TECHSUPPORTEVAL: An Automated Evaluation Framework for Technical Support Question Answering

Bohan Chen[1], Yongqian Sun [2,*], Yuhe Liu[1], Longlong Xu[1], Zhe Xie[1], Changhua Pei[3], Jing Han[4], Fan Ni[4], Xuhui Cai[5], Ce Yang[5], and Dan Pei[1]

[1]Tsinghua University, {cbh22, lyh23, xull23, xiez22}@mails.tsinghua.edu.cn, peidan@tsinghua.edu.cn
[2]Nankai University, sunyongqian@nankai.edu.cn
[3]Computer Network Information Center, Chinese Academy of Sciences, chpei@cnic.cn
[4]ZTE Corporation, {han.jing28, ni.fan1}@zte.com.cn
[5]China Mobile Communications Group Co., Ltd., {caixuhui, yangce}@chinamobile.com

***Abstract*—Technical support question-answering (QA) systems assist users in diagnosing and resolving technical issues, but ensuring their reliability remains a challenge. Existing QA systems may generate inaccurate responses due to LLM hallucinations and retrieval errors, which can lead to misleading guidance. A reliable evaluation framework is essential for systematically improving technical support QA systems, ensuring they generate accurate guidance. However, existing evaluation methods for QA systems struggle to precisely match key terms, and verify step order and completeness.**

To address these challenges, we propose TECHSUPPORTEVAL, an automated evaluation framework for technical support QA. Our framework introduces two novel techniques: (1) ClozeFact, which formulates fact verification as a cloze test and uses an LLM to fill in missing key terms to ensure precise key term matching, and (2) StepRestore, which shuffles ground truth steps and uses an LLM to reconstruct the actionable instructions in the correct order, verifying step order and completeness.

To support comprehensive evaluation, we propose a benchmark dataset built upon the publicly available TechQA dataset, containing responses generated by different levels of QA systems. TECHSUPPORTEVAL achieves an AUC of 0.91, outperforming the state-of-the-art method by 7.6%. The code and dataset are available at https://github.com/NetManAIOps/TechSupportEval.

***Index Terms*—Technical Support, Question Answering, Automated Evaluation, LLM**

I. INTRODUCTION

Technical support is a critical domain focused on diagnosing and resolving technical issues to maintain the reliability of IT services, forming a notable part of the trillion-dollar IT services industry [1], [2]. A common approach to technical support is the question answering (QA) [3]–[6], where users (such as customers or developers) describe their issues and seek guidance from technical experts on platforms such as Microsoft Forums and IBM Support Forums. Technical experts typically rely on knowledge bases, e.g., product manuals and troubleshooting guides, to analyze issues and provide actionable instructions [2].

Traditional human-driven technical support faces limitations in response time and scalability, as it relies on a limited number of technical experts and struggles to handle a high volume of queries efficiently [7]. With the advancement of Large Language Models (LLMs) [8], [9], QA systems leveraging Retrieval-Augmented Generation (RAG) and LLMs have emerged to process technical support questions [10], [11], enhancing both efficiency and scalability. Figure 1 presents an example of how a typical LLM-based QA system responds to a technical support question.

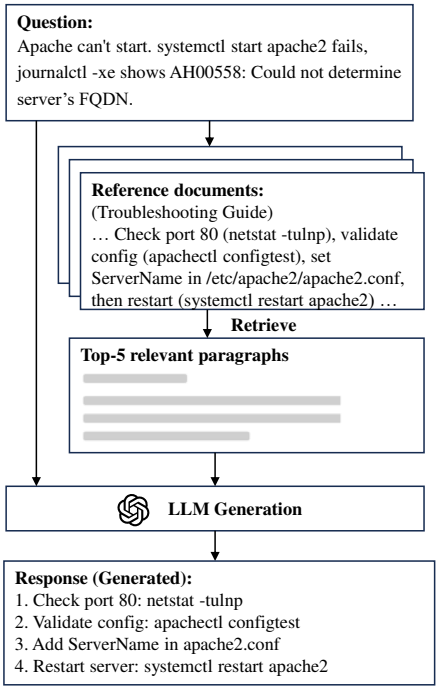

Fig. 1. Workflow of a typical LLM-based technical support QA system. The system takes a question as input, retrieves the top 5 relevant paragraphs from reference documents, and generates a response through an LLM.

However, existing RAG-based QA systems, which are built upon LLMs, may generate inaccurate responses to technical support questions. This issue primarily arises from hallucinations in LLMs [12], [13] and the omission of relevant paragraphs when retrieving from knowledge bases [14]. In-

* Yongqian Sun is the corresponding author

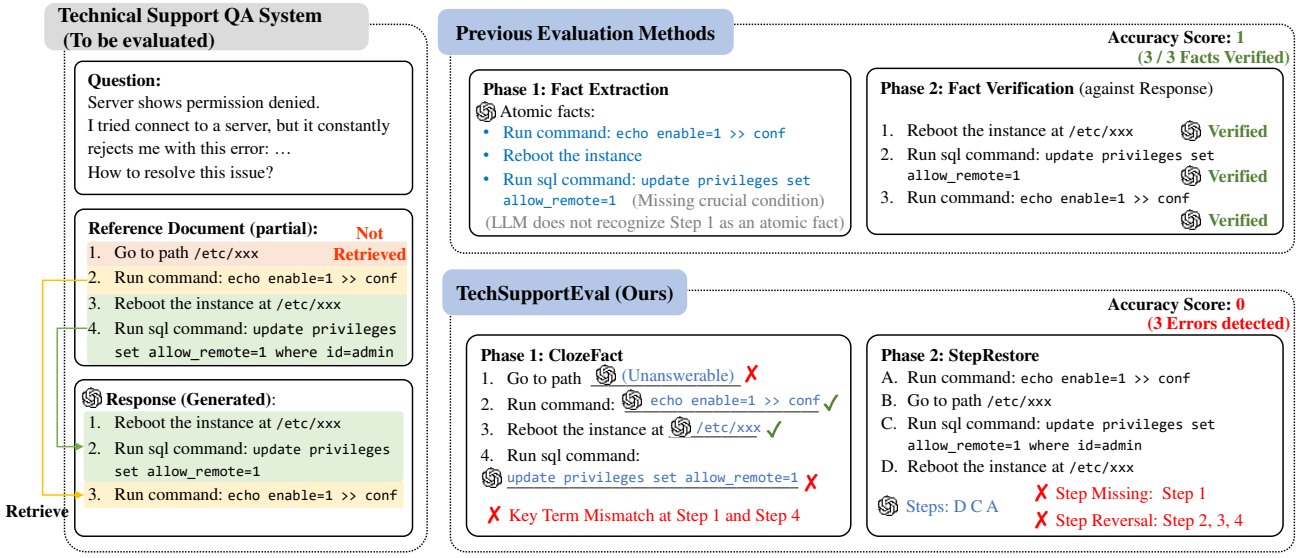

Fig. 2. Comparison of TECHSUPPORTEVAL with previous evaluation methods. The left plot shows a QA system inaccurately answering a technical support question. The upper right plot illustrates how previous evaluation methods failed to detect the errors, missing a step during fact extraction and omitting a crucial condition in the SQL command, leading to incorrect judgments. The lower right plot shows TECHSUPPORTEVAL identifying the errors using **ClozeFact** and **StepRestore** through a comprehensive mechanism, detecting mismatched key terms and verifying that the steps are neither in the correct order nor complete.

accurate responses lead to misleading guidance, making it impossible for users to resolve their issues successfully. Thus, an automated evaluation framework is essential for detecting inaccurate responses, which in turn provides a foundation for improving the reliability of technical support QA systems.

Various evaluation methods [15]–[28] have been proposed to assess the accuracy of QA system responses automatically. A detailed discussion of these methods is provided in Section II-B. However, these methods cannot reliably assess the accuracy of responses generated by a technical support QA system.

In this paper, the accuracy of technical support QA corresponds to whether the response mentions all necessary steps of actionable instructions in the correct order, along with precise key terms such as commands, file paths, and URLs. This criterion aligns with the primary goal of technical support, which is to help users resolve their issues successfully. Based on this criterion, we identify three key challenges that previous evaluation methods have not fully overcome:

1) **Key Term Matching**: LLM hallucinations may introduce incorrect or non-existent key terms, leading to erroneous operations, e.g., modifying the wrong file. Existing evaluation methods often fail to detect these mismatches.
2) **Step Order Verification**: RAG retrieval may return fragments in the wrong order, causing incorrect execution. Prior work typically treats extracted facts as an unordered set, making it difficult to verify step order.
3) **Step Completeness Verification**: RAG retrieval may omit intermediate steps, leading to incomplete guidance. Existing evaluation methods may fail to recognize all necessary steps, resulting in omissions during validation.

The limitations of the previous evaluation methods are further discussed in Section II-B. To address these challenges, we propose an LLM-based automatic evaluation framework TECHSUPPORTEVAL, that incorporates two key techniques:

1) **ClozeFact** identifies key terms within the ground truth and converts the ground truth into a cloze test by blanking out these key terms. The LLM used for evaluation (denoted as LLM_{eval}) is then required to fill in the blanks based on the response (generated by the QA system being evaluated), ensuring precise key term matching.
2) **StepRestore** shuffles the steps from the ground truth and asks LLM_{eval} to reorder the steps by selecting the mentioned steps from the response. This ensures reliable verification of step order and completeness.

Figure 2 compares our approach to a typical category of evaluation methods [22]–[28]. When adopting these evaluation methods in this example, a step was missed in the fact extraction phase and an SQL command missed a crucial condition, leading to a false positive. In contrast, TECHSUPPORTEVAL avoided these issues using **ClozeFact** and **StepRestore**.

The contributions of this paper are concluded as follows:

1) We investigate the evaluation of technical support QA and pinpoint three key challenges it presents.
2) We propose an LLM-based automated evaluation framework TECHSUPPORTEVAL for technical support QA with two novel techniques, **ClozeFact** and **StepRestore** to address the challenges effectively.
3) We introduce a benchmark dataset based on the publicly available TechQA dataset [1], containing responses generated by different levels of QA systems for comprehensive evaluation. Our approach achieves an AUC of 0.91, outperforming RefChecker [27], the state-of-the-art method, by 7.6%. The code and dataset are available at https://github.com/NetManAIOps/TechSupportEval.

II. Related Works

A. Technical Support QA

Technical support QA is a practical domain, with real-world datasets proposed to facilitate research. TechQA [1] introduced a dataset based on user queries from the IBM Developer Forum, while MSQA [29] collected QA pairs from Microsoft Forums, both reflecting authentic technical support scenarios.

Various techniques have been proposed for technical support QA. Before the LLM era, retrieval-based methods dominated, often combined with learning-based models. For instance, TransTQA [6] integrated retrieval with transfer learning, and [7] explored retrieval with seq2seq models. With the rise of LLMs, the field has shifted towards LLM-enhanced approaches, e.g., RAG-based methods [11] and LLM-integrated methods based on knowledge graph reasoning [10]. The progress in technical support QA highlights the need for a reliable evaluation method.

B. Automated Evaluation of QA Systems

Automated evaluation of QA systems has traditionally relied on lexical-based and semantic-based methods. Lexical-based methods, like ROUGE [15] and BLEU [16], compare n-grams between responses and ground truths, while semantic-based methods, such as BERTScore [17], use contextual embeddings to assess semantic similarity. However, both struggle to capture the step-by-step structure in the text.

With the rise of LLMs, evaluation methods such as G-Eval [19] and the evaluation modules in LangChain [20] and LlamaIndex [21] use few-shot prompting to let LLMs assign scores. However, these approaches are black-box, prompt-sensitive, and lack consistency in their evaluations.

Fact extraction and verification approaches validate responses at the atomic fact level, where each atomic fact aligns with a step in the actionable instructions defined in this study. FActScore [22] extracts facts from the response and verifies them against knowledge sources. RAGAS [23] extends this with multiple evaluation metrics, while ARES [24] fine-tunes LLM_{eval} and automates test data generation. RAGQuestEval [25] generates fact-based questions from the ground truth and validates the response through LLM_{eval}. RefChecker [27] refines fact representation into triplets, and L-Face4RAG [28] further incorporates logical consistency checks. However, these evaluation methods remain inadequate for technical support QA due to three key issues:

1) **Key Term Matching** – LLM_{eval} may hallucinate, altering or omitting key terms, leading to incorrect validation.
2) **Step Order Verification** – Extracted facts are treated as an unordered set, making it difficult to verify the correct order of the steps in the actionable instructions.
3) **Step Completeness Verification** – LLM_{eval} may fail to recognize all necessary steps as atomic facts, resulting in incomplete verification.

Figure 2 illustrates the above issues. It shows the process of these evaluation methods and explains why they lead to incorrect assessments.

III. Methodology

In this section, we first formulate the problem in Section III-A. Next, we introduce an error typology to categorize evaluation errors in technical support QA, as presented in Section III-B. We then describe the detailed evaluation pipeline of our framework, TechSupportEval, in Section III-C. Finally, we illustrate our scoring strategy in Section III-D.

A. Problem Formulation

Given a technical support question Q and a set of reference documents $\mathcal{D}$, a QA system based on LLM_{QA} retrieves relevant paragraphs and generates a response:

$$A = \text{Generate}(\mathcal{D}', Q; \text{LLM}_{QA}), \tag{1}$$

where $\mathcal{D}' = \text{Retrieve}_{\text{K}}(Q, \mathcal{D})$ represents the top-K relevant paragraphs retrieved from $\mathcal{D}$. Here, $\text{Retrieve}_{\text{K}}(\cdot)$ identifies the most relevant paragraphs based on Q, and $\text{Generate}(\cdot)$ uses LLM_{QA} to synthesize the final response A from Q and $\mathcal{D}'$.

The ground truth GT represents the actionable instructions required to resolve the underlying issues of the question Q. GT can be formulated as an ordered list of steps:

$$\text{GT} = [g_1, g_2, ..., g_n], \quad g_i \text{ is an intermediate step.} \tag{2}$$

The LLM-based evaluation method $\text{Evaluate}(\cdot)$, which is the main focus of our study, assigns an accuracy score $S \in [0, 1]$ to the response A generated by the QA system:

$$S = \text{Evaluate}(A, \text{GT}, Q; \text{LLM}_{eval}). \tag{3}$$

Our goal is to design $\text{Evaluate}(\cdot)$ such that S serves as a reliable predictor of the true accuracy $S^* \in \{0, 1\}$ of A, maximizing $\text{AUC}(S, S^*)$, which measures how well S distinguishes between accurate and inaccurate responses. A detailed discussion of AUC is provided in Section IV-A(3).

B. Error Typology

A technical support QA system may generate responses that contain three types of errors: (1) **Key Term Mismatch**, (2) **Step Reversal**, and (3) **Step Missing**. Table I provides an example illustrating these errors. Such errors often arise due to LLM hallucinations and retrieval inconsistencies in RAG-based QA systems, where the retrieved information may not match the original reference documents, resulting in incomplete or misordered steps in the response A.

1) **Key Term Mismatch** occurs when the generated response contains key terms that do not match those in the reference documents, such as mismatched commands, file paths, or configuration parameters. This typically results from LLM hallucinations or retrieval errors, leading users to perform erroneous operations.
2) **Step Reversal** refers to cases where the response presents necessary steps in the wrong order. This can be caused by retrieval misalignment in RAG-based systems. Executing steps in the correct order is crucial, as performing certain

TABLE I
EXAMPLES OF ERRORS IN TECHNICAL SUPPORT QA

Question (Q): My Apache server fails to start. Running `systemctl start apache2` shows an error. How can I fix this?	
Ground Truth (GT): 1. Identify the process using port 80 with `netstat -tulnp`. 2. Stop the process. 3. Restart the server.	
Error Type	**Response** (A)
Key Term Mismatch	1. Identify the process with `netstat -anp`. 2. Stop the process. 3. Restart the server.
Step Missing	1. Identify the process with `netstat -tulnp`. 2. Restart the server. (Missing step 2 in ground truth)
Step Reversal	1. Restart the server. (This should be the last step) 2. Identify the process with `netstat -tulnp`. 3. Stop the process.

actions too early may render later steps ineffective, then failing to resolve the issue.

3) **Step Missing** occurs when the response omits the necessary steps required to resolve the issue. This often happens when relevant paragraphs are not retrieved or when LLM_{QA} fails to recognize implicit but necessary steps. A missing step can disrupt the resolution process, making it difficult for users to reach a successful outcome.

These errors significantly impact the reliability of technical support QA systems by causing misleading guidance, failed troubleshooting attempts, and increased resolution time.

C. Evaluation Pipeline

Our evaluation pipeline consists of two phases, **ClozeFact** and **StepRestore**, designed to detect the three types of errors previously discussed in the generated response A. Each phase assesses A from a specific perspective of accuracy and identifies corresponding errors. The final accuracy score S is computed using our scoring strategy based on the detected errors. In this section, we detail the design and implementation of these two phases.

Phase 1. ClozeFact

To ensure precise verification of key terms in the response A, we introduce an evaluation method inspired by the cloze test. First, key terms are extracted from the ground truth GT using predefined rules. These terms include critical elements such as commands, file paths, and URLs, which are essential for correctly executing each step. Misidentifying or omitting these terms could lead to erroneous operations, such as modifying the wrong file or executing an invalid command.

Next, each extracted key term is replaced with a placeholder of the form ⟨BLANK [ID]⟩, producing a masked version of GT. This ensures that the evaluation does not rely on direct string matching but instead requires semantic understanding. The masked GT is then presented to LLM_{eval}, which is prompted to fill in the blanks based on A or return "Unanswerable" if the key term is missing. This design forces LLM_{eval} to explicitly extract key terms from A rather than relying on prior knowledge.

We use the following prompt to instruct LLM_{eval}:

> *Given the provided text, replace each placeholder ⟨BLANK ∗⟩ with the corresponding key term based on the given response. If the required information is not explicitly mentioned, return "Unanswerable". Ensure that the filled terms exactly match those in the reference.*

This approach enforces precise fact verification by requiring LLM_{eval} to reconstruct key terms rather than simply checking for their presence. Unlike direct boolean matching, which may overlook subtle errors or fail to detect minor discrepancies, this cloze-style verification method forces LLM_{eval} to extract key terms directly from A, ensuring alignment with GT. By actively reconstructing key terms, LLM_{eval} minimizes reliance on prior knowledge, reducing hallucinations and improving reliability in technical support QA evaluation.

Phase 2. StepRestore

To ensure that the response A follows the correct logical order of steps, we introduce an approach inspired by ordering tasks. The ordered steps in GT are first shuffled to create a randomized step list, with each step assigned a unique uppercase letter as an identifier. The shuffled steps are then presented to LLM_{eval}, which must reconstruct the correct execution order by selecting and arranging only the steps explicitly mentioned in A. This prevents the model from inferring missing steps based on prior knowledge and ensures that only the steps present in A contribute to the evaluation.

We use the following prompt to accomplish this task:

> *Based on the provided text, identify and arrange the mentioned steps in the correct logical execution order. Only include steps explicitly stated in the text, and ignore any steps not mentioned, as they are misleading options. Only use the steps listed in the given options.*

By requiring LLM_{eval} to reconstruct the step sequence rather than merely verifying individual steps, this approach ensures that LLM_{eval} captures logical dependencies within the response. Unlike treating the response as an unordered set of atomic facts, which may overlook misordered steps, this approach ensures that steps are presented in the correct order and remain complete, preventing potential issues when users follow the steps to resolve the issue. As a result, this phase reliably verifies both the step order and the step completeness of the response.

D. Scoring

After both evaluation phases are completed, the detected errors are aggregated into an error set, which is then converted into a final score reflecting the accuracy of the response generated by the QA system.

By default, we employ a strict binary scoring strategy: if any error is detected, the response receives a score of 0; otherwise,

it is assigned 1. This approach reflects the critical nature of technical support QA, where even a single mistake—such as an incorrect command, a missing step, or a misordered step—can cause troubleshooting failures or unintended system behavior. Given the importance of accuracy in this domain, this conservative strategy ensures that only fully reliable responses are considered valid.

Beyond this strict mode, our evaluation framework supports customization based on detected errors. For instance, alternative strategies can penalize step omissions or reversals with a score of 0 while proportionally scoring responses based on precisely matched key terms. This flexibility allows adaptation to different evaluation needs while maintaining reliability.

IV. Experiments

This section evaluates our proposed evaluation framework, TechSupportEval, with the following research questions:

1) **RQ1**: How effective is our evaluation framework, and what is the impact of its design choices?
2) **RQ2**: How does our evaluation framework perform across different LLM_{eval}s?
3) **RQ3**: How efficient is our evaluation method, and how much does it cost to run?

We first describe the experimental setup in Section IV-A, including datasets, baselines, and evaluation metrics. Section IV-B presents results demonstrating the effectiveness of our evaluation framework, followed by an ablation study in Section IV-C. In Section IV-D, we analyze the impact of different LLM_{eval}s on performance, while Section IV-E evaluates the efficiency and computational cost. Finally, Section IV-F discusses lessons learned and known limitations.

A. Experimental Setup

1) Dataset: To comprehensively evaluate our proposed evaluation framework, we first examined existing technical support QA datasets [1], [29]. However, we found no dataset specifically designed for benchmarking evaluation methods. To address this gap, we constructed a benchmark dataset based on TechQA, the most comprehensive publicly available technical support QA dataset. We generated responses using multiple QA systems with varying capability levels and obtained human expert annotations for their true accuracy.

TechQA is a domain-adaptation QA dataset tailored for the technical support domain. It contains a total of 910 question-answer pairs collected from the IBM Developer Forum. Among these, 610 questions are labeled as answerable. We filtered questions that requested actionable instructions and obtained 282 valid questions. To ensure consistency, we manually standardized the ground truths into step-by-step instructions. Table II shows the statistics of the filtered dataset.

To evaluate the performance of our evaluation method across QA systems with varying capability levels, we implemented three RAG-based QA systems using LangChain [20], each leveraging a different foundation model: (1) GPT-4o Mini, (2) LLaMA 3 (70B), and (3) LLaMA 3 (8B). All three QA systems utilize `RecursiveTextSplitter` for document chunking and FAISS as the vector database for retrieval. Each QA system was used to generate responses for all 282 filtered questions. We then conducted a human evaluation with 5 domain experts, who annotated each response for accuracy. Table III presents the accuracy results based on human evaluation, showing a clear correlation between model size and QA performance. This benchmark provides a solid foundation to benchmark our proposed evaluation framework.

TABLE II
Statistics of the Filtered TechQA Dataset

Metric	Value
Number of Questions	282
Avg. Length of Questions	366.48
Avg. Length of Ground Truths	220.87
Avg. Length of Reference Documents	4844.93
Avg. Steps in Ground Truths	2.04
Max. Steps in Ground Truths	14

TABLE III
Accuracy of Different RAG-Based QA Systems

QA System (LLM_{QA})	**Accuracy**
GPT 4o Mini	0.8440
LLaMA 3 (70B)	0.7092
LLaMA 3 (8B)	0.5284

2) Baselines: We evaluate our approach against 10 baselines, including lexical-based, semantic-based, and LLM-based methods. These methods include all available approaches that align with our problem setting and can be implemented.

Lexical methods, such as ROUGE [15] and BLEU [16], and the semantic-based BERTScore [17], primarily measure text similarity but cannot capture step structures.

LLM-based methods can be categorized into two groups. The first group (G-Eval [19], LangChain Eval. [20], LlamaIndex Eval. [21]) relies on few-shot prompting but is sensitive to prompt variations and lacks interpretability. The second group consists of fact extraction and verification methods (RAGAS [23], RAGQuestEval [23], RefChecker [27]), which validate extracted facts against the ground truth.

To ensure fairness, all LLM-based evaluation methods default to using `GPT-4o-mini-2024-07-18` in experiments unless explicitly specified, as it is OpenAI's latest model that balances cost and performance.

In our implementation, we normalize the output scores of all baselines to $[0, 1]$ scale. For RefChecker [27], since it only provides the number of Entailment, Neutral, and Contradictory triplets without a default scoring strategy, we adopt a scoring strategy tailored for the technical support QA setting. Specifically, we compute the proportion of Entailment and Neutral triplets among all extracted triplets.

3) Metrics: To evaluate the performance of TechSupportEval and baselines, we employ the following metrics:

AUC (Area Under the ROC Curve): AUC measures a model's ability to distinguish between accurate and inaccurate responses. It represents the area under the Receiver Operating

TABLE IV
COMPARISON OF RESULTS ON DIFFERENT EVALUATION METHODS

Type	Method	LLM of Evaluated QA Systems					
		GPT 4o Mini		LLaMA 3 (70B)		LLaMA 3 (8B)	
		AUC	Pearson r	AUC	Pearson r	AUC	Pearson r
Lexical-based	ROUGE-1	0.5321	0.0311	0.5420	0.0648	0.5288	0.0484
	ROUGE-L	0.5631	0.0872	0.5932	0.1615	0.5752	0.1554
	BLEU	0.6061	0.1138	0.6252	0.1940	0.6158	0.1959
Semantic-based	BERTScore	0.6584	0.2243	0.6793	0.2892	0.6894	0.3095
LLM-based	LangChain Eval.	0.6608	0.4034	0.6310	0.3525	0.7015	0.4431
	LlamaIndex Eval.	0.6651	0.3061	0.6849	0.4117	0.7899	0.5131
	RAGAS	0.6728	0.1934	0.6894	0.2730	0.6544	0.2531
	RAGQuestEval	0.7416	0.3546	0.7205	0.3768	0.6899	0.3380
	G-Eval	0.8233	0.5192	0.8169	0.5419	0.8532	0.6109
	RefChecker	0.8348	0.4627	0.8313	0.5493	0.8309	0.5862
LLM-based	TECHSUPPORTEVAL	0.9109	**0.6616**	**0.8876**	**0.7430**	**0.8970**	**0.7938**
	w/o **ClozeFact**	0.8486	0.4641	0.8463	0.5752	0.8323	0.5914
	w/o **StepRestore**	**0.9129**	0.5669	0.8517	0.5884	0.8693	0.6635

Characteristic (ROC) curve, which plots the true positive rate (TPR) against the false positive rate (FPR):

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}.$$

Pearson Correlation Coefficient (r): Pearson's r quantifies the correlation between model-assigned and human-annotated scores. A value closer to 1 indicates stronger alignment with human judgments.

These metrics offer a comprehensive evaluation that does not depend on a fixed threshold. AUC ranges from 0 to 1, while Pearson's r ranges from -1 to 1, with higher values indicating better performance.

B. Effectiveness

We evaluate the effectiveness of TECHSUPPORTEVAL by comparing it with existing evaluation methods. Table IV shows that TECHSUPPORTEVAL consistently outperforms all baselines in identifying inaccurate responses for technical support QA. It achieves the highest AUC and Pearson correlation, surpassing the previous state-of-the-art, RefChecker [27], by 7.6% on a benchmark dataset for evaluating GPT 4o Mini, demonstrating its superior evaluation performance.

Our analysis reveals key limitations in existing methods. Lexical-based approaches perform poorly as they rely on shallow similarity and fail to capture step-by-step structures, making them ineffective for technical support QA evaluation.

Evaluation methods based on LLM-generated judgments, such as G-Eval, LangChain Eval, and LlamaIndex Eval, rely entirely on the model's inherent reasoning capabilities. While sometimes effective, these methods lack interpretability and offer limited control over the evaluation process.

Fact extraction and verification approaches, including RAGAS and RAGQuestEval, assess accuracy by extracting and verifying key facts but struggle with fact granularity, making it difficult to determine relevant facts. RefChecker improves granularity with triplet-based verification but still faces challenges in verifying step order and completeness.

These findings highlight the advantages of TECHSUPPORTEVAL in technical support QA evaluation.

C. Ablation Study

To assess the contribution of each phase in our evaluation framework, we implement two ablated variants: one without **ClozeFact** and one without **StepRestore**. In these settings, if the output contains only errors related to the disabled phase, the evaluation simplifies to fact extraction and verification.

Table IV shows that while both ablated versions outperform previous methods, they are less effective than the full approach. This confirms that **ClozeFact** and **StepRestore** both contribute significantly to performance.

Notably, on the dataset where GPT 4o Mini serves as the QA system, the variant without **StepRestore** achieves a slightly higher AUC. This is because GPT 4o Mini performs the best among the three models and exhibits fewer Step Missing or Step Reversal errors. As a result, disabling **StepRestore** has minimal impact on evaluation performance for this dataset.

D. Impact of LLM_{eval}

We evaluate TECHSUPPORTEVAL against the top three evaluation methods among the baselines across four foundation models as LLM_{eval} to analyze the impact of LLM_{eval} choice on overall effectiveness.

Table V (on the next page) shows that TECHSUPPORTEVAL consistently outperforms all baselines, demonstrating robustness to variations in LLM_{eval} and indicating that our framework is less constrained by model selection.

Evaluation effectiveness depends on LLM_{eval} capability. Weaker LLMs lack the reasoning and factual verification needed for reliable evaluation, causing a significant performance drop in methods with loosely defined fact granularity, such as RAGAS. While TECHSUPPORTEVAL also declines, the drop is smaller.

TABLE V
IMPACT OF LLM_{eval} ON LLM-BASED EVALUATION METHODS

LLM_{eval}	Method	LLM of Evaluated QA Systems					
		GPT 4o Mini		LLaMA 3 (70B)		LLaMA 3 (8B)	
		AUC	Pearson r	AUC	Pearson r	AUC	Pearson r
GPT 4o Mini	RAGAS	0.6728	0.1934	0.6544	0.2531	0.6894	0.2730
	RAGQuestEval	0.7416	0.3546	0.6899	0.3380	0.7205	0.3768
	RefChecker	0.8348	0.4627	0.8309	0.5862	0.8313	0.5493
	TECHSUPPORTEVAL	**0.9109**	**0.6616**	**0.8970**	**0.7938**	**0.8876**	**0.7430**
Claude 3.5 Haiku	RAGAS	0.7548	0.3489	0.7495	0.4285	0.7301	0.3767
	RAGQuestEval	0.7368	0.3337	0.7483	0.4319	0.7287	0.4197
	RefChecker	0.8132	0.4826	0.7704	0.4956	0.7681	**0.5391**
	TECHSUPPORTEVAL	**0.8651**	**0.5701**	**0.8029**	**0.6083**	**0.7737**	0.5332
LLaMA 3.3 70B	RAGAS	0.7705	0.3240	0.7579	0.4387	0.7066	0.3303
	RAGQuestEval	0.7807	0.3881	0.7056	0.3680	0.7394	0.4198
	RefChecker	0.8094	0.4533	0.7426	0.4256	0.7471	0.4492
	TECHSUPPORTEVAL	**0.8395**	**0.5021**	**0.8237**	**0.6464**	**0.7859**	**0.5538**
Qwen 2.5 72B	RAGAS	0.5199	0.0146	0.6919	0.3283	0.6017	0.1628
	RAGQuestEval	0.7342	0.3193	0.7481	0.4300	0.6746	0.2885
	RefChecker	0.7951	0.4041	0.7800	0.5033	0.7894	0.5339
	TECHSUPPORTEVAL	**0.8234**	**0.4954**	**0.8004**	**0.6013**	**0.8080**	**0.6002**

These findings confirm that TECHSUPPORTEVAL adapts well across different LLM_{eval}, though stronger models yield better performance, ensuring more accurate assessment in technical support QA.

E. Efficiency and Cost Analysis

LLM-based evaluation methods typically involve notable computation time and cost. To assess the trade-offs between effectiveness, cost, and time, we measure the elapsed time and the total cost for each LLM-based evaluation method. The results are visualized in a 2D plot, with the X-axis representing the mean elapsed time (in seconds), the Y-axis representing the performance (AUC), and the circle radius indicating the cost.

TABLE VI
COMPARISON OF EFFICIENCY AND COST ON LLM-BASED METHODS

Method	AUC (avg.)	Time (sec.)	Cost (10^{-3}\$)
LangChain Eval.	0.6644	8.85	0.30
RAGAS	0.6722	23.55	2.37
LlamaIndex Eval.	0.7133	**2.09**	**0.13**
RAGQuestEval	0.7173	8.18	0.39
G-Eval	0.8311	8.41	**0.13**
RefChecker	0.8323	4.06	0.45
TECHSUPPORTEVAL	**0.8985**	2.43	0.31

Table VI and Figure 3 show that TECHSUPPORTEVAL not only outperforms other methods in evaluation effectiveness but also strikes a balanced trade-off between time and cost. Compared to other LLM-based evaluation methods, it offers an efficient and economical approach.

These findings demonstrate the practicality of our evaluation framework as a cost-effective and time-efficient solution for large-scale evaluation tasks in technical support QA.

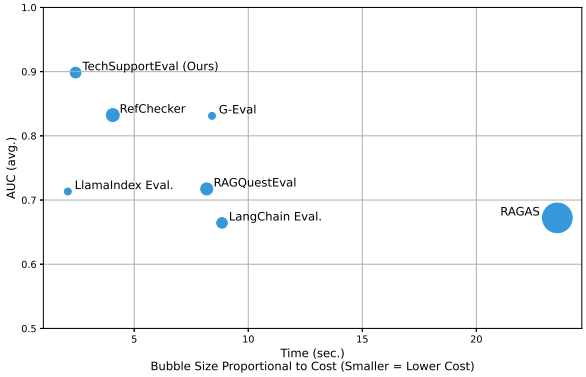

Fig. 3. Comparison of AUC, time, and cost on LLM-based evaluation methods. Smaller bubbles indicate lower costs. TECHSUPPORTEVAL achieves strong performance with competitive efficiency and cost.

F. Discussion

1) Lessons Learned: Our results highlight key differences between LLM-based evaluation methods and demonstrate why our approach performs better. Methods such as G-Eval rely on LLMs' implicit reasoning capabilities but lack interpretability and are sensitive to prompt variations. In contrast, works like RefChecker improve transparency by verifying atomic facts, but they struggle with the granularity of facts and ensuring the correct order of steps. Our approach, which formulates evaluation as fact-based answering, ensures consistency in key terms, step order, and completeness, leading to reliable evaluation in technical support QA.

2) Known Limitations: Our evaluation framework primarily focuses on questions that seek actionable instructions, but technical support responses often contain valuable informative content as well. Future work will explore how to evaluate the completeness and helpfulness of such content.

Additionally, RAG-based QA systems involve many design choices, making it challenging to cover all possible variations. We evaluate three representative solutions with different levels of QA system capabilities, but future research could assess the effectiveness of TECHSUPPORTEVAL across a broader range of system designs and configurations.

V. CONCLUSION

We propose TECHSUPPORTEVAL, an automated evaluation framework for technical support QA, designed to systematically improve the reliability of QA systems in this domain. TECHSUPPORTEVAL addresses three key challenges: (1) Key Term Matching, (2) Step Order Verification, and (3) Step Completeness Verification, using two novel techniques: (1) **ClozeFact**, which formulates fact verification as a cloze test, leveraging an LLM to fill in key terms for accurate key term matching, and (2) **StepRestore**, which shuffles ground truth steps and uses an LLM to reconstruct actionable instructions in the correct order, ensuring step order and completeness.

For a comprehensive evaluation, we introduce a benchmark dataset based on the publicly available TechQA dataset, featuring responses generated by QA systems of varying capabilities. Our results show that TECHSUPPORTEVAL achieves an AUC of 0.91, outperforming the previous state-of-the-art by 7.6%, while remaining competitive in terms of efficiency and cost. Additionally, TECHSUPPORTEVAL demonstrates stable performance across different choices of LLM_{eval}, maintaining its superiority over other methods regardless of model selection. To support future research, we release our code and dataset at https://github.com/NetManAIOps/TechSupportEval.

ACKNOWLEDGEMENTS

This work is supported by the Tsinghua-ZTE Joint Research Fund, the National Natural Science Foundation of China (62202445), and the National Natural Science Foundation of China-Research Grants Council (RGC) Joint Research Scheme (62321166652).

REFERENCES

[1] V. Castelli, R. Chakravarti, S. Dana, A. Ferritto, R. Florian, M. Franz, D. Garg, D. Khandelwal, S. McCarley, M. McCawley, *et al.*, "The techqa dataset," *arXiv:1911.02984*, 2019.

[2] Wikipedia, "Technical support — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Technical%20support&oldid=1269780780, 2025. [Online; accessed 29-January-2025].

[3] Y. Li, Q. Miao, J. Geng, C. Alt, R. Schwarzenberg, L. Hennig, C. Hu, and F. Xu, "Question answering for technical customer support," in *Natural Language Processing and Chinese Computing: 7th CCF International Conference, NLPCC 2018, Hohhot, China, August 26–30, 2018, Proceedings, Part I 7*, pp. 3–15, Springer, 2018.

[4] L. Zhang, A. Hu, J. Zhang, S. Hu, and Q. Jin, "Mpmqa: multimodal question answering on product manuals," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 13958–13966, 2023.

[5] A. Nandy, S. Sharma, S. Maddhashiya, K. Sachdeva, P. Goyal, and N. Ganguly, "Question answering over electronic devices: A new benchmark dataset and a multi-task learning based qa framework," *arXiv:2109.05897*, 2021.

[6] W. Yu, L. Wu, Y. Deng, R. Mahindru, Q. Zeng, S. Guven, and M. Jiang, "A technical question answering system with transfer learning," in *Proc. EMNLP 2020, System Demonstrations*, pp. 92–99, 2020.

[7] M. Hardalov, I. Koychev, and P. Nakov, "Towards automated customer support," in *Artificial Intelligence: Methodology, Systems, and Applications: 18th International Conference, AIMSA 2018, Varna, Bulgaria, September 12–14, 2018, Proceedings 18*, pp. 48–59, Springer, 2018.

[8] 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:2303.18223*, 2023.

[9] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv:2312.10997*, 2023.

[10] Z. Xu, M. J. Cruz, M. Guevara, T. Wang, M. Deshpande, X. Wang, and Z. Li, "Retrieval-augmented generation with knowledge graphs for customer service question answering," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2905–2909, 2024.

[11] O. Cederlund, S. Alawadi, and F. M. Awaysheh, "Llmrag: An optimized digital support service using llm and retrieval-augmented generation," in *2024 9th International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 54–62, IEEE, 2024.

[12] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, *et al.*, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Transactions on Information Systems*, 2024.

[13] V. Rawte, A. Sheth, and A. Das, "A survey of hallucination in large foundation models," *arXiv:2309.05922*, 2023.

[14] G. Agrawal, T. Kumarage, Z. Alghamdi, and H. Liu, "Mindful-rag: A study of points of failure in retrieval augmented generation," *arXiv:2407.12216*, 2024.

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

[16] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

[17] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with BERT," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.

[18] G. Bekoulis, C. Papagiannopoulou, and N. Deligiannis, "A review on fact extraction and verification," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–35, 2021.

[19] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, "G-eval: Nlg evaluation using gpt-4 with better human alignment," *arXiv:2303.16634*, 2023.

[20] H. Chase, "LangChain," Oct. 2022.

[21] J. Liu, "LlamaIndex," Nov. 2022.

[22] S. Min, K. Krishna, X. Lyu, M. Lewis, W.-t. Yih, P. Koh, M. Iyyer, L. Zettlemoyer, and H. Hajishirzi, "FActScore: Fine-grained atomic evaluation of factual precision in long form text generation," in *Proc. EMNLP 2023*, pp. 12076–12100, 2023.

[23] S. Es, J. James, L. Espinosa Anke, and S. Schockaert, "RAGAs: Automated evaluation of retrieval augmented generation," in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pp. 150–158, 2024.

[24] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, "Ares: An automated evaluation framework for retrieval-augmented generation systems," *arXiv:2311.09476*, 2023.

[25] Y. Lyu, Z. Li, S. Niu, F. Xiong, B. Tang, W. Wang, H. Wu, H. Liu, T. Xu, and E. Chen, "Crud-rag: A comprehensive chinese benchmark for retrieval-augmented generation of large language models," *ACM Transactions on Information Systems*, 2024.

[26] Y. Zhao, J. Zhang, I. Chern, S. Gao, P. Liu, and J. He, "Felm: Benchmarking factuality evaluation of large language models," *NeurIPS*, vol. 36, pp. 44502–44523, 2023.

[27] X. Hu, D. Ru, L. Qiu, Q. Guo, T. Zhang, Y. Xu, Y. Luo, P. Liu, Y. Zhang, and Z. Zhang, "Knowledge-centric hallucination detection," in *Proc. EMNLP 2024*, pp. 6953–6975, 2024.

[28] Y. Xu, T. Cai, J. Jiang, and X. Song, "Face4rag: Factual consistency evaluation for retrieval augmented generation in chinese," in *Proc. ACM SIGKDD 2024*, pp. 6083–6094, 2024.

[29] F. Yang, P. Zhao, Z. Wang, L. Wang, B. Qiao, J. Zhang, M. Garg, Q. Lin, S. Rajmohan, and D. Zhang, "Empower large language model to perform better on industrial domain-specific question answering," in *Proc. EMNLP 2023, Industry Track*, pp. 294–312, 2023.