

ChatGPT Inaccuracy Mitigation during Technical Report Understanding: Are We There Yet?

Salma Begum Tamanna*, Gias Uddin†, Song Wang†, Lan Xia‡ and Longyu Zhang‡

*University of Calgary, Calgary, Canada, †York University, Toronto, Canada, ‡IBM, Markham, Canada

*salmabegum.tamanna@ucalgary.ca, †{guddin, wangsong}@yorku.ca, ‡{lan_xia, longyu.zhang}@ca.ibm.com

Abstract—Hallucinations, the tendency to produce irrelevant/incorrect responses, are prevalent concerns in generative AI-based tools like ChatGPT. Although hallucinations in ChatGPT are studied for textual responses, it is unknown how ChatGPT hallucinates for technical texts that contain both textual and technical terms. We surveyed 47 software engineers and produced a benchmark of 412 Q&A pairs from the bug reports of two OSS projects. We find that a RAG-based ChatGPT (i.e., ChatGPT tuned with the benchmark issue reports) is 36.4% correct when producing answers to the questions, due to two reasons 1) limitations to understand complex technical contents in code snippets like stack traces, and 2) limitations to integrate contexts denoted in the technical terms and texts. We present CHIME (ChatGPT Inaccuracy Mitigation Engine) whose underlying principle is that if we can preprocess the technical reports better and guide the query validation process in ChatGPT, we can address the observed limitations. CHIME uses context-free grammar (CFG) to parse stack traces in technical reports. CHIME then verifies and fixes ChatGPT responses by applying metamorphic testing and query transformation. In our benchmark, CHIME shows 30.3% more correction over ChatGPT responses. In a user study, we find that the improved responses with CHIME are considered more useful than those generated from ChatGPT without CHIME.

Index Terms—ChatGPT, Hallucination, Software Issue Reports

I. INTRODUCTION

The reliability of LLMs is often questioned due to their tendency to produce nonsensical or incorrect outputs, a phenomenon commonly referred to as hallucination [1], [2], [3]. Like any LLM, ChatGPT can also suffer from hallucination issues like inconsistency in responses [4], [5] or factual inaccuracies. These problems can arise even when the model is provided with the context as a document/paragraph. While progress is made to assess hallucinations in textual data [6], [7], we are not aware of how hallucinations can be detected and mitigated for software technical reports that contain both textual and technical terms (e.g., crash dumps, code snippets, etc.).

This paper studies the detection and mitigation of ChatGPT inaccuracies in technical reports. We pick software bug reports for our study, because bug reports often contain a blend of descriptive text, technical terminology, code references, and snippets of crash/system dumps [8]. These documents are crucial for tracking and resolving software issues but can be overwhelming due to their volume and complexity [9]. An AI chatbot, trained to understand these reports, may streamline the process by extracting information. But for that, first we need to ensure that the responses from the chatbot are correct.

In the first phase of our study, we conducted a survey of 47 software engineers to understand the types of questions they ask while exploring bug reports and for which they wish for an automated Q&A tool like a chatbot. We found that developers ask diverse questions during bug exploration, which we could group into five types: 1) issue analytics, 2) issue trends, 3) issue summary, 4) issue labeling, and 5) issue backlogs. Based on the survey findings, we produced a benchmark of 412 Q&A pairs by consulting our industry partner (with whom we conducted regular bi-weekly sessions) and the literature. The Q&A pairs are collected by analyzing the issue reports of two popular open-source software (OSS).

In the second phase of our study, we tuned ChatGPT with issue reports from the two studied OSS based on the Retrieval Augmented Generation (RAG) techniques [10], [11], [12], [13]. We then evaluated the correctness of ChatGPT responses against our benchmark. Each question was asked and compared automatically to its expected answer. Correctness was assessed as the ratio of questions whose answers were found as correct. We found that our RAG-based ChatGPT was correct in only 36.4% cases. For the rest of the questions, it hallucinated by producing incorrect or irrelevant answers. We manually examined each hallucination case and identified two limitations in ChatGPT to process technical documents like bug reports: 1) limitations to understand complex technical contents in code snippets like stack traces (e.g., when a partial code snippets/crash dump is provided and the question is about determining the cause of the crash by assessing both the crash dump and the textual contents), and 2) limitations to integrate contexts denoted in the technical terms and texts (e.g., when ChatGPT was required to assess the relationships among multiple metadata and the technical terms).

In the third phase of our study, we designed CHIME (ChatGPT Inaccuracy Mitigation Engine) to address the above two limitations. The underlying principle in CHIME is that (1) if we can preprocess the technical reports better and store information relevant to an issue report as a combination of metadata and actual contents and (2) then guide the query validation process in ChatGPT with guided iterative prompting approaches, we can address the observed limitations. The usage of metadata is found to improve LLM search capabilities [14]. For us, such metadata could be generated by organizing the mix of textual and technical contents into a structured form. A challenge was how to separate the textual and technical

TABLE I: Demography of Survey Participants

Current Role	Years of Experience					Total
	0-3	4-5	6-10	11-15	16-20	
Developer	13	20	4	2	-	39
QA Engineer	1	1	-	-	-	2
Project Manager	-	1	-	-	-	1
Other	2	-	2	-	1	5
Total	16	22	6	2	1	47

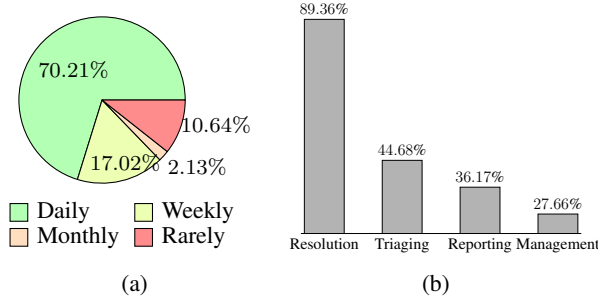


Fig. 1: (a) Frequency & (b) Reasons of Bug Report Exploration

contents and process the code terms within a crash dump and then organize those within a structure. We introduce a novel context-free grammar (CFG) in CHIME to efficiently parse stack traces in technical reports. As for the second principle (i.e., guided prompting for verification), we extended recent similar work on textual content. CHIME verifies and fixes ChatGPT responses by using query transformation [15] and by extending CoVe [16] with metamorphic testing (MT) [17]. CoVe is a zero-shot iterative prompting-based query verification technique. We evaluated CoVe’s response by further mutating the question using MT, because CoVe may discard correct responses or promote incorrect responses.

We evaluated CHIME using our benchmark. CHIME shows on average 30.3% improvement over ChatGPT responses by offering more correct answers. In a user study, we find that the improved responses with CHIME are considered more useful than those generated from ChatGPT without CHIME.

Our **replication package** (<https://bit.ly/4fyaMIP>) contains all the data and code developed in the study.

II. RELATED WORK

A. Hallucinations in Large Language Models

Extensive studies in the literature identified the causes of hallucinations as sub optimal training, inference [18], [19], [20], [21], [22], and insufficient/low-quality data [23], [24], [25]. Techniques such as bidirectional auto-regressive models [26] and attention-sharpening mechanisms [27] have been developed to address training-related hallucinations. Inference issues, primarily due to decoding strategies, often result in inaccurate outputs. Strategies like factual-nucleus sampling [28] and in-context pretraining [29] are implemented to mitigate these inaccuracies. Challenges posed by flawed data sources introduce biases and inaccuracies into models, stemming from misinformation, duplication biases, and social biases in

the training datasets. Mitigating data biases involves manual dataset creation [30], integrating high-quality sources such as the Pile [31], and up-sampling factual data [32]. Furthermore, knowledge editing [33], [34] and Retrieval-Augmented Generation (RAG) [35], [11], [12], [13] are employed to bridge knowledge gaps, utilizing external sources for more accurate text generation.

Our study utilizes RAG-based ChatGPT for technical bug report understanding. We enhance ChatGPT’s knowledge base by integrating it with a database of bug reports through RAG methods. We then develop CHIME, which refines both the preprocessing of input data and the validation of RAG-based ChatGPT responses.

B. LLMs for Software Engineering

In recent years, the application of LLMs has been widely utilized in Software Engineering (SE) tasks, ranging from code analysis to bug detection [36], [37], [38], [39], [40], [41], [42], [43], [44]. Encoder-only models like BERT [45] and its derivatives, including CodeBERT [46] and GraphCodeBERT [47], excel in processing code. While encoder-decoder models, such as T5 and PLBART excel in understanding semantics of code for tasks like code summarization [48], [49]. Decoder-only models like the GPT series and specialized versions like CodeGPT and Codex generate direct responses from prompts. However, the specific challenge of mitigating inaccuracies in software technical reports remains unexplored. Addressing this gap, we introduce CHIME to reduce inaccuracies in ChatGPT-generated responses during bug report exploration.

III. AI CHATBOT NEEDS FOR BUG REPORT EXPLORATION

To evaluate a chatbot on software technical documents, we needed a benchmark, which at the time of our study was not available. We thus adopted a systematic approach to create such a benchmark. First, we conducted a survey of software developers to produce a catalog of questions that they ask during bug reports. Second, we used the catalog to produce our benchmark (see Section IV). This section discusses the survey, which answers the following research question (RQ):

RQ1. What types of questions would software practitioners like to ask a chatbot during bug report exploration?

A. Survey Participants

We employed the snowball sampling approach [50] to recruit participants, resulting in 47 software practitioners. The majority of respondents (83%) held developer roles. The largest proportion of participants (55%) reported having 4-5 years of experience in the field. Table I illustrates the distribution of participants across their roles in the software industry and years of experience. We ensured that all selected participants regularly use Issue Tracking Systems (ITS), such as Jira, GitHub, or in-house systems for several key purposes. Around 89.36% of respondents used ITS for issue resolution—addressing and for solving bugs; followed by issue triaging which involves prioritizing and assigning issues. Project Management and documenting issues for records or

TABLE II: Identified Task Types for Bug Report Exploration

T#	Task Title	Task Description
T1	Issue Analytics	Explores complex details within/across bug reports, including technical jargon, error codes, and contextual nuances, etc.
T2	Issue Trends	Patterns and trends in bug occurrences.
T3	Issue Summaries	Summaries of key topics discussed in an issue or across multiple issues.
T4	Issue Labeling	Inquires labels for bug reports to help organize and categorize them effectively.
T5	Issue Backlog	Explores whether the issue remains open for long and why

TABLE III: Key Questions (KQ) of the Survey. C/O=Close/Open-ended question

KQ#	Questions	O/C
1	How would you like to utilize a chatbot during bug report exploration?	O
2	Would you want to use the chatbot for T#?	C
3	Rate the usefulness of T# for your work	C
4	Rate the potential usefulness of the following benchmark questions of T#	C

stakeholder communication were also noted. A vast majority of respondents (70.21%) reported engaging with issue reports daily. Figure 1a presents the distribution of issue report interaction frequency and Figure 1b illustrates the survey responses for the primary reason for issue report exploration.

B. Survey Questions

Before the main survey, we consulted two software professionals from a reputed software company to get insights on the potential tasks that an AI chatbot can support during bug report understanding. Each had 16 and 35 years of experience respectively. We finally settled on five distinct tasks as shown in Table II). For our survey, we formulated questions to address these tasks only.

During the survey, participants were prompted to envision an AI chatbot with comprehensive access to bug reports. Subsequently, they were asked whether they would like to utilize the chatbot for the specified tasks (T#) and to rate the perceived usefulness of each task for their work. We condensed the key survey questions into Table III for clarity. All survey questions can be found in our online appendix.

C. Understanding Preferences of Software Practitioners on Identified Task Types (RQ1)

In Table III, with Key Question 1 (KQ1), we tried to identify the desired chatbot roles during bug report explorations. With KQ2-4, we examined how participants perceived the usefulness of each task type. The questions are summarized in Table IV. For each question, we show quantitative evidence from the survey about what the participants thought of the usefulness of the question.

1) *KQ1. Desired Chatbot Roles*: We asked KQ1 as an open-ended question to check whether the participants considered the same five task types that we identified in Table II. As such, we did not show them KQ2-4 until they answered KQ1.

Participants desired a tool whose capabilities closely aligned with our predefined tasks (T1–T5). They emphasized the chatbot’s potential to analyze issues (T1), such as extracting pivotal details like exceptions or log entries from similar bug reports. According to R43: “*Queries for potential duplicate issues could be really helpful*”. The capability to analyze trends (T2) within reported issues to uncover recurring problems was seen as crucial. Participants also noted the importance of the chatbot’s ability to summarize (T3) and categorize (T4) issues. The survey responses indicated a significant interest in features that would allow users to query backlogged items (T5), as R43 stated: “*Queries for issues with very little recent activity and no clear resolution would be super helpful*”. Apart from these tasks, participants also expressed desires for additional capabilities such as sorting issues based on priority, severity, or difficulty and forecasting resolution times using historical data. We leave support for those tasks as our future work.

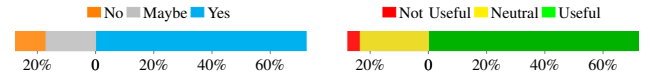


Fig. 2: (a) Interest for T1 (b) Usefulness Perception of T1

2) *KQ2-4. Issue Analytics (T1)*: The survey results reveal a strong preference for the chatbot’s analytical capabilities, particularly in the context of analyzing multiple issues, with more than 80% of participants expressing interest in utilizing the chatbot for detecting similar or duplicate issues and finding it useful; while 72.34% participants value the chatbot’s utility in analyzing individual issues as illustrated in Figure 2.

In Table IV, we show 10 questions under T1 that each participant assessed. On average, 73.62% of participants marked these useful. The capability to identify and summarize stack traces (Q1.1) within the single issue analysis domain was highly valued, evidenced by a utility score of 78.72% and remark from respondent R11, “*Summary of stack trace is a good idea*”. In comparison, the importance of determining the environment linked to an exception (Q1.2) was rated lower, at 57.45%. For the analysis of multiple issues, the ability to find similar issues (Q1.3) was highly valued at 82.98%, as quoted by R09, “*They are all extremely time-consuming when done manually. A chatbot will definitely help with this.*”



Fig. 3: (a) Interest for T2 (b) Usefulness Perception of T2

3) *KQ2-4. Issue Trend (T2)*: 87.2% of participants expressed interest in using this feature and 85.1% found the corresponding questions on identifying and analyzing trends within bug reports useful (see Figure 3). In Table IV, we show two questions that we asked under this task. Both received a favorable response. Respondent R31 noted, “*By focusing on recurring errors and themes, these questions provide valuable insights that can guide decision-making, resource allocation, and issue resolution efforts.*” When participants were asked

TABLE IV: Perceived Usefulness of Benchmark Questions Presented in the Survey.

■ Not Useful ■ Neutral ■ Useful

Q#	Question	Perceived Usefulness of T# (KQ4)
T1 - Issue Analytics — Extracting Information from Issue Details or Find Similarities Among Issues		
Q1.1	Is there a stack trace provided in issue 123, and can you summarize it?	
Q1.2	Where in the code does the exception in issue 123 occur?	
Q1.3	What is the exception reported in issue 123?	
Q1.4	How many tests failed as reported in issue 123?	
Q1.5	Which environment is associated with the exception reported in issue 123?	
Q1.6	Are there any issues similar to issue 123?	
Q1.7	Find duplicate reports of the X error (or other) in 'A' module	
Q1.8	Find all similar issues related to X failures]	
Q1.9	Identify any performance degradation issues reported on last month	
Q1.10	Has there been a report of a crash on a X machine running the "A" Test recently?	
T2 - Issue Trend — Detect and Analyze the Trends and Patterns among Issues		
Q2.1	What are the frequently encountered errors in the nightly builds?	
Q2.2	What are the recurring themes in bug reports post the latest OS update?	
T3 - Issue Summary — Obtain a Comprehensive Overview of Reports Selected by Different Criterion		
Q3.1	List all issues related to an X feature and their current status	
Q3.2	Generate a report detailing the distribution of issues across different project modules	
Q3.3	Compile a summary of unresolved issues not older than 60 days	
Q3.4	Can you generate a summary of all issues tagged as 'bug' in the last 30 days?	
Q3.5	Create a summary of user-reported issues versus internally identified issues	
T4 - Issue Label — Provide Suggestions for Categorizing and Tagging Issues with Appropriate Labels		
Q4.1	Suggest existing labels to tag issue 123	
Q4.2	Can you recommend labels for performance-related issues?	
T5 - Issue Backlog — Analyze Unresolved Issues Reported but not yet Addressed		
Q5.1	Are there any long-standing issues that have been consistently postponed?	
Q5.2	Find issues that have not been assigned to any milestone but are older than 60 days	
Q5.3	List issues that have missed two or more release cycles	
Q5.4	Identify issues with no activity in the last 30 days.	



Fig. 5: (a) Interest for T4 (b) Usefulness Perception of T4

about the utility of chatbots in identifying frequently encountered errors in the nightly builds of their development environment (Q2.1), e.g., in CI/CD pipelines, 76.60% perceived this functionality as useful.



Fig. 4: (a) Interest for T3 (b) Usefulness Perception of T3

4) *KQ2-4. Issue Summary (T3)*: 80.6% of participants were keen on a chatbot to produce summaries of issues and 70.2% considered the asked questions useful for efficiently understanding and resolving software issues. In Table IV, regarding the chatbot's ability to report on how issues are distributed across different project modules (Q3.1), 76.6% found this function useful. On the other hand, the feature for distinguishing between user-reported and internally identified issues (Q3.2) was seen as useful by 51.06% of participants, indicating a notable but more moderate interest in differentiating the sources of issues.

5) *KQ2-4. Issue Labeling (T4)*: 70.2% of participants are interested in leveraging chatbots for the task of issue labeling (see Figure 5). However, it's worth noting that this task received the lowest percentage of perceived usefulness (61.70%) compared to others. Regarding the chatbot's ability to suggest appropriate labels for an issue (Q4.1) and to recommend labels for performance-related issues (Q4.2), about 68% of respondents considered these features to be useful.

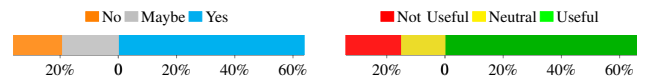


Fig. 6: (a) Interest for T5 (b) Usefulness Perception of T5

6) *KQ2-4. Issue Backlog (T5)*: The management of Issue Backlogs is an essential aspect of software development. R16 highlighted the challenge: "Sometimes change of priorities pushes issue out of find and stay unresolved for days. So it is good to find out long-running or inactive issues." Despite its importance, this task garnered the least interest (63.8%) among all tasks for potential chatbot utilization. Notably, T5 records the highest percentage of "No" responses (17.02%) regarding interest and "Not Useful" perceptions (19.15%). 70.2% of respondents see value in identifying long-standing, postponed issues (Q5.1) but interest slightly drops to 61.70%

for detecting issues with no recent activity over the last 30 days (Q5.2).

Summary of RQ1. When examining software practitioners’ preferences for AI chatbot capabilities in exploring bug reports, the identification of similar issues and the analysis of recurring error trends were highly favored. In contrast, capabilities related to categorizing issues and handling pending bugs were deemed less critical.

IV. A BENCHMARK OF Q&A PAIRS TO EVALUATE AI CHATBOTS FOR BUG REPORT EXPLORATION

In Table IV, we showed a catalog of 23 questions that we validated with our survey participants and for which they wished for chatbot support. Each question is a template, which can be used to produce multiple similar questions. Based on the question templates in Table IV, we produced a total of 412 questions from the issue reports of two popular OSS repos, OpenJ9 and ElasticSearch. OpenJ9 was chosen due to its alignment with our industrial partner. ElasticSearch [51], [52], [53] is frequently referenced in academic studies. We then produced an answer to each question by assessing the two OSS repos and by consulting among the authors. Four authors (the first two and last two) engaged in many hours of discussions that spanned over six months (both in-person and over formal presentations). The last two authors are also among the maintainers of OpenJ9. Given the benchmark was created via mutual discussion, we did not compute any standard agreement analysis metrics.

We created the benchmark by selecting 80 complex issues (40 from each repository). Following Deeksha et al. [9], we define an issue as complex if it is excessively long and/or it has stack traces. We picked issues within the last year of our analysis because those are likely to be explored more by developers. We sorted issues by length and selected 40 issues with stack traces (per repo). Following the standard chatbot evaluation process, we contained three types of answers: binary [54], factual [55], [56], and summary [57].

The binary (i.e., Query Type = Y/N) queries have questions with answers as Y/N. These are designed to assess the chatbot’s accuracy in identifying clear-cut, definitive binary decisions based on information available in bug reports; such as the presence of a particular error code or the applicability of a specific scenario. To verify a chatbot response for these queries, we simply need to check for Y/N in their responses and match those against the benchmark answer.

Q1. Type: Y/N. Source: ElasticSearch

Question: Is there any issue similar to issue 100071?

Expected Answer: No.

The factual (i.e., Query Type = Factual) queries assessed the chatbot’s ability to extract concrete information from bug reports, such as identifying, retrieving, and presenting specific details from the dataset, such as error messages, stack traces,

configuration settings, etc. Like binary queries, this method also allows for a straightforward assessment of the chatbot’s accuracy, and thus direct matching can be used for verification.

Q2. Type: Factual. Source: ElasticSearch

Question: What existing label is recommended for issues that need immediate triaging?

Expected Answer: ‘needs:triage’

The summary-based (i.e., Query Type = Summary) queries challenge the chatbot to engage in deeper analysis and synthesis of data. These queries require the chatbot to identify patterns and even to propose potential solutions based on the analysis of multiple data points. Since these queries demand a synthesis of information and provide insights or summaries, we need a similarity analysis between a response and the expected answer for verification.

Q3. Type: Summary. Source: ElasticSearch

Question: Summarize similarities between issues 103072 & 103344

Expected Answer: Issues 103072 and 103344 both involve test failures within the LearningToRankRescorerIT class. The root cause of these failures stems from a named_object_not_found_exception and x_content_parse_exception, resulting in ...

TABLE V: Distribution of Benchmark Questions over Survey-identified Tasks from Table IV.

T#	Y/N	Fact	Summarization	
T1 - Issue Anlys(S)	48	140	24	212
T1 - Issue Anlys(M)	12	20	8	40
T2 - Issue Trend	16	16	8	40
T3 - Issue Summary	8	32		40
T4 - Issue Labeling	12	20	8	40
T5 - Issue Backlog	12	24	4	40

Table V shows the distribution of question types—Yes/No, Fact, and Summarization—across OpenJ9 and ElasticSearch, totaling 206 questions per project. OpenJ9 and ElasticSearch have a similar overall structure, with a strong emphasis on factual questions (114 for OpenJ9, 114 for ElasticSearch), followed by binary (yes/no) and summarization questions. Our online appendix contains details about each of the 412 questions and how each question maps to our catalog of 23 survey questions.

V. EFFECTIVENESS OF CHATGPT ON THE BENCHMARK

In this section, we answer the following research question:

RQ2. How effective is a RAG-enhanced ChatGPT to answer to the benchmark questions while exploring the corresponding bug reports?

A. Approach

The RAG architecture combines ChatGPT with an external knowledge retriever to provide responses to queries. This framework utilizes external database sources, primarily issue reports with structural data and metadata fetched by the GitHub API. Figure 7 illustrates the pipeline for this. It functions by first retrieving pertinent information from the database based on the input query. This step is crucial as it aligns the model’s focus with the most relevant data. Then, the augmented data from the retrieval step are combined with the inherent generative capabilities of ChatGPT to help ChatGPT provide high-quality responses. We used ChatGPT 3.5-turbo within LangChain framework [58] to implement this pipeline. We used a temperature setting of 0. A temperature value above 0 produces slightly different answers to a prompt across multiple runs, which is unnecessary when we expect consistent answers from ChatGPT. We ran it multiple times on our benchmark dataset to ensure that the answers were indeed consistent across multiple runs.

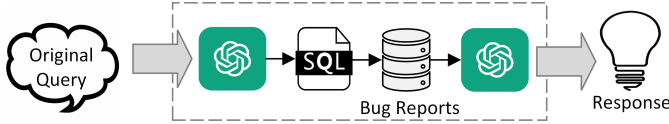


Fig. 7: Pipeline of the RAG-based ChatGPT

We measure the effectiveness of the above RAG-based ChatGPT on our benchmark by calculating correctness (C):

$$C = \frac{\text{Number of Correct Responses}}{\text{Total Number of Queries}} \times 100\% \quad (1)$$

Correctness analysis involved comparing the generated responses against the predefined correct answer for a query in the benchmark. We adopted the following automated approaches to measure the correctness of the responses for the three types of queries in our benchmark (i.e., Y/N, Fact, and Summaries).

For evaluating yes/no responses, we used a zero-shot classification approach, enabling us to automatically determine if detailed answers from the chatbot align with a simple “Yes” or “No” expectation. For evaluating factual query responses, we combine two approaches: direct comparison of extracted information (such as issue numbers) and semantic similarity assessment for non-listed facts. First, we extract and compare key factual elements. If the response and expectation directly match or share common elements, the correctness is assessed accordingly. For responses without explicit factual elements, we utilize the SentenceTransformer library, employing the *all-MiniLM-L6-v2* model [59] to encode the actual and expected answers into embeddings. Subsequently, we gauge the semantic similarity between these embeddings by computing the cosine similarity [60] and assessing how closely the actual answer aligns with the expected fact, considering nuances in phrasing and context. For evaluating Summarization queries, we compute the semantic similarity like before between the actual summary provided by ChatGPT and the expected sum-

mary. Based on empirical observations (see Section VIII-B), we used a similarity threshold of 0.7.

B. Results

The RAG-based pipeline achieved 36.4% accuracy in our benchmark (see Table VI). We manually assessed each of the 262 incorrect answers to determine the causes of its incorrectness. Given that ChatGPT is a black-box model, our assessment is based on the nature of the questions asked and the provided answers. We observed two limitations in ChatGPT:

- L1. Limitations in Understanding Complex Technical Content (42.7% cases).
- L2. Contextual Understanding Challenges (57.3% cases).

TABLE VI: Correctness of RAG Based ChatGPT

T#	Y/N	Fact	Sum	Total
Total	49.0%	30.7%	36.9%	36.4%
T1 - Issue Anlys(S)	41.7%	32.1%	33.3%	34.4%
T1 - Issue Anlys(M)	50.0%	15.0%	0.0%	22.5%
T2 - Issue Trend	50.0%	12.5%	37.5%	32.5%
T3 - Issue Summary	0.0%	75.0%	43.8%	50.0%
T4 - Issue Labeling	58.3%	40.0%	62.5%	50.0%
T5 - Issue Backlog	66.7%	25.0%	25.0%	37.5%

The challenge in understanding technical contents (L1) occurred when ChatGPT faced challenges in deeply understanding and accurately processing highly technical content, particularly when it involved intricate programming/technical concepts. For example, when asked to provide the root location of the exception encountered in issue 18151; it gives a non-useful answer; even though the issue provides a stack trace.

Q4. Asking ChatGPT - OpenJ9

Question: Where in code exception of issue 18151 occur?

Incorrect Answer: The exception of issue 18151 occurs in the body of the issue.

Again, for asking to find the line number where the error occurs, ChatGPT just gives an irrelevant answer by showing the filename.

Q5. Asking ChatGPT - OpenJ9

Question: Which line in CharacterDataLatin1 class triggered ArrayIndexOutOfBoundsException in issue 17063?

Irrelevant Answer: The line of code that is “java/lang/String/ToLowerCase.java”.

The Contextual Understanding Challenges category (i.e., L2) shows a fundamental difficulty in ChatGPT to process and respond to queries about technical issues. This category primarily involves the model’s struggles with:

- Integrating and interpreting the context in which queries are made. Some context is explicitly stated within the query or the referenced issue, such as a specific error message or stack trace. Other times, the context is implicit, requiring

the model to infer based on its broader knowledge or related data points. Handling ambiguous or insufficiently detailed queries necessitates the chatbot to fill in the gaps with assumptions or inferred knowledge. Not all contextual information holds equal relevance to a given query, requiring the chatbot to prioritize the most pertinent context based on the nuances of the query.

- Adapting to the technical conventions of specific domains. Technical domains often have their own conventions for documentation, communication, and issue tracking. For instance, understanding that a particular label in an issue tracking system denotes the responsible team, requires domain-specific knowledge that the AI must possess. On asking to find a responsible team for an issue, ChatGPT searches on the assignee list, but the team details are on the issue labels. Due to this lack of contextual information, it fails to answer the question.

Q6. Asking ChatGPT - ElasticSearch

Question: Which team is responsible for issue 104160?

Incorrect Answer: The team responsible for issue 104160 is not specified in the database.

- Even when the relevant context is identified, retrieving and applying it accurately to generate a response is challenging. This includes understanding the specific ways in which information is structured or presented within data sources and how it relates to the user's query. Effectively bridging this gap is crucial for generating accurate and contextually appropriate responses.

Summary of RQ2. A RAG-based ChatGPT showed an average correctness of 36.4% on our benchmark. The pipeline encountered challenges in comprehending complex technical content and grasping contextual nuances, leading to inaccuracies in its responses.

VI. CHIME: CHATGPT INACCURACY MITIGATION ENGINE

Our observations in Section V of ChatGPT limitations contributed to the design of CHIME, as a suite of techniques to detect and fix incorrectness in ChatGPT responses. The underlying principle of CHIME is that by offering ChatGPT with a more structured representation of bug reports and by applying a systematic approach to assess ChatGPT responses, we can address the two limitations we observed in Section V-B. A more structured representation of bug reports can be achieved if we can process the different technical and textual terms properly and store those in a structured way, e.g., in a database with metadata offering more information about those terms. A systematic approach to verify the responses can be achieved by applying/adapting the techniques of guided iterative prompting of LLM responses that are used in the literature for textual content. As such, we designed to preprocess the inputs (both the bug report and the query) and to verify the ChatGPT responses.

CHIME treats ChatGPT as an API, where the inputs (questions) and outputs (answers) are processed for inaccuracy detection and mitigation. We can use another LLM as an API in CHIME and apply all the techniques we developed. Doing so would simply require changing the API endpoints to point to the other LLM within the LangChain toolkit. CHIME will need to be updated significantly while using multi-modal LLMs, e.g., to process/validate modalities other than texts, etc.

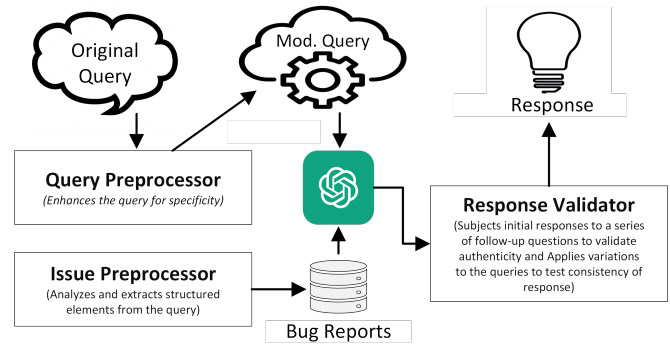


Fig. 8: The Architecture of CHIME

In Figure 8, we show the architectural diagram of CHIME. We apply an ‘Issue Preprocessor’ component to parse crash dumps and other technical terms. Issue Preprocessor employs Context-Free Grammars (CFGs) to interpret complex technical data, addressing ChatGPT’s limitations in understanding complex technical content. To address the second limitation (i.e., Contextual understanding challenges), we introduce two more components in CHIME. First, we apply ‘Query Preprocessor’ component to decipher users’ intents. Second, we designed the ‘Response Validator’ component, which evaluates the accuracy in a response using a combination of two techniques: Chain of Verification (CoVe) [16] and Metamorphic Testing (MT) [17]. A query goes through each of these components. We discuss these components below.

A. Issue Preprocessor

We developed a CFG as shown in Listing 1 which supports parsing stack traces to the level of individual code elements. Stack traces can also contain auxiliary information like timestamps and memory addresses (e.g., crash dumps). We preprocessed these auxiliary information and applied our CFG to parse the code elements in a Java stack trace. Finally, our issue preprocessor stored the code blocks in the database by separating the code blocks from the textual contents. The issue preprocessor saves the following information by parsing an issue report: title, body, stack trace, and other metadata (creation date, status, etc). When ChatGPT, enhanced with Issue Preprocessor, is presented with the query regarding the type of exception in issue 18151 (Q4) or 17063 (Q5), it correctly identifies the location of the exception detailed in the stack trace.

Listing 1: CFG Grammar for Stack Traces

```

1 Root ::= StackTraceElems
2 StackTraceElems ::= StackTraceElem StackTraceElems |
  StackTraceElem
3 StackTraceElem ::= ExceptionElems | CodeDetails
4 ExceptionElems ::= ExceptionElem ExceptionElems |
  ExceptionElem
5 ExceptionElem ::= ExceptionType | ExceptionMessage
6 CodeDetails ::= ClassElem | MethodElem | FileElem |
  LineElem

```

Asking CHIME with Q4 from Section V. OpenJ9**Question:** Where in the code exception of issue 18151?**Correct Answer:** The exception in issue 18151 can be triggered at line 98 in the JvmErgonomics.java file.**Asking CHIME with Q5 from Section V. OpenJ9****Question:** Which code in CharacterDataLatin1 triggered the ArrayIndexOutOfBoundsException in issue 17063?**Correct Answer:** Line 72 in the CharacterDataLatin1 class.**B. Query Preprocessor**

A user query may not provide enough details, leading ChatGPT to generate responses based on faulty assumptions or context. We leveraged the query transformation algorithm from [15] that strategically rephrases and modifies a query. Each query goes through this component for error correction as follows. We prompt ChatGPT with instructions on how to transform a query based on the question type. For example, for a Yes/No query type, we instruct ChatGPT as follows “Change ‘is/are/have there issues’ to check if there are any issues with the provided condition”. For a summarization type question, the instruction is “Summarize the contents from issue title, exceptions, body, and labels”. We then provide some example transformations (in a few-shot setting) to clarify the instruction, like the following.

Query Preprocessor.**Original Query:** Are issue 18102 and 18669 similar?**Transformed Query:** Compare the exceptions, stack traces, and descriptions of issues 18102 and 18669 to determine similarity.

Finally, GPT answers based on the transformed query. For example, when presented with the query, Q6 from Section V, the Query Preprocessor directs the question with additional instructions: “Summarize the team responsible for issue 104160 based on the assignee and labels.” This instruction results in generating a more precise SQL query to our database (we used LangChain SQL engine + RAG ChatGPT pipeline).

Asking CHIME with Q6 from Section V. ElasticSearch**Question:** Which team is responsible for issue 104160?**Correct Answer:** The team responsible for issue 104160 is the ML team, with no specific assignee and labels indicating it is related to ML.**C. Response Validator**

The third component of CHIME is added to validate initial responses from RAG-based ChatGPT. We employed two strategies sequentially, CoVe [16] and Metamorphic Testing (MT) [17]. Drawing inspiration from investigative interrogation techniques, CoVe’s methodology is designed to test the consistency and authenticity of ChatGPT’s responses through a series of challenge questions. CoVe’s process can be unfolded into three distinct stages 1) Upon receipt of a query, ChatGPT formulates an initial response based on its interpretation of the query within the context of available data. 2) CoVe then autonomously generates a set of follow-up questions related to the initial response. CoVe dynamically determines the number of follow-up questions. These questions aim to clarify its preceding statements. 3) The subsequent responses are analyzed for consistency with the initial response. Any observed contradictions signal the need for a review and adjustment of the initial response, leading to the refinement or generation of a new, more accurate answer.

Listing 2: How CoVe validates in CHIME

Initial Question: Is issue 18102 and 18669 similar?**Initial ChatGPT****Response:** No, issue 18102 and 18669 are not similar.**Follow-up Questions Generated by CoVe**

1. Can issue 18102 and 18669 be considered similar based on their descriptions and reported problems?
2. Have there been any interactions or dependencies between the teams working on issue 18102 and 18669 that could indicate similarity?
3. Are the issues 18102 and 18669 related to the same error?

ChatGPT Responses to Follow-up Questions

1. Yes, based on the descriptions and reported problems, they can be considered similar as both involve ArrayIndexOutOfBoundsException.
2. Yes, there have been interactions between the teams working on issue 18102 and 18669.
3. No, issues 18102 and 18669 are not related to same error.

Final Response: Yes, issue 18102 and 18669 are similar.

However, CoVe can discard correct responses (if not found in responses to challenge questions) or promote incorrect responses (if responses are found both in base and challenge questions). We thus refine CoVe with MT as follows.

Using CoVe to verify ChatGPT’s initial responses, we get an intermediate response which is then passed for consistency verification to MT. Central to MT is the concept of metamorphic relations (MRs), which involve applying transformations to input queries to anticipate changes in output. In CHIME, this approach is leveraged to introduce nuanced variations to the queries for testing the consistency of its responses under diverse conditions. A specific implementation of MRs in CHIME involves Sentence-Level Mutation, where equivalent

questions are generated to test. This process includes 1) using predefined MRs to subtly alter the phrasing or focus of the original queries. In our implementation, MT generates three mutated questions each time in CHIME. For instance, the original query might be “What causes error X in module Y?”, and its mutated equivalent could be “Why does error X occur in module Y?”. 2) The responses ChatGPT provides for both the original and mutated queries are compared to assess consistency with MR expectations. If an MR implies that adding specific details should refine the response, the evaluation checks for this level of refinement. After the refinement via MR, CHIME generates the final response for the users. The user in CHIME does not see the mutated questions from CoVe or the mutated questions from MRs. Instead, the user only sees the final response from CHIME. However, CHIME can print the mutated questions to the user if needed.

Listing 3: How CoVe + MT validates in CHIME

Initial Question: Does Elasticsearch require the UseG1GC option to be present during issue 18151 startup stage?

Initial ChatGPT Response: No, Elasticsearch does not require the UseG1GC option to be present during its startup stage in issue 18151.

Follow-up Questions Generated by CoVE:

1. Is Elasticsearch the software mentioned in the response?
2. Is the UseG1GC option not required?
3. Is issue 18151 not requiring the UseG1GC option?

ChatGPT Responses to Follow-up Questions:

1. Yes, Elasticsearch is mentioned in the response.
2. No, the UseG1GC option is not required.
3. No, issue 18151 is not requiring the UseG1GC option.

Intermediate Response from

CoVE: No, the UseG1GC option is not required during Elasticsearch’s startup stage.

Mutated Questions Generated by MT:

1. Is the UseG1GC option necessary for Elasticsearch to be present during its startup stage in issue 18151?
2. Must the UseG1GC option be included during Elasticsearch’s startup stage for issue 18151?
3. Is it required to have the UseG1GC option present during Elasticsearch’s startup stage for issue 18151?

ChatGPT Responses to Mutated Questions:

1. No, the UseG1GC option is not necessary for Elasticsearch’s to be present during its startup stage in issue 18151.
2. Yes, the UseG1GC option must be included during Elasticsearch’s startup stage for issue 18151.
3. Yes, it is required to have the UseG1GC option present during Elasticsearch’s startup stage for issue 18151.

Final Response: Yes, it is required to have the UseG1GC option during Elasticsearch’s startup stage .

VII. EFFECTIVENESS OF CHIME

We evaluate CHIME by answering three RQs:

RQ3. How well can CHIME fix inaccuracies in ChatGPT responses?

RQ4. How well do individual components in CHIME perform over ChatGPT?

RQ5. Would responses from CHIME be favored like those from ChatGPT when both are correct?

RQ3 and RQ4 investigate whether CHIME and its components can fix inaccuracies in ChatGPT while we use our benchmark. Similar to RQ2, we use the correctness metric from Equation 1 to answer RQ3 and RQ4. RQ5 assesses the usability of CHIME when it is used by developers instead of a RAG-based ChatGPT. We conduct a user study to answer RQ5.

TABLE VII: Correctness of CHIME. Column ‘Improv’ shows percent improvement over RAG-based ChatGPT

T#	Y/N	Fact	Sum	Total	Improv
Total	80.0%	61.4%	65.5%	66.7%	+30.3%
T1 - Issue Anlys(S)	83.3%	67.1%	66.7%	70.8%	+36.3%
T1 - Issue Anlys(M)	58.3%	30.0%	50.0%	42.5%	+20.0%
T2 - Issue Trend	68.8%	43.8%	50.0%	55.0%	+22.5%
T3 - Issue Summary	0.0%	87.5%	68.8%	72.5%	+22.5%
T4 - Issue Labeling	83.3%	60.0%	87.5%	72.5%	+22.5%
T5 - Issue Backlog	100.0%	58.3%	50.0%	70.0%	+32.5%

A. How well can CHIME fix ChatGPT inaccuracies? (RQ3)

Table VII presents the assessments of the correctness of CHIME in our benchmark by offering overall results, across the three types of queries and also across the five task types in our benchmark. Overall, CHIME offers around 30.3% improvement over the RAG-based ChatGPT pipeline from Section V. The improvement is consistent across all five task types, with issue analytics and backlog tasks benefiting the most from CHIME. CHIME showcases enhancements over ChatGPT across all tasks for both OpenJ9 and ElasticSearch: 29.6% and 31.1% improvement over ChatGPT for OpenJ9 and ElasticSearch respectively. The detailed result for each project is provided in our online appendix.

We manually assessed the responses where CHIME was inaccurate and observed three main reasons as follows.

Query-Directed Retrieval Failure (60.6%): CHIME relies on its ability to query a database of stored data and generate SQL queries based on the provided questions. However, when user or verifying queries lack clarity, the query fails to provide clear instructions for formulating SQL queries. In such cases, CHIME produces incorrect or irrelevant responses.

Logical Inference Errors (27%): This pertains to cases where CHIME fails to accurately apply logical inference principles. It occurs when CHIME incorrectly deduces information from the data or makes faulty assumptions during reasoning.

Semantic Discrepancy (10.9%): CHIME relies on similarity scores to match user queries with existing data or responses. However, discrepancies in semantic similarity assessments can lead to incorrect matches or associations.

Summary of RQ3. CHIME achieves an average correctness of 66.7% and an improvement of 30.3% over a RAG-based ChatGPT on our benchmark of bug report questions.

TABLE VIII: Correctness of components of CHIME. Column ‘Improv’ shows percent improvement over RAG-based ChatGPT

Component	Y/N	Fact	Sum	Total	Improv
Issue Preprocessor	59.0%	43.9%	46.4%	48.1%	+11.7%
Query Preprocessor	57.0%	35.1%	45.2%	42.5%	+6.1%
Response Validator	55.0%	36.0%	47.6%	43.0%	+6.6%
CoVe	58.0%	28.9%	38.1%	37.9%	+1.5%
MT	69.0%	37.7%	53.6%	48.5%	+12.1%

B. How do individual components in CHIME perform? (RQ4)

We ran each component of CHIME individually and determined the contribution of the component within the pipeline. In Table VIII, we show the performance of each component per query type and also show whether the component offered an improvement over a RAG-based ChatGPT. We discuss how we ran each component while analyzing the results below.

Issue Preprocessor. In our CHIME pipeline, we kept this component and removed the other two components (i.e., Query Processor and Response Validator). Hence, issue reports are preprocessed by this component and then stored in the database. From here, we utilize RAG-based ChatGPT for Q&A. The integration of the Issue Processor enhances the accuracy of the baseline GPT model by 11.7% on average for both projects. This improvement is particularly notable in technical question comprehension and analysis tasks.

Query Preprocessor. Similar to the above setup, we only kept this component and removed the other two components in our CHIME pipeline (i.e., Issue Preprocessor and Response Validator). On average, this process demonstrates an improvement of 6.1% over a RAG-based ChatGPT.

Response Validator. We used it to validate responses from an RAG-based ChatGPT. Overall, this component contributed to a 6.6% improvement over a RAG-based ChatGPT. The bottom two rows in Table VIII further illustrate the performance of the two modules in the Response Validator, i.e., CoVe and MT. Interestingly, MT as an individual module worked even better than the Response Validator component. MT offered a 12.1% improvement over RAG-ChatGPT while CoVe offered a 1.5% improvement. However, we kept the combinations of CoVe and MT in the response validator, because CoVe + MT may become more useful for other repos where responses may need a sequence of challenges via both CoVe and MT. For example, when the responses from CoVe contain references to the fact (but with incorrect summarization), MT can double-check those facts via follow-up mutated questions.

As we can see from Tables VII and VIII, CHIME as an end-to-end pipeline offers 30.3% improvement over RAG-based ChatGPT, while none of the individual components in CHIME could offer more than 12% improvement over RAG-based ChatGPT. This means that the ensemble of all the components in CHIME’s pipeline helped the fixing of one’s mistake by others. For instance, when a user queries “List all pending issues” rather than simply providing the

count of pending issues, the transformed query from Query Preprocessor prompts the system to generate a list of issue numbers, which increases its accuracy.

Summary of RQ4. Each component in CHIME can offer an improvement over a RAG-based ChatGPT by correcting the inaccuracies in ChatGPT responses. The components work best when they are all put together in CHIME as an end-to-end pipeline.

C. Would responses from CHIME favored like those from ChatGPT when both are correct? (RQ5)

A comparative study was conducted to assess the practical efficacy of CHIME, involving 31 participants. The majority (93%) had 0-5 years of experience in the software industry, with 57% being software developers and 33% researchers.

1) *Survey Setup:* Participants were presented with two random questions from each task in our benchmark dataset. The questions include a summarization of the failure of an issue, similarities between multiple issues, recurring themes in a component, pending issues, identification of error-prone components, unresolved or blocker issues, guidelines for labeling, and label suggestions. For these questions, responses from both CHIME and ChatGPT, along with links to associated bug reports, were provided for evaluation. To ensure a fair comparison, only questions with correct responses from both systems were selected. Participants were then asked to rate the correctness and perceived usefulness of the responses in addressing software bug-related queries. The survey questions are provided in our online appendix.

2) *Survey Result:* Participant feedback in Table IX indicates that CHIME was the preferred choice for the majority of tasks when the answers were correct and selected more frequently for 6 out of 10 questions. It was favored in issue analysis (T1) with a 79% participant preference. On average, for this task, 63.6% of participants found the responses to be comprehensive and covering all necessary aspects, 45.1% felt that they provided additional information helpful for a better understanding of the problem, and 33.3% thought the responses were clear and easy to follow. Their preference also extended to issue summarization (T3) with a 65% preference and to issue labeling (T4), with a 63% preference. However, for the issue trending task (T2), there was a slight preference for ChatGPT. Nonetheless, for two questions, participants seemed undecided, indicating a comparable level of usefulness between CHIME and ChatGPT when the responses were correct.

Summary of RQ5. In a comparative study with 31 participants, CHIME responses were preferred over a stand-alone ChatGPT for the majority of tasks when both provided correct answers. This preference was particularly evident for tasks related to issue analysis, summarization, and labeling.

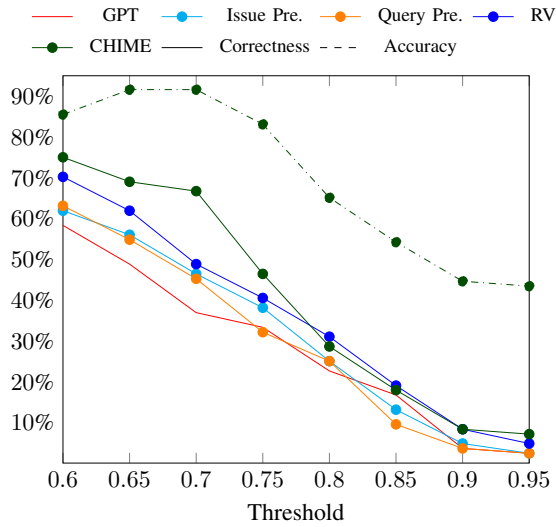


Fig. 9: Impact of Threshold on Similarity Analysis. Here, Issue Pre. = Issue Preprocessor, Query Pre. = Query Preprocessor, RV = Response validator

TABLE IX: Selection Preference of CHIME and ChatGPT Provided Correct Responses across Tasks

T#	ChatGPT	CHIME
Overall	40%	60%
T1 - Issue Analytics	21%	79%
T2 - Issue Trend	58%	42%
T3 - Issue Summary	35%	65%
T4 - Issue Labeling	37%	63%
T5 - Issue Backlog	48%	52%

VIII. DISCUSSION

A. Accuracy of our CFG

We evaluated the CFG-based stack trace parsing by assessing the 80 issue reports that we used to create our benchmark dataset. The CFG is designed to identify key elements, such as exception types, messages, and code details (e.g., class/method/file names, etc.). For each stack trace, we checked whether the parser found all key elements as expected. We used three metrics to compute accuracy: precision, recall, and F1-score. Precision is the ratio of correctly identified elements to the total elements identified by the parser. Recall is the ratio of correctly identified elements to the total actual elements in the stack trace. F1-score ($F1$) is the harmonic mean of precision and recall. We manually created a list of the expected elements for accurate comparison for each of the 80 issue reports. We observed an average precision of 0.99 and recall of 0.91 ($F1$ -score = 0.93). The few errors in parsing were mainly due to the limitations in our regular expressions used in the CFG parser, and the variations in stack trace formats across

issue reports. Our replication package contains the details of the assessment.

B. Threats to Validity

Concerns regarding **construct validity** arise from the benchmark's design. However, we derived our benchmark queries from survey responses. The selection of issues from OpenJ9 and ElasticSearch may affect the generalizability of the findings across various software engineering contexts. The participant pool in surveys might not comprehensively represent the diverse perspectives in the broader software engineering community. Finally, the methodology used for evaluating CHIME could affect the accuracy and objectivity of our effectiveness assessment and may introduce **methodological bias**. However, we have analyzed the accuracy rates of summary queries across various similarity threshold values of CHIME, ranging from 0.60 to 0.95, and chose the threshold of 0.7 as it gives the highest accuracy (see Figure 9), an optimal trade-off between capturing relevant information and minimizing false positives.

IX. CONCLUSION

We have introduced CHIME to mitigate the inaccuracy of ChatGPT response during bug report exploration. CHIME demonstrates 30.3% improvements over ChatGPT in terms of providing more correct responses for bug exploration tasks. Our industrial partner is working on deploying CHIME as a Slack bot. The conceptualization of CHIME originated from an internal demo of a similar chatbot created by our partner one year ago. Their initial chatbot lacked the required accuracy, which we sought to address by developing CHIME. To further motivate the need for such a chatbot beyond our industrial partner, we conducted a survey of 47 software practitioners (see Section III). The survey findings highlight the necessity of such chatbots in the real world. Feedback from 31 industry participants, presented in Section VII-C shows that CHIME is preferred for its ability to analyze, summarise, and label issues. Our industrial partner was involved in the design and evaluation of CHIME, which was crucial for advancing CHIME from the proof-of-concept stage to the current deployment stage within the company. Like any innovation, we expect to improve CHIME in an agile manner, i.e., based on user feedback once deployed.

In the future, we will also focus on expanding CHIME's grasp of more technical terminologies and other documents. To handle other documents, in CHIME we will improve the issue preprocessor module e.g., to separate code and textual contents, and to adapt the CFG to handle code snippets/traces/crash dumps in those documents or using a static partial program analyzer to handle code examples in API documentation. We expect that the other modules in CHIME can be used with minimal changes.

DATA AVAILABILITY

The code and data used for this study can be found here: <https://bit.ly/4fyaMIP>.

REFERENCES

- [1] Z. Ji *et al.*, “Survey of hallucination in natural language generation,” *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.
- [2] K. Filippova, “Controlled hallucinations: Learning to generate faithfully from noisy data,” in *Findings of the Association for Computational Linguistics: EMNLP 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020, pp. 864–870. [Online]. Available: <https://aclanthology.org/2020.findings-emnlp.76>
- [3] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, “A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions,” *ArXiv*, vol. abs/2311.05232, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:265067168>
- [4] M. Jang, D. S. Kwon, and T. Lukasiewicz, “BECCEL: Benchmark for consistency evaluation of language models,” in *Proceedings of the 29th International Conference on Computational Linguistics*, N. Calzolari, C.-R. Huang, H. Kim, J. Pustejovsky, L. Wanner, K.-S. Choi, P.-M. Ryu, H.-H. Chen, L. Donatelli, H. Ji, S. Kurohashi, P. Paggio, N. Xue, S. Kim, Y. Hahm, Z. He, T. K. Lee, E. Santus, F. Bond, and S.-H. Na, Eds. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, Oct. 2022, pp. 3680–3696. [Online]. Available: <https://aclanthology.org/2022.coling-1.324>
- [5] M. Jang and T. Lukasiewicz, “Consistency analysis of chatgpt,” in *Conference on Empirical Methods in Natural Language Processing*, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:257496270>
- [6] R. Cohen, M. Hamri, M. Geva, and A. Globerson, “LM vs LM: Detecting factual errors via cross examination,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 12 621–12 640. [Online]. Available: <https://aclanthology.org/2023.emnlp-main.778>
- [7] B. A. Galitsky, “Truth-o-meter: Collaborating with llm in fighting its hallucinations,” 2023.
- [8] OpenJ9. (2024, February) Issue 18151: To accept openjdk option +useJgc to enable startup of elasticsearch application]. [Online]. Available: <https://github.com/eclipse-openj9/openj9/issues/18151>
- [9] D. Arya, W. Wang, J. L. C. Guo, and J. Cheng, “Analysis and detection of information types of open source software issue discussions,” in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE ’19. IEEE Press, 2019, p. 454–464. [Online]. Available: <https://doi.org/10.1109/ICSE.2019.00058>
- [10] Z. Jiang, F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig, “Active retrieval augmented generation,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 7969–7992. [Online]. Available: <https://aclanthology.org/2023.emnlp-main.495>
- [11] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS’20. Red Hook, NY, USA: Curran Associates Inc., 2020.
- [12] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang, “Realm: retrieval-augmented language model pre-training,” in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML’20. JMLR.org, 2020.
- [13] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, “Retrieval augmentation reduces hallucination in conversation,” in *Findings of the Association for Computational Linguistics: EMNLP 2021*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 3784–3803. [Online]. Available: <https://aclanthology.org/2021.findings-emnlp.320>
- [14] K. Beelen and D. van Strien, “Metadata might make language models better,” 2022. [Online]. Available: <https://arxiv.org/abs/2211.10086>
- [15] X. Ma, Y. Gong, P. He, hai zhao, and N. Duan, “Query rewriting in retrieval-augmented large language models,” in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. [Online]. Available: <https://openreview.net/forum?id=gXq1cwKUZc>
- [16] S. Z. Dhuliawala, M. Komeili, J. Xu, R. Raileanu, X. Li, A. Celikyilmaz, and J. E. Weston, “Chain-of-verification reduces hallucination in large language models,” 2024. [Online]. Available: <https://openreview.net/forum?id=VP20ZB6DHL>
- [17] T. Chen, F. Kuo, H. Liu, P. Poon, D. Towey, T. Tse, and Z. Zhou, “Metamorphic testing: A review of challenges and opportunities,” *ACM Computing Surveys*, vol. 51, no. 1, pp. 1–27, 2018.
- [18] C. Wang and R. Sennrich, “On exposure bias, hallucination and domain shift in neural machine translation,” 07 2020.
- [19] B. Liu, J. T. Ash, S. Goel, A. Krishnamurthy, and C. Zhang, “Exposing attention glitches with flip-flop language modeling,” in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: <https://openreview.net/forum?id=VzmpXQAn6E>
- [20] K. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison-Burch, and N. Carlini, “Deduplicating training data makes language models better,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 8424–8445. [Online]. Available: <https://aclanthology.org/2022.acl-long.577>
- [21] N. Dziri, A. Madotto, O. Zaiane, and A. J. Bose, “Neural path hunter: Reducing hallucination in dialogue systems via path grounding,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 2197–2214. [Online]. Available: <https://aclanthology.org/2021.emnlp-main.168>
- [22] H.-S. Chang and A. McCallum, “Softmax bottleneck makes language models unable to represent multi-modal word distributions,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 8048–8073. [Online]. Available: <https://aclanthology.org/2022.acl-long.554>
- [23] B. Dhingra, M. Faruqi, A. P. Parikh, M.-W. Chang, D. Das, and W. H. Cohen, “Handling divergent reference texts when evaluating table-to-text generation,” *ArXiv*, vol. abs/1906.01081, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:174797747>
- [24] S. Lin, J. Hilton, and O. Evans, “TruthfulQA: Measuring how models mimic human falsehoods,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3214–3252. [Online]. Available: <https://aclanthology.org/2022.acl-long.229>
- [25] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. B. Brown, D. X. Song, Ú. Erlingsson, A. Oprea, and C. Raffel, “Extracting training data from large language models,” in *USENIX Security Symposium*, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:229156229>
- [26] Z. Li, S. Zhang, H. Zhao, Y. Yang, and D. Yang, “Batgpt: A bidirectional autoregressive talker from generative pre-trained transformer,” *ArXiv preprint*, vol. abs/2307.00360, 2023.
- [27] Y. Liu, Y. Yao, J.-F. Ton, X. Zhang, R. Guo, H. Cheng, Y. Klockov, M. F. Taufiq, and H. Li, “Trustworthy llms: A survey and guideline for evaluating large language models’ alignment,” *ArXiv preprint*, vol. abs/2308.05374, 2023.
- [28] N. Lee, W. Ping, P. Xu, M. Patwary, M. Shoenybi, and B. Catanzaro, “Factuality enhanced language models for open-ended text generation,” *ArXiv*, vol. abs/2206.04624, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:249538460>
- [29] W. Shi, X. Han, M. Lewis, Y. Tsvetkov, L. Zettlemoyer, and S. W.-t. Yih, “Trusting your evidence: Hallucinate less with context-aware decoding,” *ArXiv preprint*, vol. abs/2305.14739, 2023.
- [30] A. Radford *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [31] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima *et al.*, “The pile: An 800gb dataset of diverse text for language modeling,” *arXiv preprint arXiv:2101.00027*, 2021.
- [32] H. Touvron *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” *arXiv preprint arXiv:2306.00186*, 2023.
- [33] A. Sinitin, V. Plokhomyuk, D. Pyrkyn, S. Popov, and A. Babenko, “Editable neural networks,” in *8th International Conference on Learning Representations, ICLR 2020*. Addis Ababa, Ethiopia: OpenReview.net, April 26–30 2020.

- [34] Y. Yao, P. Wang, B. Tian, S. Cheng, Z. Li, S. Deng, H. Chen, and N. Zhang, "Editing large language models: Problems, methods, and opportunities," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 10 222–10 240. [Online]. Available: <https://aclanthology.org/2023.emnlp-main.632>
- [35] Z. Jiang *et al.*, "Active retrieval augmented generation," *arXiv preprint arXiv:2305.06983*, 2023.
- [36] D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of chatgpt," in *2023 IEEE/ACM International Workshop on Automated Program Repair (APR)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 23–30. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/APR59189.2023.00012>
- [37] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:49313245>
- [38] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. 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," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. [Online]. Available: <https://openreview.net/forum?id=6IE4dQXaUcb>
- [39] J. Lanchantin *et al.*, "Learning to reason and memorize with self-notes," *arXiv preprint arXiv:2305.00833*, 2023.
- [40] B. Peng *et al.*, "Check your facts and try again: Improving large language models with external knowledge and automated feedback," *arXiv preprint arXiv:2302.12813*, 2023.
- [41] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, "Retrieval augmentation reduces hallucination in conversation," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 3784–3803. [Online]. Available: <https://aclanthology.org/2021.findings-emnlp.320>
- [42] A. Madaan *et al.*, "Self-refine: Iterative refinement with self-feedback," *arXiv preprint arXiv:2303.17651*, 2023.
- [43] C. Malaviya *et al.*, "Quest: A retrieval dataset of entity-seeking queries with implicit set operations," *arXiv preprint arXiv:2305.11694*, 2023.
- [44] P. Manakul *et al.*, "Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models," *arXiv preprint arXiv:2303.08896*, 2023.
- [45] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:52967399>
- [46] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547. [Online]. Available: <https://aclanthology.org/2020.findings-emnlp.139>
- [47] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, J. Yin, D. Jiang, and M. Zhou, "Graphcodebert: Pre-training code representations with data flow," *ArXiv*, vol. abs/2009.08366, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:221761146>
- [48] 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*, K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, Eds. Online: Association for Computational Linguistics, Jun. 2021, pp. 2655–2668. [Online]. Available: <https://aclanthology.org/2021.naacl-main.211>
- [49] C. Raffel, N. M. 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," *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:204838007>
- [50] L. A. Goodman, "Snowball sampling," *Annals of Mathematical Statistics*, vol. 32, no. 1, pp. 148–170, 1961.
- [51] G. Rodriguez-Perez, G. Robles, and J. M. Gonzalez-Barahona, "How much time did it take to notify a bug? two case studies: Elasticsearch and nova," in *2017 IEEE/ACM 8th Workshop on Emerging Trends in Software Metrics (WETSoM)*, 2017, pp. 29–35.
- [52] G. Rodríguez-Pérez, Gregorio, Robles, A. Serebrenik, Andy, Zaidman, D. M. Germán, Jesus, and J. M. Gonzalez-Barahona, "How bugs are born: a model to identify how bugs are introduced in software components," *Empirical Software Engineering*, vol. 25, pp. 1294 – 1340, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:203693795>
- [53] T. Zhang, D. Han, V. Vinayakara, I. C. Irsan, B. Xu, F. Thung, D. Lo, and L. Jiang, "Duplicate bug report detection: How far are we?" *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 4, may 2023. [Online]. Available: <https://doi.org/10.1145/3576042>
- [54] Z. Rasool, S. Barnett, S. Kurniawan, S. Balugo, R. Vasa, C. Chesser, and A. Bahar-Fuchs, "Evaluating llms on document-based qa: Exact answer selection and numerical extraction using cogtale dataset," *ArXiv*, vol. abs/2311.07878, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:265158057>
- [55] C. Wang, X. Liu, Y. Yue, X. Tang, T. Zhang, C. Jiayang, Y. Yao, W. Gao, X. Hu, Z. Qi, Y. Wang, L. Yang, J. Wang, X. Xie, Z. Zhang, and Y. Zhang, "Survey on factuality in large language models: Knowledge, retrieval and domain-specificity," *ArXiv*, vol. abs/2310.07521, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:263835211>
- [56] M. P. Polak and D. Morgan, "Extracting accurate materials data from research papers with conversational language models and prompt engineering," *Nature Communications*, vol. 15, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:257427054>
- [57] L. Basyal and M. Sanghvi, "Text summarization using large language models: A comparative study of mpt-7b-instruct, falcon-7b-instruct, and openai chat-gpt models," *arXiv preprint arXiv:2310.10449*, October 2023. [Online]. Available: <https://arxiv.org/abs/2310.10449>
- [58] LangChain, "SQLDatabaseChain," Webpage. [Online]. Available: https://api.python.langchain.com/en/latest/sql/langchain_experimental.sql.base.SQLDatabaseChain.html
- [59] Hugging Face, "all-MiniLM-L6-v2," Webpage. [Online]. Available: <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>
- [60] F. Rahunoto, T. Kitasuka, and M. Aritsugi, "Semantic cosine similarity," 2012. [Online]. Available: <https://api.semanticscholar.org/CorpusID:18411090>