Eagle: Leveraging Operations Documents for Comprehensive Benchmark Question Generation

Yuhe Liu
lyh23@mails.tsinghua.edu.cn
Tsinghua University & BNRist
Beijing, China

Changhua Pei
Hang Wang
chpei@cnic.cn
wanghang@cnic.cn
CNIC, CAS
Beijing, China

Longlong Xu
xull23@mails.tsinghua.edu.cn
Tsinghua University & BNRist
Beijing, China

Xiaogang Dong
Zhen Feng
dongxiaogang@huawei.com
fz1995tvxq@outlook.com
Huawei
Xi'an, China

Li Zheng
Kehang Ji
zhengli@caict.ac.cn
jikehang@caict.ac.cn
CAICT
Beijing, China

Dan Pei*
peidan@tsinghua.edu.cn
Tsinghua University & BNRist
Beijing, China

Abstract

The rapid growth in scale and complexity of modern software systems has intensified the need for intelligent and reliable IT operations. While Artificial Intelligence for IT Operations (AIOps) addresses some challenges, existing solutions predominantly rely on isolated, task-specific models that struggle with interpreting multimodal data, incur high maintenance costs, and lack sufficient transparency. Operations Large Language Models (OpsLLMs) offer unified, knowledge-rich reasoning capabilities, yet their evaluation faces significant barriers, including the absence of Ops-centric evaluation taxonomies, limited availability of public datasets, simplistic question-generation methods, and inadequate quality standards for comprehensive operations tasks.

We present Eagle, a comprehensive benchmarking framework tailored for evaluating OpsLLMs. Deployed inside Huawei, Eagle ingests enterprise product documentation and synthesizes 4,845 domain-grounded QA pairs across logs, metrics, traces, and configurations, which are paired with a standardized model evaluation system. This deployment supported multiple evaluations of OpsLLM and culminated in an internal horizontal benchmarking report that informed model selection and rollout decisions. Methodologically, Eagle (i) defines an operations-centric taxonomy aligning core LLM abilities with end-to-end operations tasks; (ii) implements an automated question-generation pipeline with multi-granular quality controls validated by human annotation; and (iii) provides reproducible evaluation suites and metrics for scenario-driven reasoning. In offline studies, Eagle-generated test suites improve expert-rated rubric scores by 22%–49% over state-of-the-art baselines, enabling more precise assessments of anomaly detection, fault diagnosis, and root-cause analysis abilities in OpsLLMs. To foster community adoption and reproducibility, we open-source the framework[1] and a sanitized dataset[2]. By bridging general LLM evaluation and operations practice, Eagle delivers a deployable foundation for advancing large-model applications in AIOps.

CCS Concepts

• **Computer systems organization** → **Availability**; • **Computing methodologies** → **Natural language generation**; • **Software and its engineering** → **Software creation and management**.

Keywords

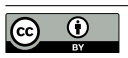
Operations, Large Language Model, Benchmark, Dataset, Question Generation

ACM Reference Format:
Yuhe Liu, Changhua Pei, Hang Wang, Longlong Xu, Xiaogang Dong, Zhen Feng, Li Zheng, Kehang Ji, and Dan Pei. 2026. ***Eagle***: Leveraging Operations Documents for Comprehensive Benchmark Question Generation. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 5–9, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3803437.3805214

1 Introduction

IT Operations (Ops) is crucial for the reliable functioning of complex software systems, including cloud computing, 5G networks, and financial platforms. As software system scale and complexity rapidly expand, traditional operations face significant challenges. To address this, artificial intelligence-assisted operations (AIOps) have emerged [12], utilizing AI for key software engineering tasks like anomaly detection, fault diagnosis, and performance optimization. However, current AIOps implementations struggle with multimodal data and sufficient domain-specific knowledge, face increased complexity and maintenance costs due to multiple specialized models.

*Dan Pei is the corresponding author.



FSE Companion '26, Montreal, QC, Canada

https://doi.org/10.1145/3803437.3805214

[1] https://github.com/NickLennonLiu/eagle_code/
[2] https://github.com/NickLennonLiu/eagle_data/

To overcome these limitations, Operations Large Language Models (OpsLLMs) have emerged, leveraging the advanced natural language processing capabilities of general-purpose large language models (LLMs). OpsLLMs incorporate domain-specific operations knowledge through pre-training and fine-tuning, enabling them to effectively handle specialized operations queries and complex tasks such as fault diagnosis and root cause analysis.

Despite recent progress, evaluating OpsLLMs remains challenging. Rigorous evaluation is crucial to determine their readiness for deployment, guide model improvements, and support informed adoption. However, general-purpose LLM benchmarks do not align with the unique demands of operations tasks.

Ops Taxonomy. There is a significant gap in evaluation dimensions tailored specifically to Ops scenarios. Current benchmarks emphasize general linguistic abilities like fluency and coherence, while OpsLLMs require assessment metrics explicitly designed to reflect their proficiency in domain-specific capabilities, including anomaly detection accuracy, fault diagnosis precision, and efficacy in root cause analysis. Bridging this gap necessitates establishing a set of standardized evaluation criteria tailored explicitly to operations tasks. **Data Availability.** Constructing high-quality Ops evaluation datasets is difficult due to the sensitive, specialized nature of the data. Moreover, Ops data is often multimodal—logs, metrics, traces, configs—making dataset creation costly and labor-intensive. **Task Complexity.** Existing dataset generation methods are overly simplistic, limiting their effectiveness in rigorously testing the advanced capabilities of OpsLLMs. Traditional automated methods frequently rely on shallow heuristics or template-based approaches, which fail to capture complex real-world operations scenarios adequately. The oversimplifications hinder the accurate assessment of whether an OpsLLM can handle nuanced operational challenges. **Quality Standards for Question Generation.** Current question generation and evaluation methodologies lack suitable standards for long-form, scenario-based operations tasks. In practical operations environments, tasks often require detailed, structured responses involving multiple steps, considerations of context, and integration of multimodal information.

To overcome these challenges, this paper introduces ***Eagle***, a comprehensive benchmarking framework explicitly designed for evaluating OpsLLMs. We established a standardized operations taxonomy for evaluation that aligns closely with real-world Ops tasks; We developed an automated question generation method based on Ops documentation, ensuring deeper assessment of OpsLLM capabilities. By defining quality evaluation standards for scenario-based QA tasks, we assess the quality of generated questions using our method and other baselines. To support adoption and reproducibility, we release the framework and a sanitized dataset.

Industrial Deployment. Beyond offline experiments, ***Eagle*** has been *deployed inside Huawei for six months upon submission* as part of a company-internal data–evaluation loop. The documentation-driven pipeline synthesized **4,845** domain-grounded QA pairs, which feed a standardized *Model Evaluation System*. Using these assets, we conducted internal model comparisons across abilities, Ops phases, and data modalities, and delivered *horizontal benchmarking reports* that informed model selection and rollout decisions. While this deployment demonstrates practicality in one enterprise setting, broader cross-organization validation remains future work.

The contributions of our paper are summarized as follows:

- We propose a systematic evaluation framework, ***Eagle***,that integrates both the characteristics of operations tasks and the core capabilities of large models, providing a structured approach to assess their effectiveness in real-world AIOps scenarios.
- We introduce a novel automatic QA generation method based on operations documentation, making it a valuable tool for evaluating LLMs in operations tasks.
- We design rigorous evaluation methodology to assess the quality of automatically generated QA datasets. Through both automated quality metrics and human annotations, we demonstrate that our method produces higher-quality questions compared to baseline approaches.

2 Related Works

2.1 OpsLLM

Recent advances in OpsLLM demonstrate how infusing general-purpose LLMs with domain-specific operations knowledge can tackle complex IT operations tasks at scale. MonitorAssistant [24] transforms raw monitoring data into fault reports using prompt selection, intent recognition, and anomaly detection. D-Bot [27] performs end-to-end fault diagnosis by combining tool use, time-series analysis, and report generation. RCAgent [20] treats root-cause analysis as a multi-step process involving action selection and tool invocation. OWL [9] provides a QA interface to retrieve and synthesise O&M knowledge. DivLog [22] parses semi-structured logs into structured records via LLM-based information extraction.

2.2 LLM Benchmark

While numerous efforts apply LLMs to O&M scenarios, the corresponding evaluation resources remain fragmented. NetOps[14] contributes a network-engineering dataset that emphasise protocol configuration and troubleshooting. LogEval[3] measures a model's ability to interpret, cluster, and summarise logs, stressing robustness to noisy and domain-specific formats. OpsEval[13] integrates these strands into a broader, cross-domain collection covering finance, telecom, and cloud services; it introduces the FAE-Score to quantify answer quality without human grading, making large-scale comparison of OpsLLMs feasible.

2.3 QA Generation

Automated QA generation is an efficient way to construct instructional datasets without expensive manual annotation. Bonito [15] synthesises diverse QA pairs from unlabelled technical documents via meta-prompts, while 5G Instruct Forge [11] converts 3GPP specifications into an instruction set for expert-level 5G O&M queries.

Evaluation of QA-generation systems typically relies on human ratings of fluency and relevance [2, 4, 8, 26]; task-supportive works further assess answerability[2] and answer consistency[8], while studies focused on linguistic quality analyse paraphrase, multi-fact, and topic-selection errors [5]. Educational datasets such as FairyTaleQA[23] additionally measure diversity of question types, drawing connections to model bias and content safety.

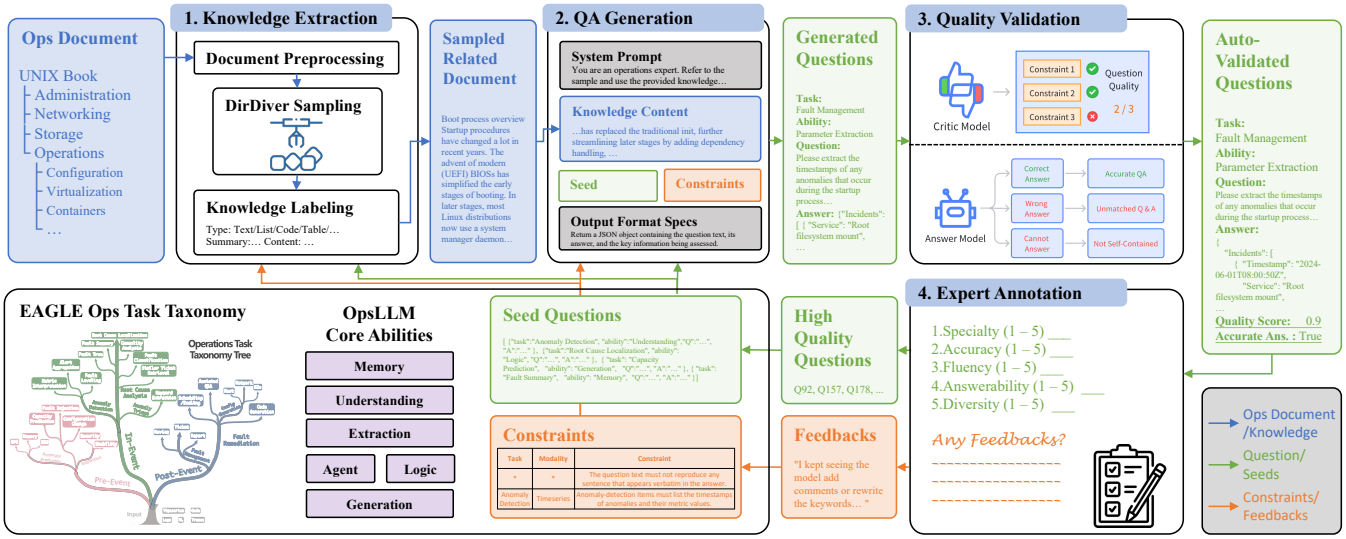


Figure 1: Overall Framework of *Eagle*. The figure illustrates the pipeline from document processing to QA generation and validation, highlighting key stages and interactions. The blue color indicates the flow of Ops document and the knowledge within. The green color indicates the general flow of seed questions and generated questions. The orange color indicates the flow of expert feedbacks and corresponding constraints in QA generation.

3 *Eagle* Benchmark

This section focuses on establishing a scientific and rigorous evaluation framework, devising specialized metrics and methods covering tasks such as alert analysis, log processing, and fault diagnosis, to ensure accurate and reliable model assessment across diverse Ops scenarios. Specifically, our design principles include:

(1) Integrating general LLM evaluation methodologies with Ops characteristics, we propose the Ops evaluation framework ***Eagle***, abstracting essential capabilities into five dimensions: Extraction, Agent, Generation, Logic, and Understanding.

(2) Establishing an evaluation system structured as "Ops Task × Ops Modality" allowing performance assessment of Ops LLMs on specific tasks or modalities, and meeting targeted requirements from different companies or Ops contexts.

3.1 Abilities for OpsLLM

Survey [18] proposes a framework for evaluating general-purpose LLMs across four dimensions: **Understanding**, **Memory**, **Generation**, and **Logic**, corresponding respectively to semantic comprehension, contextual memory, generative capability, and logical reasoning. Building upon this framework, we extend the evaluation scope to accommodate AIOps-specific characteristics. In particular, we introduce an additional **Agent** dimension to assess task planning and tool usage, and an **Extraction** dimension to evaluate parameter extraction, intent recognition, and instruction parsing, which are crucial in AIOps, where prompts are often generated with structured templates rather than natural expressions. Furthermore, we expand the original Understanding dimension to cover comprehension of logs, time series, and traces. The complete structure of our ***Eagle*** evaluation framework is shown in Table 1.

Table 1: Core Abilities of OpsLLM

Core Ability	Sub-abilities
Understanding	Text, Log, Time Series, Trace
Memory	Knowledge, Context
Generation	Text, Log, Configuration, Code
Agent	Task Planning, Tool Selection, Tool Invocation, Task Decomposition
Logic	Logical Reasoning, Mathematical Calculation
Extraction	Parameter Extraction, Intent Understanding, Format Compliance

3.2 Ops Tasks

In Figure 1, we illustrate various Ops tasks categorized according to the three Ops stages—pre-event, in-event, and post-event—based on multimodal inputs. The **pre-event** stage primarily focuses on anomaly detection and failure prediction, encompassing tasks such as fault injection, capacity forecasting, and routine inspection. The **in-event** stage emphasizes fault diagnosis and root cause localization, including tasks like alert classification, root cause analysis, and similar ticket retrieval. The **post-event** stage covers fault management and assistive question-answering, including tasks such as event report generation, ticket management, configuration generation, and fault remediation.

3.3 Data Modalities

In the domain of Ops, a broad range of textual data modalities is available for OpsLLMs . In our benchmark, we consider six key data modalities commonly encountered in AIOps scenarios: **Natural**

Language: Ops manuals, documentation, and alert messages providing essential task context and knowledge. **Logs:** Semi-structured or unstructured records capturing system and application events, crucial for anomaly detection and root cause analysis. **Time Series:** Structured metrics (e.g., CPU, memory, latency) sampled over time for monitoring and forecasting. **Traces:** Execution paths of transactions or requests, used for analyzing bottlenecks and dependencies. **Configuration:** System and application settings; vital for understanding failures due to misconfigurations. **Code:** Scripts and automation tools (e.g., deployments, probes) that encode operational procedures.

Table 2: Examples of the six data modalities in IT Operations

Modality	Example
Natural Language	`If a pod enters CrashLoopBackOff, first run kubectl describe pod <name> to inspect recent events and verify the image pull policy.`
Logs	`[2025-05-10 08:32:15] ERROR 502#0: *12345 upstream timed out (110: Connection timed out) while connecting to upstream, client: 10.0.0.5, server: api.example.com`
Time Series	`0.15, 0.12, 0.17, 0.14, 0.20, 0.18, 0.22, 0.19, 0.21, . . .`
Traces	`checkout → payment (28 ms) → inventory (12 ms) → shipping (8 ms)`
Config	`upstream backend { server 10.0.0.10 weight=5; server 10.0.0.11 weight=5; }`

Code	`#/usr/bin/env python3;`
	`import psutil, subprocess;`
	`if psutil.cpu_percent() > 90:`
	`  subprocess.run(["systemctl",`
	`"restart","web.service"])`

Table 2 shows examples for the modalities described above. These six modalities reflect the diversity and richness of data in practical IT operations. By supporting and evaluating OpsLLMs across these varied inputs, EAGLE ensures comprehensive assessment of their capabilities in realistic and complex operations environments.

3.4 Seed Questions

To clearly guide LLMs in generating appropriate questions aligned with our benchmark design, we adopted methodologies from [19] and [21] to create initial seed questions using GPT-4. These seed questions exemplify the desired format, complexity, and scope based on the defined core capabilities of OpsLLMs. Additionally, specific constraints ensures generated questions effectively cover relevant data modalities and task types. Example constraints include:

- **General**: The question description must not contain the answer.
- **Task (Anomaly Detection)**: Questions must include the timestamp and specific cause of the anomaly.
- **Ability (Task Decomposition)**: Knowledge points must involve procedural or multi-step instructions.
- **Modality (Trace)**: Trace content must reflect temporal correlation and modular invocation relationships.

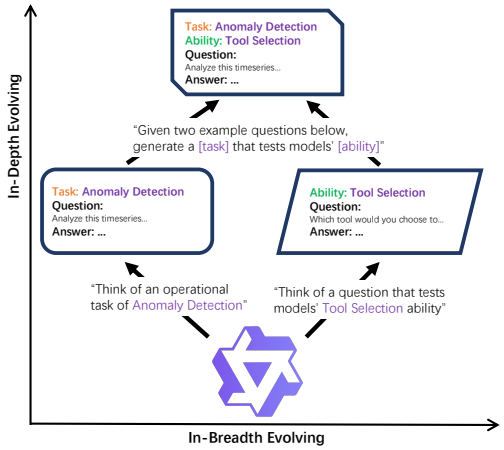


Figure 2: Illustration of the seed QA generation process.

Our empirical study indicated that directly providing constraints across all three dimensions simultaneously often results in questions failing to meet each dimension's classification requirements. Inspired by the approach in [21], we first independently generated seed questions for each individual dimension (In-Breadth Evolving). Subsequently, pairs of seed questions from two different dimensions served as in-context examples, enabling GPT-4 to produce questions covering combinations across multiple dimensions (In-Depth Evolving). Figure 2 illustrates an example of this multi-dimensional seed QA generation process.

In total, we generated 4252 seed questions and defined 55 generation constraints.

4 Automated Ops QA Generation

Figure 1 illustrates the basic workflow of the ***Eagle*** QA automated generation framework. It takes Ops Documents and predefined classification dimensions as input, and undergoes four main stages to complete an iteration of QA generation, ultimately producing high-quality Ops task instructions based on original Ops documents.

4.1 Knowledge Extraction

The first stage of the ***Eagle*** framework involves knowledge extraction from Ops documents. This stage accomplishes three tasks: First, documents of various formats (PDF, HTML, ZIP) are preprocessed to extract textual data while preserving the document's directory structure. Second, given a target category from the ***Eagle*** classification dimensions, we use seed questions corresponding to that category to guide an LLM through the document hierarchy, progressively identifying the most relevant textual content. Finally, we leverage an LLM to summarize knowledge from the extracted text, annotating each piece of information with its data type and content summary.

4.1.1 Document Preprocessing. Ops documents frequently come in PDF and ZIP formats. In this phase, we decompress the documents according to their specific formats to generate a flattened document directory. Hierarchical information is retrieved from directory files

```
You are an operations expert. Your task is to locate content in a
operational document that can evaluate a model's {ability}
capability on the {task} task.

Example Question:
{content}

Document Directory:
{path}

Return a JSON object with the following fields:
{
    "candid_subdirs": ["Document path (must be one of the
    entries in the document directory)", ...],
    "reason": "Why these sub-chapters were chosen"
}
```

Figure 3: Prompt used for DirDiver subdirectory selection.

such as "navi.xml", and the original document hierarchy is restored by creating corresponding subfolders. Additionally, regular expressions are used to eliminate redundant information (headers and footers), and all external references (hyperlinks and citations) are uniformly replaced with standardized tokens.

4.1.2 DirDiver Sampling. To generate Ops questions corresponding to specific task categories, it is crucial to retrieve relevant textual content from the Ops documents. Leveraging the inherent directory structure within Ops documentation, we designed the DirDiver algorithm, which guides an LLM through a hierarchical traversal process to identify textual content closely aligned with target task categories and provided seed questions.

Algorithm 1 illustrates the prompt-guided directory traversal procedure. DirDiver initiates from the document root and iteratively explores subdirectories guided by dynamically generated prompts. At each hierarchical step, if multiple candidate directories exceed a predefined limit k, the algorithm constructs a prompt based on the current query and candidate subdirectories. The LLM then evaluates these candidates and selects the most relevant subdirectory for deeper exploration. The prompt used for subdirectory selection is shown in Figure 3. If no further valid subdirectories exist, or the LLM response is invalid, the algorithm backtracks through previously visited directories to locate a valid file. Ultimately, DirDiver returns a relevant file path, providing targeted textual content for subsequent QA generation tasks. By default, DirDiver sets the candidate subdirectory limit to $k = 5$, the maximum traversal depth to $D = 6$, and the maximum number of steps to $S = 64$.

4.1.3 Knowledge Labeling. Before utilizing the retrieved knowledge for QA generation, we use LLM to summarize the extracted content. This process includes labeling the data types (e.g., List, Code, Table) contained within the text and providing concise knowledge summaries. This step enables the QA generation model to effectively comprehend the content, using the annotated data types to determine whether specific knowledge points are suitable for generating certain types of Ops tasks.

4.2 QA Generation

As depicted in Figure 1, we integrate the previously obtained knowledge content, seed questions, and constraints into a unified prompt, accompanied by a consistent system prompt and explicit output format specifications. This prompt is then processed by an LLM_{gen}

Algorithm 1 DirDiver: Document Layers Traversal

Require: Root path P, content q, prompt template $\mathcal{T}$, LLM model $\mathcal{M}$, candidate limit k
Ensure: Relevant file path f

$path \leftarrow P$, $visited \leftarrow [\,]$
while True **do**
  Append $path$ to $visited$
  $dirs, files \leftarrow$ list subdirectories and valid files in $path$
  if $dirs = \emptyset$ **then**
    return random file from $files$
  end if
  $C \leftarrow$ random sample of up to k elements from $dirs$
  $resp \leftarrow \mathcal{M}.chat(\mathcal{T}(q, C))$
  $d^\star \leftarrow$ pick one from $(resp.\text{candid\_subdirs} \cap C)$ if not empty else random element from C
  $path \leftarrow path/d^\star$
end while
for p in $visited$ (reversed) **do**
  if valid file exists in p **then**
    return random file from p
  end if
end for

```
You are an operations (Ops) expert. Refer to the sample below and use
the provided knowledge-point content to create an Ops question that
assesses those knowledge points, while adhering to all generation
requirements and constraints.

<Sample Question>
{seed_question_formatted}
</Sample Question>

<Knowledge Points>
{kp_content}
</Knowledge Points>

<Requirements & Constraints>
{constraints}
</Requirements & Constraints>

Using the sample question and knowledge-point content above, generate
an {dimension_description} Ops question whose question and answer
align with the document content. If the requested modality conflicts with
the Ops task, or the knowledge points cannot reasonably be assessed,
return an empty string. Otherwise, return a JSON object containing the
question text, its answer, and the key information being assessed:

{
    "question": "Question text",
    "answer": "Corresponding answer",
    "key": "Key information assessed by the question"
}

Do not prefix your response with ```json; output the JSON object directly.
```

Figure 4: Prompt for QA Generation.

to generate QA pairs. Specifically, we instruct the model to simultaneously output the generated question, corresponding answer, and key knowledge points involved in the QA pair. These key knowledge annotations allow experts to subsequently verify the QA pairs against the original knowledge sources.

Fig. 4 illustrates the prompt template utilized in the core QA generation stage. The prompt specifies the question context, task

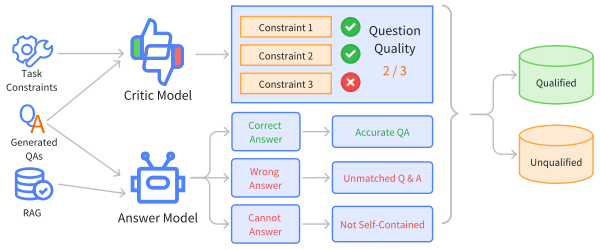


Figure 5: Quality Validation Methodology.

constraints, and expected answer format, which improves consistency and quality across generated QA pairs.

4.3 Quality Validation

Quality validation is a critical step to ensure the generated QAs meet predefined standards. Figure 5 illustrates our quality validation methodology, which includes two main evaluation dimensions: question quality and answer accuracy.

4.3.1 Question Quality. To assess the quality of the generated questions, we employ a dedicated Critic Model that evaluates each question against a set of predefined task constraints. Each constraint represents an essential criterion, such as clarity, specificity, relevance to Ops contexts, or adherence to domain-specific guidelines. Questions are scored based on the number of constraints they fulfill. Only questions that meet a predetermined threshold (e.g., satisfying at least two out of three constraints) are categorized as qualified, with the rest marked as unqualified, requiring further refinement or exclusion from the dataset. The prompts for question quality assessment are shown in Figure 6.

4.3.2 Answer Accuracy. To verify answer accuracy, we utilize an Answer Model in conjunction with retrieval-augmented generation (RAG). We embed the related documents using the BGE-M3 model and perform cosine similarity retrieval with FAISS to support the RAG-based answer verification process. The Answer Model processes generated QA pairs by attempting to independently generate answers based on retrieved knowledge content. Outcomes from the Answer Model evaluation fall into three categories:

- Correct Answer: The Answer Model independently produces a response matching the original QA pair, confirming the QA as accurate.
- Wrong Answer: A mismatch between the original answer and the Answer Model's response indicates inaccuracies, labeling the QA as "Unmatched Q&A."
- Cannot Answer: When the Answer Model fails to response due to insufficient or ambiguous context, the QA is labeled as "Not Self-Contained," suggesting issues with completeness or clarity of the question or supporting knowledge.

QA pairs falling under "Unmatched Q&A" or "Not Self-Contained" are marked as unqualified, highlighting the necessity for revision or removal. This design favors conservative, document-grounded benchmark items: questions that require unstated incident facts or external assumptions are filtered out to reduce unverifiable or

```
You are an operations (Ops) expert. A model has generated a question-and-answer pair from a specific knowledge point. Use the checklist below to evaluate its quality and try to improve the question.

<Knowledge-Point>
{kp_content}
</Knowledge-Point>

<Question>
{question}
</Question>

<Answer>
{answer}
</Answer>

Checklist:
{constraints}

Return a JSON object with two fields:
- score – the ratio of checklist items satisfied to the total number of items
- reason – why the QA pair fails to meet any checklist requirements

Example formats:
{
    "score": 0.5,
    "reason": "The QA pair fails the first checklist item because …"
}
Output the JSON object directly—do not prefix it with ```json.
```

Figure 6: Prompt for quality validation

hallucinated scenarios. The trade-off is that some otherwise useful questions may be rejected because they depend on broader retrieval context or tool support beyond the current validation setup.

5 Experiment

This section evaluates our proposed QA Generation framework, ***Eagle***, with the following research questions:

(1) **RQ1:** How effective is our QA generation framework in producing Ops QAs compared to existing methods?
(2) **RQ2:** How does our QA generation framework perform across different LLM_{gen}s?
(3) **RQ3:** What impact do task constraints and seed questions have on the quality of the generated QA pairs?

We first describe the experimental setup in Section 5.1, including datasets, baselines, and evaluation metrics. Section 5.2 presents results demonstrating the effectiveness of our framework compared to baseline methods, and the model evaulation results on ***Eagle***. An ablation study is conducted in Section 5.3 examining the contributions of specifics components. In Section 5.4, we conduct a case analysis to provide qualitative insights into the strengths and limitations of our framework.

5.1 Experimental Setup

5.1.1 Dataset. Due to the lack of QA datasets grounded in operations (Ops) documents, we curated corpora from publicly available technical sources to support QA generation across methods and domains. Specifically, we chose three representative documents: **UNIX:** *UNIX and Linux System Administration Handbook* (5th Ed.) [10], an authoritative English reference covering system administration procedures and troubleshooting tasks. **Redis:** *Redis Development and Operations* [6], a widely used textbook detailing

database Ops practices and real-world maintenance scenarios. **Zabbix:** *Zabbix Manual* [25], an official guide for the Zabbix monitoring system, reflecting monitoring and observability workflows in practical software Ops.[3] Together, these corpora cover complementary slices of Ops practice: system administration, database operations, and monitoring/observability. Table 3 summarizes key information of these datasets, including volume, the number of directories, predominant data modalities, and domain classification.

For public release, we provide a sanitized dataset rather than raw enterprise documents. Organization-specific identifiers and sensitive incident context are removed or abstracted, while task structure, modality combinations, and reasoning requirements are preserved.

Table 3: Overview of the datasets used for QA generation.

Dataset	Volume	#Directories	Domain
UNIX	197MB	225	Operating Systems
Redis	109MB	142	Database
Zabbix	128MB	165	Monitoring Software

We generate 1000 questions per dataset and per method, using approximately 80% of each dataset's text, as the remainder falls outside the defined Operations taxonomy context. Each experiment randomly samples 100 generated questions per method for comparative analysis, involving both directional pairwise comparisons.

5.1.2 Baselines. To comprehensively evaluate our proposed QA generation framework, we benchmarked against two representative baselines. Bonito[15] is a QA generation framework that synthesizes diverse question styles—including yes/no, extractive, and entailment—directly from unlabelled technical documents using designed meta-prompts. Forge[11], by contrast, focuses on the telecommunications field; it transforms standardized 3GPP specifications into a instructional dataset. These baselines reflect state-of-the-art approaches in automated QA generation and serve as robust points of comparison for our evaluation.

5.1.3 Metrics. We evaluate the quality of the QA pairs using comprehensive metrics that combine rubric-based annotations and comparative assessments.

Rubrics. Experts individually score QA pairs across five dimensions: **1) Specialty** evaluates the Ops expertise required, emphasizing domain-specific knowledge rather than general comprehension. **2) Accuracy** measures the correctness and completeness of the provided answer relative to the question requirements. **3) Fluency** assesses clarity, conciseness, and explicitness of instructions provided in the question. **4) Answerability** determines if sufficient context is provided for a model to accurately answer the question. **5) Diversity** evaluates question variation within a batch of QA pairs rather than individually. Annotators assess diversity based on the variation of output formats requested within a group of five QA pairs. The diversity score also ranges from 1 (no variation) to 5 (complete variation in formats).

[3]In addition to these public corpora, we evaluated our approach on proprietary industrial Ops documents from Huawei. Due to confidentiality constraints, we exclude the raw internal documents from the experiments and public release.

Each question-answer pair should be assessed along the following four dimensions, with scores ranging from 1 (lowest) to 5 (highest):

Specialty (Ops-Specificity): Does the question assess the model's capabilities in operations (Ops) tasks, rather than general knowledge or reading comprehension?
- 1 = General knowledge only, no Ops relevance
- 3 = Some Ops concepts mentioned, but no real skill assessment
- 5 = Clearly tests model performance in complex or realistic Ops scenarios

Accuracy: Does the provided answer correctly and completely address the question?
- 1 = Completely wrong or irrelevant
- 3 = Mostly correct, but missing some details
- 5 = Fully correct, complete, and precise

Fluency: Is the question clearly stated with sufficient context and clear answer format requirements, without being overly brief or verbose?
- 1 = Poorly phrased, unclear meaning or excessive raw text
- 3 = Understandable but needs refinement
- 5 = Clear, concise, and informative with explicit answer instructions

Answerability: Does the question provide enough context for a model to generate an accurate answer?
- 1 = Lacks critical context, nearly impossible to answer
- 3 = Mostly sufficient but with some missing information
- 5 = Fully answerable based on the provided content

Please use these criteria consistently when scoring each QA.

Figure 7: Details of scoring rubrics for expert annotators.

Each dimension is scored independently for every QA pair, using integer ratings from 1 (lowest) to 5 (highest). The detailed scoring criteria for each rubric is listed in Appendix. The aggregated rubric score for each dimension d is calculated as below, where $s_d^{(i)}(q)$ represents the d dimension score given by annotator i on question q, and N is the total number of annotators:

$$\text{Score}_d(q) = \frac{1}{N}\sum_{i=1}^{N} s_d^{(i)}(q) \tag{1}$$

Win Rate. Additionally, we perform pairwise comparisons between QA pairs generated by anonymous methods. Experts select the superior QA pair based on Specialty, Accuracy, Fluency, and Answerability. The *Win Rate* is defined as the percentage of pairwise comparisons where our proposed method's QA pairs are preferred. This metric provides a direct measure of our method's relative effectiveness against baseline approaches.

$$\text{Win Rate} = \frac{\text{Number of wins}}{\text{Total comparisons}} \times 100\% \tag{2}$$

Annotators. All QA pair evaluations were conducted by experienced Ops engineers from industry, each with at least three years of hands-on experience in system administration, monitoring, and incident response. To ensure consistency and reliability in scoring, we provided annotators with detailed scoring rubrics, including example QA pairs for each score level (1 to 5) along every dimension. The detailed rubrics can be found in Fig. 7. Annotators were also asked to provide justifications or feedback when assigning lower scores, especially in cases of disagreement or ambiguity. We recruited 8 annotators to assess the quality of generated QA pairs. Each question underwent annotation by at least three independent experts, with disagreements resolved through majority voting.

Table 4: Performance comparison across datasets. We report average question length in characters (Len.), five expert-rated dimensions—Specialty (Spec.), Accuracy (Acc.), Fluency (Flu.), Answerability (Ans.), and Diversity (Div.)—and total score (Tot.). Win. denotes the pairwise win rate over other methods/models on the same dataset.

Method	Len.	Spec.	Acc.	Flu.	Ans.	Div.	Tot.	Win.
UNIX								
Bonito	7337	2.63	3.75	3.56	3.98	3.33	17.25	35.3%
Forge	68	2.74	3.51	3.21	3.52	1.25	14.23	17.9%
Eagle	322	**3.95**	**4.57**	**4.06**	**4.60**	**4.01**	**21.19**	**96.4%**
Redis								
Bonito	1822	2.82	3.63	3.42	3.90	3.52	17.29	28.8%
Forge	75	2.96	3.89	3.25	3.71	1.49	15.30	22.7%
Eagle	325	**3.93**	**4.52**	**4.07**	**4.58**	**4.01**	**21.11**	**97.5%**
Zabbix								
Bonito	3750	2.70	3.69	3.30	3.83	3.65	17.17	28.8%
Forge	78	2.70	3.46	3.05	3.31	1.57	14.09	22.1%
Eagle	313	**3.90**	**4.57**	**4.01**	**4.60**	**3.99**	**21.07**	**98.5%**

5.2 Results

5.2.1 RQ1 - Comparison with existing methods. Across all three Ops domains (**UNIX**, **Redis**, and **Zabbix**), ***Eagle*** consistently surpasses the two strongest baselines, *Bonito* and *Forge*. From Table 3 and the rubric scores in Table 4, ***Eagle*** exceeds the best baseline by **4.0–6.9** points in the aggregate *Tot.* score. The largest gains are observed on *Specialty* (+1.20) and *Accuracy* (+0.90), showing that our framework produces questions that probe deeper Ops knowledge and yield more precise answers.

Win-rate analysis in Table 4 substantiates these findings: ***Eagle*** wins over baselines in more than **95%** of expert pairwise comparisons, whereas *Bonito* and *Forge* win fewer than **30%**. These results confirm the effectiveness of ***Eagle*** in generating markedly higher-quality Ops QA pairs.

Practical Lesson: A lightweight but carefully crafted prompt pipeline—combining task constraints, domain-aware seed questions, and targeted directory filtering—can lift *Ops specialty* and *answer accuracy* by 20 %+ over state-of-the-art QA generators.

5.2.2 RQ2 - Robustness across LLM_{gen}. By default, we adopt the locally deployed Qwen2.5-72B [17] as the QA generation model (LLM_{gen}), ensuring controllable inference cost and deployment flexibility. To evaluate the robustness of our pipeline across different generator models, we further test with three additional LLMs: GPT-4.1 [16], Gemini 2.5 [7], and Claude 3.7 Sonnet [1].

Table 5: Performance comparison across LLM_{gen}.

LLM_{gen}	Len.	Spec.	Acc.	Flu.	Ans.	Div.	Tot.	Win.
GPT-4	411	**3.98**	4.59	4.05	4.63	**4.05**	21.30	57.7%
Gemini	483	3.96	4.55	3.99	4.64	3.95	21.09	54.7%
Claude	541	3.95	**4.79**	**4.10**	**4.81**	4.03	**21.68**	**66.3%**
Qwen2.5	322	3.95	4.57	4.06	4.60	4.01	21.19	21.3%

Table 5 demonstrate ***Eagle*** maintains high quality regardless of the underlying generator model. All four LLMs attain an aggregate rubric score above 21, with differences within ±0.6 points. Claude-3.7-Sonnet achieves the highest total (21.68), but even the open-sourced Qwen2.5-72B reaches 21.19—only 0.49 behind, indicating that our methodology transfer well to models of diverse sizes and training paradigms.

Practical Lesson: Different base LLMs exhibit noticeable variation in *surface characteristics*—average question length and stylistic flavour—yet a robust prompt-constraint-retrieval pipeline can absorb these differences and drive all models to comparable rubric scores and win-rate performance. In practice, careful pipeline engineering mitigates model-specific quirks, yielding stable QA quality across proprietary and open-source backbones.

5.2.3 Model Evaluation. Table 6 summarizes per-metric scores (0–10) across abilities, Ops phases, and modalities. Memory and Logic are consistently strong across models (often > 9.5 for top-tier systems), suggesting these capabilities are largely saturated. In contrast, Generation is the weakest and most variable dimension, reflecting persistent challenges in schema-compliant and structurally precise output. Agentic behavior and External tool usage perform well overall but show moderate dispersion among mid-sized models.

From an Ops-phase perspective, Post-Event tasks are the most robust across all models, typically exceeding 9.0, suggesting that retrospective diagnosis and summarization are well supported. In-Event performance is competitive but model-dependent, while Pre-Event exhibits the largest variance (e.g., scores ranging from 5.9 to 9.0), highlighting that proactive risk assessment, early triage, and planning remain more sensitive to model capacity and alignment.

In terms of input modalities, models demonstrate reliable processing of Natural Language, Logs, and Configurations. However, a performance cleavage is observed in Time-Series and Trace data, where top-tier models outperform the mid-tier ones. These findings reinforce the conclusion that larger or domain-specialized models achieve more uniform performance. We conclude that the primary focus for future research lies in enhancing structure-aware generation and enhancing Pre-Event proactive capabilities. Future benchmarks should prioritize more stringent schema constraints and adversarial scenarios to better differentiate advanced LLMs.

Practical Lesson: Model comparisons suggest that conventional strengths (memory, logic, post-event analysis) no longer differentiate models. Instead, the decisive gaps emerge in *generation fidelity*, *Pre-Event reasoning*, and *complex modalities*. Effective benchmarks should thus emphasize strict structural constraints and proactive scenarios, where model robustness and autonomy truly diverge.

5.3 Ablation Study

To quantify the contribution of each component, we sequentially remove (1) task constraints, (2) seed questions, and (3) the *DirDiver* document layers traversal method. Table 7 reveal clear performance drops: **1) w/o Constraint** removing constraints mainly hurts *Accuracy* and *Fluency* (-0.29 and -0.19), reducing the total score to 20.70 (-0.49) and lowering the win rate by ≈ 9%. **2) w/o Seed** discarding example seeds causes the sharpest decline in *Specialty*, *Fluency*, and *Diversity*, dropping the total to 19.86 (-1.33) and cutting the win rate almost in half. **3) w/o DirDiver** disabling directory selection

Table 6: Per-metric scores by model (Ability, Ops Task & Modality)

Model	Ability						Ops Phase			Modality					
	Und.	Mem.	Gen.	Age.	Ext.	Log.	Pre	In	Post	NL	Log	TS	Trace	Code	Conf
Pangu-Pro-MOE	9.04	9.83	7.50	9.00	9.75	9.83	8.55	8.42	9.70	8.71	9.64	8.00	9.70	9.38	9.57
Qwen2.5-72B-Instruct	8.62	8.67	7.75	9.33	9.62	9.67	7.82	8.83	9.38	9.07	9.27	7.50	8.80	9.38	9.71
DeepSeek-R1-138	9.42	9.67	6.00	9.11	9.00	9.67	8.36	9.08	9.43	8.43	8.82	9.50	9.80	9.13	9.71
DeepSeek-R1-Distill-Qwen-32B	8.88	9.83	7.00	9.44	8.88	9.92	9.00	8.50	9.35	9.00	8.91	9.00	9.00	9.13	9.43
QwQ-32B	8.71	9.83	7.25	8.89	8.75	9.58	8.73	8.83	9.00	9.00	8.27	8.83	9.70	8.88	9.14
Yi-1.5-34B-Chat	8.33	9.67	7.50	8.78	7.50	8.50	6.55	8.08	9.00	8.43	8.27	7.83	9.20	7.13	8.86
glm-4-9b-chat-hf	7.12	9.67	6.00	9.33	8.00	8.50	5.91	7.83	8.60	7.43	8.36	6.67	9.30	8.88	8.29
InternLM3-8B-Instruct	8.42	7.00	6.25	8.89	9.00	9.00	8.00	7.42	8.80	7.86	8.91	6.83	8.30	8.50	8.86
GLM-4-32B-0414	8.88	7.67	7.00	8.56	8.62	9.75	8.09	8.00	9.12	8.14	9.36	7.50	8.90	8.63	8.86
Meta-Llama-3-70B-Instruct	8.88	8.67	7.50	9.11	9.12	9.67	8.73	7.92	9.38	8.57	8.91	8.83	9.20	9.50	9.00

mainly impacts *Specialty* and *Answerability*; the total falls to 19.71 (-1.48). **4) Naive Prompt** omitting all three components yields the weakest variant, confirming the additive benefit of each module.

Table 7: Results of ablation study on our method. "↓" marks a drop in score compared to the full method, indicating a degradation in quality.

Variant	Spec.	Acc.	Flu.	Ans.	Div.	Tot.
Eagle	3.95	4.57	4.06	4.60	4.01	21.19
w/o Constraint	3.90	4.28↓	3.87↓	4.61	4.04	20.70
w/o Seed	3.77↓	4.39	3.78↓	4.45	3.47↓	19.86
w/o DirDiver	3.53↓	4.57	4.07	3.62↓	4.02	19.71
Naive Prompt	3.59↓	4.44↓	3.99↓	3.47↓	3.15↓	18.64

Overall, seed questions provide the largest single boost, while constraints and DirDiver further refine quality. These observations answer **RQ3**: each component is essential, and their synergy enables ***Eagle*** to outperform competitive baselines substantially.

Practical Lesson: Component ablations reveal a clear hierarchy of impact—*seed questions* provide the biggest single boost, but the full benefit comes only when combined with explicit *task constraints* and *DirDiver*-guided retrieval.

5.4 Case Analysis

In this section, we present and analyze representative examples of generated QA pairs from different methods, highlighting their relative strengths and weaknesses. We further summarize common issues and insights gained from these examples.

Figure 8 shows four QA pairs generated by Bonito, with the corresponding document context omitted for brevity. The examples illustrate that Bonito generally produces simple questions, including true-or-false judgments, generic question generation, and fill-in-the-blank formats. This indicates that Bonito's generation strategy primarily relies on basic and general domain questions. Occasionally, it produces meaningful relevant questions (e.g., Question 1); however, due to the simplicity of its question types, many questions have limited relevance to Ops contexts (e.g., Questions 2 and 3) or are entirely unrelated to Ops tasks (e.g., Question 4).

Figure 9 displays two QA pairs generated by Forge, highlighting its primary focus on evaluating the model's memory and knowledge recall capabilities through short-answer questions. Forge successfully generates a meaningful question related to operating system concepts in Question 1. However, due to Forge's reliance on vectorization and similarity-based retrieval mechanisms, it occasionally generates questions that the model itself cannot answer, rendering some QA pairs ineffective (e.g., Question 2).

Question 1: Which of the following is an advantage of centralized logging? Yes or no, is the answer "Having a well-defined, site-wide logging strategy."?
Answer: Yes

Question 2: What would be the best title for the text?
Yes or no, is the answer "Recommended vendors"?
Answer: Yes

Question 3: Write a multi-choice question for the following article, with the given choices and answer:
Options:
A www.dockerbook.com
B docs.docker.com
C O'Reilly Media
D container-solutions.com/blog
Answer: A www.dockerbook.com
Answer: If you want to read a book about Docker, go to _ .

Question 4: Question: After the end of the story, the author probably:
The answer to the above question is
Answer: is happy.

Figure 8: Examples of questions generated by Bonito

Question 1: What is the purpose of using `cp --reflink` in Btrfs?

Answer: The `cp --reflink` command in Btrfs is used to create a copy-on-write (COW) clone of a file. When the file is initially copied, it does not consume additional disk space because it references the same data blocks as the original. Additional space is only used when the cloned file is modified, thus optimizing storage usage and improving performance.

Question 2: How has the role of information technology in business and daily life evolved over the past four decades?

Answer: I don't have specific information about the evolution of the role of information technology in business and daily life over the past four decades based on the given context.

Figure 9: Examples of questions generated by Forge

Task: Code Generation **Ability:** Timeseries Understanding
Modality: Time Series
Question: Write a Python script to analyze time-series resource usage data from a given server and identify system performance bottlenecks. Assume you have a CSV file named server_metrics.csv with the following contents:
Timestamp,CPU(%),Memory(%),Disk IO (Read MB/s),Network IO (MB/s)
2024-06-01 10:00:00,45,67,12,5
2024-06-01 10:05:00,92,68,11,5
…
Please complete the following tasks:
1. Read the file and analyze resource usage at each timestamp.
2. For each timestamp, determine the primary performance bottleneck based on the following rules:
   - If CPU usage exceeds 90% and all other metrics are below 80%, the bottleneck is CPU.
   - …

Output a new CSV file named performance_bottlenecks.csv with the following format:
Timestamp,Bottleneck

Figure 10: An example of generated QA pair by *Eagle*

Task: Fault Management
Ability: Parameter Extraction
Modality: Text
Question: Here is a passage from an operations manual on password management. Based on it, document a security incident caused by the failure to regularly inspect the /etc/shadow file. Output a JSON object with the incident time, location, affected systems or users, and impact scope. Do not include root cause analysis or mitigation steps.

Excerpt:
Poor password management is a common vulnerability ... the encrypted password field in /etc/shadow should contain * or ! to prevent login.

Answer: {
"incident_time": "2024-06-18 08:30",
"incident_location": "Production server containing the /etc/shadow file",
"affected_system": "Company's core business server",
"affected_user": "daemon (pseudo-user)",
"impact_scope": "Due to the lack of ... risk to the overall server."
}

Figure 11: An example of a relatively low-quality generated QA pair, which sheds light on potential pitfalls in *Eagle*.

An example of a high-quality QA pair generated by our method ***Eagle*** (Figure 10) involves analyzing server resource usage data to identify performance bottlenecks. The scenario is well-defined, presenting clear and detailed conditions for identifying bottlenecks (CPU, memory, disk IO, and network IO), each accompanied by explicit thresholds. The task incorporates multiple data modalities, specifically numerical time-series data, and clearly outlines the expected output format, ensuring that the generated solution is both Ops relevant and directly actionable. This clarity in scenario, modality, and expected outcomes significantly enhances the quality and practical applicability of the generated QA pair.

Conversely, an example of a lower-quality QA pair generated by our method (Figure 11) addresses documenting a security incident based on password management practices described in an Ops manual. The primary issue with this task is its inherent requirement for real-world data input, which the document-derived scenario fails to provide. Consequently, the generated answer includes fabricated details about incident timing, location, and impact, undermining the credibility and realism of the scenario. This limitation emphasizes the necessity for QA generation tasks to align closely with the provided inputs to avoid unrealistic or unverifiable outputs.

Practical Lesson: A high-quality QA pair must include clearly defined scenarios, explicitly detailed modalities, and precise output requirements. It is crucial that the input context supports the generation of realistic and verifiable outputs to ensure the practical relevance and credibility of the generated QA.

6 Threats to Validity

External Validity: Our study evaluates documentation-driven QA generation under a unified taxonomy and pipeline. The evidence covers three public Ops corpora and one enterprise deployment setting; more specialized domains or noisier documents may require additional task definitions and constraints.

Internal and Construct Validity: Automatic screening combines a Critic model and a RAG-based Answer model, so imperfect constraint checking, retrieval failure, or answer-model mismatch can misclassify qualified items. Our "Not Self-Contained" rule favors document-grounded QA pairs, but may introduce false negatives for questions that would become answerable under broader retrieval context.

7 Conclusion

We introduced ***Eagle***, a benchmarking framework explicitly designed for evaluating OpsLLMs in realistic operational scenarios, addressing critical gaps in current methodologies:

1) Operations Taxonomy. A standardized taxonomy aligning core OpsLLM capabilities—understanding, extraction, generation, logic, and agent interactions—with real-world operational tasks across diverse, multimodal contexts (logs, metrics, traces, configurations). **2) Automated QA Generation**. An advanced pipeline leveraging domain-specific seed questions, targeted constraints, and structured document retrieval (DirDiver) to enhance the relevance, complexity, and specificity of generated QA pairs. Our QA pairs outperform state-of-the-art methods (Bonito, Forge) by 22%–49% in overall rubric scores, particularly excelling in operations specificity, accuracy, and diversity. **3) Evaluation Methodology**. Rigorous validation combining automated metrics and expert annotations ensures high-quality QA pairs essential for meaningful evaluation. Comprehensive ablations across multiple LLMs confirm the robustness and consistency of ***Eagle***.

Eagle has been *deployed inside Huawei for six months upon submission.* The framework has produced 4,845 domain-grounded QA pairs. Using these assets, we conducted internal evaluations of OpsLLMs and delivered benchmarking reports that informed model-selection and rollout decisions in one enterprise setting.

In conclusion, ***Eagle*** provides a foundation for benchmarking OpsLLMs with a direct path to industrial impact. Future work includes extending supported data modalities and operational contexts, broadening cross-organization validation, and encouraging community-driven enhancements.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (No.2024YFB4505903), National Natural Science Foundation of China (No.62202445), and the Beijing National Research Center for Information Science and Technology (BNRist) key projects.

References

[1] Anthropic. 2025. Claude 3.7 Sonnet and Claude Code. https://www.anthropic.com/news/claude-3-7-sonnet. Accessed: 2025-05-12.

[2] Woon Sang Cho, Yizhe Zhang, Sudha Rao, Asli Celikyilmaz, Chenyan Xiong, Jianfeng Gao, Mengdi Wang, and Bill Dolan. 2021. Contrastive Multi-document Question Generation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (Eds.). Association for Computational Linguistics, Online, 12–30. doi:10.18653/v1/2021.eacl-main.2

[3] Tianyu Cui, Shiyu Ma, Ziang Chen, Tong Xiao, Shimin Tao, Yilun Liu, Shenglin Zhang, Duoming Lin, Changchang Liu, Yuzhe Cai, and others. 2024. Logeval: A comprehensive benchmark suite for large language models in log analysis. *arXiv preprint arXiv:2407.01896* (2024).

[4] Bidyut Das, Mukta Majumder, Santanu Phadikar, and Arif Ahmed Sekh. 2021. Multiple-choice question generation with auto-generated distractors for computer-assisted educational assessment. *Multimedia Tools Appl.* 80, 21–23 (Sept. 2021), 31907–31925. doi:10.1007/s11042-021-11222-2

[5] Nan Duan, Duyu Tang, Peng Chen, and Ming Zhou. 2017. Question Generation for Question Answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Martha Palmer, Rebecca Hwa, and Sebastian Riedel (Eds.). Association for Computational Linguistics, Copenhagen, Denmark, 866–874. doi:10.18653/v1/D17-1090

[6] Lei Fu and Yijun Zhang. 2017. *Redis Development and Operations.* China Machine Press.

[7] Google DeepMind. 2025. Gemini 2.5: Our Most Intelligent AI Model. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/. Accessed: 2025-05-12.

[8] Jing Gu, Mostafa Mirshekari, Zhou Yu, and Aaron Sisto. 2021. ChainCQG: Flow-Aware Conversational Question Generation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (Eds.). Association for Computational Linguistics, Online, 2061–2070. doi:10.18653/v1/2021.eacl-main.177

[9] Hongcheng Guo, Jian Yang, Jiaheng Liu, Liqun Yang, Linzheng Chai, Jiaqi Bai, Junran Peng, Xiaorong Hu, Chao Chen, Dongfeng Zhang, xu Shi, Tieqiao Zheng, liangfan zheng, Bo Zhang, Ke Xu, and Zhoujun Li. 2024. OWL: A Large Language Model for IT Operations. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=SZOQ9RKYJu

[10] Trent R. Hein, Evi Nemeth, Garth Snyder, Ben Whaley, and Dan Mackin. 2017. *UNIX and Linux System Administration Handbook* (5th ed.). Addison-Wesley Professional.

[11] Azzedine Idir Ait Said, Abdelkader Mekrache, Karim Boutiba, Kostas Ramantas, Adlen Ksentini, and Moufida Rahmani. 2025. 5G INSTRUCT Forge: An Advanced Data Engineering Pipeline for Making LLMs Learn 5G. *IEEE Transactions on Cognitive Communications and Networking* 11, 2 (2025), 974–986. doi:10.1109/TCCN.2024.3516055

[12] Andrew Lerner. 2017. AIOps Platforms—Gartner.

[13] Yuhe Liu, Changhua Pei, Longlong Xu, Bohan Chen, Mingze Sun, Zhirui Zhang, Yongqian Sun, Shenglin Zhang, Kun Wang, Haiming Zhang, Jianhui Li, Gaogang Xie, Xidao Wen, Xiaohui Nie, Minghua Ma, and Dan Pei. 2024. OpsEval: A Comprehensive IT Operations Benchmark Suite for Large Language Models. http://arxiv.org/abs/2310.07637 arXiv:2310.07637.

[14] Yukai Miao, Yu Bai, Li Chen, Dan Li, Haifeng Sun, Xizheng Wang, Ziqiu Luo, Dapeng Sun, Xiuting Xu, Qi Zhang, Chao Xiang, and Xinchi Li. 2023. An Empirical Study of NetOps Capability of Pre-Trained Large Language Models. *CoRR* abs/2309.05557 (2023). https://doi.org/10.48550/arXiv.2309.05557

[15] Nihal V. Nayak, Yiyang Nan, Avi Trost, and Stephen H. Bach. 2024. Learning to Generate Instruction Tuning Datasets for Zero-Shot Task Adaptation. In *Findings of the Association for Computational Linguistics: ACL 2024*.

[16] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).

[17] QwenLM. 2023. QwenLM/Qwen-7B. https://github.com/QwenLM/Qwen-7B

[18] Ke Wang, Jiahui Zhu, Minjie Ren, Zeming Liu, Shiwei Li, Zongye Zhang, Chenkai Zhang, Xiaoyu Wu, Qiqi Zhan, Qingjie Liu, and Yunhong Wang. 2024. A Survey on Data Synthesis and Augmentation for Large Language Models. http://arxiv.org/abs/2410.12896 arXiv:2410.12896.

[19] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-Instruct: Aligning Language Models with Self-Generated Instructions. In *The 61st Annual Meeting Of The Association For Computational Linguistics*. doi:10.18653/v1/2023.acl-long.754

[20] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2023. Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 4966–4974.

[21] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. WizardLM: Empowering Large Pre-Trained Language Models to Follow Complex Instructions. In *International Conference on Learning Representations (ICLR)*.

[22] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2024. DivLog: Log Parsing with Prompt Enhanced In-Context Learning. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.

[23] Ying Xu, Dakuo Wang, Mo Yu, Daniel Ritchie, Bingsheng Yao, Tongshuang Wu, Zheng Zhang, Toby Jia-Jun Li, Nora Bradford, Branda Sun, Tran Bao Hoang, Yisi Sang, Yufang Hou, Xiaojuan Ma, Diyi Yang, Nanyun Peng, Zhou Yu, and Mark Warschauer. 2022. Fantastic Questions and Where to Find Them: FairytaleQA – An Authentic Dataset for Narrative Comprehension. In *ACL 2022*. Association for Computational Linguistics.

[24] Zhaoyang Yu, Minghua Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, Saravan Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, and others. 2024. Monitorassistant: Simplifying cloud service monitoring via large language models. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 38–49.

[25] Zabbix Documentation Team. 2018. *Zabbix Documentation 4.0 (Chinese Version)*. Zabbix. https://www.zabbix.com/documentation/4.0/zh/manual Accessed: 2025-05-05.

[26] Zhenjie Zhao, Yufang Hou, Dakuo Wang, Mo Yu, Chengzhong Liu, and Xiaojuan Ma. 2022. Educational Question Generation of Children Storybooks via Question Type Distribution Learning and Event-centric Summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 5073–5085. doi:10.18653/v1/2022.acl-long.348

[27] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2023. D-bot: Database diagnosis system using large language models. *arXiv preprint arXiv:2312.01454* (2023).