# Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models

Toufique Ahmed*§, Supriyo Ghosh†, Chetan Bansal†
Thomas Zimmermann‡, Xuchao Zhang†, Saravan Rajmohan†
*UC Davis
†Microsoft
‡Microsoft Research

***Abstract*—Incident management for cloud services is a complex process involving several steps and has a huge impact on both service health and developer productivity. On-call engineers require significant amount of domain knowledge and manual effort for root causing and mitigation of production incidents. Recent advances in artificial intelligence has resulted in state-of-the-art large language models like GPT-3.x (both GPT-3.0 and GPT-3.5), which have been used to solve a variety of problems ranging from question answering to text summarization. In this work, we do the first large-scale study to evaluate the effectiveness of these models for helping engineers root cause and mitigate production incidents. We do a rigorous study at Microsoft, on more than 40,000 incidents and compare several large language models in zero-shot, fine-tuned and multi-task setting using semantic and lexical metrics. Lastly, our human evaluation with actual incident owners show the efficacy and future potential of using artificial intelligence for resolving cloud incidents.**

***Index Terms*—Incident Management, Service Quality, GPT-3.x, Large Language Models**

## I. Introduction

Large IT enterprises such as Amazon, Google, Microsoft, and Salesforce have replaced the traditional shrink-wrapped software and moved towards deploying applications and services on cloud platforms. In today's cloud systems, production incidents (e.g., outage or performance degradation, unplanned interruptions) adversely impact the customers and can be expensive in terms of penalty associated with service level agreement violations and engineering efforts required to mitigate the incidents. For example, one hour of downtime is estimated to cost Amazon US$100 million on major shopping days [1]. Despite continuous reliability efforts over the years, cloud services still experience inevitable severe incidents.

Artificial Intelligence (AI) for IT Operations, also known as AIOps, has increased in popularity. Data-driven and AI techniques have been leveraged for automating parts of the incident life-cycle, for example, incident prioritization [2], retrieval of incidents with similar symptoms [3], and reducing the time to mitigate incidents [4], [5]. However, on-call engineers (OCEs) still spend a significant amount of manual toil through multiple rounds of back and forth communication for identifying *root causes* and *mitigation steps*. Motivated by the recent successes of leveraging GPT-3 models for non-trivial tasks [6], [7] and code generation [8], we apply such models to incident management. We identified the following two scenarios:

1) **Find the incident's root cause.** Diagnosing incidents typically requires significant time and communication before engineers can identify the root cause of the incident. We investigate how effective large language models are at suggesting root causes for incidents (RQ1).
2) **Suggest the mitigation steps for the incident.** After a root cause has been located, engineers take actions to mitigate the problem. We investigate how effective large language models are at recommending the mitigation steps for incidents (RQ2).

When applying large language models several considerations and decisions need to be taken. Since the models were not trained with incident management data, is *fine-tuning* of the models necessary (RQ3)? Is it more effective to build one model for each task (*single-task*) or one combined model that supports both root causes and incidents (*multiple task*) (RQ4)? Does the root cause help language models to find better mitigation steps (RQ5)? Do the models perform better for certain types of incidents (RQ6)? We address these questions with a rigorous large-scale evaluation of 44,340 incidents from 1,759 services of Microsoft. In addition to lexical and semantic evaluation metrics that are typically reported for such experiments, we present the results from a human validation, where we asked incident owners to assess the correctness and readability of suggested root causes and mitigation steps. The original incident owners are the most qualified to assess the performance of the models on incidents. In this paper, we make the following contributions:

1) This is the first work to demonstrate the usefulness of state-of-the-art large language models (LLMs) such as GPT-3.x (both GPT-3.0 and GPT-3.5) for resolving production incidents in a real world setting. (Section III)
2) We present a rigorous and large-scale study in Microsoft on over 40,000 incidents from 1000+ cloud services with six semantic and lexical metrics. (Section IV)
   - Fine-tuning significantly improves the effectiveness of LLMs for incident data.
   - GPT-3 and GPT-3.5 models significantly outperform encoder-decoder models in our experiments.
   - Metrics such as BLEU-4 are useful to measure relative

[§] This work is done during the author's internship at Microsoft Research.

performance of models in different settings. However, manual inspection and validation with experts is needed to assess the actual performance.

3) Our human study with the actual incident owners of production incidents helps prove the efficacy of the proposed approach. (Section V)

## II. Overview

### A. Incident management

Production incidents are inevitable in large-scale cloud services and often severely affect the customer experience. Also, they can be extremely expensive in terms of engineering resources required to root cause and mitigate them. An incident life-cycle typically has the following four stages: (1) **Detection:** The first step in the incident life-cycle is detection where the incidents are either reported by internal or external customers of a given service after they notice anomalous behavior. Also, incidents can also be reported via automated monitors which are configured by the service owners. (2) **Triaging:** Once an incident is reported, a team of OCEs analyze the problem and route the incident ticket to appropriate engineering team. This process is often referred as incident triaging. (3) **Diagnosis:** The incident diagnosis and root cause identification process requires multiple iterations of back and forth communication between engineers inspecting the different aspects to understand the broad nature of the incident and identify the root cause. (4) **Mitigation:** Based on the identified root causes, actions are taken to mitigate the problem so as to recover the service health and minimize the impact on the service users.

Lately, AIOps (AI for IT Operations) has gained popularity for automating various parts of the incident life-cycle by combining data-driven and AI techniques with data-sources like application logs, time series performance metrics and service traces [2], [4], [5], [9]. Albeit significant efforts, incident management in large cloud systems still requires a huge amount of engineering effort and cost. More specifically, even with plethora of historical incident data, root cause identification and mitigation remains notoriously challenging and time consuming tasks. In this work, we propose to use large language models such as GPT-3.x to automatically recommend root causes and mitigation for new incidents by leveraging historical incident data.

### B. The promise of LLMs/GPT-3.x models

Large language models (LLMs) such as GPT-3.x [7] have emerged as one of the hottest trends in natural language processing over the last few years. With 175 billion parameters, the GPT-3.x language models, which held the record for being the largest neural network ever developed, is an order of magnitude larger than prior language models. Using this massive model architecture, GPT-3.x were trained using almost all accessible data from the Internet, including CommonCrawl [10], WebText [11], Wikipedia [12], and a corpus of books. GPT-3.x models surpass the state-of-the-art models in a variety of NLP tasks, including machine translation, question-answering, and close tasks. Furthermore, the GPT-3.x models achieved a significant milestone by showing that unsupervised language models trained with adequate data can multi-task to the same level of fine-tuned models using just a few examples of the new tasks. As a result of its powerful text generation capabilities in new tasks, GPT-3.x are used in a wide range of categories and industries, from productivity and education to creativity and gaming. For instance, GPT-3.x are used to produce creative writing, including blog posts, advertisements, and poetry, that mimics the literary style of well-known writers like Shakespeare.

> **Title:** Attach vm fails with connection timeout
>
> **Summary:** The workspace is not associated with any vnet. Customer has a vm which is already running inside a vnet. They like to attach that vm into [product omitted]. We tried the UI and CLI route, but still fails with same connection timeout error. Error points that it resolves to some public ip [...]
>
> **Reference root cause:** It is not supported to attach a private vm to a public workspace directly.
>
> **Reference mitigation:** Open a task to provide better official document for customer on the topic of virtual machine.

Fig. 1: A sample production incident.

### C. Root-causing and mitigating incidents

Incident root-causing and mitigation is a complex process which requires significant amount of manual effort and, also, domain knowledge about the services. Incidents can be caused by various kind of issues such as code bugs, dependency failures, infrastructure issues, configuration bugs, etc. Due to the vast number of possibilities, it is non-trivial for the OCEs to root cause the incidents. Similarly, once the root cause is identified, there can be various mitigation steps which can be taken such as code rollback, hotfix, infrastructure changes, configuration update, etc. Identifying the correct mitigation step is again non-trivial and requires domain knowledge and experience. Human errors in root causing or mitigation of incidents results in not just more effort and human toil but also impact on the customers and the revenue. Fig. 1 shows a real incident from a service where we can see the title and summary provided by the customer along with the actual root cause and mitigation.

In this study, we evaluate the effectiveness of large language models like GPT-3.x and Codex for root causing and mitigating production incidents. When an incident is created, the author would specify a title for the incident and describe any relevant details such as any error messages, anomalous behavior and other details which could potentially help with resolution. Once the OCE starts investigating the incident, they might get more details by communicating with the incident author or by looking at telemetry and logs. During the course of the investigation, the OCE might often updates the incident details. For our evaluation, we use the title and the summary of a given incident at the time of incident creation as input and generate the root cause and mitigation steps. This is to ensure that we only use the information which was available to the OCE when they started investigating the incident.

### D. Research questions

We investigated several OpenAI GPT-3.x models (*i.e.,* Curie, Codex-cushman, Davinci, Code-davinci-002) to generate root causes and mitigation plans for the incident. This leads to several RQs.

*RQ1 Are fine-tuned GPT-3.x models effective at finding the incident's root cause?*
The OpenAI models are not trained with the incident management data since the data contain sensitive privacy information, and Microsoft follows standard protocols to ensure the security of the data. Therefore, the GPT-3.x models are not expected to perform well in zero-shot/few-shot settings. In this paper, we fine-tuned four different GPT-3.x models with different capacities and observed how the models performed at proposing the root causes of the incident.

*RQ2 Are fine-tuned GPT-3.x models capable of suggesting the mitigation plan for the incident?*
We are also interested in generating mitigation plans for the incident using GPT-3.x models. Like root cause generation, we fine-tune and evaluate the model using the input and criteria we use for RQ1.

*RQ3 How much fine-tuning improves over zero-shot learning performance of GPT-3.x models?*
Though we primarily focus on fine-tuning, GPT-3.x models are reported to be effective at various downstream tasks with zero-shot and few-shot training [7], [8]. In few-shot learning, we use a few examples in the prompt as input to the model, and the model generates the expected output. Zero-shot is similar to few-shot training, but none of the examples are given. These two settings are economically and environmentally beneficial (reduced carbon footprint) because we are not updating any parameters of the models. This paper will investigate how the models perform at zero-shot settings. Note that few-shot learning is unsuitable for our project because we have long sequences in our dataset, and we observe the truncation of the sequences when we infer only one sequence after fine-tuning.

*RQ4 Does multi-task learning improve the performance of GPT-3.x models at finding root causes and mitigation plans?*
Multi-task learning is effective for some pre-trained models [13]. So far, we have discussed separate training models and using the input independently to generate the incident's root cause and mitigation plans. We are interested in how GPT-3.x models react to multi-task learning in our specific setting. We combine all the training data for this experiment for both tasks. However, during evaluation, we used the same test sets used in RQ1 and RQ2.

*RQ5 Do GPT-3.x models get better at proposing mitigation plans if the root cause is given?*
Mitigation plans for an incident depend on the specific root cause. Different root causes may lead to different mitigation plans. Moreover, the GPT-3.x models can be improved by making the input larger or more informative. We will also investigate whether providing the root cause in the input help models find the mitigation plans.

*RQ6 Do the models better propose mitigation plans for machine-detected incidents than human-detected ones?*
Incidents can be machine-detected (by some monitors) or human-detected. Both types of incidents have specific characteristics. Machine-detected incidents are generally triggered when the monitor observes system changes like build failures, resource availability, request counts, etc. On the contrary, human-detected incidents are unique and may apply to a specific customer (*e.g.,* webpage is not loading). In the research question, we will investigate if the model performs well for incidents belonging to a specific class.

### E. Human validation

Root causes and mitigation plans can be written in different forms. Unlike natural language translation or code summarization, root causes and mitigation steps are much more open-ended. Depending on the author, the root causes and mitigation plans can vary from generic to specific. Automatic metrics may fail to reflect the overall performance of the models ideally because these metrics compare the models' suggestions with one reference, which may be completely different from the models' correct and relevant outputs. To better understand the model's performance, we went to the owner/resolver of the specific incidents and presented the solutions from our models and baselines. They assigned correctness and readability scores to the model's output. We will discuss our methodology and findings from the human validation in Section V.

## III. Methodology

### A. Dataset Preparation

Thousands of incidents with different severity are being detected (by both machines and humans) every day at Microsoft. The on-call engineers (OCEs) are working relentlessly to provide seamless service to the customers. To manage incidents at that scale, Microsoft has a well-designed website for reporting and managing the incident. A database also keeps track of the website's data insertion, modification, and deletion from incident reporting to mitigation. One of the inputs to the model is the summary written at the time of incident reporting or creation to prevent any data leakage from input to output.

In most cases, the OCEs do not follow any specific format to write incident summaries, root causes, and mitigation plans. The fields, especially summaries, contain information in multiple forms, including tables, links to prior incidents, and images of individual monitor output or code snippets. This is because the incidents are very different from each other, and the utmost priority of the OCEs is to resolve the incident rather than document the symptoms. Also, some incidents are transient and auto-mitigated. No postmortem is done if the severity is low. Since GPT-3.x are text models, we discarded the tables and images from the summaries. Hence, there is a chance that we lost some critical information while discarding that information.

We collected data for incidents from the database that has the creation date between January 1, 2018, to July 15, 2022. Initially, we collected 123,953 instances for root causes and 23,544 mitigations from the "Resolved" or "Mitigated" incidents with severity levels 0-3 (most severe incidents belong to level 0). The samples for mitigation are low because they can be found in the postmortem of the incident, and postmortem are not done for every incident. After collecting the data, we observe many incidents with duplicate root causes and mitigations. Some severe incidents/ denial of service trigger hundreds of incident reports for the same event, all of which have the exact root causes and mitigations. To fairly evaluate the model, we remove the exact duplicates for root causes and mitigation plans and end up with 57,520 root causes and 8,300 mitigation plans. The average root causes and mitigations lengths are 87 and 12 tokens, respectively. Some root causes are very long, and it is difficult for the models and human evaluators to generate and evaluate the models' output. We kept the root causes up to 100 tokens, allowing us to keep 73% of the instances for root causes. We also discarded root causes and mitigation plans with less than three tokens because those are not informative.

After deduplication and filtering, we sorted the data according to the creation date to use only historical data for training the model. We selected 35820, 3000 and 2000 root causes for training, testing and validation. We have fewer instances for mitigations. Hence, the training, test and validation sets for mitigations have 5455, 2000 and 500 data, respectively. Even after this rigorous filtering and deduplication of data, some root causes and mitigations do not carry any useful information (*e.g.,* root cause in a different link, transient, and auto-mitigated incidents). We manually went through 3000 root causes and 2000 mitigation plans from test sets and selected 2,621 root causes and 1,780 mitigation plans. [1]

### B. OpenAI models and baselines

The recent advancement of the deep neural network models is greatly influenced by the introduction of Transformer models [14]. Prior approaches (*i.e.,* LSTM [15] and GRU [16]) modeled the sequential dependencies of the generated text using recurrent architecture. These recurrent models use "Back-Propagation through Time" (BPTT) to recursively propagate loss values over gradients within the same recurrent units prohibiting the possibility of parallel computation while capturing the long-distance dependencies of the tokens in the sequence. Bahdanau *et al.* introduced an attention mechanism that works on top recurrent architecture and improves the performance of recurrent neural models by providing an attention vector that indicates the relevant part of the input to the target output [17]. Transformer model completely removes the recurrence unit and entirely relies on the attention mechanism. It uses a multi-layer, multi-head self-attention architecture where the attention mechanism can relate different positions of a single sequence to compute a sequence representation.

Pre-trained models are currently achieving state-of-the-art performance for various natural language and code tasks. These pre-trained models work in 2 stages (*i.e.,* pre-training and fine-tuning). In the pre-training stage, we train the model to learn statistics of language (or code) in a self-supervised fashion from large-scale corpora. After that, we use a smaller labeled dataset to fine-tune the model for specific tasks. It is nearly infeasible to have sufficient labeled data to train such high-capacity deep learning models. Pre-trained models enable us to train such big models with the unlabeled data in a self-supervised way in the pre-training stage. All the recent pre-trained (encoder-only and encoder-decoder) models (*e.g.,* BERT [18], RoBERTA [19], BART [20], T5 [21]) and decoder-only generative models (*e.g.,* GPT [22], GPT-2 [23], GPT-3 [7], OPT [24]) are basically Transformer models of various capacity trained with different pre-training objectives. The following subsections briefly discuss the baselines and OpenAI models we used for our experiments.

*1) Baselines encoder-decoder models:* We can apply the encoder-decoder models for both root cause and mitigation. The encoder will encode the input, and the decoder will generate the root cause or mitigation using the encoded representation provided by the encoder.

Pre-trained NLP models (*e.g.,* BERT [18], RoBERTa [19], BART [20], T5 [21]) use different self-supervised pre-training objectives to learn robust language representations. NLP models have programming language counterparts (*e.g.,* CodeBERT [25], GraphCodeBERT [26], PLBART [27], CodeT5 [13], NatGen [28]) where the models are initialized with the NLP models' weights and continued pre-training with code and associated natural language comments in most cases. Though root cause and mitigation are natural language descriptions, the vocabulary (*e.g.,* identifiers) overlaps more with the comments used in code models. Therefore we picked both NLP and code models from OpenAI and baseline criteria to see if the performance differs depending on the domain used for pre-training. For baselining, we pick RoBERTa [19] and CodeBERT [25] models because of two reasons: i) these two models are architecturally identical with 125M parameters, ii) Both models are widely used as baselines (in fact, CodeBERT is the primary baseline model of the CodeXGLUE [29] dataset, which is a popular benchmark of 10 SE tasks including encoder-decoder tasks like code summarization and code translation). Note that many transformer-based encoder-decoder models can be applied to this problem. However, comparing with all the models is beyond the scope of the paper.

***RoBERTa:*** BERT is the first model that introduced the pre-training strategy that outperforms the traditional Transformer models. It applied two pre-training strategies: Masked Language Modeling (MLM) and NSP (Next Sentence Prediction). In MLM pre-training, we randomly mask out 15% of the tokens and ask the model to recover those tokens, whereas, in NSP, we train the model to learn to predict the next sentence following an input sentence. Liu *et al.* [19] propose RoBERTa (A Robustly Optimized BERT Pre-training Approach), which outperforms the BERT model with a few changes, such as

[1] We cannot share the dataset because incident data can contain confidential and private data and sharing such data would violate the terms of service.

dynamic masking and dropping NSP, achieves better performance. We apply RoBERTa as NLP baseline model.

***CodeBERT:*** CodeBERT is architecturally identical to RoBERTa model that uses two pre-training objectives: MLM and Replaced Token Detection (RTD) [30]. We can define RTD as a binary classification problem where two data generators (i.e., NL and PL) generate plausible alternatives for a set of randomly masked positions. A discriminator is trained to determine whether a word is the original one or not. CodeBERT is pre-trained on CodeSerachNet [31] dataset.

*2) OpenAI generative models:* Radford *et al.* introduced general task-agnostic generative pre-training of language models (GPT) and outperformed 9 out of 12 discriminatively trained models that use architectures designed for the specific task [22]. In generative pre-training, we autoregressively predict the probability of a token given the previous tokens moving from left to right. This left-to-right autoregressive training prevents the model from retrieving information from future tokens. All the subsequent generative models (*e.g.,* GPT-2, GPT-3) use very similar pre-training objectives but have a higher capacity than previous ones and are pre-trained on a much larger dataset. Very large language models (LLMs) like GPT-3.x have 175 billion parameters and are found to be effective with few-shot learning replacing the need for fine-tuning for a specific set of tasks. However, fine-tuning GPT-3.x models are still beneficial for some tasks [7]. This paper evaluates our approach using four OpenAI [32] GPT-3.x models: Curie, Codex, Davinci, and Code-davinci-002.

***Curie:*** Curie is the fastest GPT-3 model with 6.7B parameters. This model is trained with natural language data and performs well on language translation, complex classification, text sentiment, and summarization tasks. This is the smallest model we use for our experiments.

***Codex:*** The Codex models are also GPT-3 models trained for understanding and generating code. The training data contains both natural language and billions of lines of public code from GitHub. We use one model, Codex-cushman from Codex family, with 12 billion parameters. Though the models are pre-trained for code-related tasks, it somehow relevant to incident management. Root cause and mitigation contain a lot of terminology (*e.g.,* filenames, identifiers) which relate more to comments used in software development projects.

***Davinci:*** Davinci is the biggest GPT-3 model (175 billion parameters) we use for our experiments. It can perform tasks with fewer instructions than other GPT-3 models. Davinci usually performs better at understanding the content or creative content generation task. It is also very good at solving logic problems. However, training the 175 billion parameters model is costly and requires a much longer period (almost four times compared to Curie with the same dataset) and more resources. Davinci is not trained to understand or generate code.

***Code-davinci-002:*** Code-davinci-002 is the 175 billion parameters GPT-3.5 model we use for our experiments. Code-davinci-002 is an upgraded and more capable version of Codex model that was trained on a more recent dataset of text and code corpus.

### *C. Model configuration*

One of the limitations of pre-trained encoder-decoder models is that they can only encode 512 tokens. We observe that several samples from our test set truncated in GPT-3 model even though GPT-3 models support from 2048 tokens (*e.g.,* Curie, Codex) to 4000 tokens (*e.g.,* Code-davinci-002). Therefore, we can assume that the traditional encoder-encoder models do not have enough capacity to encode our sequences.

Encoder-decoder models have been successful for problems like code summarization [13], [25], [27], code translation [29], and natural language translation [14], [20], [21]. We usually generate one sample using beam search for each input and compare the results with the reference. Generating one sample is sufficient for these problems because the target text is less open-ended. Besides, most of the information needed for successful generation can be found in the input for this set of problems. The models need to learn the syntactic alignment between two programming languages for code translation. Learning to transform conditional statements and loops from one programming language to another may be enough to do a successful translation, which is learnable from a few thousand samples. For natural language translation learning the mapping between the words from different natural languages is essential to generate good quality translation. Code summarization is slightly different from these two, where the input is much longer than the output. However, Ahmed and Devanbu found that all the necessary information for code summarization is extracted from the identifiers, and obfuscating the identifiers hurts the models [33]. Generating root causes and mitigation plans is much more complex than these problems, where the input may not contain handy information. The models need to be able to generate more diverse and creative solutions to answer the question. Our problem is more aligned with code generation problems where the input does not carry most information. For these types of problems, it is found that instead of using the encoder-decoder model, decoder-only models (*e.g.,* GPT-3.x) are more successful where we only focus on the following tokens considering the prior tokens generated by the models. It is well-established that encoder-decoder models are not as successful as decoder-only models in code generation tasks. However, we still apply encoder-decoder models to our problems and discuss our findings in the following sections. For RoBERTa [19] and CodeBERT [25] we use the exact setup that is used for the code summarization task [31], [34]. We adjust the length to 512 tokens with a batch size of 8 to provide as much as information to the model.

Full fine-tuning that retrains all the parameters is very costly and challenging for the OpenAI models with billions of parameters. We use LoRA (Low-Rank Adaptation), a novel approach that significantly reduces the number of trainable parameters by freezing the pre-trained model weights and injecting trainable rank decomposition matrices into each layer of the Transformer architecture [35]. Even though LoRA reduces trainable parameters, it performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-

2, and GPT-3. We fine-tuned the OpenAI GPT-3 (*i.e.,* Curie, Codex, Davinci) and GPT-3.5 (Code-davinci-002) models for root causes and mitigation plans generation. We train both models for 2000 steps (4 epochs) which OpenAI recommends. For fine-tuning smaller models (*i.e.,* Curie and Codex), we use one NVIDIA V100 GPU, and for Davinci, we use four NVIDIA V100 GPUs. For finetuning Code-davinci-002 model, we use four NVIDIA A100 GPUs. We evaluated the models on the validation set after every 100 steps and chose the model that showed minimum training loss on the validation set.

As discussed earlier, the model needs to generate more diverse and creative recommendations to solve problems like the predictions of root causes and mitigation plans. Two critical parameters to control the quality of the generated outputs are *temperature* and *top_p*, and it is recommended to update one parameter. Following prior works [8], [36], we decided to update the value of temperature. Higher temperature encourages the model to take more risk, which is necessary for the creative application [32]. Lower value performs argmax sampling, which is very similar to what we do in encoder-decoder model models like CodeBERT. Typically, a temperature between 0.50–0.90 is the most common for creative tasks. However, a high temperature is hurtful (makes the output too diverge) [36]. We perform a grid search and choose 0.7 for Curie, Codex, and Davinci models and 0.5 for Code-davinci-002 experiments to minimize the divergence issue for generating five samples.

### *D. Evaluation Metrics*

We briefly describe the evaluation metrics used for the two downstream tasks, root cause and mitigation generation.

*1) Lexical Metrics:* For lexical metrics, we employ the smooth sentence **BLEU-4** (Bilingual Evaluation Understudy) [37] metric to calculate n-grams overlap from 1 to 4 between the reference and generated texts. In addition, the Rouge metric (Recall Oriented Understudy for Gisting Evaluation) [38] is used to compare a candidate document to a set of reference texts. Specifically, we choose **ROUGE-L** [38], which takes into account sentence-level structural similarity and identifies longest co-occurring in sequence n-grams based on Longest Common Subsequence (LCS) [39] statistics. **METEOR** (Metric for Evaluation of Translation with Explicit Ordering) [40] is the final lexical metric we selected, which is based on the harmonic mean of unigram precision and recall as well as stemming and synonymy matching as extra features.

*2) Semantic Metrics:* Since the lexical metrics usually conduct exact word matches and disregard the meaning of words, we choose three semantic metrics to evaluate our outcomes according to their semantic meanings. We use the **BERTScore** [41], which leverages the pre-trained contextual embeddings from the BERT [18] model and matches candidate and reference sentence words based on cosine similarity. Then, the **BLEURT** score [42] is selected to demonstrate the degree to what extent the candidate is fluent and conveys the meaning of the reference. Last, we select **NUBIA** (NeUral Based Interchangeability Assessor) [43], a recent neural-based measure that incorporates the semantic similarity, logical inference and sentence legibility from exposing layers of pre-trained language models, including RoBERTa STS [19], RoBERTa MNLI and GPT-2 [23].

The semantic metric calculation takes significant time and requires expensive GPU resources (Tables I and II took two days on a single GPU). Therefore, we reported semantic metrics for the first two research questions, and for the remaining research questions, we restricted ourselves to lexical ones that are computationally less expensive.

## IV. Result

### *A. How effective are fine-tuned GPT-3.x models in generating incidents' root cause recommendation? (RQ1)*

Table I presents the effectiveness of our baseline encoder-decoder models and fine-tuned GPT-3.x models for root cause recommendation. We have 2621 test samples for evaluating the models. We generated ten samples for the OpenAI models for two reasons: i) using temperature, we can generate very diverse and creative samples from GPT-3.x models. ii) we found that GPT-3.x models can generate valuable suggestions even with lower ranks. We observed the average BLEU-4 of all the samples at a particular rank, and we found that all the OpenAI GPT-3.x models produce examples with higher BLEU-4 even at rank eight or lower. However, ten examples are too many for a human OCE, and we restrict ourselves to five top suggestions from the model. In Table I, for each metric, we have Top 1 and Top 5. Top 1 presents the mean of the first candidates for all the test samples; while calculating Top 5, we take the maximum value from the first five candidates and then find the average for all samples. This Top 5 gives an overall view of how the models are performing. For our baseline encoder-decoder models, we have only one sample for each model.

Surprisingly, the encoder-decoder models are doing really good compared to GPT-3 models in all six automatic metrics. In fact, all six metrics fail to distinguish significant differences between the OpenAI models. The reason behind the success of encoder-decoder models in automatic metrics is that these models are less explorative and try to maximize the success depending on argmax probabilities during decoding. Now "There is a bug in the code" is a very common and generic sentence that can be a part of any root causes. The models maximize the success just by copying that particular segment, and automatic metrics also fail here. We tried three semantic metrics to resolve that issue, but the encoder-decoder models still benefit from the automatic metric. Table III presents the number of unique samples generated by the models. For OpenAI models we only consider the first candidate to make a fair comparison. We observe that the unique candidate count for RoBERTa and CodeBERT are 6.10% and 16.67% of the total count, whereas, for all the OpenAI GPT-3.x models, the percentages are above 97%. Remember that we deduplicated the dataset, and repeatedly generating the same samples should not help here. In Section V, we interviewed the incident owners, and the majority of them complained about the generic nature of encoder-decoder models' recommendations, and these models

TABLE I: Effectiveness of fine-tuned GPT-3.x models at finding **root causes** of the incidents

| Model | BLEU-4 | | ROUGE-L | | METEOR | | BERTScore | | BLEURT | | NUBIA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 |
| RoBERTa | 4.21 | NA | 12.83 | NA | 9.89 | NA | 85.38 | NA | 35.66 | NA | 33.94 | NA |
| CodeBERT | 3.38 | NA | 10.17 | NA | 6.58 | NA | 84.88 | NA | 33.19 | NA | 39.05 | NA |
| Curie | 3.40 | 6.29 | 9.04 | 15.44 | 7.21 | 13.65 | 84.90 | 86.36 | 32.62 | 40.08 | 33.52 | 49.76 |
| Codex | 3.44 | 6.25 | 8.98 | 15.51 | 7.33 | 13.82 | 84.85 | 86.33 | 32.50 | 40.11 | 33.64 | 49.77 |
| Davinci | 3.34 | 5.94 | 8.53 | 15.10 | 6.67 | 12.95 | 83.13 | 84.41 | 31.06 | 38.61 | **35.28** | 50.79 |
| Davinci-002 | **4.24** | **7.15** | **11.43** | **17.2** | **10.42** | **16.8** | **85.42** | **86.78** | **36.77** | **42.87** | 32.3 | **51.34** |
| %gain for Davinci-002 | 23.26 | 13.67 | 26.44 | 10.90 | 42.16 | 21.56 | 0.61 | 0.49 | 12.72 | 6.88 | -8.45 | 1.08 |

underperform at correctness criteria. Among OpenAI models, GPT-3.5 (i.e., Code-davinci-002) model significantly outperforms all GPT-3 models as well as other baselines in terms of all the 6 automated metrics.

Though the automatic metrics fail to detect the weaknesses of the encoder-decoder models, these metrics are still widely used. Human evaluation is hard to perform in every scenario, and these metrics can be useful to find the models' relative performance. Therefore, even though we achieve a low score on these metrics, these are useful while trying to capture the relative performance of the model in different settings. Also, getting a lower score with lexical metrics is not surprising because lexical metrics only consider token overlaps and root cause and mitigation are open-ended, and the same root cause/mitigation can be written differently. In Section V, from the interviews with OCEs, we found that suggestions with lower BLEU-4 or other metrics are still helpful.

### *B. How effective are fine-tuned GPT-3.x models in recommending mitigation plans for an incident? (RQ2)*

Table II shows that we achieved a slightly higher mitigation score (4.44-6.76 BLEU-4) than the root cause recommendation (3.38-4.24 BLEU-4). We observed a similar and consistent pattern (Table III) of the output as observed with root causes. The encoder-decoder models generate generic comments (*e.g.,* "the issue is self-mitigated", "fix deployed to all regions") like before, and those recommendations are mostly useless for the OCEs. For both RQ1 and RQ2, the fine-tuned Davinci model (even with 175 Billion parameters) is significantly underperforming other baseline methods according to automatic metrics. However, the Davinci and Code-davinci-002 models are the best performing models according to the incident owners (see Section V)

### *C. How much fine-tuning improves over zero-shot learning performance of GPT-3.x models? (RQ3)*

As discussed in Section II-D, we will investigate the performance of OpenAI models in the zero-shot setting. Table IV presents the performance of the OpenAI models for root cause and mitigation. As expected, the model did not perform well in this setting since the models were not trained on confidential data from the incident management space. The models achieve 0.80-2.18 BLEU-4 for the top candidate, which is much lower (210%) than what we achieved with fine-tuning the models (5.47-6.76) while recommending mitigation steps. Though we achieved a higher score for mitigation than root cause during fine-tuning, in the zero-shot setting, the numbers for root cause are slightly high (1.18-2.83 for the top candidates). The model tries to complete the sequence depending on the given input. Copying a few tokens from input may help the model because the root cause is usually longer than mitigation and tends to share more tokens with the input. Because of unigram overlaps METEOR is doing better compared to other metrics (BLEU-4 and ROUGE-L) because it looks for the unigram precision and recall, making it lenient compared to BLEU-4 and ROUGE-L. We observe another interesting phenomenon here. Though the Davinci model was underperforming in RQ1 and RQ2, it significantly outperforms the other OpenAI models at zero-shot settings for both root cause and mitigation. This is because the model has higher parameters and is trained on more data enabling it to infer better without explicit training.

### *D. Does multi-task learning improve the performance of GPT-3.x models at finding root causes and mitigation plans? (RQ4)*

To evaluate the results of multi-task training in the root cause recommendation and mitigating planning tasks, we combine the training set of the two tasks for GPT-3.x models. The models are then individually tested using the corresponding test sets. Table V shows the results of root cause and mitigation with multi-task training. Overall, we observe that multi-task training does not significantly outperform training for a single task. The performance of Curie and Codex models has fallen by an average of 2.8% for BLEU-4, 2.0% for Rouge-L and 7.2% for Meteor. Only the Davinci model is marginally 6.2% better than single task training in terms of BLEU-4 metric. The performance of Code-davinci-002 is almost always lower across all lexical metrics in a multi-task setting. Similar to this, the results of mitigation generation reveals a 4.1% performance decline in average for all the four models. The lack of connection between the root cause and mitigation is what mostly contributes to the decline in performance. It is challenging to transfer knowledge from one task to the other because of the distinct distribution in their answer spaces, such as the variations in root cause and mitigation length and concreteness.

### *E. Do GPT-3.x models get better at proposing mitigation plans if the root cause is given? (RQ5)*

We assess the performance of the mitigation generation while the root cause is being revealed. Our training set of mitigation is reduced from 5,455 to 2,973 as a result of the missing root causes in the incidents, and we have 166 test

TABLE II: Effectiveness of fine-tuned GPT-3.x models at finding mitigation plans of the incidents

| Model | BLEU-4 | | ROUGE-L | | METEOR | | BERTScore | | BLEURT | | NUBIA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 |
| RoBERTa | 4.44 | NA | 7.10 | NA | 4.52 | NA | 86.33 | NA | 26.80 | NA | 14.90 | NA |
| CodeBERT | 6.02 | NA | 4.40 | NA | 3.37 | NA | 86.83 | NA | 28.44 | NA | 27.89 | NA |
| Curie | 5.47 | 10.62 | 8.03 | 16.31 | 6.22 | 12.75 | 85.65 | 87.13 | 27.20 | 37.23 | 15.30 | 25.46 |
| Codex | 5.53 | 10.62 | 8.15 | 16.23 | 6.19 | 13.15 | 85.68 | 87.35 | 28.43 | 37.92 | 15.77 | 26.33 |
| Davinci | 5.54 | 10.66 | 8.10 | 15.96 | 6.08 | 12.49 | 85.72 | 87.19 | 27.15 | 37.00 | 15.71 | 25.61 |
| Davinci-002 | **6.76** | **11.66** | **10.22** | **18.14** | **8.23** | **15.13** | **86.17** | **87.65** | **30.19** | **38.96** | **17.58** | **28.81** |
| %gain for Davinci-002 | 22.02 | 9.38 | 25.40 | 11.22 | 32.32 | 15.06 | 0.52 | 0.34 | 6.19 | 2.74 | 11.48 | 9.42 |

TABLE III: Uniqueness of the models' suggestions

| Model | Root cause | | Mitigation | |
|---|---|---|---|---|
| | # of unique recommendations | In % of total | # of unique recommendations | In % of total |
| RoBERTa | 160 | 6.10 | 4 | 0.22 |
| CodeBERT | 437 | 16.67 | 2 | 0.1 |
| Curie | 2612 | 99.65 | 1669 | 93.76 |
| Codex | **2614** | **99.73** | **1743** | **97.92** |
| Davinci | 2587 | 98.70 | 1731 | 97.24 |
| Davinci-002 | **2614** | **99.73** | 1696 | 95.28 |

TABLE IV: Effectiveness of OpenAI models for recommending root causes and mitigation steps at zero-shot setting

| Objective | Model | BLEU-4 | | ROUGE-L | | METEOR | |
|---|---|---|---|---|---|---|---|
| | | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 |
| Root cause | Curie | 1.26 | 2.01 | 4.75 | 7.80 | 7.94 | 13.30 |
| | Codex | 1.18 | 1.94 | 3.80 | 7.07 | 6.58 | 12.20 |
| | Davinci | 2.83 | 4.37 | 6.11 | 11.55 | 6.04 | 11.87 |
| | Davinci-002 | 1.35 | 2.5 | 4.89 | 8.58 | 7.65 | 13.55 |
| | Finetuned-Davinci-002 | **4.24** | **7.15** | **11.43** | **17.2** | **10.42** | **16.8** |
| | % gain for Finetuning | 49.82 | 63.62 | 87.07 | 48.92 | 31.23 | 23.99 |
| Mitigation | Curie | 0.81 | 1.50 | 2.45 | 4.59 | 5.33 | 9.40 |
| | Codex | 0.80 | 1.57 | 1.97 | 4.05 | 4.56 | 8.55 |
| | Davinci | 2.18 | 3.67 | 3.84 | 7.84 | 4.99 | 10.44 |
| | Davinci-002 | 0.92 | 1.89 | 2.31 | 4.52 | 4.92 | 9.2 |
| | Finetuned-Davinci-002 | **6.76** | **11.66** | **10.22** | **18.14** | **8.23** | **15.13** |
| | % gain for Finetuning | 210.1 | 217.7 | 166.2 | 131.4 | 54.4 | 44.9 |

samples to evaluate the model. Despite the sample reduction in the training set, Table VI reveals a considerable performance gain with the additional root cause information: the average for all three metrics is improved by 9.8% for the Curie model, 8.3% for the Codex model, 5.4% for the Davinci model and 26% for the Code-davinci-002. Nevertheless, we observe that the performance gain of the Code-davinci-002 model's Top-5 recommendations is modest compared to the improvement of the Top-1 results. Despite this, the overall promising results highlight the significance of root cause information in generating mitigation plans.

### *F. Do the models better propose mitigation plans for machine-detected incidents than human-detected ones? (RQ6)*

We analyze the mitigation generation performance of GPT-3.x models for both machine and human detected incidents in Table VII. We employ the same training set but separate the test samples by the categories of human and machine detected incidents. The testing samples consist of 592 incidents recognized by machines and 1188 incidents detected by humans. Table VII demonstrates that machine-recognized incidents can outperform those detected by humans by a factor of 9.5% for BLEU-4, 20% for ROUGE-L and 23% for METEOR in the context of Top-1 recommendations of Code-davinci-002 model. It is due to the fact that machine detected incidents usually adhere to certain patterns, which are easier for machine learning models to recognize.

TABLE V: Effectiveness of multi-task learning

| Objective | Model | Multi-tasking? | BLEU-4 | | ROUGE-L | | METEOR | |
|---|---|---|---|---|---|---|---|---|
| | | | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 |
| Root Cause | Curie | No | 3.40 | 6.29 | 9.04 | 15.44 | 7.21 | 13.65 |
| | | Yes | 3.30 | 6.13 | 8.66 | 15.51 | 6.60 | 12.97 |
| | Codex | No | 3.44 | 6.25 | 8.98 | 15.51 | 7.33 | 13.82 |
| | | Yes | 3.42 | 6.11 | 8.64 | 15.24 | 6.53 | 12.81 |
| | Davinci | No | 3.34 | 5.94 | 8.53 | 15.10 | 6.67 | 12.95 |
| | | Yes | 3.60 | 6.27 | 9.11 | 15.66 | 7.31 | 13.64 |
| | Davinci-002 | No | **4.24** | **7.15** | **11.43** | **17.2** | **10.42** | **16.8** |
| | | Yes | **4.24** | 7.09 | 11.32 | 17.14 | 10.32 | 16.34 |
| Mitigation | Curie | No | 5.47 | 10.62 | 8.03 | 16.31 | 6.22 | 12.75 |
| | | Yes | 5.49 | 10.89 | 7.98 | 16.14 | 5.92 | 12.54 |
| | Codex | No | 5.53 | 10.62 | 8.15 | 16.23 | 6.19 | 13.15 |
| | | Yes | 5.15 | 10.88 | 7.49 | 15.87 | 5.55 | 11.85 |
| | Davinci | No | 5.54 | 10.66 | 8.10 | 15.96 | 6.18 | 12.49 |
| | | Yes | 5.64 | 10.74 | 7.88 | 15.97 | 6.13 | 12.99 |
| | Davinci-002 | No | **6.76** | **11.66** | **10.22** | **18.14** | **8.23** | **15.13** |
| | | Yes | 6.58 | 11.36 | 10.04 | 17.76 | 7.91 | 14.36 |

TABLE VI: Effectiveness of GPT-3 models at proposing mitigation plans given root causes

| Model | Root-cause given? | BLEU-4 | | ROUGE-L | | METEOR | |
|---|---|---|---|---|---|---|---|
| | | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 |
| Curie | No | 5.92 | 11.29 | 9.46 | 17.76 | 7.34 | 13.35 |
| | Yes | 6.59 | 12.40 | 10.25 | 18.61 | 8.24 | 16.00 |
| Codex | No | 6.25 | 11.23 | 8.94 | 17.62 | 6.46 | 13.00 |
| | Yes | 6.23 | 12.03 | 9.32 | 18.48 | 7.73 | 15.96 |
| Davinci | No | 6.35 | 12.05 | 8.75 | 18.21 | 7.28 | 15.07 |
| | Yes | 7.02 | 11.47 | 9.49 | 18.20 | 8.40 | 16.17 |
| Davinci-002 | No | 6.8 | 12 | 9.48 | 17.37 | 8.15 | 15.53 |
| | Yes | **8.6** | **13.28** | **11.56** | **19.46** | **10.9** | **18.08** |
| | %gain | 26.47 | 10.21 | 21.94 | 12.03 | 33.74 | 16.42 |

## V. Looking through the Incident Owners' eyes

### *A. Methodology*

From our test sets for root causes and mitigation plans, we selected the incidents with both root causes and mitigation, so that each incident owner could evaluate both the models in the same interview. Incident resolution is a complex task requiring significant context and domain knowledge about the service and also about the specific incidents. Hence, we conducted this human evaluation with the actual owners who root caused and mitigated the incidents. We chose 50 recent incidents which occurred in the last two months, to

TABLE VII: Models' performance on machine vs human detected incidents

| Model | Machine detected? | BLEU-4 | | ROUGE-L | | METEOR | |
|---|---|---|---|---|---|---|---|
| | | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 |
| Curie | Yes | 5.49 | 10.54 | 8.54 | 16.63 | 6.45 | 13.13 |
| | No | 5.45 | 10.65 | 7.78 | 16.15 | 6.10 | 12.56 |
| Codex | Yes | 5.76 | 10.54 | 9.10 | 16.84 | 6.80 | 13.88 |
| | No | 5.41 | 10.67 | 7.68 | 15.93 | 5.88 | 12.78 |
| Davinci | Yes | 5.56 | 10.51 | 8.49 | 16.17 | 6.34 | 12.59 |
| | No | 5.52 | 10.74 | 7.91 | 15.86 | 5.95 | 12.44 |
| Davinci-002 | Yes | **7.18** | **11.83** | **11.5** | **18.59** | **9.41** | **15.66** |
| | No | 6.56 | 11.57 | 9.58 | 17.92 | 7.65 | 14.87 |
| | %gain | 9.45 | 2.25 | 20.04 | 3.74 | 23.01 | 5.31 |

evaluate the models' performance so that the incident owners could precisely remember what happened during managing particular incidents. We reached out to all the incident owners and 25 incident owners responded and each interview took around 20-30 minutes.

We presented the outputs from all the models under consideration. For both root causes and mitigation plans, we have six pools of candidates. The first four pools are for OpenAI models, each with six options (including "none"), and the last two are for RoBERTa and CodeBERT, which has only one candidate. For the OpenAI models, we ask the OCEs to select the best option that might be relevant to the incident. After that, we ask the OCEs to assign correctness and readability for the chosen candidate on a scale of 1-5, with 5 being the best score. Please note that for RoBERTa and CodeBERT, we only have one option. Hence, we only ask to assign correctness and readability scores to those candidates. We define correctness and readability as follows:

Correctness: For this metric, we ask the incident owner to check whether the model provides a helpful and relevant suggestion compared to the actual root cause/mitigation.

Readability: Readability is the ease with which a reader can understand a generated text. A text is readable if it is grammatically correct, meaningful and easy to understand. Note that a readable text does not need to be correct.

At the end, we asked the incident owners to assign an overall score (1-5) indicating their perception about the usefulness of LLMs for incident resolution and, also, asked them to share their thoughts and comments regarding this.

### B. Results

Table VIII presents the correctness and readability scores assigned by the incident owners. We can see that candidates from the Davinci and Code-davinci-002 pools have achieved higher mean correctness scores than those selected from Curie and Codex models for both root causes (2.88 and 2.56) and mitigation plans (3.04 and 3.16). The mean readability score ranges from 2.52 to 4.08 for all the models. The incident owners expressed positive opinions about the readability of the outputs, and all the models achieved higher readability than correctness scores. We received a few recommendations on how to improve the readability in the future (*e.g.,* avoiding use of acronyms and generating more specific or informative comments).

As discussed before, the baseline encoder-decoder models generate very generic comments, and the automatic metrics fail to detect that. We can see the incident owners assign a lower correctness score to RoBERTa and CodeBERT model, and several OCEs pointed out the generic nature of the recommendations generated by the encoder-decoder models. Though the correctness score of the OpenAI models ranges from 2.28 to 3.16, several OCEs pointed out that the models recommend beneficial root causes and mitigation plans. For example, the models succeeded in pinpointing some hard to detect root causes:

***"I am very impressed because one model found the right root cause, which was very hard to detect. We found it in the postmortem phase. However, I am a little worried that there would not be enough information on the incident website. Overall, I am impressed with the efficacy of the models."***

***"Even if not always correct, these suggestions can guide the OCE towards actual root cause. ML model can give directions and can be valuable suggestions."***

We also took the maximum score assigned by the OpenAI models and reported the average correctness and readability score. The mean correctness and readability score ranges from 3.52 to 4.64 (median score 3-5), presenting the overall strength of the models. We asked for the overall scores (1-5), and Table IX shows that the incident owners found the overall contribution promising and useful. More than 70% of incident owners gave three or above for the recommendations of the models. We found that at least one model is effective for most incidents. We also found out why the automatic metrics fail to provide valuable insights.

There is always another side to the coin, and we observe that the models' outputs are not helpful for some incidents. The OCEs assigned lower scores to those incidents and here are some of the concerns they mentioned:

***"Based on just incident data it is difficult for the model to predict root-cause and mitigation because not all data are recorded in the database and some of them are classified."***

***"Major concern is if the suggestion is incorrect, on-call engineers may take longer time to investigate the problem."***

We observed some negative samples for the model because a lack of discussion or other information results in the deprivation of valuable signals from the input. However, the model's overall performance is quite promising, which can be considered a stepping stone toward the automation of root causes and mitigation plans in the future.

### C. Two illustrative examples

Table X exhibits two samples to show the effectiveness of GPT-3.x model for generating root causes and mitigation plans in cloud incidents. We present the actual texts written by the OCEs and generated texts by one of the models (fine-tuned Code-davinci-002) side by side. Though the human-written and generated texts are different, the generated texts provide very relevant and valuable suggestions that can help the OCEs.

TABLE VIII: Correctness and readability scores assigned by the incident owners

| Objective | Criteria | RoBERTA | | CodeBERT | | Curie | | Codex | | Davinci | | Davinci-002 | | Max OpenAI | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Mean | Median | Mean | Median | Mean | Median | Mean | Median | Mean | Median | Mean | Median | Mean | Median |
| Root cause | Correctness | 1.56 | 1 | 1.72 | 1 | 2.40 | 2 | 2.40 | 2 | 2.88 | **3** | 2.56 | 2 | **3.52** | **3** |
| | Readability | 3.56 | **5** | 3.68 | **5** | 3.08 | 4 | 3.52 | 4 | 3.56 | 5 | 3.8 | 4 | **4.52** | **5** |
| Mitigation | Correctness | 1.6 | 1 | 1.52 | 1 | 2.28 | 2 | 2.28 | 1 | 3.04 | 3 | 3.16 | 3 | **4.04** | **4** |
| | Readability | 2.88 | 2 | 3.04 | 4 | 2.52 | 2 | 2.8 | 3 | 3.52 | 4 | 4.08 | 4 | **4.64** | **5** |

TABLE IX: Usefulness of LLMs for incident resolution

| Score | # of incident owners | In percent (%) of total |
|---|---|---|
| 5 | 2 | 7.41 |
| 4 | **9** | **33.33** |
| 3 | 8 | 29.63 |
| 2 | 6 | 22.22 |
| 1 | 2 | 7.41 |

These textual dissimilarities also show why the OCEs found the GPT-3.x models promising, but the automatic evaluation metrics failed to do that.

## VI. Discussion & Threats

### A. Do automatic metrics reflect human perception?

Automatic evaluation metrics are known to be representative of human perception and are widely used in problems like natural language translation [14], [20], [21]. Though some recent works looked into the effectiveness of these metrics in code summarization and reported many pitfalls and weaknesses of these metrics [44]–[47], researchers are still using them for benchmarking. The best possible alternative to automatic metrics is human validation or some form of automatic test case evaluation (done in code generation tasks). The main challenge in incident management is that even experts face difficulties evaluating the incidents if they are not involved in resolving particular incidents. In some cases, the OCEs could not clearly remember the incidents if they happened two months ago. Thus conducting a large-scale study is quite challenging in this area. However, we interviewed 25 incident owners and found that the models perform pretty well even after achieving lower scores with automatic metrics. We calculated the Pearson coefficient for all three lexical metrics (*i.e.,* BLEU-4, ROUGE-L, and METEOR) with the correctness and readability score assigned by the OCEs. We observed that the co-efficient varies from -0.42 to +0.62, preventing us from getting specific patterns in the value. That also indicates that these automatic metrics may not be coherent with human perception for resolving cloud incidents. However, more sample cases are needed to reach any concrete resolution.

### B. Natural language or code? Which family of models are better for incident management?

While choosing the models, we selected both natural language (*i.e.,* RoBERTa, Curie, Davinci) and code models (*i.e.,* CodeBERT, Codex-cushman, Code-davinci-002) to see which family of models is beneficial for incident management. We did not find any winners from these two groups. Davinci and Code-davinci-002 models are found to be producing correct and readable suggestions compared to other models. Note that both of them have 175 billion parameters. We leave fine-tuning larger code models or pre-training a model from scratch with incident data for future research.

### C. How the models' performance can be improved?

We received several recommendations from the incident owners. The main recommendation is to incorporate the discussions among the OCEs into the model. This will guide the model to locate better suggestions. We also dropped many incidents with summaries that written or updated at the time of incident resolution. To fairly evaluate the model and prevent possible data leakage (root cause and mitigation can be written in summary if updated later), we discarded them from our dataset. Incorporating them into our dataset after preventing data leakage may improve the performance of the models. We also lost some critical information while cleaning the summaries (*e.g.,* discarding images and tables). Incorporating that information may also help.

### D. Threats to Validity

There are several threats to our study. The semantic metrics use pre-trained models at the core, and we use the default, natural language models for the evaluation. A model pre-trained with incident management text may result in some changes in the performance evaluation. Also, we train and evaluate the models with the services available within our organization. These models may show unexpected behaviors if evaluated on a different set of services from other organizations. Some incidents owners expressed concerns about the models' efficacy with rare incidents, and rare incidents are frequently reported at Microsoft. Another threat to our study is the sample size of our human subject study. It is difficult to achieve statistical significance on correctness and readability scores with such small samples. However, it is challenging to scale depending on the nature of the study.

## VII. Related Work

### A. Incident management

Incident management in large cloud services has become a popular topic of research in the Systems and Software Engineering communities. Prior work in this space has focused on two main directions. First, there has been several empirical studies on analyzing incidents and outages in production systems which have focused on studying incidents caused by certain type of issues [48]–[51] or issues from specific services and systems [52]–[54]. Second and more related to our work is the use of machine learning and data driven techniques for automating different aspects of incident life-cycle such as triaging [55]–[57], diagnosis [58]–[60] and

TABLE X: Examples exhibiting the effectiveness of GPT-3.x model for incident resolution

| Examples | Task | Original text | Generated by DaVinci model |
|---|---|---|---|
| I | Root cause | The customer was sending very large amounts of data as their payload. The size in addition to the rate at which these transactions were being sent led to the initial network crash. | Large payloads that exceed the max payload size limit of the ledger caused the ledger to crash. |
| | Mitigation | A code upgrade with higher payload size solved the problem. | Republished the ledger with a higher payload size |
| II | Root cause | An unexpected power loss caused multiple VMs to go down in East US causing DB services. | Datacenter outage impacting multiple services |
| | Mitigation | DB team mitigated the upstream issue in the West US region | DB team mitigated the issue by restarting the service. |

mitigation [5]. Different from prior work, this is the first effort on leveraging state-of-the art language models for assisting OCEs with incident resolution. We hope that this work will also motivate future work which will merge traditional task-specific discriminative models with LLMs to do end-to-end automation of production incidents.

### *B. LLMs in Software Engineering*

Even though this is the first work leveraging LLMs for AIOps, several works in Software Engineering have tried to solve other challenging problems with LLMs. Github Copilot uses GPT-3 for automated code generation from natural language inputs [8]. Several researchers have addressed code generation [8], [36], docstring generation [8], [61], and code repair [62], [63] problems. Bareiß *et al.* [64] show how few-shot learning can be effective at (i) code mutation; (ii) test oracle generation from natural language documentation; and (iii) test case generation task. Jain *et al.* propose an approach to augment large language models with post-processing steps based on program analysis and synthesis techniques and achieve better performance [65]. However, unlike code generation where we have both lexical and structural information along with massive amount of training data, we explore the problem of incident resolution using state-of-the-art LLMs which has not been done before.

## VIII. Conclusion

With this work, we show that state-of-the-art large language models such as GPT-3 and GPT-3.5 are effective to help with incident management, specifically, to identify root causes and mitigation steps. To compare the effectiveness of the models, we conducted a rigorous and large-scale study at Microsoft, on over 40,000 incidents. To assess the actual usefulness of the approach, we involved the actual owners of production incidents. We expect that this paper is the first of many studies that leverage LLMs to make incident management more effective. Our next steps are to deploy the models in production to assist the OCEs with incident resolution. We are also planning to explore other usage scenarios for LLMs such as incident summarization.

## IX. Acknowledgements

We would like to thank the engineers who participated in the validation of root causes and mitigation steps. We would like to also acknowledge the contributions of the following people across Microsoft: Oleg Losinets, Jim Jernigan and Jim Kleewein.

## References

[1] S. Wolfe, "Amazon's one hour of downtime on prime day may have cost it up to $100 million in lost sales," 2018. [Online]. Available: https://www.businessinsider.com/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7

[2] J. Chen, S. Zhang, X. He, Q. Lin, H. Zhang, D. Hao, Y. Kang, F. Gao, Z. Xu, Y. Dang *et al.*, "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*, 2020, pp. 373–384.

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

[4] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "Continuous incident triage for large-scale online service systems," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 364–375.

[5] J. Jiang, W. Lu, J. Chen, Q. Lin, P. Zhao, Y. Kang, H. Zhang, Y. Xiong, F. Gao, Z. Xu *et al.*, "How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1410–1420.

[6] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," *arXiv preprint arXiv:2201.11903*, 2022.

[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[8] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[9] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu *et al.*, "Towards intelligent incident management: why we need it and how we make it," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1487–1497.

[10] "Common Crawl." [Online]. Available: https://commoncrawl.org/

[11] S. Kulkarni, A. Singh, G. Ramakrishnan, and S. Chakrabarti, "Collective annotation of wikipedia entities in web text," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 457–466.

[12] "Wikipedia." [Online]. Available: https://www.wikipedia.org/

[13] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," *arXiv preprint arXiv:2109.00859*, 2021.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] 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.

[17] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[19] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[20] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[21] 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," *arXiv preprint arXiv:1910.10683*, 2019.

[22] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[23] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[24] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[25] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, 2020, pp. 1536–1547.

[26] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, L. Shujie, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, "Graphcodebert: Pre-training code representations with data flow," in *International Conference on Learning Representations*, 2020.

[27] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Unified pre-training for program understanding and generation," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, Jun. 2021, pp. 2655–2668. [Online]. Available: https://www.aclweb.org/anthology/2021.naacl-main.211

[28] S. Chakraborty, T. Ahmed, Y. Ding, P. T. Devanbu, and B. Ray, "Natgen: generative pre-training by "naturalizing" source code," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 18–30.

[29] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. B. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *CoRR*, vol. abs/2102.04664, 2021.

[30] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[31] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019.

[32] "Openai." [Online]. Available: https://openai.com/

[33] T. Ahmed and P. Devanbu, "Multilingual training for software engineering," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1443–1455.

[34] "Codexglue – code-to-text." [Online]. Available: https://github.com/microsoft/CodeXGLUE/tree/main/Code-Text/code-to-text

[35] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[36] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, 2022, pp. 1–10.

[37] C.-Y. Lin and F. J. Och, "Orange: a method for evaluating automatic evaluation metrics for machine translation," in *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, 2004, pp. 501–507.

[38] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[39] D. S. Hirschberg, "Algorithms for the longest common subsequence problem," *Journal of the ACM (JACM)*, vol. 24, no. 4, pp. 664–675, 1977.

[40] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[41] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," *arXiv preprint arXiv:1904.09675*, 2019.

[42] T. Sellam, D. Das, and A. P. Parikh, "Bleurt: Learning robust metrics for text generation," *arXiv preprint arXiv:2004.04696*, 2020.

[43] H. Kane, M. Y. Kocyigit, A. Abdalla, P. Ajanoh, and M. Coulibali, "Nubia: Neural based interchangeability assessor for text generation," 2020.

[44] E. Shia, Y. Wangb, L. Dub, J. Chenc, S. Hanb, H. Zhangd, D. Zhangb, and H. Suna, "On the evaluation of neural code summarization," in *Proceedings of the 44th International Conference on Software Engineering (ICSE)*, 2022.

[45] D. Roy, S. Fakhoury, and V. Arnaoudova, "Reassessing automatic evaluation metrics for code summarization tasks," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1105–1116.

[46] D. Gros, H. Sezhiyan, P. Devanbu, and Z. Yu, "Code to comment ?translation?: Data, metrics, baselining & evaluation," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 746–757.

[47] S. Haque, Z. Eberhart, A. Bansal, and C. McMillan, "Semantic similarity metrics for evaluating source code summarization," *arXiv preprint arXiv:2204.01632*, 2022.

[48] 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*, 2016, pp. 517–530.

[49] A. Alquraan, H. Takruri, M. Alfatafta, and S. Al-Kiswany, "An analysis of {Network-Partitioning} failures in cloud systems," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 51–68.

[50] 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*, 2018, pp. 539–550.

[51] 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*, 2021, pp. 116–131.

[52] S. Ghosh, M. Shetty, C. Bansal, and S. Nath, "How to fight production incidents? an empirical study on a large-scale cloud service," in *Proceedings of the 13th Symposium on Cloud Computing*, 2022, pp. 126–141.

[53] H. Liu, S. Lu, M. Musuvathi, and S. Nath, "What bugs cause production cloud incidents?" in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2019, pp. 155–162.

[54] 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)*, 2014, pp. 249–265.

[55] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "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)*, 2019, pp. 111–120.

[56] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "Continuous incident triage for large-scale online service systems," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 364–375.

[57] A. P. Azad, S. Ghosh, A. Gupta, H. Kumar, P. Mohapatra, L. Eckstein, L. Posner, and R. Kern, "Picking pearl from seabed: Extracting artefacts from noisy issue triaging collaborative conversations for hybrid cloud services," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, 2022, pp. 12 440–12 446.

[58] V. Nair, A. Raul, S. Khanduja, V. Bahirwani, Q. Shao, S. Sellamanickam, S. Keerthi, S. Herbert, and S. Dhulipalla, "Learning a hierarchical

monitoring system for detecting and diagnosing service issues," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 2029–2038.

[59] C. Bansal, S. Renganathan, A. Asudani, O. Midy, and M. Janakiraman, "Decaf: Diagnosing and triaging performance issues in large-scale cloud services," in *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2020.

[60] C. Luo, J.-G. Lou, Q. Lin, Q. Fu, R. Ding, D. Zhang, and Z. Wang, "Correlating events with time series for incident diagnosis," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1583–1592.

[61] T. Ahmed and P. Devanbu, "Few-shot training llms for project-specific code-summarization," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–5.

[62] Z. Fan, X. Gao, A. Roychoudhury, and S. H. Tan, "Improving automatically generated code from codex via automated program repair," *arXiv preprint arXiv:2205.10583*, 2022.

[63] H. Joshi, J. Cambronero, S. Gulwani, V. Le, I. Radicek, and G. Verbruggen, "Repair is nearly generation: Multilingual program repair with llms," *arXiv preprint arXiv:2208.11640*, 2022.

[64] P. Bareiß, B. Souza, M. d'Amorim, and M. Pradel, "Code generation tools (almost) for free? a study of few-shot, pre-trained language models on code," *arXiv preprint arXiv:2206.01335*, 2022.

[65] N. Jain, S. Vaidyanath, A. Iyer, N. Natarajan, S. Parthasarathy, S. Rajamani, and R. Sharma, "Jigsaw: Large language models meet program synthesis," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1219–1231.

通过这项工作，我们展示了像GPT-3和GPT-3.5这样的最新大型语言模型在事故管理方面的有效性，

具体来说，用于识别根本原因和缓解步骤。

为了比较这些模型的有效性，我们在微软进行了一项严格而大规模的研究，涵盖了超过40,000个事故案例。

为了评估这种方法的实际效用，我们让实际的生产事故所有者参与其中。

我们期望这篇论文是许多利用大型语言模型使事故管理更加有效的研究之一。

我们接下来的步骤是在生产环境中部署这些模型，以协助OCE（运营中心工程师）解决事故。我们还计划探索LLM在事故摘要等其他使用场景中的应用。

# AI 理解论文

## Key Points

本文介绍了在大型 IT 企业（如亚马逊、谷歌、微软和 Salesforce）中，部署在云平台上的应用程序和服务已经取代 了传统的软件。在当今的云系统中，生产事件（例如停机或性能下降）对客户产生不利影响，可能违反服务级别协议，并需要投入 大量工程努力来减轻事件。例如，在一次主要购物日，亚马逊预计一小时的停机时间将造成 1 亿美元的损失。尽管进行了多年的可 靠性努力，但云服务仍在经历不可避免的严重事件。人工智能（AI）在 IT 运营中，也称为 AIOps，越来越受欢迎。数据驱动和 AI 技术已经用于自动化事件生命周期的某些部分，例如，事件优先级、具有类似症状的事件的检索和减轻事件所需时间、减轻措施等。然而，在应对事件时，电话工程师（OCEs）仍然需要通过多次往返的通信来确定根本原因和减轻措施。最近成功利用 GPT-3 模型完成非技术任务、代码生成和自动化任务等的启发，我们将在事件管理中应用这些模型。本文介绍了模型在大规模云服务中的 表现，重点讨论了生产事件在大规模云服务中是不可避免的，同时还需要投入大量工程资源来分析和解决。此外，还介绍了 incident 的生命周期通常分为四个阶段：检测、分类、诊断和缓解，以及缓解过程基于确定的根原因，采取措施以恢复服务健康， 最小化对服务用户的影响。最后，本文介绍了在处理虚拟机问题时，建议客户提供有关虚拟机问题的更好官方文档。

我们识别了以下两种情况：

1）找出事件的根本原因。诊断事件通常需要在与工程师确定事件根本原因之前花费大量时间和通信。我们研究大型语言模型在提示事件根本原因方面的有效性（RQ1）。

2）建议事件的减轻措施。在找到根本原因后，工程师采取措施来减轻问题。我们研究大型语言模型在建议事件减轻措施方面的有效性（RQ2）。

当应用大型语言模型时，需要考虑多个因素和决策。由于这些模型未用事件管理数据进行训练，是否需要对模型进行微调（RQ3）？是否更有效的是建立一个支持根本原因和事件减轻措施的单一模型（单 任务）或一个支持根本原因和事件减轻措施的集成模型（多任务）（RQ4）？根本原因是否有助于语言 模型找到更好的减轻措施（RQ5）？对于某些类型的故障，模型表现是否更好（RQ6）？

我们在微软 1759 个服务中的 44,340 个事件上进行了严格的大规模评估。除了通常报道的词法和语义评估指标外，我们还进行了人类验证，询问事件所有者对建议的根本原因和减轻措施的正确性和可读性进行评估。事件所有者是最有资格评估模型在事件上表现的人。

该论文介绍了如何使用大型语言模型（如 GPT-3.x）对生产事件进行有效分析。结果表明，这种方法可以帮助确定事件的根本原因，并建议减轻措施。在未来的研究中，可以进一步探讨如何将这种方法应用于更复杂的生产场景中。