

XPert: Empowering Incident Management with Query Recommendations via Large Language Models

Yuxuan Jiang*
University of Michigan
Ann-Arbor
USA

Chaoyun Zhang†
Microsoft
China

Shilin He
Microsoft
China

Zhihao Yang*
Peking University
China

Minghua Ma
Microsoft
China

Si Qin
Microsoft
China

Yu Kang
Microsoft
China

Yingnong Dang
Microsoft
China

Saravan Rajmohan
Microsoft
China

Qingwei Lin
Microsoft
China

Dongmei Zhang
Microsoft
China

ABSTRACT

Large-scale cloud systems play a pivotal role in modern IT infrastructure. However, incidents occurring within these systems can lead to service disruptions and adversely affect user experience. To swiftly resolve such incidents, on-call engineers depend on crafting domain-specific language (DSL) queries to analyze telemetry data. However, writing these queries can be challenging and time-consuming. This paper presents a thorough empirical study on the utilization of queries of KQL, a DSL employed for incident management in a large-scale cloud management system at MICROSOFT. The findings obtained underscore the importance and viability of KQL queries recommendation to enhance incident management.

Building upon these valuable insights, we introduce XPert, an end-to-end machine learning framework that automates KQL recommendation process. By leveraging historical incident data and large language models, XPert generates customized KQL queries tailored to new incidents. Furthermore, XPert incorporates a novel performance metric called Xcore, enabling a thorough evaluation of query quality from three comprehensive perspectives. We conduct extensive evaluations of XPert, demonstrating its effectiveness in offline settings. Notably, we deploy XPert in the real production environment of a large-scale incident management system in MICROSOFT, validating its efficiency in supporting incident management. To the best of our knowledge, this paper represents the first empirical study of its kind, and XPert stands as a pioneering DSL query recommendation framework designed for incident management.

1 INTRODUCTION

Large-scale cloud systems, such as AWS, Azure, and Google Cloud, are indispensable pillars of modern IT infrastructure [1]. These platforms cater to a diverse range of online products and services, attracting a substantial global user base and generating significant revenue. In this context, ensuring the reliability of these cloud services becomes of utmost importance, as it directly impacts revenue generation and customer satisfaction [2]. Despite considerable efforts invested in constructing robust systems, incidents continue

to be a prevalent issue in cloud infrastructures [3]. Such incidents lead to service disruptions and have a detrimental impact on the overall user experience [4].

In the event of an incident, on-call engineers (OCEs) diligently adhere to established procedures to swiftly identify the potential root cause and initiate prompt mitigation or resolution efforts [5, 6]. To accomplish this objective, OCEs heavily rely on the analysis of telemetry data, encompassing vital runtime information such as logs [7], time series [8–11] and traces [12–14]. These data hold a paramount significance in software systems and are consistently recorded and stored in databases throughout the service’s operation. Similar to formulating standard database queries, OCEs manually compose domain-specific language (DSL) queries to extract the necessary telemetry data and thoroughly investigate the circumstances surrounding the incidents, enabling them to conduct effective triage and expedite the implementation of appropriate mitigation actions.

Writing appropriate queries for incident management is however a challenging task, as it necessitates considerable domain expertise to accurately select the appropriate databases, tables, and columns, and subsequently construct a query that incorporates various operations such as join, count, aggregation, *etc.*. This process can be time-consuming and requires careful attention to detail. As a common practice, engineers often resort to finding suitable queries in existing troubleshooting guides (TSGs) to expedite the process [15]. However, this approach relies heavily on the documentation ability of engineering teams, and OCEs may struggle to find the relevant queries if the TSGs are poorly organized or non-existent. Furthermore, queries can become highly complex, particularly when they span multiple databases. Even a minor mistake in the query can result in significant discrepancies in the retrieved results. Providing a useful query to OCEs can therefore significantly reduce their effort and expedite the incident mitigation process.

In this paper, we present a comprehensive empirical study on Kusto Query Language (KQL) queries utilized for incident management in a world-wide cloud computing company MICROSOFT. Our study delves into the frequency, complexity, and diversity aspects of these KQL queries. Key findings from our investigation reveal the following: (i) the majority of incidents can be managed effectively with a small number of KQL queries; (ii) Most of KQL queries used

*This work was completed during their internship at Microsoft Research Asia.

† Corresponding author.

in incidents tend to be relatively simple in structure; and (iii) KQL queries exhibit long-tail pattern in templates and significant time variation. These insights underscore the necessity and practicality of automating KQL recommendation for OCEs to streamline the incident management.

Based on the insights gained from our empirical study, we introduce XPert, an end-to-end framework designed to empower the incident management process by automatically recommending or generating KQL queries. Drawing from the abundant historical incidents and their corresponding KQL records in the past two years, XPert provides customized KQL recommendations based on the specific context of new incidents. The framework efficiently extracts common patterns, such as tables and templates, from historical similar incidents, facilitating effective automation in XPert. To address the limitations of traditional natural language processing (NLP) metrics [16] in evaluating domain specific queries, XPert incorporates a novel performance metric called Xcore. This tailored metric allows for more comprehensive evaluation from three different perspectives, enhancing the overall quality assessment of the generated KQL queries.

To mitigate the challenges posed by costly pre-training, fine-tuning, and frequent updates of conventional NLP models, XPert leverages the exceptional few-shot learning capabilities of Large Language Models (LLMs) to generate incident-specific KQL queries with only a few examples provided, without the need for parameter tuning. LLMs have demonstrated remarkable proficiency in parsing complex data [17], extracting essential information [18], and producing concise, insightful outputs in both natural language [19, 20] and code [21] domains. This makes them well-suited for the context of XPert, where incident descriptions are often intricate and unstructured, posing challenges for traditional smaller language models. Moreover, the few-shot learning ability of LLMs in specific domains allows them to quickly adapt to novel and evolving incident types by leveraging historical data in an online fashion. This adaptability significantly enhances the quality of the generated KQL queries, rendering LLMs an ideal solution for this task.

We thoroughly evaluate XPert, deploying it as a KQL recommendation framework, which serves as a pivotal component within the incident management system at MICROSOFT. Experiments show the effectiveness of XPert from both offline and online viewpoints. In summary, this paper presents the following contributions:

- A comprehensive empirical study on the utilization of KQLs in incident management, revealing interesting insights that inspire the KQL recommendation.
- Development of XPert, an end-to-end KQL recommendation framework that leverages the few-shot learning capabilities of LLMs to empower automated KQL generation.
- Introduction of Xcore, an evaluation metric tailored to assessing the quality of generated KQL queries from various perspectives, addressing the limitations of traditional NLP metrics.
- Extensive offline evaluation of XPert using a large-scale dataset from a real incident management system, showcasing its superior KQL quality compared to several strong baselines.
- Successful deployment of XPert as a critical KQL recommendation framework within XPert’s incident management system, with pilot results demonstrating its exceptional performance.

To the best of our knowledge, this paper is the first to present an empirical study on the characteristics of DSL queries in incident management, and XPert stands as the pioneering KQL recommendation framework specifically tailored for incident management.

2 BACKGROUND

This section provides an overview of incident management in the context of cloud computing, with a particular focus on the utilization of domain-specific language, KQL queries.

2.1 Incident Management in Cloud

The increasing popularity of cloud systems in recent years can be attributed to their inherent advantages, including scalability, accessibility, and cost-effectiveness [5, 22]. However, unplanned disruptions in cloud services, commonly referred to as incidents, remains a frequent phenomenon within cloud infrastructures [23, 24]. To address incidents, OCEs typically rely on an incident management system, which involves various measures such as executing DSL queries, analyzing logs, and discussing with other engineers [25, 26].

Taking the incident management system of MICROSOFT as an example, when an incident is detected, a ticket is created in the system, with a title and summary provided by engineers manually or monitors automatically to describe the incident’s context. During the incident management process, OCEs frequently compose and execute KQL queries on the service telemetry to comprehend the incident, identify the affected scope, and diagnose the underlying cause [7, 27]. Typical telemetry data such as traces and logs related to the incidents may then be extracted and analyzed to aid in the diagnosis and mitigation processes [14, 28, 29]. These efforts yield valuable insights that greatly contribute to the triage process and subsequent mitigation actions [30].

2.2 KQL and KQL Queries

A domain-specific language (DSL) [31] denotes a language that is specifically crafted for a distinct domain and customized to cater to specific tasks, industries, or areas of expertise. Illustrative examples of DSL queries encompass SQL for managing databases [32], GraphQL for querying graph databases [33], PromQL employed for querying metrics in Prometheus monitoring system [34], Search Processing Language (SPL) in Splunk [35].

Kusto Query Language (KQL) is a DSL developed by MICROSOFT, and it has been widely adopted within the organization. The language is designed to work with schema entities arranged in a hierarchy similar to SQL, comprising databases, tables, and columns. Behind the scenes, a big data analytics cloud service is optimized to handle KQL queries over various types of data, including structured, semi-structured, and unstructured data. For example, the services team regularly stores telemetry data such as traces, logs, and metrics in KQL databases, facilitating future queries for diagnostic and analytical purposes. By utilizing KQL, engineers can effectively explore the data, uncover patterns, identify anomalies, and perform other important tasks during the incident management process.

In Fig. 1, we present an example of a KQL query. A typical KQL query comprises a primary *table name*, which signifies the main data source for the query. It may also include filter operators (e.g., “where”), selection operators (e.g., “take”), and join/union operators

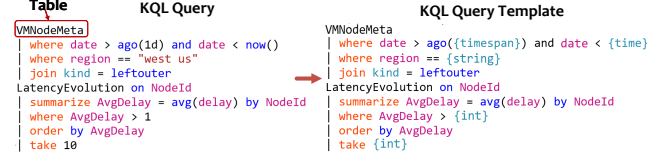


Figure 1: An example of a KQL query and its template.

(e.g., “join”). In essence, a query template can be derived from a KQL query by replacing the actual values in the query with placeholders. The resulting template is depicted on the right side of Fig. 1. These placeholders represent data types and query templates can be reused in different incident tickets. As a result, both the template and the full KQL query provide valuable information to OCEs for efficient incident management.

2.3 System Objective

The primary objective of XPERT is to recommend KQL queries to OCEs within incident management systems, utilizing the available incident context and information. This recommendation process aims to facilitate various aspects of incident management, including triage [36] and diagnosis [37], with the ultimate goal of reducing the time to mitigate (TTM) [23] for incidents. XPERT offers two hierarchical levels of KQL query recommendation: (i) **Template Recommendation**, which involves suggesting a query template (see Fig. 1 right), that is the skeleton of queries for OCEs to fill in more concrete values. (ii) **Query Recommendation**, refers to predicting the full KQL query, with all values in the query filled. The recommended template and query are both submitted to the incident management system and presented to OCEs.

The rationale behind this hierarchical approach lies in the understanding that incident context may not always provide sufficient information for all values and fields in the full query. Some of these details may require domain-specific knowledge possessed by OCEs. By leaving certain fields as placeholders in the template recommendations, OCEs can complete the missing information based on their expertise. However, it is important to emphasize that template recommendations still provide valuable insights to OCEs, thereby enhancing the incident management process.

3 EMPIRICAL STUDY

To better understand KQL in the context of incident management, we conducted a large-scale empirical study using incident tickets from the incident management system at MICROSOFT. Our objective was to analyze the characteristics of KQL queries and gain insights to guide the design of XPERT. For this purpose, we collected 346,508 incident tickets that were categorized into four severity levels (0-3, from highest to lowest) from the top-30 services with the highest number of incidents, spanning from January 2021 to November 2022. These incidents represent over 60% of total incidents and resulted in a dataset of 712,222 KQL queries. Note that only incident tickets that contain at least one KQL query are included. Specifically, we aimed to answer the following research questions (RQs):

- RQ1: How frequently are KQL queries used by OCEs?
- RQ2: How complex are the KQL queries used by OCEs?
- RQ3: How diverse are the KQL queries employed?

The following subsections present our findings to these RQs.

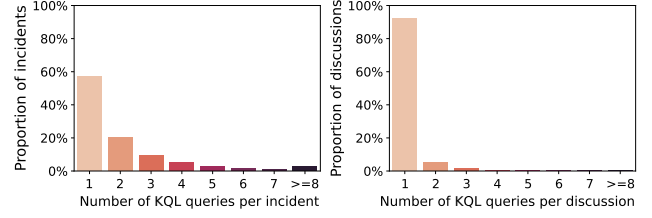


Figure 2: Distributions of KQL query number per incident/discussion.

3.1 RQ1: Frequency of KQL Queries

In the incident management platform, human-written KQL queries are typically posted in the “discussion” section of an incident ticket. The ticket may contain multiple discussions, and within each discussion, multiple KQL queries can be included. Fig. 2 displays the distributions of the number of KQL queries per incident and per discussion across all incidents. Upon observation, it is evident that over half of the incidents have only one KQL query, and over 90% of the discussions within an incident contain just one query. These findings indicate that a concise set of well-targeted queries is often sufficient to manage various incidents effectively. Given these findings, we set the primary objective of XPERT to recommend the initial KQL query when a new incident is created, as this is sufficient to effectively address the majority of incident tickets.

Table 1: Query reaction time statistics of incidents.

Severity	Mean	Median	P90	P95	P99
0	553	130	45	30	28
1	992	114	15	10	3
2	801	49	6	4	2
3	4,563	839	22	10	2
Overall	3,197	216	11	6	2

Table 1 presents the statistical analysis of the time duration between the occurrence of the first KQL query and the incident’s creation time, referred to as “query reaction time”. This analysis is further categorized based on incident severity levels. Note that time unit in the table are normalized for anonymity. The query reaction time reflects the duration within which an OCE writes a KQL query in response to a reported incident. Notably, for high-severity incidents (severity levels 0-2), the query reaction time is significantly shorter, indicating that OCEs respond much more promptly to these critical and impactful incidents. This observation aligns with our expectations, as high-severity incidents demand swift attention and action. The faster response times for high-severity incidents underscore the importance of timely intervention in critical situations, which can be supported by efficient KQL recommendations offered by XPERT. Additionally, we observe that 95% of incidents have a query reaction time greater than 6 units. This finding is valuable for informing the design of the time window of input that we include in XPERT, and its detailed design will be discussed in Sec. 4.2.1.

Takeaways 1: The number of queries required for effective incident resolution is small. The query reaction time for high-severity incidents is significantly shorter compared to lower-severity incidents.

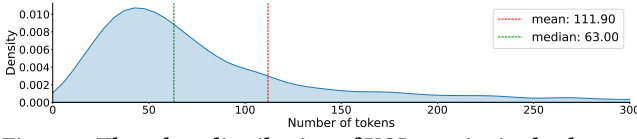


Figure 3: The token distribution of KQL queries in the dataset.

3.2 RQ2: Complexity of KQL Queries

We explore the complexity of KQL queries that OCEs write for incident management. The complexity of queries often signifies the level of filtering (e.g., “where” operator) and the number of data sources involved, making it a valuable metric for gauging the comprehension and understanding of the incident. To quantify the query complexity, we evaluate the number of tokens in the query, which provides a measure of the overall query length.

Fig. 3 presents the distribution of tokens in KQL queries within the studied dataset. The analysis reveals that the token count follows a long-tail distribution, with the majority of queries being relatively short (less than 63 tokens). This aligns with our expectations, since during an incident, OCEs may have limited time for in-depth understanding, leading to the formulation of shorter queries. Consequently, this characteristic makes the task of query recommendation more feasible, as shorter queries are generally less diverse and easier to generate compared to more complex ones.

This findings reinforce the fundamental objective of XPert. It highlights that in general, XPert does not require the generation of overly complicated queries for most incidents, as the majority of incidents can be effectively managed with relatively concise and straightforward queries. This observation aligns with the design philosophy of XPert, which aims to recommend KQL queries that are concise yet effective in addressing incident management tasks.

Takeaways 2: The majority of incidents can be effectively managed using relatively concise and straightforward queries, rendering the query recommendation task more feasible.

3.3 RQ3: Diversity of KQL queries

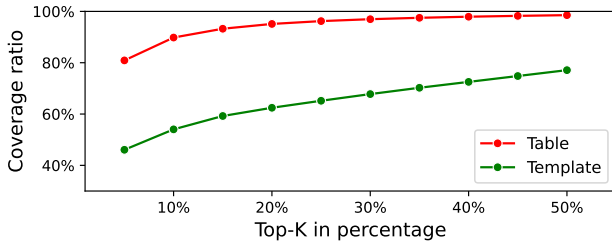


Figure 4: Query coverage ratio w.r.t. top-K tables/templates in percentage.

Lastly, we turn our attention to investigating the diversity of KQL queries for incident management. Fig. 4 displays the coverage ratio with respect to the top-k percentage in terms of tables and templates. The results show that the top 5% of tables used by most queries already cover 80.9% of the queries, while the top 5% of templates can cover 46.1% of the queries. This long-tail pattern further supports the notion of low diversity in employed KQL queries, as common patterns are widely shared among them. This finding enhances the feasibility of the query recommendation goal, as these shared

patterns can be effectively extracted by the LLMs. In addition, our analysis reveals an interesting finding regarding the sharing of query templates across different services. It is observed that query templates are rarely shared between services, with the percentage of shared templates being lower than 4.75%. This low percentage indicates a high level of isolation between services, meaning that each service tends to have its own unique set of query templates specific to its incident management requirements. This finding holds important implications for the design of the hierarchical data retrieval approach in XPert to limit the retrieval to incidents within the same service. We will provide details on this design in Sec. 4.3.1.

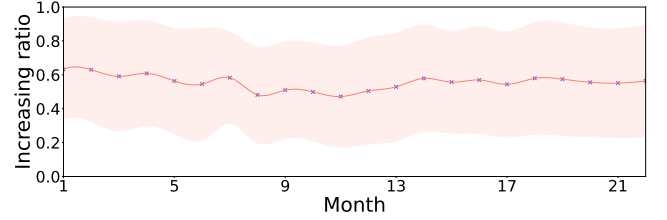


Figure 5: The mean±std. of the monthly ratio of KQL queries covered by novel templates across all services.

Finally, we present the mean±std. of the monthly ratio of KQL queries covered by novel templates across all services, as depicted in Fig. 5. This metric measures the proportion of queries that are associated with previously unseen or unique templates within a given month, offering insights into the degree of change and data drift observed in KQL queries over time. We observe that KQL queries exhibit significant variation on a monthly basis, with an average of over 60% of KQL queries utilizing different templates compared to the previous months, even within the same service. This finding highlights the dynamic and evolving nature of incidents, and underscores the need for adaptable and tailored KQL queries for incident management. Consequently, we have designed XPert to accommodate such time-varying patterns in an online manner, as elaborated in Sec. 7.

Takeaways 3: (i) KQL queries exhibit low diversity in terms of both tables and templates; (ii) templates used in different service are rarely shared across each others; (iii) KQL queries exhibit significant variation and data drift on a monthly basis.

Summary. In summary, Takeaway 1 elucidates the primary objectives behind the design of our KQL query recommendation system XPert. It underscores the importance of targeting the initial query and template of KQL queries for incidents, as these encompass a substantial portion of incident tickets. Furthermore, Takeaways 2 and 3 provide valuable insights into the relatively modest complexity and diversity observed in the KQL queries employed for incident management. These insights underscore the practicality of automating the recommendation of KQL queries. Subsequently, we introduce the architecture and design principles of XPert, which are founded upon the discoveries derived from our empirical study.

4 THE DESIGN OF XPert

We provide an overview of the architecture of XPert in Section 4.1, and delve into its components in the following subsections.

4.1 XPert in a Nutshell

Fig. 6 presents an overview of the XPert framework. Upon receiving a new incident ticket in the incident management system, the data processor gathers relevant incident context and preprocesses it to a format compatible with the language model. Subsequently, an embedding model is employed to vectorize the incident context and conduct a search for similar historical incidents along with their corresponding KQL queries. By combining these retrieved samples with the target incident context in a prompt sequence, XPert feeds this input into the LLM to generate an initial KQL template and query simultaneously.

The generated KQL template and query then undergoes a post-validation process to verify its adherence to correct grammar. Valid query is presented to the OCEs for recommendation, while invalid query undergoes necessary rectification to ensure its correctness before being forwarded to the OCEs. At last, the true queries crafted by OCE based on our recommendations are promptly added to the vector database to keep it up-to-date. We also designed a dedicated Xcore to evaluate the quality of the recommended query, which we will elaborate in Sec. 5

4.2 Incident Data Processor

The incident data processor gathers comprehensive information from the incident ticket and performs appropriate pre-processing to optimize the utilization of this data, as elaborated below.

4.2.1 Information Collection. To equip the LLM with sufficient information for effective query recommendation, XPert employs a comprehensive approach in collecting rich incident data from various resources within the incident management system [38]. These resources encompass: (i) **Metadata**, which entails fundamental incident details such as the creation time, the service which triggers the incident, and other essential information. (ii) **Title** of the incident, which may be system-generated or written by an engineer. (iii) **Summary** of the incident, serving as a high-level overview either generated by the monitoring system or written by an engineer. (iv) **Discussion** pertaining to the incident, encompassing system logs related to the incident as well as discussions among the engineers. It is important to note that the discussions included in the incident context are confined to a time window of 5 time units from the creation time of the incident, ensuring the timeliness of the query recommendation. The selection of this 5-unit time window is based on the findings presented in Table 1, which indicate that it covers at least 95% of the incidents before the initial KQL query is posted. By incorporating these diverse contextual elements, XPert maximizes the information available for the LLM, while upholding the timeliness of the query recommendation.

4.2.2 Data Pre-processing. Upon collecting all information from the incident tickets, the data is concatenated into a text sequence. XPert performs two pre-processing steps on the incident context: (i) Repetitive information that appears multiple times in the context is removed. (ii) If the incident context exceeds a certain token threshold, the sample is clipped to avoid over-length. This is necessary as the input of the LLM is subject to token limitations. This pre-processing ensures improved information utilization in the data while adhering to the LLM’s input constraints on token length.

4.3 KQL Query Recommendation

Pretrained Language Models, such as the GPT series [39] and LLaMA [40], are typically trained on vast amounts of general information from publicly available domains or the Internet. However, their training corpus often lacks specialization in certain domains, such as internal incident management data. Fine-tuning a LLM with domain-specific data can be extremely costly [41], and in some cases, it may be infeasible due to resource constraints or restrictions imposed by the model provider [39].

To address this limitation, we explore the few-shot learning capability of LLMs [42], which allows us to demonstrate a few examples in the prompt sequence provided to the LLM for prediction. This approach, known as in-context learning (ICL), has shown to be effective in various domain-specific scenarios [43]. By combining general knowledge from the LLM itself with context from specialized samples provided in the prompt sequence, ICL leverages both sources of information to make accurate predictions.

We therefore adopt a combination of LLM and ICL to generate KQL queries tailored for incident management. This approach utilizes the pre-processed incident context and similar historical incidents as demonstration and context for the LLM. The entire process encompasses three crucial steps, namely (i) similar incident retrieval, (ii) prompt construction; and (iii) DSL query generation. We provide details of each step in the following subsections.

4.3.1 Similar Incident Retrieval. The first crucial step in the ICL process is to retrieve incidents that are similar to the target incident, along with their corresponding queries. The ground truth queries are compiled when OCEs submit their initial query within the incident management system following the reception of an incident ticket.

This retrieval process involves utilizing an embedding model to vectorize historical incidents and the target incident, making them searchable using distance metrics. In this study, we utilize an embedding model [44], which encodes all incident contexts into 1,536-dimensional vectors. These vectors, along with their corresponding queries, are stored in a vector database called Faiss [45], enabling efficient data retrieval.

When a new incident request is received, its context is embedded using the same embedding model. We then retrieve the top- K similar incidents using the cosine similarity [46] metric. Since the retrieval process is performed exclusively by considering only incidents falling within the same service as the target incident request, it narrows the retrieval to a more relevant range since data sources and templates are rarely shared across services, as suggested in Sec. 3.3. Furthermore, this exclusive retrieval significantly reduces retrieval costs. Once the top- K similar incidents are obtained, we utilize them as context and construct the prompt sequence for the LLM, as detailed in the subsequent section.

4.3.2 Prompt Construction and Query Generation. Prompts as language sequence inputs to LLMs, serve as a means of interaction to accomplish specific tasks [47]. In the context of ICL, a typical prompt comprises three key elements, as illustrated in Fig. 7: (i) An instruction, to inform the LLM about the goal of the task and the rules that apply to the task. (ii) Retrieved similar incident samples

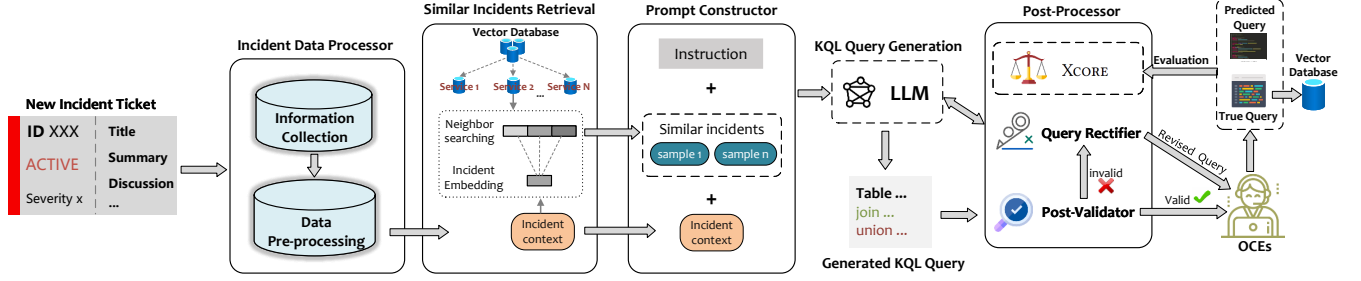


Figure 6: The overall architecture of XPert.

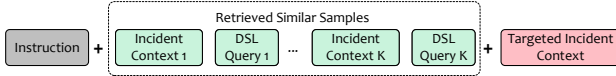


Figure 7: The structure of a prompt employed in XPert.

from history, including the incident context and their corresponding true queries for incident management. (iii) Targeted incident context, representing the incident context for which a KQL query needs to be recommended.

Note that the retrieved incident contexts undergo similar pre-processing steps as described in Sec. 4.2.2. The number of retrieved samples included in the prompt depends on the total token length of the prompt. We adopt a greedy approach to add as many samples as possible to the prompt, maximizing the context and information provided to the LLM [48], while ensuring that the number of tokens in the prompt does not exceed the LLM’s constraint (8k). The constructed prompts are directly fed into the LLM to facilitate KQL query recommendation through the OpenAI API. The API returns a raw query that has been generated based on the prompt input.

4.4 Post-Processor

To address the issue of potentially non-executable or grammatically incorrect KQL queries generated by LLMs, which may arise due to noise in retrieval data or mispredictions, we have integrated a post-processor into XPert. The post-processor plays a crucial role in checking the validity of generated queries and rectifying any issues whenever possible. It comprises two key components:

- **Post-Validator:** This component performs a grammar and syntax check on the query using the intrinsic compiler abstract syntax tree (AST) [49]. By analyzing the data flow of the query, it determines if the query is executable. If the query fails this check, it is passed on to the post-rectifier for revision.
- **Post-Rectifier:** The post-rectifier carries out a two-step revision process to rectify invalid queries. In the first step, it cleans extraneous tokens from the query, such as spacing and tabs that might have been mistakenly generated. If the query still remains invalid, the post-rectifier proceeds to the second step, where we provide the LLM with the incident context, retrieved examples, the invalid query, error messages from the post-validator, and select usage handbook of the KQL. We then prompt the LLM to attempt fixing the query, resolving more complex cases that cannot be addressed by simple token removal.

This post-processing mechanism ensures that the KQL queries generated by XPert are refined and enhanced to achieve executability

and grammatical correctness, minimizing the need for manual intervention by OCEs. Once an OCE provides a ground-truth query for an incident, it is promptly added to the vector database. This ensures that the query becomes available for retrieval by future incidents, allowing the system to capture any potential data drift, as discussed in Sec. 3.3.

5 EVALUATION WITH XCORE

Assessing the quality of generated KQL or other DSL queries poses a significant challenge, as effective metrics for such evaluations are lacking. Traditional NLP metrics, like BLEU [50] and METEOR [51], primarily focus on lexical similarity and do not take into account code executability and execution accuracy. On the other hand, recent code metrics, such as CodeBLEU [52], are designed to support various mainstream languages. However, these metrics need to be customized and tailored to the specific operators or DSL of a given language, rendering them less scalable. Moreover, evaluating the execution accuracy of KQL queries is a difficult task, further complicating the evaluation process, especially in the context of incident management. Execution results are sometimes not provided in the incident context and may not be easily reproducible. This limitation makes it challenging for other code metrics, such as Spider’s SQL evaluation metric [53] and HumanEval [54], which rely on execution accuracy, to be effectively applied for KQL evaluation. Consequently, the overall evaluation of KQL quality remains a challenge.

5.1 Design of XCORE

To address the limitations of current evaluation metrics, we introduce XCORE, an innovative assessment metric customized for evaluating the quality of generated queries, with a particular focus on KQL while also being adaptable to other DSLs. XCORE utilizes static code analysis [55] to parse KQL queries into distinct components using Abstract Syntax Trees (ASTs), and extract name reference nodes representing table columns. This parsing process enables us to perform a comprehensive evaluation of KQL queries from three key perspectives, namely (i) Syntax and Semantic Check, (ii) Sub-component Matching, (iii) Output-Schema Matching. We detail each component in the following subsections.

5.2 The Syntax and Semantic Check

To evaluate the executability of a query, we utilize a two-step process. First, we employ the built-in syntax checker of KQL to detect any syntactical errors that may exist in the generated query. This step is crucial as it ensures that the query conforms to the language’s grammar rules and structure. Next, the static analysis approach

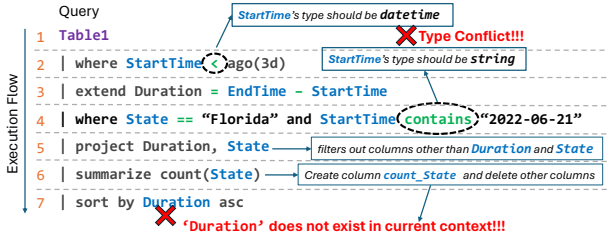


Figure 8: A semantic check example on an KQL query.

allows to approximately infer the semantic correctness of the query, by inferring the data types of these columns based on the semantics of the associated operations in the query. This analysis allows us to determine whether the query is semantically sound and executable.

Fig. 8 illustrates an example of the semantic check performed on an KQL query. In line 2 of the example, the type of `StartTime` is inferred as `datetime` based on the operand `ago(3d)`. However, in line 4, the query calls a `contains` operation, which only applies to string data types. As a result, this operand becomes inexecutable due to a type mismatch. Additionally, in line 6, the `summarize` operation drops the `Duration` column, as it performs a “groupby” operation. However, in line 7, the `Duration` column is called again, which makes the query invalid. Consequently, the semantic check assigns a zero score to this check, as the query contains invalid operations and is not executable. By conducting such semantic checks, we aim to estimate the validity of KQL queries, highlighting any issues that may arise during query execution.

It is important to note that this checker primarily assesses the query at the validity level, meaning that queries passing this checker are likely to be free of grammar mistakes. However, it is possible that queries passing this check may still encounter runtime errors during actual execution. Despite this limitation, the checker provides a valuable approximation of the query’s validity and ensures that generated queries are less prone to syntax errors.

5.3 Sub-component Matching

Leveraging the parsing results from the static analysis, we employ the sub-component matching technique to assess the lexical similarity between the generated queries and the ground truths. Following the design of Spider [53], we extract the operands of tabular components from both the ground truth and generated queries. For instance, for the code where `colA > 20`, we extract the sets (`colA`, where, `>`, `20`) as representations of the operands of different tabular components. To measure the similarity, we compute the F1 score by comparing the sets of operands for both the ground truth and generated queries. The sub-component matching approach offers the advantage of considering operators that do not have strict order requirements [56]. This means that changing the order of certain sub-components, such as filters, in a query may not necessarily alter the execution results. Therefore, the sub-component matching technique provides a more flexible and comprehensive evaluation of lexical similarity between generated queries and ground truths.

5.4 Output-Schema Matching

The evaluation of execution accuracy includes a crucial step of assessing the output data schema, which comprises essential information such as column names, data source names, and output

types (e.g., tables or charts). This information plays a pivotal role in aiding engineers during incident troubleshooting, providing insights into aspects like impact start time and machine IDs. For instance, consider a scenario where valuable information pertains to the response delay field, which should be included in the query output. However, the final recommended query lacks this specific column, despite a high degree of similarity between the query code and the ground truth. The absence of such crucial information in the output diminishes the usefulness of the recommended query. Consequently, it becomes imperative to evaluate the correctness of the output-schema to address this issue.

To infer the output schema for both the generated and ground truth queries, we employ a similar data-flow analysis, as depicted in Fig. 8. Once the output schema is obtained from the static code analysis, we parse it into a list of columns, a data source, and the return type. The final assessment of the output-schema matching is conducted in a two-fold manner: First, we use the F1 score to evaluate the similarity of output columns between the generated queries and the ground truth. Second, we employ binary accuracy to assess the correctness of the data source and the return types. This output-schema matching offers a different perspective on the information that should be returned to provide valuable insights, whose accuracy is particularly crucial in the context of incident management.

5.5 Summarizing the Final Xcore

Finally, we combine the results of the three evaluation components to encapsulate the overall quality of the generated query. We adopt a linear weighting scheme as follows:

$$X_{core} = \alpha \cdot \mathcal{V} + \beta \cdot S + \gamma \cdot O. \quad (1)$$

Here, \mathcal{V} represents the binary validity score examined in Sec.5.2, S indicates the F1 score of the sub-component matching score described in Sec.5.3, and O represents the summarized output schema detailed in Sec. 5.4. α , β and γ are their corresponding weights, and they sum up to 1. In X_{core} , we assign equal weights to \mathcal{V} , S , and O , as all three components hold comparable significance within the context of incident management. However, these weights can be easily adjusted to prioritize specific aspects in different applications, if required. All three scores range from 0 to 1, making X_{core} a continuous value ranging from 0 to 1 as well. A higher X_{core} indicates a better quality of the generated query.

By integrating these three evaluation perspectives, X_{core} provides a robust and comprehensive assessment of the quality of generated queries, effectively addressing the limitations of existing metrics and catering to the specific requirements of KQL. It is worth noting that while X_{core} is specifically tailored to KQL, its adaptability to other DSL queries is straightforward. This can be achieved by simply substituting its AST and the built-in syntax checker. Hence, X_{core} emerges as an adaptive and scalable metric applicable to various languages and scenarios beyond KQL.

6 OFFLINE EVALUATION

We conduct a comprehensive offline evaluation of XPert, utilizing real incident data and KQL queries from the production environment of MICROSOFT. We compare its performance against several baselines, to answer the following research questions (RQs):

Table 2: Performance evaluation of XPert and other baselines in recommending KQL templates and full queries.

Model	Template						Full Query					
	BLEU	METEOR	XCORE	TableAcc	Identicality	Validity	BLEU	METEOR	XCORE	TableAcc	Identicality	Validity
Bart	2.91	24.90	39.29	31.15	0.43	75.23	8.54	26.96	36.06	33.02	0.50	66.04
T5	60.90	60.84	38.17	31.02	10.52	49.98	30.29	46.65	31.10	23.94	4.64	33.89
CodeT5	70.48	69.64	59.35	48.21	18.70	77.60	64.44	64.88	57.42	52.39	14.59	73.49
CodeT5+	73.50	69.77	61.38	48.58	20.00	82.90	66.17	65.35	60.75	55.53	16.13	80.37
XPert (GPT-3.5)	75.55	67.18	58.19	45.68	30.58	80.87	66.51	64.36	60.27	53.46	24.44	83.01
XPert (GPT-4)	76.89	71.61	62.40	48.81	35.46	83.27	70.45	67.98	64.2	57.56	29.18	86.34

- **RQ1:** How effective is XPert in recommending KQL templates and queries?
- **RQ2:** How effective is the post-processor in correcting and refining the generated KQL queries?
- **RQ3:** How does XCORE outperform other NLP metrics?

We provide answers to these RQs in the subsequent subsections.

6.1 Experiment Setup

6.1.1 Dataset. We use a large-scale dataset comprising incident context and corresponding KQL queries from the top-10 services with the most incident amount in MICROSOFT. These services are the most representative and have the most significant impact. The data was partitioned into 197,666 instances for training and validation (for non-LLMs baselines) / retrieval (for LLMs), and 3,000 instances for testing. To prevent data leakage, incidents in the test data were strictly created after incidents in the training data. Note that we perform one-shot offline evaluation and do not add the incidents to the vector database in an online manner during this evaluation.

6.1.2 Baselines. We evaluate the performance of XPert against several baselines, including smaller language models, namely: (i) Bart [57], a popular sequence-to-sequence transformer model trained with denoising autoencoder fashion; (ii) T5 [58], an encoder-decoder transformer model pre-trained on a mixture of unsupervised and supervised tasks; (iii) CodeT5 [59], an extension of T5 specifically designed for code-related tasks; and (iv) CodeT5+ [60], a more advanced version of CodeT5 pretrained with more diverse programming tasks and uses instruction tuning. All of the smaller language models are fine-tuned with training data. In this study, the terms “T5”, “CodeT5” and “CodeT5+” refer to their base and 220M versions. For XPert, we compare its GPT-3.5 and GPT-4 versions.

6.1.3 Evaluation Metrics. We employ 6 evaluation metrics to comprehensively assess the quality of the generated KQL queries from various perspectives. In line with prior research [30, 61, 62], we adopt two traditional NLP metrics, namely SacreBLEU [50] and METEOR [51], to evaluate the quality of the generated KQL queries. While these metrics have inherent limitations when assessing code quality, they can still offer valuable insights into certain aspects, such as the lexical similarities. The proposed metric XCORE, addresses these limitations by evaluating the code from both syntax and semantic perspectives, making it a crucial measure for assessing the quality of KQL queries.

Additionally, we utilize TableAcc to quantify the accuracy of the predicted table names in the queries, which provides a partial data source assessment. Furthermore, Identicality offers a more stringent evaluation of the recommended queries, measuring the proportion of recommended queries that are identical to the ground

truth. To ensure the executability of the generated queries, we also employ the Validity metric, evaluating the percentage of generated KQL queries that comply with the KQL grammar. This is a critical aspect to guarantee the executability of the generated queries.

6.1.4 Implementation. The smaller language models were implemented using Pytorch [63] and fine-tuned on 2 NVIDIA A100 GPUs. XPert interacts with GPT Ada Embedding, GPT-3.5 and GPT-4 through the Python API provided by OpenAI. The entire implementation of XPert consists of 1,929 lines of C# code and 2,694 lines of Python code.

6.2 KQLs Recommendation Performance (RQ1)

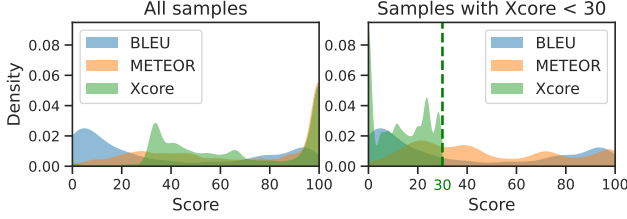
Table 2 presents the performance evaluation of XPert in recommending KQL templates and full queries, compared to various baseline models. Notably, XPert equipped with GPT-4 demonstrates superior performance across all metrics for both template and query recommendation. This outcome suggests that XPert represents the state-of-the-art approach from various perspectives, encompassing improved lexical similarity metrics (BLEU and METEOR), enhanced syntax correctness and semantic analogy assessments (XCORE), higher table accuracy (TableAcc), and superior exact matching with the ground truth (Identicality) and executability (Validity). Particularly, its advantage is most pronounced in the Identicality dimension, where both GPT-3.5 and GPT-4 versions of XPert significantly outperform other smaller language models. This aspect bears particular significance in the incident management scenario, as refining a query can be a laborious task for OCEs. A high Identicality rate in XPert recommendations renders the process more user-friendly and accessible, particularly for less experienced OCEs.

CodeT5+ emerges as a stronger baseline compared to other baselines, exhibiting comparable performance with XPert (GPT-3.5) in several metrics. However, note that CodeT5+ requires a costly fine-tuning process, rendering it difficult and expensive to adapt to evolving incident types in an online manner. In contrast, XPert offers a more effective, simpler, and cost-efficient solution by effortlessly adding a new incident and its ground truth query to the vector database upon arrival, thereby capturing online data drift, as will be demonstrated in Sec. 7.

Moreover, as templates of KQLs are less diverse compared to full queries, XPert achieves greater identicality in template recommendation compared to query recommendation. This can be particularly helpful in situations where incident context is incomplete. In such cases, OCEs can refine KQL queries based on the predicted templates, saving the query writing time. Overall, XPert delivers remarkable performance in every dimension, especially when empowered by GPT-4. These inspiring results establish XPert

Table 3: Comparison before and after the post-processing.

Metric	Template		Full Query	
	Before	After	Before	After
BLEU	34.95	36.61	30.52	27.88
METEOR	42.24	46.30	47.77	46.14
XCORE	11.63	35.99	11.19	35.93
TableAcc	22.54	38.03	17.39	31.68
Identity	0.00	7.04	0.00	10.56
Validity	0.00	50.70	0.00	56.52


Figure 9: Distributions of BLEU, METEOR and XPert in the test set (left), and samples with Xcore below 30 (right).

as an effective solution for recommending KQL queries, thereby significantly facilitating the job of OCEs during incidents.

Lastly, it is important to highlight the impressive few-shot learning capability of LLMs observed during our experiments. The average and median number of similar samples retrieved for the prompt sequence provided to XPert, with a maximum of 8k tokens, is 7.41 and 6, respectively. These values are surprisingly small, indicating that LLMs can achieve remarkable performance with very few examples for demonstration. In fact, LLMs outperform other smaller models that are trained with approximately 200 thousand samples in all evaluated dimensions. This outstanding few-shot learning ability not only reduces the effort of pre-training and fine-tuning but also positions LLMs as an ideal solution for real productions.

6.3 Post-processor Effectiveness (RQ2)

We now shift our attention to evaluating the effectiveness of the post-processor in improving the query generation quality. In the raw predictions of XPert (GPT-4), a total of 71 templates and 161 queries are found to be invalid. After applying the post-processor, 36 (50.70%) templated and 91 (56.52%) queries are successfully fixed. Taking a closer look at the impact of the post-processor on these invalid cases as shown in Table 3, we observe that it significantly improves the quality of the generated queries across various metrics. The Xcore of generated templates and queries improved by 24.36 and 24.74 respectively, showcasing its efficacy in enhancing the overall query quality. Moreover, the identity metric saw an improvement of 7.04% and 10.56% for templates and queries respectively, further highlighting the necessity and effectiveness of the post-processor. These results demonstrate that the post-processor is an indispensable component of XPert, as it plays a vital role in refining and enhancing the generated queries.

6.4 Xcore Evaluation (RQ3)

Finally, we present a comparison of Xcore with other NLP metrics to emphasize the effectiveness of our proposed design. In Fig. 9,

Ground truth	Generated sample
1 Errors	JobErrors
2 where TIMESTAMP datetime(2022-03-27 22:04:06Z)	where TIMESTAMP datetime(2022-03-27 22:04:06Z)
3 where operationName "ImagePull"	where operationName "ImagePull"
4 where SourceNamespace contains "us-east"	where SourceNamespace contains "us-east"
5 where Deployment == "svc-mesh-3d21"	where Deployment "svc-mesh-3d21"
6 summarize count() by subscriptionId	summarize count(subscriptionId)

Figure 10: A representative generated sample and its corresponding ground truth.
Table 4: Performance comparison of XPert in productions.

Model	Template		Full Query	
	XCORE	Identity	XCORE	Identity
CodeT5+ (Offline)	58.5	10.04	64.94	2.62
XPert (Offline)	55.63	14.19	66.55	11.14
CodeT5+ (Online)	58.79	10.04	65.43	2.62
XPert (Online)	58.74	18.12	72.96	17.69

we depict the distributions of BLEU, METEOR, and Xcore on all samples in the test set (left), as well as on selected samples where Xcore is below 30 (right). Upon examination, we observe distinct differences in the distributions of Xcore compared to the other two metrics. While XPert and METEOR show significant overlap in the high score region (over 90), they exhibit substantial differences in the low score region (<30). Upon closer examination of the samples where Xcore falls below 30 in the right subplot, we observe that although Xcore values are low, both BLEU and METEOR metrics are evenly distributed across their entire range from 0 to 100. This observation prompts us to investigate the discrepancy among these metrics in these particular samples to determine which metric provides a more reasonable evaluation.

To this end, we present a representative generated sample and its corresponding ground truth in Fig. 10. The generated sample achieves satisfactory scores on NLP metrics (75.67 for BLEU and 87.02 for METEOR), as there is a high lexical similarity between the two queries. However, it only achieves a score of 3.54 on Xcore, which is significantly lower. Upon closer examination, we observe the following issues: (i) the generated sample fails the validity check (Sec. 5.2) due to the incorrect usage of = in the where operator (line 5); (ii) various operations are inverted in the generated sample (lines 2 and 3), resulting in substantial sub-component mismatches (Sec. 5.3); and (iii) examination of the output schema (Sec. 5.4) uncovers discrepancies in the source tables and output columns in the two queries (lines 1 and 6). These discrepancies result in low scores of V, S, and O, leading to the low overall Xcore.

This discrepancy between Xcore and other NLP metrics highlights the effectiveness and reliability of Xcore for evaluating the quality of KQL queries. While the generated sample may appear plausible and lexically similar based on BLEU and METEOR, closer examination reveals its invalidity and inaccuracies, making Xcore a more robust and meaningful metric in this context.

7 PRODUCTION IMPACT

The proposed framework XPert, has been seamlessly integrated as a crucial component within the incident management system of MICROSOFT. A pilot study was conducted, wherein XPert was

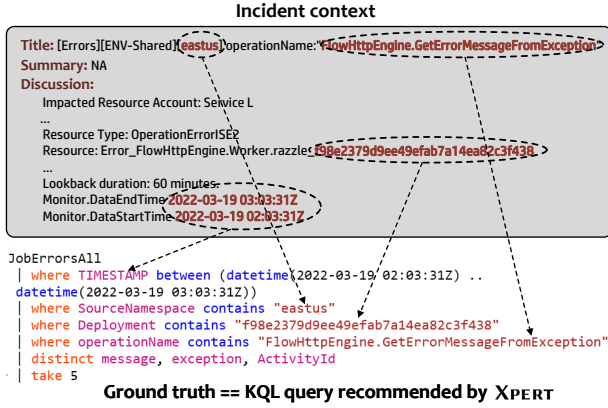


Figure 11: A successful case that XPERT recommends an KQL query identical to the ground truth.

deployed for approximately one month on incidents in a representative service. Whenever an incident ticket is received, XPERT is triggered to recommend the first XQL query for incident management, following the workflow depicted in Fig. 6. Unlike the offline evaluation, each time a new incident ticket and its corresponding ground-truth query written by OCEs are recorded, we promptly add this instance to the vector database. This functionality enables subsequent requests to retrieve this recorded sample, ensuring that XPERT can continuously adapt to new types of incidents without requiring adjustments to other models or settings. We refer to this framework equipped with GPT-4 deployed in the production environment, as “Online XPERT”. To compare the effectiveness of XPERT in the real production environment, we conducted fine-tuning on CodeT5+ weekly (referred to as “Online CodeT5+”) and used its latest version for predictions. For both XPERT and CodeT5+, we compare their performances without vector database updates and fine-tuning, denoting them as “Offline” versions.

The piloting results are presented in Table 4. The online experiments yield inspiring outcomes, revealing the superiority of XPERT over CodeT5+ in various aspects. Firstly, both online and offline versions of XPERT significantly outperform CodeT5+ in terms of Xcore and Identity in XQL query recommendation, affirming XPERT’s consistent advantage in over smaller language models. Secondly, the online version of XPERT demonstrates superior performance compared to its offline counterpart. This suggests that XPERT effectively captures the significant time variation of incidents through effortless vector database updates, enabling it to adapt more flexibly to the ever-evolving real production environment. Furthermore, the average online response time for XPERT is approximately 5 seconds, demonstrating its efficiency in meeting the online responsiveness requirement. All these results further underscore XPERT’s potential as a more robust, efficient and adaptive solution for XQL recommendation in practical scenarios.

8 DISCUSSION

8.1 Case Study

8.1.1 Successful Case. In Fig. 11, we present a successful case where XPERT recommends an KQL query that perfectly matches the ground truth written by an OCE. In this case, XPERT efficiently

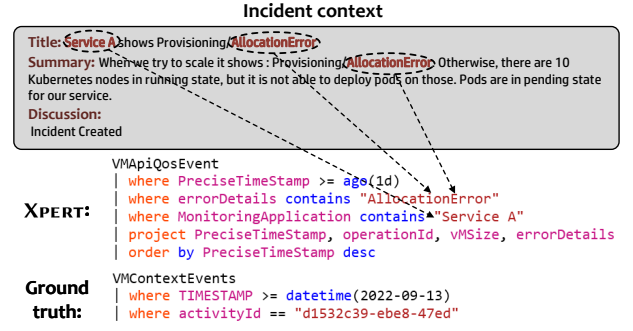


Figure 12: A failed case that XPERT recommends a different KQL query to the ground truth.

extracts relevant information from the incident title and discussion, including details such as time, region, and resource ID. It skillfully places this extracted information in the appropriate positions within the query to construct the operands. By leveraging this context, XPERT successfully generates a query that is not only syntactically and semantically correct but also identical to the query written by the OCE. This outcome demonstrates the superior information extraction capability of LLMs, enabling them to distill valuable insights from complex and unstructured data. Consequently, LLMs are well-suited for the KQL query recommendation task and can provide significant assistance to OCEs.

8.1.2 Failed Case. However, it is essential to acknowledge that XPERT may not always provide optimal recommendations and may encounter challenges when dealing with incidents that lack crucial information. In Fig. 12, we illustrate a case where XPERT fails to recommend the correct query to OCEs. Although the ground truth query is short and simple, crucial information, such as TIMESTAMP and activityId, is absent from the incident context, rendering it impossible to predict these elements accurately. Upon closer examination of the query recommended by XPERT, we observe that it still extracts useful information from the incident, such as the error and service details. Additionally, we found that historically, queries similar to the predicted one have occurred frequently in this service. This indicates that the prediction does not come from nowhere, as XPERT encapsulates significant incident information and its predicted query can serve as a valuable reference for OCEs.

8.2 Threats to Validity

8.2.1 Internal Validity. Throughout this study, we encountered challenges related to the instability of LLMs provided by OpenAI. The LLM service underwent continuous upgrades with multiple model versions and API changes released. Moreover, hyperparameter settings, such as temperature and top_p, may also result in unstable outcomes. To address this issue, we fix the API version and model version and set both temperature and top_p to zero. Despite these configurations, we observed occasional differences in the LLM response even for the same request. This may be attributed to float precision differences in the underlying hosting machines.

The datasets in this study were collected from the incident management platform at Microsoft. Due to the vast volume of incidents over years, we focused on selecting incidents of high severity and top cloud services starting from the year 2022, as they are

of greater significance and relevance. While preparing the KQL queries, we extracted the queries from free-text discussions using domain rules. However, this process may introduce inaccuracies or incompleteness in the extracted queries. To mitigate this concern, we meticulously designed and iterated the rules through multiple rounds, and we also removed queries that were found to be invalid.

8.2.2 External Validity. While our work focuses on the KQL, our approach is easily applicable to other query languages, such as SQL. The key advantage lies in our method’s ability to generalize without requiring model fine-tuning, leveraging the in-context learning capability of LLMs. Additionally, since popular query languages like SQL are extensively encountered during LLM pre-training, we anticipate even better performance when applied to such languages.

Our experiments were conducted on a select set of core services within MICROSOFT. However, the fundamental concept and workflow of our proposed method can be adapted to other services and scenarios with ease. By incorporating incident-query data from new services, our approach can be readily extended to support additional services. Furthermore, in scenarios with abundant pairwise data, our approach can be adopted to build a versatile pipeline.

9 RELATED WORK

In this section, we examine a selection of notable research studies pertaining to the generation of DSL and the application of language models in the field of software engineering.

9.1 Domain-Specific Language Generation

Deep learning techniques have gained significant traction in the DSL generation domain, particularly for languages like SQL [64, 65], LaTeX [66], and GraphQL [67]. These methods aim to make programming languages in specific domains more accessible to non-technical users. Notably, text-to-SQL has received considerable research attention, as SQL is widely used for data querying and insights discovery. In [68], a fine-tuned T5 model is employed to guide the auto-regressive decoders of language models through incremental parsing. This approach helps minimize the generation of invalid code during the process. Hui *et al.*, propose S²SQL [69], which leverages syntactic dependency information from text-to-SQL questions to enhance performance, surpassing a set of pretrained models. Additionally, Liu *et al.*, evaluate the text-to-SQL generation capabilities of ChatGPT [70], demonstrating its remarkable performance in zero-shot scenarios [71].

9.2 LLMs for Software Engineering

The advancement of LLMs has greatly contributed to the feasibility and capabilities of these models in the field of software engineering [4, 30, 38]. For instance, Ahmed [30] employed GPT models to recommend root causes and mitigation steps for incidents, facilitating incident management. Promising results showcased the potential of LLMs in incident management. Similarly, RCACopilot [38] utilized LLMs to improve root cause analysis accuracy by leveraging additional information from troubleshooting guides and chain-of-thought techniques [72]. Furthermore, Jin *et al.*, [4] employed LLMs to summarize outages in cloud environments. Online performance evaluations demonstrated that LLMs can achieve human-level outage summaries at significantly faster speeds (251.2×). In a departure

from the aforementioned practices, XPert presents a pioneering framework for automatically recommending DSL queries to support incident management tasks.

10 CONCLUSION

Incident management plays a critical role in ensuring the smooth functioning of cloud infrastructure, requiring OCEs to execute DSL queries for understanding incidents and support the management processes. This paper conducts a comprehensive empirical study on KQL queries used in a large-scale incident management system at MICROSOFT, revealing valuable insights into the frequency, complexity, and diversity of KQL queries. Based on these findings, we propose XPert, an end-to-end framework that leverages LLMs and ICL to provide customized query recommendations tailored to new incidents, thus empowering the incident management process. To evaluate the quality of the generated queries, we introduce Xcore, a dedicated evaluation metric that comprehensively assesses query performance from three different perspectives. In offline evaluations using real-world data, XPert outperforms other baseline models across various metrics. Furthermore, we have successfully deployed XPert in the real production environment of MICROSOFT, and the piloting results confirm XPert’s reliability in supporting incident management. To the best of our knowledge, this paper presents the first empirical study of DSL queries for incident management, and XPert stands as a pioneering KQL recommendation framework specifically designed to enhance incident management processes.

11 DATA AVAILABILITY

The data used in this work originates from the production of MICROSOFT and contains highly confidential information. We are therefore unable to make the data publicly available due to security concerns.

REFERENCES

- [1] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 27–33. Ieee, 2010.
- [2] Xiaohan Yan, Ken Hsieh, Yasitha Liyanage, Minghua Ma, Murali Chintalapati, Qingwei Lin, Yingnong Dang, and Dongmei Zhang. Aegis: Attribution of control plane change impact across layers and components for cloud systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 222–233. IEEE, 2023.
- [3] Bernd Grobauer and Thomas Schreck. Towards incident handling in the cloud: challenges and approaches. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pages 77–86, 2010.
- [4] Pengxiang Jin, Shenglin Zhang, Minghua Ma, Haozhe Li, Yu Kang, Liqun Li, Yudong Liu, Bo Qiao, Chaoyun Zhang, Pu Zhao, et al. Assess and summarize: Improve outage understanding with large language models. *arXiv preprint arXiv:2305.18084*, 2023.
- [5] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1487–1497, 2020.
- [6] Liqun Li, Xu Zhang, Shilin He, Yu Kang, Hongyu Zhang, Minghua Ma, Yingnong Dang, Zhangwei Xu, Saravan Rajmohan, Qingwei Lin, et al. Conan: Diagnosing batch failures for cloud systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 138–149. IEEE, 2023.
- [7] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Kukun Li, Yingnong Dang, Qingwei Lin, et al. Onion: identifying incident-indicating logs for cloud systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1253–1263, 2021.
- [8] Chaoyun Zhang, Marco Fiore, Cezary Ziemlicki, and Paul Patras. Microscope: mobile service traffic decomposition for network slicing as a service. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.
- [9] Chenyu Zhao, Minghua Ma, Zhenyu Zhong, Shenglin Zhang, Zhiyuan Tan, Xiao Xiong, LuLu Yu, Jiayi Feng, Yongqian Sun, Yuzhi Zhang, et al. Robust multimodal failure detection for microservice systems. *arXiv preprint arXiv:2305.18985*, 2023.
- [10] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications surveys & tutorials*, 21(3):2224–2287, 2019.
- [11] Chaoyun Zhang, Marco Fiore, Iain Murray, and Paul Patras. Cloudlstm: A recurrent neural model for spatiotemporal point-cloud stream forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10851–10858, 2021.
- [12] Zhengran Zeng, Yuqun Zhang, Yong Xu, Minghua Ma, Bo Qiao, Wentao Zou, Qingjun Chen, Meng Zhang, Xu Zhang, Hongyu Zhang, et al. Traceark: Towards actionable performance anomaly alerting for online service systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 258–269. IEEE, 2023.
- [13] Lu Wang, Chaoyun Zhang, Ruomeng Ding, Yong Xu, Qihang Chen, Wentao Zou, Qingjun Chen, Meng Zhang, Xuedong Gao, Hao Fan, et al. Root cause analysis for microservice systems via hierarchical reinforcement learning from human feedback. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5116–5125, 2023.
- [14] Yuhang Chen, Chaoyun Zhang, Minghua Ma, Yudong Liu, Ruomeng Ding, Bowen Li, Shilin He, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Imdiffusion: Imputed diffusion models for multivariate time series anomaly detection. *arXiv preprint arXiv:2307.00754*, 2023.
- [15] Manish Shetty, Chetan Bansal, Sai Pramod Upadhyayula, Arjun Radhakrishna, and Anurag Gupta. Autots: learning and synthesis for incident troubleshooting. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1477–1488, 2022.
- [16] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL ’02, page 311–318, USA, 2002. Association for Computational Linguistics.
- [17] Van-Hoang Le and Hongyu Zhang. An evaluation of log parsing with chatgpt. *arXiv preprint arXiv:2306.01590*, 2023.
- [18] Le Xiao and Xiaolin Chen. Enhancing llm with evolutionary fine tuning for news summary generation. *arXiv preprint arXiv:2307.02839*, 2023.
- [19] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Everything of thoughts: Defying the law of penrose triangle for thought generation. *arXiv preprint arXiv:2311.04254*, 2023.
- [20] Dheeraj Mekala, Jason Wolfe, and Subhro Roy. Zerotop: Zero-shot task-oriented semantic parsing using large language models. *arXiv preprint arXiv:2212.10815*, 2022.
- [21] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*, pages 1–7, 2022.
- [22] Bulbul Gupta, Pooja Mittal, and Tabish Mufti. A review on amazon web service (aws), microsoft azure & google cloud platform (gcp) services. In *Proceedings of the 2nd International Conference on ICT for Digital, Smart, and Sustainable Development, ICIDSSD 2020, 27-28 February 2020, Jamia Hamdard, New Delhi, India*, 2021.
- [23] Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 126–141, 2022.
- [24] Weijing Wang, Junjie Chen, Lin Yang, Hongyu Zhang, Pu Zhao, Bo Qiao, Yu Kang, Qingwei Lin, Saravanakumar Rajmohan, Feng Gao, et al. How long will it take to mitigate this incident for online service systems? In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 36–46. IEEE, 2021.
- [25] Manish Shetty, Chetan Bansal, Sumit Kumar, Nikitha Rao, Nachiappan Nagappan, and Thomas Zimmermann. Neural knowledge extraction from cloud service incidents. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 218–227. IEEE, 2021.
- [26] Alex Khang, Vladimir Hahanov, Gardashova Latafat Abbas, and Vugar Abdullayev Hajimahmud. System and incident management. *AI-centric smart city ecosystems: technologies, design and implementation*, page 21, 2022.
- [27] Chen Luo, Jian-Guang Lou, Qingwei Lin, Qiang Fu, Rui Ding, Dongmei Zhang, and Zhe Wang. Correlating events with time series for incident diagnosis. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1583–1592, 2014.
- [28] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3009–3017, 2019.
- [29] Liqun Li, Xu Zhang, Xin Zhao, Hongyu Zhang, Yu Kang, Pu Zhao, Bo Qiao, Shilin He, Pochian Lee, Jeffrey Sun, et al. Fighting the fog of war: Automated incident detection for cloud systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 131–146, 2021.
- [30] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *ICSE 2023*, 2023.
- [31] Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.
- [32] Chris J Date. *A Guide to the SQL Standard*. Addison-Wesley Longman Publishing Co., Inc., 1989.
- [33] Olaf Hartig and Jorge Pérez. Semantics and complexity of graphql. In *Proceedings of the 2018 World Wide Web Conference*, pages 1155–1164, 2018.
- [34] Navin Sabharwal, Piyush Pandey, Navin Sabharwal, and Piyush Pandey. Working with prometheus query language (promql). *Monitoring Microservices and Containerized Applications: Deployment, Configuration, and Best Practices for Prometheus and Alert Manager*, pages 141–167, 2020.
- [35] Splunk Search Processing Language (SPL), howpublished = <https://docs.splunk.com/documentation/splunk/latest/searchreference/sqltosplunk>, note = Accessed: 2023-07-06.
- [36] Phuong Pham, Vivek Jain, Lukas Dauterman, Justin Ormont, and Navendu Jain. Deeptriage: Automated transfer assistance for incidents in cloud services. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3281–3289, 2020.
- [37] Zilong He, Pengfei Chen, Yu Luo, Qiuyu Yan, Hongyang Chen, Guangba Yu, and Fangyuan Li. Graph based incident extraction and diagnosis in large-scale online systems. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.
- [38] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, et al. Empowering practical root cause analysis by large language models for cloud incidents. *arXiv preprint arXiv:2305.15778*, 2023.
- [39] OpenAI. GPT-4 technical report, 2023.
- [40] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [41] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2021.
- [42] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

- [43] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, 2022.
- [44] Niklas Muennighoff. Sgpt: GPT sentence embeddings for semantic search. *arXiv preprint arXiv:2202.08904*, 2022.
- [45] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.
- [46] Baoli Li and Liping Han. Distance weighted cosine similarity measure for text classification. In *Intelligent Data Engineering and Automated Learning–IDEAL 2013: 14th International Conference, IDEAL 2013, Hefei, China, October 20–23, 2013. Proceedings 14*, pages 611–618. Springer, 2013.
- [47] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.
- [48] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*, 2022.
- [49] Iulian Neamtiu, Jeffrey S Foster, and Michael Hicks. Understanding source code evolution using abstract syntax tree matching. In *Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, 2005.
- [50] Matt Post. A call for clarity in reporting bleu scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, page 186. Association for Computational Linguistics, 2018.
- [51] Satandeep Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.
- [52] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis, 2020.
- [53] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2019.
- [54] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation, 2023.
- [55] Panagiotis Louridas. Static code analysis. *Ieee Software*, 23(4):58–61, 2006.
- [56] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning, 2017.
- [57] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, 2020.
- [58] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.
- [59] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, 2021.
- [60] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*, 2023.
- [61] David Gros, Hariharan Sezhiyan, Prem Devanbu, and Zhou Yu. Code to comment" translation" data, metrics, baselining & evaluation. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 746–757, 2020.
- [62] Toufique Ahmed and Premkumar Devanbu. Multilingual training for software engineering. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1443–1455, 2022.
- [63] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [64] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, et al. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*, 2022.
- [65] George Katsogiannis-Meimarakis and Georgia Koutrika. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, pages 1–32, 2023.
- [66] Zelun Wang and Jyh-Charn Liu. Translating math formula images to latex sequences using deep neural networks with sequence-level training. *International Journal on Document Analysis and Recognition (IJDAR)*, 24(1-2):63–75, 2021.
- [67] Pin Ni, Ramin Okhrati, Steven Guan, and Victor Chang. Knowledge graph and deep learning-based text-to-graphql model for intelligent medical consultation chatbot. *Information Systems Frontiers*, pages 1–20, 2022.
- [68] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, 2021.
- [69] Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Yanyang Li, Bowen Li, Jian Sun, and Yongbin Li. S2sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1254–1262, 2022.
- [70] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.
- [71] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*, 2023.
- [72] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.