar to an extractive summary. Hence, Pegasus is amenable for context extraction from PIRs, because we can mask the key sentences identified in our analysis (§3.1) to finetune the model parameters. Using the standard Transformer encoder-decoder, Pegasus model is pre-trained on two enormous text corpora: 1) Colossal and Cleaned version of Common Crawl (C4) [27], which comprises of text from 350 million web pages with a size of 750 gigabytes; 2) HugeNews [39], a dataset of 1.5 billion news articles gathered from 2013 to 2019. Similar to MLM tasks for predicting masked tokens, a new pre-training task called Gap Sentences Generation (GSG), is applied to fill the masked sentences. Three different strategies are applied for selecting m gap sentences without replacement from a document. The first method is to uniformly select m sentences at random, whereas the second strategy is to simply select the first m sentences. The aforementioned two strategies are combined with the Principal strategy, in which top-m scored sentences are chosen based on their significance as measured by rouge score [20] without the selected sentence. Formally, the

The root cause of this monitor alert was that a lot of subscriptions could not deploy VMs on indiacentral region.
... ... *(omit 132 words)*
This problem occurred because the traffic that was re routed to AZSM could not be handled. This problem occurred on indiacentral prod b and indiacentral prod b. As part of increasing inventory we have introduced news sets of AMD clusters. The AZSM services on these clusters still needed some configuration and build out related processed to be completed. Hence these clusters stamps could not handled the traffic re routed to them. The traffic was routed as part of default behavior. We are going to change this. The Fabricator clusters started taking tenant traffic even though their corresponding Az SM clusters weren't ready. This was done as part of flighting on Broad clusters. India Central was one of the region for this flighting. We did not anticipate a case where new build out clusters would not be able to take the new traffic. This was detected as part of the API failure monitor. We will be working on adding more robust feature specific monitoring and more strict rollout to not encounter this failure again.

Figure 4: **Context extraction from a redacted PIR. Green sentences are extracted by both our model and human expert, red are extracted by model only and blue are extracted by human only.**

score s_i of the i-th sentence x_i can be expressed as follows:

$$s_i = \text{rouge}\big(\text{x}_\text{i}, \text{D} \setminus \{\text{x}_\text{i}\}\big), \tag{3}$$

where the D is the set of all the sentences in the document and rouge function is a commonly employed metric for evaluating how good an automatically produced summary against a reference summary.

Even though Pegasus has been pre-trained on massive datasets, it is not trained to generate context from root cause descriptions in software engineering domain. To completely comprehend the context extraction task in our domain, we utilize the human-labeled context to further fine-tune the Pegasus model as a sequence-to-sequence task.

Based on the empirical results in §6.3, we found the Pegasus model can outperform the state-of-the-art abstractive summarization technique on our dataset for the reasons listed below. The human-labeled contexts are chosen straight from the original root cause details with no alteration to the phrase structure. Pegasus, an abstractive summarization model that pre-trained on the Gap Sentences Generation task, may immediately replicate the important sentences from the input, resulting in a higher overall rouge score than the traditional abstractive technique.

Figure 4 illustrates an example of context extraction from a PIR report consisting of 328 words. The human-labeled context is shown in blue and green, whereas the context extracted from Pegasus is shown in the green and red. This example shows that ≈50% tokens can be filtered, which can considerably enhance the efficiency with which on-call engineers read the PIR report. Also, we discover that the extracted context from Pegasus has a high recall compared to the human labels, allowing engineers to seldom overlook vital information.

Section	Micro-F1	Weighted-F1
Whole PIR	**0.55**	**0.40**
Title	0.53	0.45
Summary	0.47	0.46
RC-Details	0.52	0.45
5-Whys	0.54	0.40
Discussion	0.53	0.40
Mitigation	0.47	0.40
RC-Details + 5-Whys	**0.56**	**0.42**

Table 3: **Study on the utility of different PIR sections in top-level root cause classification using Random Forests.**

6 Evaluation

We now empirically evaluate the performance of AutoARTS.

Dataset. Our dataset consists of 1120 PIRs that are expert-annotated with ARTS root cause tags and contextual sentences to justify them. We use stratified sampling to divide this dataset into train(72%) and validation(8%) splits to train and tune the hyperparameters of different models and test(20%) split to report the results with the trained models.

Data Pre-processing. We found that engineers often included various types of data such as debugging queries issued on logs, error messages, stack traces, screenshots, etc., in PIRs (also identified in [31]). These add significant noise to the vocabulary of the language processed by NLP models, without contributing to performance. We carefully remove such noise with regular-expression based filters and only select alphabetic text for our evaluation. For experiments in §6.1, we also use the NLTK [3] library to remove stop-words and extract stems of words to construct vocabulary.

Evaluation Metrics. For root cause classification, we use micro-F1 score to analyze performance across different incidents with multiple labels. We also use weighted-F1 score to analyze performance across different classes since our dataset is imbalanced as shown in Table 1. For context extraction, we use ROUGE (Recall-Oriented Understudy of Gisting Evaluation) [20] and BLEU (Bi-Lingual Evaluation Understudy) [25] scores to evaluate the similarity of extracted context against the ground truth. Rouge-N score is based on the percentage (higher the better) of N-grams from the ground truth that are present in the extracted context. BLEU-N score indicates the percentage of N-grams from the extracted context that are present in the ground truth. Rouge-L F1-score is based on the longest common subsequence (not necessarily consecutive) between the extracted context and target context.

6.1 Featurization and Feature Selection

R1: Given the input text sequence length limitations of DL models, what information should be used from PIRs?

Sophisticated DL language models impose constraints on input sequence length (e.g., 512 tokens for BERT [8]). The

Model	ROUGE			BLEU			
	Rouge-1	Rouge-2	Rouge-L	BLEU	BLEU-1	BLEU-2	BLEU-3
Pegasus - Pretrained	32.55	18.72	24.30	9.61	18.03	10.31	8.93
Pegasus - Finetuned	**45.46**	**35.65**	**38.43**	**24.60**	**32.19**	**24.98**	**23.41**
T5 - Pretrained	34.38	23.31	28.03	10.06	15.68	10.83	9.43
T5 - Finetuned	41.63	33.86	35.76	23.81	29.81	24.10	22.70
BERT-cased - Pretrained	40.05	27.03	31.01	18.62	28.43	18.95	16.83
BERT-cased - Finetuned	40.08	27.35	31.20	18.80	28.32	19.03	16.95
BERT-uncased - Pretrained	39.52	26.58	30.74	17.63	27.47	17.98	15.89
BERT-uncased - Finetuned	39.92	27.44	31.57	18.64	28.08	18.91	16.90

Table 4: **Performance of Pegasus and T5 models with their corresponding pre-trained versions and fine-tuned versions. We also present performance of unsupervised clustering based approach for extractive summarization using BERT.**

Model	Micro-F1	Weighted-F1
HiAGM	83.16	89.63
HiAGM_Flat	45.40	68.66
BERT_MLC	42.29	46.85

Table 5: **Performance of HiAGM compared to using flattened root cause taxonomy (HiAGM_Flat) and a finetuned-BERT based multilabel classifier (BERT_MLC).**

Model	Test Perplexity	Test Accuracy
BERT-uncased	7.57	34.83%
BERT-cased	6.69	35.26%

Table 6: **Performance of MLM based pre-finetuning and OCE-assigned root cause based finetuning of BERT.**

limit is much smaller than our preprocessed PIRs (avg. length of $\approx$ 1900 words). However, a PIR is organized into multiple sections and we conduct an ablation study by featurizing each section in the PIR into Bag-of-Words encodings and classify them to top-level root cause categories using a Random Forest classifier. Table 3 highlights that "root cause details" and "5-Whys" sections achieve better micro-F1(1.8% higher) and weighted-F1 (5% higher) scores compared to using the whole PIR. These sections capture information relevant to root cause classification and by only using them, we minimize sequence length to meet constraints imposed by DL models.

6.2 Hierarchical Classification

R2: Is the hierarchical nature of our taxonomy beneficial in leveraging relationships between root causes to classify incidents?

Table 5 compares the level-3 root cause classification performance of our trained HiAGM model against a flattened version of our hierarchical taxonomy (HiAGM_Flat), where we remove parent-child relationships between different root causes in the taxonomy and consider all the root causes tags to be opaque and independent of each other. We also compare HiAGM against a multi-label classifier (BERT_MLC) with the flattened version of the taxonomy using finetuned BERT [8] model (details in §6.4) to encode the PIR text. We observe that HiAGM performs significantly better (31% higher weighted-F1 measure) than HiAGM_Flat indicating the utility of GCN to leverage neighboring relationships between root causes and the need for root cause taxonomies to be hierarchical. HiAGM performs significantly better (91% higher weighted-F1 measure) than BERT_MLC along with HiAGM_Flat (47% higher weighted-F1 measure), demonstrating no utility in finetuning existing language models on PIRs.

6.3 Context Extraction

R3: How do supervised (abstractive and extractive) or unsupervised (extractive) summarization approaches fare for context extraction? Is finetuning necessary for context extraction and does it work with limited data?

Using the train and validation splits of the dataset, we finetune Pegasus for 15 epochs and T5 (3 Billion parameters) [27] for 7 epochs and report the results on the test set. Table 4 compares the performance of finetuned Pegasus model against baseline approaches using T5 and clustering-based extractive summarization [23] using BERT. We can clearly see that our finetuned Pegasus model achieves the highest performance across various ROUGE and BLEU metrics. We observe a significant (58.15%) improvement in Rouge-L score as a result of finetuning Pegasus, because pre-trained version of Pegasus is trained on significantly different domains of language such as news articles, etc., and is trained to summarize them, which is different from context extraction. We also observe a 7.6% increase in ROUGE-L score compared to the finetuned-T5 model, because Pegasus extracts sequences of text from the PIR as opposed to T5 which generates new sequences of words, which might not represent the content present in the PIR which is where engineers derive their context from. Finetuned-Pegasus performs significantly higher (21.73%) than unsupervised clustering based summarization approaches using finetuned-BERT, because summary of the PIR doesn't represent the context.

Metric	Response	Description
(Q1) Usefulness of generated context to identify contributing factors	4.6/5	1 - Not useful at all, 5 - Very useful
(Q2) # Contexts generated with unnecessary details	0/10	No unnecessary details in generated contexts
(Q3) # New Rootcauses from generated contexts	2	False negatives identified by AutoARTS
(Q4) # Examples with a crucial Rootcause tag missing in classification	7/10	Crucial contributing factor missing from predictions

Table 7: **Quantitative user feedback from an expert over the effectiveness of AutoARTS across context generation and root cause classification tasks over a randomly chosen set of 10 incidents.**

6.4 Fine-tuning Language Models

R4: Can existing language models like BERT be finetuned to model software incident reports?

We conducted Masked Language Modeling(MLM) based pre-finetuning of BERT to fit our domain-specific language model, using 110K PIRs (un-tagged due to scale) from several Microsoft services. We then finetuned the models by leveraging the OCE-assigned "root cause Category" tag (which are chosen from the predefined taxonomies in Microsoft) of each of the PIRs. As mentioned in §2, OCE-assigned root cause category tags are not accurate; however, they are available at a large scale and this classification task is semantically the closest to our target classification task using the ARTS taxonomy. Table 6 shows the results for both pre-finetuning and finetuning tasks, highlighting a high perplexity of 7.57 for BERT-uncased (perplexed between choosing 8 candidate words for a blank in a given sentence) and poor classification accuracy ($\approx 35\%$) on the finetuning task. Errors in tagging of PIRs (by OCEs) coupled with lack of sufficient training data makes finetuning language models infeasible. Due to space constraints, we omit the results from other variations of BERT, but we had similar experience with them.

7 User Study

To evaluate the utility of AutoARTS, we randomly sampled 10 example incidents and the tool's generated contexts, the corresponding root cause categories and presented them to one of the leads that developed the ARTS taxonomy (by studying PIRs). These were examples that were tagged by them in the past that are not used for training any of our models. The goal of this study is to understand, for each example: (Q1) How useful the generated context is in identifying all the contributing factors that they identified, (Q2) If the generated context has extra details that are not useful for identifying contributing factors (to evaluate the succinctness of our generated contexts), (Q3) Whether the generated context can help them identify any new contributing factor that they have not identified previously (to evaluate the generalization of the model's outputs beyond accidental False Negatives in ground-truth) and (Q4) Whether the tool missed the most important contributing factors (to evaluate the importance of False Negatives).

Although we quantitatively evaluated the syntactic similarity of generated contexts to the ground truth, the developer study helps us understand if they are semantically similar and ultimately usable by a human (OCE). Similarly for Root cause classification task, the relative severity of each individual contributing factor is not identifiable from ground truth (no ranking). Q4 helps us understand if the predicted contributing factors miss any crucial tag from the ARTS taxonomy.

We quantify response to Q1 on a Likert scale of 1 to 5, where 1 meant 'not useful at all' and 5 meant 'Very useful'. Q2-Q4 were answered as a binary Yes/No, by providing clarifying responses wherever necessary for sanity check. Table 7 shows the utility of the tool based on the subject's responses to Q1-Q4. We find from the study that the contexts generated by the tool are extremely useful in identifying all the contributing factors and they are succinct enough without presenting additional information that is not useful in identifying contributing factors. In addition to this, we also found that our tool helped the subject find 2 new root cause tags that should have been assigned to these incidents in the past, highlighting the difficulty in manually sifting through postmortem reports to identify contributing factors.

At the end of the experiment, the subject was asked to answer on a scale of 1-5 (5 being very useful, 1 being not useful at all), indicating the overall usefulness of our tool to assist them in their task based on the 10 examples. The subject rated our tool 'above 4.5'. In addition to this, the subject's verbatim feedback on our tool — **'This tool is very useful from context generator perspective for the root cause classification task. From the Tags perspective, if we had 4th level for just code change related tags this is very useful for change management standards team. Need to fix the dependency tags related logic as it's defaulting to "Data Bricks". Over all I am very happy with this tool'** — highlights the utility of our context generation and the lapses in automated root cause classification. The imbalance in tag distribution over our training set resulted in misclassifying incidents with tags that do not have sufficient supporting training samples. Overall, the feedback indicates the promise for deploying the tool for practical use in assisting engineers by providing enough context from PIRs to assign root cause tags from ARTS taxonomy.

8 Related Work

Rootcause analysis of past incidents. Rootcause analysis of incidents and outages and defining taxonomies to capture their root causes has been a popular topic of study in the software engineering and systems community. We find that prior work can be categorized into two major buckets. The first category of prior work focuses on specific type of production issues such as software bugs [4, 5, 11, 19, 21, 40] or network issues [14]. The other category focuses on specific services or systems such as big-data systems [38], business data processing platform [9], high performance computing [17, 29] and a specific cloud service [13]. In contrast, we consider a large-scale cloud system consisting of many hundreds of services and *all* types of failures including hardware and software, infrastructure and application, software code and configuration, and so on. The scale of our study also differentiates us from existing studies (e.g., 152 incidents from 1 service considered in [13], 100 incidents from networking service in [14]). To the best of our knowledge, this is the most comprehensive effort of analyzing production incidents and building a fine-grained taxonomy. Prior work that propose solutions to simplify the task of actively identifying the root cause of a failure [10, 32] are orthogonal to our focus.

Text Summarization & Root cause classification. Text summarization [33] is the task of rewriting a long document into a condensed form while retaining its essential meaning, hence reducing the burden to read through lengthy documents. The most prevalent paradigms for summarization are extractive and abstractive based approaches. When generating summaries, abstractive summarization approaches [35] are typically considered as a sequence-to-sequence learning problem [24, 26, 30], whereas extractive summarization methods [12, 41] extract key sentences as summary directly from the text. In our context extraction task, we utilize the gap sentence based summarization technique not to condense the PIR context, but to extract essential text snippets describing different contributing factors. Saha et al. [28] construct a causal knowledge graph from postmortem reports but do not generate consistent root-cause tags for incidents. This can lead to ambiguous and non-uniform tagging of similar incidents, as we observed from manual tagging using different taxonomies (in Finding 2).

Prior work focused on diagnosing different kinds of incidents such as, Rex [22] suggests changes in potential misconfigurations using syntax trees, DeepAnalyze [32] identifies culprit frame in Windows Error Reporting (WER) crash stack traces, Orca [2] identifies buggy commits using differential code analysis and provenance tracking, Revelio [10] generates debugging queries for root cause analysis using logs and user reports, SoftNER [31] analyzes postmortem reports to extract entities. To the best of our knowledge we are the first to classify incident postmortems into an extensive high-granularity taxonomy.

9 Discussion

Generality of AutoARTS. Postmortems are routinely conducted in large scale production cloud systems to document learnings from incidents, similar to PIRs in Microsoft Azure (e.g., Google [14], AWS [1], and Cloudfare [7]). These postmortem reports may have different structures and content across different clous systems, but they all contain natural language descriptions of root cause diagnosis of incidents. Since the ARTS taxonomy is based on a diverse and large set of services and incidents, and AutoARTS is trained on postmortems, we believe that they can benefit other cloud systems.

Evolution of Taxonomy. Our open-sourced ARTS taxonomy captures a wide range of contributing factors, but new factors may emerge and the taxonomy may evolve. Therefore, we deliberately separate context generation and classification in AutoARTS so that new categories can be detected from the generated contexts (Table 7). When a new category is identified, the HiAGM model of AutoARTS can be finetuned for the new tags only (which takes a few minutes). We also hope others can contribute to the growth of ARTS taxonomy.

10 Conclusion

Incident postmortems are treasure troves of rich insights and retrospective analyses of them reveals actionable insights to improve reliability and availability of large-scale production systems. Unfortunately, it's not done at scale today because of the manual effort required in analyzing them. We developed a novel hierarchical and comprehensive taxonomy based on a painstaking extensive multi-year analysis of 2000+ severe incidents across 450+ Microsoft Azure services. To make this taxonomy accessible and assist engineers in analyzing postmortem reports, we proposed techniques to extract key context from postmortems and automatic classification to identify all the contributing factors for an incident and presented our findings. To the best of our knowledge, this is the largest study of production incident postmortem reports yet.

We envision that this paper enhances the audiences' understanding of contributing factors for production incidents and fosters future research by leveraging the open-sourced taxonomy for root cause labelling. Our experimental findings show promise in assisting engineers with classifying postmortem reports and we intend to fully automate this task incorporating engineers' feedback and leverage larger training datasets in future work.

Acknowledgements

We thank the ATC reviewers and our shepherd Yu Jiang for their insightful comments. Pradeep Dogga was partially supported by NSF grant CNS-1901510 and research grants from Google, Amazon and Cisco.

References

[1] aws.amazon.com. Post-Event Summaries. https://aws.amazon.com/premiumsupport/technology/pes/.

[2] R. Bhagwan, R. Kumar, C. Maddila, and A. A. Philip. Orca: Differential bug localization in large-scale services. In *13th USENIX Symposium on Operating Systems Design and Implementation*. USENIX, October 2018. Won the Jay Lepreau Best Paper Award.

[3] S. Bird. Nltk: The natural language toolkit. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, COLING-ACL '06, page 69–72, USA, 2006. Association for Computational Linguistics.

[4] H. Chen, W. Dou, Y. Jiang, and F. Qin. Understanding exception-related bugs in large-scale cloud systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 339–351. IEEE, 2019.

[5] X. Chen, C.-D. Lu, and K. Pattabiraman. Failure analysis of jobs in compute clouds: A google cluster case study. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 167–177. IEEE, 2014.

[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[7] Cloudfare. Cloudfare Status - Incident History. https://www.cloudflarestatus.com/history.

[8] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[9] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, G. Goel, S. Sarkar, and R. Ganesan. Characterization of operational failures from a business data processing saas platform. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 195–204, 2014.

[10] P. Dogga, K. Narasimhan, A. Sivaraman, S. K. Saini, G. Varghese, and R. Netravali. Revelio: Ml-generated debugging queries for finding root causes in distributed systems. In *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022.

[11] Y. Gao, W. Dou, F. Qin, C. Gao, D. Wang, J. Wei, R. Huang, L. Zhou, and Y. Wu. An empirical study on crash recovery bugs in large-scale distributed systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 539–550, 2018.

[12] S. Gehrmann, Y. Deng, and A. M. Rush. Bottom-up abstractive summarization. *arXiv preprint arXiv:1808.10792*, 2018.

[13] S. GHOSH, M. Shetty, C. Bansal, and S. Nath. How to fight production incidents? an empirical study on a large-scale cloud service. In *SoCC 2022*. ACM, November 2022.

[14] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat. Evolve or die: High-availability design principles drawn from googles network infrastructure. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 58–72, New York, NY, USA, 2016. Association for Computing Machinery.

[15] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, et al. What bugs live in the cloud? a study of 3000+ issues in cloud systems. In *Proceedings of the ACM symposium on cloud computing*, pages 1–14, 2014.

[16] H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, and K. J. Eliazar. Why does the cloud stop computing? lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 1–16, 2016.

[17] D. Jauk, D. Yang, and M. Schulz. Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[18] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[19] T. Leesatapornwongsa, J. F. Lukman, S. Lu, and H. S. Gunawi. Taxdc: A taxonomy of non-deterministic concurrency bugs in datacenter distributed systems. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 517–530, 2016.

[20] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[21] H. Liu, S. Lu, M. Musuvathi, and S. Nath. What bugs cause production cloud incidents? In *Workshop on Hot Topics in Operating Systems (HotOS)*, May 2019.

[22] S. Mehta, R. Bhagwan, R. Kumar, C. Bansal, C. Maddila, B. Ashok, S. Asthana, C. Bird, and A. Kumar. Rex: Preventing bugs and misconfiguration in large services using correlated change analysis. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 435–448, Santa Clara, CA, Feb. 2020. USENIX Association.

[23] D. Miller. Leveraging BERT for extractive text summarization on lectures. *CoRR*, abs/1906.04165, 2019.

[24] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

[25] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[26] R. Paulus, C. Xiong, and R. Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[27] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[28] A. Saha and S. C. Hoi. Mining root cause knowledge from cloud service incident investigations for aiops. *arXiv preprint arXiv:2204.11598*, 2022.

[29] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. *Dependable and Secure Computing, IEEE Transactions on*, 7:337 – 351, 01 2011.

[30] A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

[31] M. Shetty, C. Bansal, S. Kumar, N. Rao, N. Nagappan, and T. Zimmermann. Neural knowledge extraction from cloud service incidents. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 218–227. IEEE, 2021.

[32] M. Shetty, C. Bansal, S. Nath, S. Bowles, H. Wang, O. Arman, and S. Ahari. Deepanalyze: Learning to localize crashes at scale. In *ICSE 2022*, May 2022.

[33] T. Shi, Y. Keneshloo, N. Ramakrishnan, and C. K. Reddy. Neural abstractive text summarization with sequence-to-sequence models. *ACM Transactions on Data Science*, 2(1):1–37, 2021.

[34] A. Vidyasagar. The art of root cause analysis. *Quality Progress*, 49(1):48, 2016.

[35] S. Xu, X. Zhang, Y. Wu, and F. Wei. Sequence level contrastive learning for text summarization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11556–11565, 2022.

[36] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.

[37] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[38] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. U. Jain, and M. Stumm. Simple testing can prevent most critical failures: An analysis of production failures in distributed {Data-Intensive} systems. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 249–265, 2014.

[39] J. Zhang, Y. Zhao, M. Saleh, and P. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.

[40] Y. Zhang, J. Yang, Z. Jin, U. Sethi, K. Rodrigues, S. Lu, and D. Yuan. Understanding and detecting software upgrade failures in distributed systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 116–131, 2021.

[41] M. Zhong, P. Liu, Y. Chen, D. Wang, X. Qiu, and X. Huang. Extractive summarization as text matching. *arXiv preprint arXiv:2004.08795*, 2020.

[42] J. Zhou, C. Ma, D. Long, G. Xu, N. Ding, H. Zhang, P. Xie, and G. Liu. Hierarchy-aware global model for hierarchical text classification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1106–1117, 2020.