On Enhancing Root Cause Analysis with SQL Summaries for Failures in Database Workload Replays at SAP HANA

Neetha Jambigi
University of Cologne
Cologne, Germany
njambigi@smail.uni-koeln.de

Joshua Hammesfahr,
Moritz Mueller
SAP
Walldorf, Germany
{joshua.hammesfahr,
moritz.mueller07}@sap.com

Thomas Bach
SAP
Walldorf, Germany
0000-0002-9993-2814

Michael Felderer
German Aerospace Center
University of Cologne
Cologne, Germany
0000-0003-3818-4442

***Abstract*—Capturing the workload of a database and replaying this workload for a new version of the database can be an effective approach for regression testing. However, false positive errors caused by many factors such as data privacy limitations, time dependency or non-determinism in multi-threaded environment can negatively impact the effectiveness. Therefore, we employ a machine learning based framework to automate the root cause analysis of failures found during replays. However, handling unseen novel issues not found in the training data is one general challenge of machine learning approaches with respect to generalizability of the learned model. We describe how we continue to address this challenge for more robust long-term solutions. From our experience, retraining with new failures is inadequate due to features overlapping across distinct root causes. Hence, we leverage a large language model (LLM) to analyze failed SQL statements and extract concise failure summaries as an additional feature to enhance the classification process. Our experiments show the F1-Macro score improved by 4.77% for our data. We consider our approach beneficial for providing end users with additional information to gain more insights into the found issues and to improve the assessment of the replay results.**

I. Introduction

Replaying recorded database queries poses several practical challenges, including the complexity of large replays, recording costs, legal considerations, and the occurrence of false positives. Non-deterministic factors, such as multi-threading and concurrency, randomness, and missing functionality in the recording tool, contribute to errors that do not signify regressions [1]. External factors like network and infrastructure issues further contribute to a variety of reasons for false positives in practice [2]. This can result in up to 1 million failures per replay [1]. Deciding whether a failure is a true positive or false positive case requires to identify the root cause of an error to finally assess whether a software regression exists. However, manual root cause identification and assessment is impractical due to the large amount of failures. Previous work proposed a machine learning-based automated root cause analysis system called MIRA to attribute root causes to failures arising from a capture and replay-based testing approach [1]. MIRA has been in productive use for capture and replay testing the database management system SAP HANA [3], [4]. In addition to testing for regressions, capture and replay is also used to perform evaluations of new features, identify performance issues, and other purposes. The approach is rather successful as MIRA helped to detect one third of potential software regressions for upcoming major versions [1].

MIRA is a supervised learning setup that relies on the features of a SQL statement collected during the replays as shown in Table I. MIRA attempts to learn representations for the SQL statement strings in combination with other textual attributes like error messages using *TFIDF* [5], *Doc2Vec* [6], or *fasttext* [7] in order to use them for classifying replay failures. However, as the observed software evolves, new failures arising from new software features demand continuous adaptation of MIRA to be able to identify root causes for new replay failures. With the current set of attributes, failures from several different root cause categories tend to be almost indistinguishable by classification models, due to extremely overlapping features across failure categories. Manual exploration of regressions arising during replays encompasses a comprehensive understanding of the events preceding a statement's failure during the replay process. More than 10% of the root cause categories require this historical context to be classified under the right category. To tackle these issues arising from data evolution, we propose encoding the historical information of a failed SQL statement to further contextualize a replay failure. Previous work focus on developing effective representations for SQL queries using natural language processing techniques in order to model various aspects of databases [8], [9], [10], [11]. Our approach involves constructing a new feature by utilizing a large language model (LLM) analyzing a sequence of failed SQL statements to extract failure summaries.

In the following sections we provide an overview of MIRA and the application in a production environment, outline the encountered challenges, and present a method for enhancing the current setup to accommodate the evolving data landscape.

II. Failure Analysis with MIRA

In this section we discuss the performance of MIRA in a production environment, user feedback on its performance, and possible measures to address drifting data.

A. Performance in Production

Captured workload consists typically of SQL statements. SQL statements executed during a replay are referred to as 'events'. Failed events from a replay are forwarded to MIRA and this workflow is presented as a part of Figure 3. Each failed event is assigned a root cause category along with measures to indicate the certainty of the classification. The predictions that are deemed uncertain are manually assessed and reclassified by 'operators' who are the domain experts. During the time MIRA has been in use, there have been several instances of reclassifications by operators. These include instances where additional context is required to explain the reclassification, as well as a few instances of human error, and instances that the classifier could benefit from. The classifier undergoes weekly retraining, incorporating new failure data and operator-reclassified events. Before each training cycle, cross-validation, coupled with hyperparameter optimization, is conducted, monitoring F1-Scores. A significant decline in F1-Score from the previous iteration may signify potential issues with training items with overlapping features of different classes or human errors in the reclassification process. An expert reviews these items and makes decisions concerning training data and subsequent classifier training. The current setup in MIRA in production is a KNN [12] classifier with custom distance (CD) [1] using features as shown in Table I. We employ KNN because of its interpretability for analyzing predictions, which is important in production settings. The textual attributes are concatenated and vectorized as a single feature using *fasttext* [7] while the categorical attributes are one-hot-encoded.

Table I: Overview of Data Types

Attribute	Type(#Values)	Example
Error Code	Categorical (142)	-303, 111
Request Name	Categorical (10)	1, 2
SQL Type	Categorical (7)	1, 19, 5
SQL Sub Type	Categorical (79)	1, 2, 3
Error Message	Text (-)	'Cannot find table/view X'
SQL Statement Strings	Text (-)	'Select * from table X'

Over the course of 2.5 years, over 1,000 replays from 180 unique workload captures have been processed through MIRA. Currently, the total number of failure categories has grown to 233, which is 2.5 times the original count of 93 [1]. Over the span of the past 12 months, on average 30 replays per month have been processed through MIRA with an average of failures from 9 unique root causes per replay. The average number of failed statements per replay is 31,600 and the average size of a replay is over 670 million events. The latest model configuration achieved an average F1-Macro score of 82.45% across 20 iterations during training cross-validation. Subsequently, with the introduction of flagging predictions from *problem group*, which is further explained in Section II-B, the average F1-Macro score over 17 iterations of training has decreased to 81.48%.

B. Mitigating Data Evolution Issues

Ongoing replay analysis reveals new failure types, demanding continuous adaptation of our current setup and models in MIRA. With the evolving data, there has been a growing set of replay failures that look identical despite considerably different root causes. The uncertainty measures cannot identify the situations without additional information. There are very limited samples for new root causes in MIRA and oversampling of minority classes can have potential impacts on classes with shared vocabularies making it further challenging. However, the introduction of a new feature into a productive system requires thorough evaluation due to potentially significant ramifications. The solution is expected to make the classification better across the categories with issues while not hampering the performance of the remaining classes.

Some challenges are relatively trivial in terms of efforts needed to identify alternative mechanisms to overcome the limitations of how we vectorize textual attributes. For instance, to address changing patterns in Error Messages, replacing the original mechanism with techniques like *fasttext* works effectively. However, the identical failures with very different root causes require a more robust approach like the selection of new features or feature engineering.

Stack traces, despite being potential discriminating attributes, are not universally available for all types of errors in this data. The frequency of such failures is rare and makes it rather difficult to gather more data and conduct experiments to apply this solution productively. Additionally, obtaining precise stack traces automatically for a failed event is currently a very complex task.

In contrast to stack traces, each event is linked to a SQL statement string. Our initial assessments suggested that statement strings can aid in resolving issues of overlapping features in MIRA. We recognized the potential inclusion of SQL statement strings as an attribute, in addition to Error Messages, through experiments outlined in Table II. Additionally, we explored *fasttext* as an alternative for vectorizing Error Messages resulting in improvements. We used χ^2 [13] statistical test on both Error Messages and SQL Statement strings to identify the most important terms for classification in MIRA to inform the text preprocessing for the experiments. Concatenating SQL Statement strings with Error Messages and vectorizing them using *fasttext* due to the sub-word-embeddings tends to be a more effective feature across all classes and is used productively in MIRA.

In spite of the inclusion of SQL Statement string for classification, as the set of failures increased, instances from classes not prone to misclassification were observed to be misclassified. This indicates a deterioration in the model's quality due to changes in the training data. Given the set of features as shown in Table I, more than 20% of failure categories involve instances that exhibit notable vocabulary

Table II: Average F1-Scores of 5-fold cross-validation Error Message, Statement Strings, Both (CD: Custom Distance, ED: Euclidean Distance)

		KNN	
Feature	Vectorization	CD	ED
Error Message	TFIDF	94.77	92.21
	Doc2Vec	94.41	91.37
	Fasttext	95.41	94.30
SQL Statements	TFIDF	93.72	88.68
	Doc2Vec	97.01	95.32
Both	TFIDF	94.15	89.90
	Doc2Vec	94.51	92.25

overlap with instances from other classes. This led to the identification of problematic classes. Identification of these problematic classes involves TFIDF vectorization of Error Messages and SQL Statement strings, calculating cosine similarity between vectors. A class is considered problematic if it shares training instances with over 0.95 cosine similarity with instances from other classes, forming a reference *'problem group'*. If the classifier KNN, opts for one of the training items from the problem group as one of the K neighbors to classify a failed event in production, the prediction is flagged for manual inspection regardless of the certainty. There were 62,000 events that belong to this *problem group* over 6 months amounting to 1.56% of the total failed events. Correct yet uncertain predictions of MIRA are deemed false uncertainties, which are manually investigated. Adjusting certainty thresholds to minimize these may lead to overlooking potential new failures. False uncertainties, combined with predictions from problematic groups, amplify manual labor. Given the replay sizes, manual inspection may not remain a sustainable solution.

C. Feedback from End-Users

The workflow of MIRA necessitates operator involvement for feedback and correction, posing an initial overhead. Following the initial learning period, the utilization of MIRA appeared to remain consistent among operators. We collected assessments from operators who analyzed MIRA's results in a production environment over a period of 9 months. As an assessment, operators provided numerical ratings per replay to indicate the severity of deviations from the expected behavior observed in MIRA's predictions. These ratings ranged from 1 to 4, with higher numbers indicate more severe deviations and expected behavior is rated 1. More severe deviations encompass and supersede less severe deviations for a single replay. The categorization of severity ratings is summarized in Table III. Based on collective analysis of this feedback over 199 replays 55% of the replays are rated 1, 37% are rated 2, 6.5% are rated 3 and 1.5% are 4. An average of the rating per week over 37 weeks is presented in Figure 1. Overall, there were 2 cases where new failures were missed, and the assessment resulted in a rating of 4.

Table III: Rating Definition (*: All cases)

	Certain	Uncertain
Known class	Correctly classified → No Action Needed (1) Misclassified → Reclassified by Operator (3)	∗ → Reclassified by Operator (2)
New class	∗ → Manually detected by Operator (4)	New failure → Assessed by Operator (1)

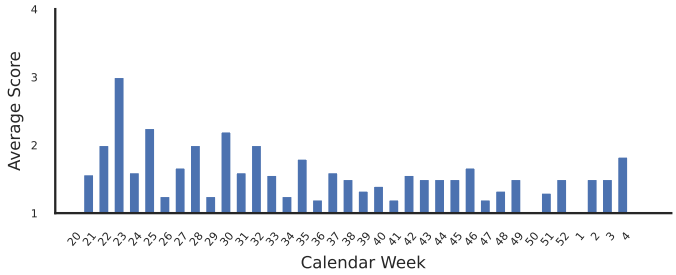

Figure 1: Weekly Average User-Review

III. Feature Extraction

Given the challenges outlined in Section II-B, we implement a mechanism that emulates operators' processes for identifying problematic root cause categories. This entails applying filtering conditions to collect a set of SQL statements and relevant attributes, which are subsequently summarized using an LLM.

Operators examine statements they deem relevant for identifying a failure's root cause. In a majority of analyzed cases, the root cause can be linked to statements executed within the same session ID. The session ID serves as an identifier for the session from a client (e.g., HANA Studio) to the Database Server. While a database can run multiple parallel sessions with different IDs, a single session in HANA can execute only one transaction at a time, ensuring sequential execution of all SQL statements in that session. Since concurrency still remains a challenge in the capture and replay framework, we constrain our collection to a single session ID.

The failure of subsequent statements, prompted by a preceding failed statement in our data, can be predominantly linked to unsuccessful or unexecuted DDL SQL statements. Identifying the root cause of failures is crucial, with factors like connection errors, privilege issues, or the omission of an SQL statement due to pre-processing of the workload before re-execution being important contributors to these failures. Prior to replaying the captured workload, preprocessing addresses inconsistencies arising from transactions that start before or end after the recording. Specific SQL statements are either skipped or executed based on assigned reasons. For instance, skipping a 'CREATE TABLE X' statement may result in statements accessing this table failing with the same information 'Cannot find table/view X.' Therefore, the attribute 'Skip Reasons' contributes to contextualizing the failures in a replay.

Extracting relevant SQL statements to contextualize a failed event 'E' is limited to replay events that satisfy specific conditions: i) event - failed or skipped ii) event must have been executed in the same session ID as 'E' iii) event must have been processed prior to 'E'. The set is further filtered to retain unique SQL statements utilizing the statement hash values.

This relevant data can also be collected using transactional dependency graphs from the capture and replay framework. For each SQL statement execution in the workload, the framework records a list of closely related database objects to identify the relevant statements for the failed event. However, the framework lacks an object list for DDLs like 'CREATE' statements, resulting in incomplete information collection. Avoiding the manual selection criteria is challenging, and it is unclear if this method provides better contextualization compared to the rule-based collection.

A. Using LLM for Summarizing Failed SQL

After applying constraints on event selection for contextualizing the failures, the sizes of these sets range from 1 to 3,314 events, with an average of 45 failed events across 25,453 instances. Vectorizing the collected statements with mechanisms like *fasttext* is feasible as we employ that solution for Error Messages and SQL statements in MIRA, but the extensive contextual data necessitates intricate preprocessing steps, training, and retraining models. However, summarizing statements using a pre-trained LLM like GPT-4 [14] can help reduce noise and retain only critical parts. Furthermore, LLMs are multimodal and can process diverse input types effectively [15], [16] allowing a single input with stack traces, SQL Statements, Error Messages, etc.

Summarizing and vectorizing SQL statements to obtain representations of the workload for machine learning has been an ongoing effort [17], [9]. However, in our case, we aim to generate a comprehensive yet succinct summary of a selected set of SQL statements and their corresponding failures to contextualize a failed event in a replay. The summaries provide an overview of relevant events leading up to a failed event in consideration. The frequency and complexity of these events play a limited role in emulating the operators' assessment process. Therefore summarizing enables us to extract the salient information necessary to to contextualize a replay failure. Figure 2 shows a prompt input and the response generated by the LLM. To achieve this, we disregard the temporal ordering of the SQL statements within this group while prompting the LLM. This approach enables us to obtain a collective summary of events that exhibit similar failures. As part of the summary, we prompt for inclusion of the list of objects. However, the LLM tends to often truncate the list of objects for brevity, particularly when there is a large number of similar failures on similar statement types. Initially, we allowed unrestricted length for summarizing Error Messages, resulting in noise such as irrelevant numbers, and texts like thread ID and session name being retained. Since they typically pertain to transaction details or specific executions, they do not contribute to identifying failure root causes. To address this, we implemented a 30-word limit, aiming to focus the generation of summaries on the most crucial parts. our choice of models is limited by the context lengths of the LLMs. We tested four models: llama-7b [18], GPT-3.5, GPT-4, and GPT-4-Turbo. While GPT-4 provided the best summaries for instances within a 32,000 token context length, GPT-4-Turbo could handle more instances with its 128,000 token context length. The llama models struggled to generate complete summaries and adhere to the desired summary structures. Therefore, we use the GPT-4-Turbo model to summarize collected statements. Prior to summarization, we preprocess the events to remove data like names, IP addresses, hostnames, etc., to anonymize the information.

```
Prompt:

Given a list of JSON objects containing SAP HANA SQL statement
strings, error messages, and skip reasons, generate a concise
summary.  Group the SQL statements based on SQL type and
execution status. Summarize each group with a parseable JSON
structure:
[
"statement type": SQL statement type,
"status": failed/ skipped
"error": generic summary of all 'critical' sub-parts of unique
error message in 30 words or less.
"objects": comma-separated list of all objects
]
Summarize 'identical' failures within the list similarly.
Strictly adhere to the summary structure and include absolutely
no additional information outside the JSON.

Input List of JSON:

[{'Statement String': 'CALL proc_name ...',
  'Error Code': 100,
  'Error Message': 'Transaction rolled back ....',
  'Skip Reason': 'Skipped'}, {....}, {....}
]
```

```
Response Summary:
[
  {
    "statement type": "CALL",
    "status": "failed",
    "error": "Operation cancelled and transaction rolled back
              due to exception.",
    "objects": "ABC1, ABC2, ABC3, ABC4"
  },
{
    "statement type": "CREATE VIEW",
    "status": "failed",
    "error": "Connection error"
    "objects": "MNO1, MNO2, MNO3, MNO4"
  }
]
```

Figure 2: Example Prompt and Response

B. Building New Training Data

Capture and replay-based testing yields large result sets, which are retained temporarily, due to storage constraints and cost considerations, for assessment by operators. MIRA's training data used in the production setup has evolved with the identification of new root causes over several years. Since original result sets from most of those replays were lost, acquiring contextualizing data for events in training data used in current production setup is not possible.

We have collected a new labeled dataset over a span of 1 year. We collect the failed events from replays and leverage the labels provided by MIRA's production setup as portrayed in Figure 3. We collect the labels for instances that have been reclassified by an operator which is an indication of manual verification or the predictions with high certainty. These conditions already limit the amount of reliable data we can collect per replay. Additionally, there can be several instances of a single type of failure within a replay. Therefore, we limit the samples per failure category per replay to a maximum of 100 samples. This limitation helps ensure the diversity of the dataset. To construct the final training dataset, we retain the same set of features as used in the productive setup, with the exception of the textual attribute. Previously, this attribute comprised the concatenation

of Error Message and SQL Statement String. Now, it will be enhanced by incorporating the response summary for each event. The augmented textual attribute is then embedded using *fasttext* to build the new training dataset.

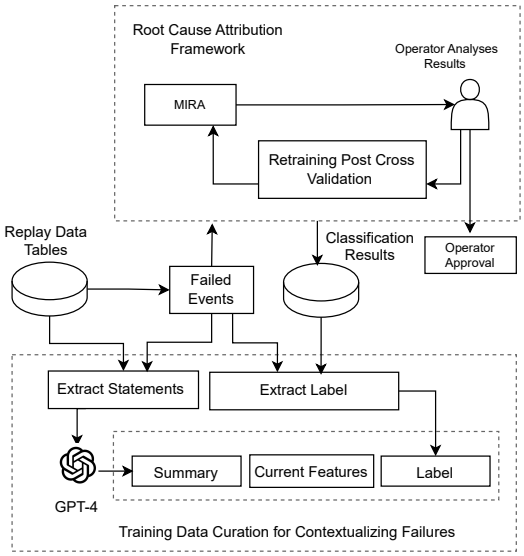

Figure 3: Constructing New Training Data

IV. Evaluation and Discussion

The final dataset, after deduplication, comprises 25,453 events and 162 classes of the 233 classes from production. There is a significant class imbalance as the top 10 root causes, by count, constitute 65% of the data. Also, several classes have only 1 sample which cannot be split uniformly across the test train split during the cross-validation despite the stratification. We perform a 5-fold stratified cross-validation [19] for hyper-parameter optimization. For classifier evaluation, we present the metrics F1-Macro, F1-Comb, and Accuracy averaged across 5-fold stratified cross-validation. F1-Macro is the average F1-score of all the classes and is a suitable metric for unbalanced datasets. F1-Comb is a combined metric of F1-Macro score and certainty metrics [1] and is calculated as a harmonic mean of both. Higher F1-Comb values indicate higher certainty for correct predictions, indicating the reliability of predictions. As stated in Section III we built a new dataset that is different than the dataset the model in production is trained on. We adopt a similar setting as our production environment with *fasttext* to vectorize the textual attributes and *KNN* as the classifier due to its interpretability. The Table IV presents an overview of the evaluation of concatenating the Summaries to the Error Messages and the SQL Statement strings. The F1-Macro improves by 4.77% with the use of SQL summary and accuracy by 0.16%.

We preprocess the summaries independently prior to the concatenation of textual attributes, to retain the most informative terms using TFIDF-based term filtering. This makes the summaries further concise. Upon closer analysis, we could not find a discernible pattern to textual features in the misclassifications. However, the misclassifications largely arose from the minority classes and it is challenging to conclusively understand the issues with only a few samples. In this data, Error Messages were consistently the most important attribute for classifying root cause. To understand the impact of the summaries, we concatenated them to Error Messages and the results are comparable to the current setting of using SQL Statements and Error Messages. This makes summarized failed SQL statements a valuable feature for our classification process. Using a t-SNE plot [20], Figure 4 visualizes the change in the *fasttext* embeddings for the 10 classes with the highest frequency of failures from the *problem group*. Summaries in concatenation with the Error Messages and SQL Statements create more cohesive clusters in contrast to the plot without the summaries.

Table IV: Average 5-fold cross-validation scores on new training data - Summary: Summarized failed SQL statements, EM: Error Message, SS: Statement String

Textual Attribute	F1-Comb	F1-Macro	Accuracy
EM + SS	78.82	63.02	91.86
EM + SS + Summary	**79.79**	**67.79**	**92.02**
EM + Summary	75.88	63.40	90.33

Our data encoding involves concatenating all textual features, which may limit the impact of one feature over another. However, treating summaries as a separate feature did not significantly change the results. While the use of summaries enhances F1-Macro by 4.77%, it is encouraging, as this appears to be a viable feature engineering method for this data. Further data collection and experimentation with summary structure and granularity could be beneficial to find more robust features.

GPT-4-Turbo may not have all the required domain knowledge of HANA failures to effectively summarize the failures. Finetuning or instruction tuning can be a possible solution to make the model better adapt to the domain. Additionally, the current constraints on data collection for contextualization presume that the root cause is present in the same session as the failed event, which may not always be accurate. Even if the LLM effectively summarizes the failed SQL statements, the necessary information for identifying the root cause might not be available in the contextual history. Furthermore, utilizing stack traces alongside failed SQL when applicable can result in a summary more informative of a root cause. Thus, an ongoing evaluation of replay failures remains essential.

V. Limitations and Improvements

Despite the applied filters, 18% of total data instances exceed 128,000 token context length as well. Addressing this challenge may need chained summarization [21]. Additionally, prompting strategies greatly influence LLM results [22]. We manually validated LLM-generated summaries for 30 samples with known contextual information and automated the summarization for the remaining dataset. An overall

(a) Before (b) After

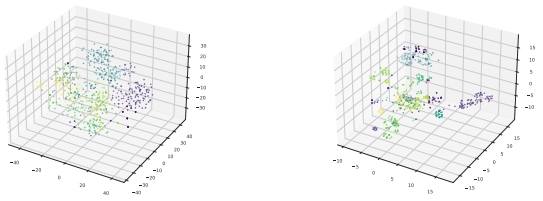

Figure 4: t-SNE Plot - Utilizing SQL Summary

understanding of generated summaries may help craft better prompts to obtain more informative summaries. Predicting future error patterns is challenging, but excluding known failures to identify unknown ones may be beneficial. We plan to test an approach by crafting precise prompts to flag unknown cases and eliminate known failures. This requires a few-shot-learning prompt structure to introduce known failures as examples into the prompt and identify a potential mismatch.

The JSON structure of the failure summary allows for experimentation with subsets. For example, objects from a DML SQL query may be irrelevant to later failures and can be omitted during summarization. However, validating parts of the summary requires extensive testing. Skipping the JSON structure frees up tokens for more informative summaries.

We construct a new dataset for evaluating our approach, using MIRA's classification results as the new ground truth for training. However, a system that continuously adapts using manually reclassified data and employs uncertainty measures to trigger such reclassifications is susceptible to performance degradation. A thorough analysis of the new training data is essential for evaluating label validity and our results. Data collection for summarization still relies on rules needing ongoing maintenance and adaptation as new issues arise.

Operators use MIRA's UI for root cause analysis but manually examine replay data for exploring new failures. Integrating condensed SQL summaries into the UI can provide immediate insights and enhance analysis.

VI. Conclusion

In this study, we address challenges arising from data evolution in our root cause analysis framework using supervised learning for replay failure analysis. As we encounter new failures, the efficacy of the machine learning framework depends upon the availability of robust data for ensuring reliable classification. We outline our efforts to improve the classification and address complexities from overlapping features in failures with different root causes. Moreover, as a long-term solution, we design an approach attempting to emulate the manual root cause analysis processes of domain experts. Leveraging LLM, we summarize a set of failed SQL statements, creating a feature that enhances our data with a potentially discriminating attribute for root cause analysis. This method improved the results by 4.77% and the experiments enabled the development of a valuable feature for classification. The failure summaries can enhance the model interpretability and facilitate a more efficient root cause analysis process for operators.

References

[1] N. Jambigi, T. Bach, F. Schabernack, and M. Felderer, "Automatic error classification and root cause determination while replaying recorded workload data at sap hana," in *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2022, pp. 323–333.

[2] T. Bach, A. Andrzejak, C. Seo, C. Bierstedt, C. Lemke, D. Ritter, D. W. Hwang, E. Sheshi, F. Schabernack, F. Renkes *et al.*, "Testing very large database management systems: The case of sap hana," *Datenbank-Spektrum*, vol. 22, no. 3, pp. 195–215, 2022.

[3] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees, "The SAP HANA database – an architecture overview," *Bulletin of the Technical Committee on Data Engineering / IEEE Computer Society*, vol. 35, no. 1, pp. 28–33, 2012.

[4] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "SAP HANA database: Data management for modern business applications," *SIGMOD Record*, vol. 40, no. 4, Jan. 2012.

[5] G. Salton and M. J. McGill, *Introduction to modern information retrieval*. USA: McGraw-Hill, Inc., 1986.

[6] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*. PMLR, 2014, pp. 1188–1196.

[7] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[8] C. Tang, B. Wang, Z. Luo, H. Wu, S. Dasan, M. Fu, Y. Li, M. Ghosh, R. Kabra, N. K. Navadiya, D. Cheng, F. Dai, V. Channapattan, and P. Mishra, "Forecasting sql query cost at twitter," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*, 2021, pp. 154–160.

[9] S. Jain, B. Howe, J. Yan, and T. Cruanes, "Query2vec: An evaluation of nlp techniques for generalized workload analytics," *arXiv preprint arXiv:1801.05613*, 2018.

[10] Y. Li and B. Zhang, "Detection of sql injection attacks based on improved tfidf algorithm," in *Journal of Physics: Conference Series*, vol. 1395, no. 1. IOP Publishing, 2019, p. 012013.

[11] J. Zahir and A. El Qadi, "A recommendation system for execution plans using machine learning," *Mathematical and Computational Applications*, vol. 21, no. 2, p. 23, 2016.

[12] A. Mucherino, P. J. Papajorgji, and P. M. Pardalos, "K-nearest neighbor classification," in *Data mining in agriculture*. Springer, 2009, pp. 83–106.

[13] A. Agresti, *An introduction to categorical data analysis*. John Wiley & Sons, 2018.

[14] OpenAI, "GPT-4 technical report," *ArXiv*, vol. abs/2303.08774, 2023. [Online]. Available: https://arxiv.org/abs/2303.08774

[15] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[16] H. Jin, Y. Zhang, D. Meng, J. Wang, and J. Tan, "A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods," *arXiv preprint arXiv:2403.02901*, 2024.

[17] S. Deep, A. Gruenheid, P. Koutris, S. Viglas, and J. Naughton, "Comprehensive and efficient workload summarization," *Datenbank-Spektrum*, vol. 22, no. 3, pp. 249–256, 2022.

[18] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[19] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

[20] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[21] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[22] L. Li, Y. Zhang, and L. Chen, "Prompt distillation for efficient llm-based recommendation," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023, pp. 1348–1357.