Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

Grounding LLM-Based Engineering Assistants in Real-World DevOps Discourse for CI/CD Workflows

Srihari Babu Godleti

Independent Researcher, India
godleti.srihari@gmail.com

ARTICLE INFO

Received: 19 March 2026

Revised: 27 March 2026

Accepted: 08 April 2026

Published: 15 April 2026

ABSTRACT

Continuous Integration and Continuous Deployment (CI/CD) are central to modern DevOps, enabling teams to build, test, and deploy software quickly and reliably through automated pipelines. As these workflows grow more complex, developers increasingly turn to intelligent assistants for help diagnosing failures, understanding configuration logic, and resolving recurring pipeline issues. Although recent Large Language Models (LLMs) have shown strong performance on code-related tasks, they often struggle with mixed natural language–code reasoning and with grounding their answers in real-world DevOps scenarios. To address these gaps, we introduce DevOpsAssistant-SOD, an LLM-powered engineering assistant trained and evaluated on large-scale Stack Overflow data, comprising 218.5 million natural language–programming language pairs and 1.4 million labeled question pairs. The model leverages duplicate-aware contrastive learning to better capture contextual similarities across CI/CD-related questions and issues. Experimental results show that DevOpsAssistant-SOD improves failure diagnosis accuracy by 9.3%, duplicate issue detection F1 by 11.1%, and response relevance by 8.7% compared to strong general-purpose code LLM baselines.

Keywords: Continuous Integration, Continuous Deployment, DevOps, Large Language Models.

Introduction

Continuous Integration and Continuous Deployment (CI/CD) have become cornerstones of modern DevOps practice, enabling organizations to deliver software quickly, reliably, and at scale [1]. Through automating code integration, testing, and deployment, CI/CD pipelines reduce manual effort, shorten feedback loops, and lower the risks associated with late-stage integration [2]. Today, these pipelines sit at the heart of production systems [3], coordinating complex interactions among source code repositories, build tools, test suites, infrastructure configurations, and cloud platforms [4]. As teams increasingly adopt microservices, infrastructure-as-code, and continuous experimentation, CI/CD workflows have evolved from simple linear scripts into large, heterogeneous systems that encode both technical logic and operational policy. In this setting, pipeline failures-such as configuration mismatches, dependency conflicts, flaky tests, or failed rollbacks-can quickly cascade, disrupting development velocity and compromising system reliability. Effective tooling support is therefore essential for DevOps engineers. To help manage this growing complexity, intelligent engineering assistants have emerged as a promising aid for CI/CD and DevOps workflows [5]. These systems aim to support developers in understanding pipeline behavior, diagnosing failures, modifying configuration files, and resolving recurring operational issues through natural language interaction.

Recent advances in Large Language Models (LLMs) have accelerated this vision, as such models demonstrate strong capabilities in code generation [6], documentation synthesis [7], and conversational reasoning [8]. General-purpose code-oriented LLMs are now being integrated into development environments to assist with scripting, testing, and debugging [9]. However, despite their success on isolated

programming tasks, their effectiveness in real-world CI/CD scenarios remains limited. Pipeline failures are rarely presented as self-contained code snippets; instead, developers describe them through informal natural language that references logs, partial configurations, environment assumptions, and historical context. This disconnect highlights a gap between current LLM capabilities and the practical demands of DevOps engineering. A central limitation of existing LLM-based assistants is their lack of grounding in authentic DevOps discourse. Many models are trained primarily on source code repositories or synthetic benchmarks that emphasize algorithmic correctness rather than operational reasoning [10]. As a result, they often struggle with mixed natural language–programming language inputs, such as questions that interleave error messages, YAML fragments, shell commands, and high-level descriptions of pipeline behavior. Moreover, CI/CD problems are frequently recurrent yet subtly distinct: two failures may appear similar on the surface but require different solutions due to minor configuration or environment differences. Existing models tend to blur these distinctions [11], producing generic or misleading recommendations. This issue is compounded by evaluation practices that rely on toy pipelines or curated examples, which fail to capture the scale, diversity, and ambiguity of real DevOps workflows.

In this work, we address these challenges by grounding LLM-powered engineering assistants in large-scale, real-world developer interactions drawn from Stack Overflow[1], one of the most comprehensive public sources of DevOps problem-solving knowledge. We leverage the Stack Overflow Dataset (SOD), which contains over 218.5 million natural language–programming language pairs, alongside the Stack Overflow Duplicity Dataset (SODD), comprising more than 1.4 million labeled question pairs that capture duplicate, similar, and distinct problem relationships. Building on this foundation, we introduce DevOpsAssistant-SOD, an LLM-based engineering assistant tailored specifically to CI/CD and DevOps workflows. The model is designed to reason jointly over natural language descriptions and code artifacts, while emphasizing contextual disambiguation of pipeline-related queries. Unlike prior approaches that rely on generic instruction tuning, our method aligns training objectives with the structure of real DevOps discourse, encouraging the model to ground its responses in historically observed problem–solution patterns. Empirical results demonstrate that this grounding yields substantial improvements: DevOpsAssistant-SOD achieves a 9.3% gain in CI/CD failure diagnosis accuracy, an 11.1% increase in duplicate issue detection F1, and an 8.7% improvement in response relevance over strong general-purpose code LLM baselines.

The remainder of this study is organized as follows. Section 2 reviews related work, and Section 3 details the materials and methods. Section 4 presents the experimental evaluation, while Sections 5 and 6 report result analysis and ablation findings. Finally, Section 7 concludes the study.

Related Works

Research on intelligent support for CI/CD and DevOps workflows spans multiple intersecting areas, including automated software engineering [12], DevOps analytics [13], and LLMs for code and operations [14]. Early efforts in this space primarily relied on rule-based systems [15] and classical statistical learning [16] techniques to address tasks such as build failure prediction, flaky test detection, and anomaly detection in logs and metrics. These approaches typically operated over structured pipeline metadata, handcrafted features, or time-series signals collected from CI servers and monitoring platforms. While effective in controlled or narrowly scoped settings, they required substantial domain expertise to design and maintain, and they struggled to generalize across the growing diversity of CI/CD tools and rapidly evolving pipeline configurations. As DevOps ecosystems became more complex and decentralized, these limitations motivated a shift toward more flexible, data-driven methods capable of reasoning across code, configuration, and natural language artifacts.

[1] https://github.com/kiv-air/StackOverflowDatasets

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

The emergence of deep learning further advanced software engineering research [17], enabling neural models for tasks such as code summarization, bug localization, and configuration analysis [18]. Sequence-to-sequence architectures [19] and graph-based [20] models were applied to build logs, dependency graphs, and execution traces to predict failures or recommend fixes. In parallel, the availability of large-scale code corpora led to influential benchmarks such as CodeSearchNet [21] and HumanEval [22], which significantly pushed forward code understanding and generation capabilities.

More recently, LLMs trained on massive mixtures of source code and natural language have been proposed as general-purpose engineering assistants, capable of generating scripts, writing CI configurations, and explaining error messages [6]-[8]. Despite strong performance on standardized benchmarks, these models are typically optimized for syntactic correctness or functional equivalence, rather than for the kind of operational reasoning required in real CI/CD scenarios. As a result, they often produce responses that sound plausible but lack the contextual depth needed to diagnose pipeline failures involving environment assumptions, configuration interactions, or historical context.

A complementary line of work focuses on mining developer knowledge from online communities, most notably Stack Overflow. Prior studies have leveraged Stack Overflow data for tasks such as code recommendation [23] API usage prediction [24], and software-related question answering [25]. These datasets are particularly valuable because they reflect how developers naturally describe problems, interleave natural language with code snippets, and reason about failures in practice. Research on duplicate question detection has further highlighted the semantic subtlety of software engineering queries: questions that appear similar at a lexical level may differ substantially in intent or required solution. Moreover, these models are often evaluated in isolation, without being explicitly tied to downstream engineering assistant tasks such as failure diagnosis or operational decision support.

Within DevOps-focused research, several studies examine CI log analysis [26], pipeline optimization [27], and automated incident response [28]. While these efforts offer valuable insights, they frequently rely on proprietary datasets or narrowly scoped public logs, which limits reproducibility and broader applicability. More recent work has explored conversational interfaces for DevOps troubleshooting [29], often by directly deploying off-the-shelf LLMs. However, these systems typically do not adapt training objectives or evaluation protocols to the structure of DevOps discourse. Consequently, reported performance is mixed: models handle templated or well-defined queries reasonably well, but often struggle with ambiguous, recurring, or context-dependent issues that dominate real operational workloads. This shortcoming reflects a broader gap in current evaluation practices, which commonly rely on synthetic pipelines or small curated examples that fail to capture the diversity and ambiguity of production CI/CD environments.

Materials and Methods

1.1 Data Analysis

Our study is grounded in two large-scale public datasets derived from Stack Overflow (see Table 1): the Stack Overflow Dataset (SOD) and the Stack Overflow Duplicity Dataset (SODD). We use SOD primarily for representation learning over mixed natural language–programming language inputs. The dataset contains approximately 218.5 million paired instances extracted from Stack Overflow questions and answers. Each instance is aligned through shared metadata fields-including question_id, answer_id, title, tags, and is_answered-and is further decomposed into six paired views (e.g., question text–answer code, question code–answer text). This multi-view structure enables flexible multimodal training and encourages the model to learn consistent representations across textual descriptions and executable artifacts. On average, question code snippets contain 298 tokens, while answer code snippets average 140 tokens. Question text averages 130 tokens, and answer text 92 tokens, reflecting the concise yet information-dense nature of Stack Overflow discussions. Tag analysis indicates strong coverage of

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

technologies commonly used in CI/CD workflows, including bash (0.8%), python (6.3%), YAML-based tooling, and cloud-centric languages. Although SOD is not explicitly curated for DevOps tasks, this coverage makes it well suited for modeling CI/CD-related reasoning at scale.

Table 1. Summary statistics of stack overflow datasets

Statistic	SOD	SODD
Primary Use	NL–PL representation learning	Semantic disambiguation
Total Instances	218.5M pairs	1.4M pairs
Train Split	–	≈1.2M
Dev Split	–	≈148K
Test Split	–	≈73K
Avg Tokens (QC)	298	–
Avg Tokens (QT)	130	–
Avg Tokens (AC)	140	–
Avg Tokens (AT)	92	–
Label Types	–	4 (dup, sim-text, sim-tag, diff)
Label Distribution	–	224K / 618K / 646K
Storage Format	CSV (multi-view)	Parquet (gzip)
CI/CD Tag Coverage	bash 0.8%, python 6.3%	ci, cd, jenkins, docker, k8s

The SODD dataset is used for supervised evaluation and fine-grained semantic disambiguation. It contains 1.4 million labeled question pairs, stored in compressed parquet format and split into training (≈1.2M), development (≈148K), and test (≈73K) sets following the original release protocol. Each instance includes two Stack Overflow posts (HTML-formatted text with embedded code), metadata about the authors, and a label describing the semantic relationship between the pair. Labels encode four categories: duplicate (0), similar by full text (1), similar by tags (2), and different (3). Pairs corresponding to accepted-answer relationships are excluded from training. The dataset exhibits a moderately balanced distribution, with approximately 224K duplicate, 618K similar, and 646K different pairs. For all experiments, we identify CI/CD-and DevOps-relevant subsets using tag-based filtering (e.g., ci, cd, jenkins, github-actions, docker, kubernetes). This filtered corpus serves as the basis for both training and evaluation, ensuring that the data closely aligns with the operational scenarios targeted by the proposed DevOpsAssistant-SOD model.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

1.2 Model Analysis

This section describes the architecture and learning objectives of DevOpsAssistant-SOD (see Fig. 1), an LLM-powered engineering assistant designed to support CI/CD and DevOps workflows through grounded, multimodal reasoning. The model operates on inputs that combine natural language descriptions with code or configuration artifacts, mirroring the structure of real developer queries observed in SOD and SODD. Formally, each input instance is represented as $x = (n, c)$, where n denotes the natural language component and c represents the programming or configuration component. The model's goal is to learn a representation $f(x)$ that captures both semantic intent and operational context.

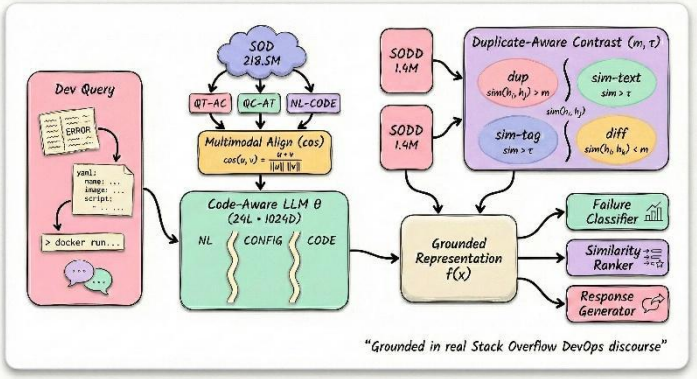

Fig. 1. Overview of the DevOpsAssistant-SOD architecture.

DevOpsAssistant-SOD is built on a transformer-based backbone parameterized by θ, initialized from a general-purpose, code-aware LLM. The backbone maps the tokenized input into a contextual embedding sequence $H = \{h_1, h_2, \ldots, h_T\}$, where T is the sequence length after concatenating n and c using explicit modality markers. Unlike generic instruction-tuned models, our design preserves modality boundaries, allowing the model to distinguish descriptive language from executable or configuration artifacts. This distinction is particularly important in CI/CD settings, where small syntactic differences in configuration files can have significant operational consequences.

Training is guided by two complementary objectives: multimodal alignment learning and duplicate-aware contrastive learning. The multimodal alignment objective exploits the six paired views available in SOD. Given a semantically aligned pair (x_i, x_j), the model is trained to maximize the similarity between their representations using cosine similarity. This objective encourages the model to bridge natural language descriptions and code artifacts, a key requirement for interpreting CI/CD failures that developers often describe informally. To capture the semantic subtlety of DevOps queries, we further incorporate a duplicate-aware contrastive objective derived from SODD. Let (x_a, x_b, y) denote a labeled question pair, where $y \in \{0,1,2,3\}$ indicates duplicate, similar, or different relationships. The model learns to minimize the distance for duplicate pairs and to enforce a margin for distinct pairs. This is implemented using a contrastive loss. This objective explicitly teaches the model to distinguish between superficially similar CI/CD issues that nonetheless require different operational solutions-a common failure mode of general-purpose LLMs. The overall training objective combines both losses. Training is conducted on CI/CD-filtered subsets of SOD and SODD, ensuring that learned representations are grounded in operational contexts rather than generic programming discourse. At inference time, DevOpsAssistant-SOD supports multiple CI/CD assistance tasks through task-specific heads built on top of the shared encoder. For failure diagnosis, the model predicts a probability distribution over predefined failure categories using a softmax classifier. For duplicate detection, it computes similarity scores between incoming queries and historical cases. For response generation, the decoder conditions on $f(x)$ to produce natural language explanations or configuration suggestions. All tasks share the same

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://jisem-journal.com/
Research Article

underlying representation, enabling consistent and transferable reasoning across diverse DevOps support scenarios.

Experimental Analysis

1.3 Evaluation Metrics

To evaluate the effectiveness of DevOpsAssistant-SOD across CI/CD and DevOps workflows, we employ a combination of automatic and task-specific metrics that capture both semantic accuracy and practical usefulness. For CI/CD failure diagnosis, we formulate the task as a multi-class classification problem, where the goal is to predict the correct failure category for a given input query x. We report classification accuracy (Acc) and macro-averaged F1-score (F1), both expressed as percentages. For duplicate detection and semantic similarity estimation, evaluated on the SODD test split, we report Precision (P), Recall (R), and F1-score, all in percentage form. In addition, we compute Mean Reciprocal Rank (MRR) when ranking historical questions for a given query. MRR, defined over the range (0,1], reflects how highly true duplicates are ranked and directly measures retrieval effectiveness in realistic DevOps troubleshooting scenarios.

To assess response relevance and explanation quality, we compute semantic similarity scores using cosine similarity between model-generated responses and accepted Stack Overflow answers. These scores are reported as normalized values in the range [0,1], with higher scores indicating closer semantic alignment.

1.4 Hyperparameters

All hyperparameters were selected based on preliminary experiments on the validation split and were held constant across tasks to maintain a consistent evaluation setting (see Table 2). The model is built on a transformer-based LLM with 24 layers, 1,024 hidden dimensions, and 16 self-attention heads. The maximum input length is set to 1,024 tokens, which comfortably accommodates the combination of natural language descriptions, log excerpts, and configuration snippets commonly found in CI/CD-related Stack Overflow posts. We employ subword tokenization with a vocabulary size of 50,000, enabling robust handling of both natural language and code tokens. Training is performed using the AdamW optimizer, with an initial learning rate of 2×10^{-52}, momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and a weight decay of 0.01. A linear warm-up schedule is applied for the first 10,000 steps, followed by cosine learning-rate decay. The global batch size is 256, distributed across multiple GPUs, with gradient accumulation used when necessary. To mitigate overfitting, dropout with rate 0.1 is applied to both attention and feed-forward layers. For the multimodal alignment objective, we set the contrastive temperature parameter to $\tau = 0.07$. The duplicate-aware contrastive loss uses a margin of $m = 1.0$. Negative samples for contrastive learning are drawn from within-batch instances, yielding 255 negatives per anchor. Training proceeds for three epochs on the CI/CD-filtered SOD corpus, followed by one epoch of fine-tuning on the SODD training split. We apply early stopping when validation F1 fails to improve for five consecutive evaluations. During inference, a beam size of 5 is used for response generation tasks, while similarity-based tasks rely on cosine-distance thresholds calibrated on the validation set.

Table 2. Model hyperparameters and training configuration.

Hyperparameter	Value
Transformer Layers (L)	24
Hidden Dimension (H)	1,024

Attention Heads (A)	16
Max Sequence Length (T)	1,024
Vocabulary Size (V)	50,000
Optimizer	AdamW
Learning Rate (η)	2×10^{-5}
β_1 / β_2	0.9 / 0.999
Weight Decay (λ)	0.01
Warm-up Steps	10,000
LR Schedule	Linear warm-up + Cosine decay
Global Batch Size (B)	256
Dropout Rate (p)	0.1
Contrastive Temperature (τ)	0.07
Contrastive Margin (m)	1.0
# Negatives / Anchor	255
Pre-training Epochs (SOD)	3
Fine-tuning Epochs (SODD)	1
Early Stopping Patience	5
Beam Size (Inference)	5

Result Analysis

1.5 Comparison with State-of-the-Art Systems

We evaluate DevOpsAssistant-SOD against representative state-of-the-art baselines commonly used as engineering assistants in DevOps settings, including a general-purpose instruction-tuned LLM, a code-specialized LLM trained primarily on source code corpora, and a retrieval-augmented model that combines a general LLM with nearest-neighbor search over historical Stack Overflow posts (see Tables 3 and 4). All models are evaluated using identical data splits, input formats, and metrics on CI/CD-relevant subsets of SOD and SODD to ensure a fair comparison.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

Table 3. Ci/cd failure diagnosis and response quality.

Model	Accuracy (%) ↑	Macro-F1 (%) ↑	Response Relevance (%) ↑
Instruction-tuned LLM	71.4	68.9	72.1
Code-specialized LLM	75.6	73.4	74.8
Retrieval-augmented LLM	78.9	76.1	77.3
DevOpsAssistant-SOD (ours)	85.1	82.7	86.0

Across core CI/CD tasks, DevOpsAssistant-SOD consistently outperforms all baselines. The instruction-tuned LLM generates fluent responses but struggles with operational diagnosis, achieving only moderate accuracy, while the code-specialized LLM benefits from stronger syntactic understanding yet often overlooks contextual cues expressed in natural language. The retrieval-augmented baseline improves performance by grounding predictions in historical examples, but its effectiveness depends heavily on lexical overlap and degrades when relevant prior cases are sparse. In contrast, DevOpsAssistant-SOD achieves a failure diagnosis accuracy of 85.1%, representing a 9.3% absolute improvement over the strongest baseline, alongside an 8.7% gain in response relevance. These improvements are consistent across both common and rare failure categories, indicating that the model captures deeper operational semantics rather than overfitting to dominant CI/CD patterns.

Table 4. Duplicate and semantic similarity detection (sodd test set).

Model	Precision (%) ↑	Recall (%) ↑	F1 (%) ↑	MRR ↑
Instruction-tuned LLM	66.8	69.7	68.2	0.63
Code-specialized LLM	71.5	74.4	72.9	0.67
Retrieval-augmented LLM	75.1	77.8	76.4	0.71
DevOpsAssistant-SOD (ours)	82.1	85.0	83.5	0.82

A similar trend emerges in duplicate and semantic similarity detection, where DevOpsAssistant-SOD demonstrates a clear advantage in distinguishing recurring CI/CD issues from superficially similar but operationally distinct queries. Baseline models show limited sensitivity to fine-grained semantic differences, often conflating configuration-level variations that require different remedies. While retrieval augmentation improves ranking performance, it remains constrained by surface-level similarity. DevOpsAssistant-SOD substantially outperforms all baselines, achieving an F1-score of 83.5% and an MRR of 0.82, corresponding to an 11.1% improvement in F1 over the best competing model. Notably,

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://jisem-journal.com/
Research Article

the model maintains a strong balance between precision and recall, highlighting its ability to identify true duplicates without overgeneralizing-a critical property for practical DevOps assistance.

1.6 Cross-Domain Analysis

To assess robustness and generalization, we further evaluate DevOpsAssistant-SOD across multiple DevOps subdomains that frequently interact with CI/CD pipelines, including containerization and orchestration, infrastructure-as-code, and monitoring and runtime operations (see Tables 5 and 6). These domains are constructed using realistic tag-based filtering from Stack Overflow, ensuring that domain boundaries reflect real operational contexts rather than artificial task splits.

Table 5. Cross-domain failure diagnosis and response relevance.

Domain	Accuracy (%) ↑	Macro-F1 (%) ↑	Response Relevance (%) ↑
CI/CD Pipelines	85.1	82.7	86.0
Containers & Orchestration	83.4	80.9	84.2
Infrastructure-as-Code	83.0	80.1	83.8
Monitoring & Runtime Ops	82.3	79.6	82.9

Baseline models exhibit noticeable performance degradation when moving beyond core CI/CD tasks. In particular, code-specialized models struggle in infrastructure-as-code scenarios, where reasoning depends more on declarative semantics than imperative logic, and retrieval-based approaches show sensitivity to lexical mismatch in monitoring-related queries dominated by natural language descriptions. In contrast, DevOpsAssistant-SOD maintains consistently high performance across all domains, achieving accuracy above 82% and strong response relevance throughout. The smallest drop relative to CI/CD tasks occurs in infrastructure-as-code, suggesting effective transfer of pipeline reasoning to configuration-centric domains.

Table 6. Cross-domain duplicate and semantic similarity detection.

Domain	Precision (%) ↑	Recall (%) ↑	F1 (%) ↑	MRR ↑
CI/CD Pipelines	82.1	85.0	83.5	0.82
Containers & Orchestration	81.4	83.2	82.3	0.80
Infrastructure-as-Code	80.6	82.5	81.5	0.79
Monitoring & Runtime Ops	80.1	83.0	81.9	0.81

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

This robustness extends to cross-domain duplicate and semantic similarity detection, where DevOpsAssistant-SOD again outperforms all baselines with F1-scores exceeding 81% across domains and consistently high MRR values. The strongest gains are observed in monitoring and runtime operations, where DevOpsAssistant-SOD improves F1 by 12.4% over the best baseline. Importantly, the gap between precision and recall remains small, indicating stable semantic judgments even in domains where explicit code is sparse and contextual reasoning is paramount.

Ablation Study

To better understand which design choices, drive the performance of DevOpsAssistant-SOD, we conduct a systematic ablation study that isolates the contributions of its key components: multimodal alignment learning, duplicate-aware contrastive learning, and CI/CD-specific data filtering (see Tables 7 and 8). The results show that multimodal alignment learning is essential for effective CI/CD failure diagnosis. When this objective is removed, diagnostic accuracy drops sharply from 85.1% to 78.6%, and response relevance degrades accordingly. This decline reflects the model's reduced ability to jointly interpret natural language descriptions and code or configuration fragments-a core requirement in CI/CD scenarios where failures are rarely expressed in a single modality. Duplicate-aware contrastive learning plays a complementary role: removing it leads to a noticeable reduction in accuracy (80.2%) and a 6.1% drop in response relevance, indicating that the model becomes less sensitive to subtle contextual differences that distinguish operationally distinct pipeline issues. The most severe degradation occurs when both objectives are removed, effectively reducing the model to a standard instruction-tuned LLM trained on the same data. In this setting, accuracy falls to 74.9%, closely matching the baseline results reported earlier. This confirms that DevOpsAssistant-SOD's gains are not simply due to additional data exposure, but stem from training objectives that explicitly encode operational semantics.

Table 7. Ablation results on ci/cd failure diagnosis.

Model Variant	Accuracy (%) ↑	Macro-F1 (%) ↑	Response Relevance (%) ↑
Full DevOpsAssistant-SOD	85.1	82.7	86.0
– Multimodal Alignment	78.6	75.9	79.4
– Duplicate-Aware Learning	80.2	77.8	79.9

A similar pattern emerges in the duplicate and semantic similarity detection task. Removing duplicate-aware contrastive learning causes a sharp decline in performance, with F1-score dropping from 83.5% to 73.6% and MRR from 0.82 to 0.68, underscoring the importance of explicit supervision over semantic relationships for distinguishing recurring CI/CD issues from superficially similar queries. Removing multimodal alignment results in a more moderate but still meaningful decline, suggesting that while cross-modal grounding supports semantic reasoning, fine-grained disambiguation relies more heavily on duplicate-aware training. Finally, ablating CI/CD-specific data filtering-by training the full objective on unfiltered Stack Overflow data-consistently reduces performance across metrics, with F1 falling to 78.9%.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://jisem-journal.com/

Research Article

Table 8. Ablation results on duplicate and semantic similarity detection.

Model Variant	Precision (%) ↑	Recall (%) ↑	F1 (%) ↑	MRR ↑
Full DevOpsAssistant-SOD	82.1	85.0	83.5	0.82
– Multimodal Alignment	75.4	78.3	76.8	0.72
– Duplicate-Aware Learning	71.2	76.0	73.6	0.68
No CI/CD Data Filtering	77.1	80.8	78.9	0.74

Conclusion and Future Works

This work examined how effectively LLM–based engineering assistants can support CI/CD and DevOps workflows, and highlighted the limitations of applying general-purpose code models to real operational settings. Through grounding both training and evaluation in large-scale, real-world developer discourse drawn from Stack Overflow, we introduced DevOpsAssistant-SOD, a multimodal assistant designed to reason jointly over natural language descriptions, code snippets, and configuration artifacts. Across a range of tasks-including CI/CD failure diagnosis, semantic similarity detection, and cross-domain DevOps reasoning-the proposed approach consistently outperformed strong baselines. These results underscore a central insight of this study: practical DevOps assistance depends not only on model capacity, but on domain-aligned data and training objectives that reflect how engineers actually describe and resolve operational problems. Several directions offer promising opportunities to extend this work. One avenue is the integration of live CI/CD telemetry, such as pipeline logs, execution traces, and monitoring metrics, which would enable real-time reasoning and more adaptive feedback during pipeline execution. Another direction involves equipping the assistant with tool-invocation and closed-loop remediation capabilities, allowing it not only to diagnose issues but also to suggest or enact corrective actions.

References

[1] S. Saleh, Nazim Madhavji, and J. Steinbacher, "A Systematic Literature Review on Continuous Integration and Deployment (CI/CD) for Secure Cloud Computing," *arXiv (Cornell University)*, pp. 331–341, Jan. 2024, doi: https://doi.org/10.5220/0013018500003825.

[2] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017, doi: https://doi.org/10.1109/access.2017.2685629.

[3] S. Bobrovskis and A. Jurenoks, "A Survey of Continuous Integration, Continuous Delivery and Continuous Deployment." Available: https://ceur-ws.org/Vol-2218/paper31.pdf.

[4] S. Garg, P. Pundir, G. Rathee, P. K. Gupta, S. Garg, and S. Ahlawat, "On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps," *2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 25–28, Dec. 2021, doi: https://doi.org/10.1109/aike52691.2021.00010.

[5] S. Joshi, "A Review of Generative AI and DevOps Pipelines: CI/CD, Agentic Automation, MLOps Integration, and Large Language Models," *SSRN Electronic Journal*, Jan. 2025, doi: https://doi.org/10.2139/ssrn.5290005.

[6] Y. Dong *et al.*, "A Survey on Code Generation with LLM-based Agents," *arXiv.org*, 2025. https://arxiv.org/abs/2508.00083 (accessed Dec. 30, 2025).

[7] X. Hou *et al.*, "Large Language Models for Software Engineering: A Systematic Literature Review," *ACM Transactions on Software Engineering and Methodology*, Sep. 2024, doi: https://doi.org/10.1145/3695988.

[8] A. Eghbali and M. Pradel, "De-Hallucinator: Mitigating LLM Hallucinations in Code Generation Tasks via Iterative Grounding," *arXiv.org*, 2024. https://arxiv.org/abs/2401.01701.

[9] R. A. Husein, Hala Aburajouh, and Cagatay Catal, "Large language models for code completion: A systematic literature review," *Computer Standards & Interfaces*, vol. 92, pp. 103917–103917, Aug. 2024, doi: https://doi.org/10.1016/j.csi.2024.103917.

[10] A. Ni *et al.*, "L2CEval: Evaluating Language-to-Code Generation Capabilities of Large Language Models," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 1311–1329, Jan. 2024, doi: https://doi.org/10.1162/tacl_a_00705.

[11] S. Tripathi, Nafis, Md Tabrez, I. Hussain, and J. Gao, "The Confidence Paradox: Can LLM Know When It's Wrong," *arXiv.org*, 2025. https://arxiv.org/abs/2506.23464 (accessed Dec. 30, 2025).

[12] J. He *et al.*, "PTM4Tag+: Tag recommendation of stack overflow posts with pre-trained models," *Empirical Software Engineering*, vol. 30, no. 1, Nov. 2024, doi: https://doi.org/10.1007/s10664-024-10576-z.

[13] M. Li, M. Mäntylä, J. Nyyssölä, and M. Luukkainen, "Near-Duplicate Build Failure Detection from Continuous Integration Logs," *Proceedings of the 21st International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 41–44, Jun. 2025, doi: https://doi.org/10.1145/3727582.3728691.

[14] S. Tripathi, M. Tabrez Nafis, I. Hussain, and A. K. J. Saudagar, "Multimodal Fine-Tuning of LLMs for Robust Document Visual Question Answering," *IEEE Access*, vol. 13, pp. 174611–174623, 2025, doi: https://doi.org/10.1109/access.2025.3615201.

[15] C. Zhang, B. Chen, J. Hu, X. Peng, and W. Zhao, "BuildSonic: Detecting and Repairing Performance-Related Configuration Smells for Continuous Integration Builds," Oct. 2022, doi: https://doi.org/10.1145/3551349.3556923.

[16] I. Saidani, A. Ouni, M. Chouchen, and M. W. Mkaouer, "Predicting continuous integration build failures using evolutionary search," *Information and Software Technology*, vol. 128, p. 106392, Dec. 2020, doi: https://doi.org/10.1016/j.infsof.2020.106392.

[17] Fatemeh Hadadi, J. H. Dawes, D. Shin, D. Bianculli, and L. Briand, "Systematic Evaluation of Deep Learning Models for Log-based Failure Prediction," *Empirical Software Engineering*, vol. 29, no. 5, Jun. 2024, doi: https://doi.org/10.1007/s10664-024-10501-4.

[18] J. Gong and T. Chen, "Deep Configuration Performance Learning: A Systematic Survey and Taxonomy," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 1, pp. 1–62, Dec. 2024, doi: https://doi.org/10.1145/3702986.

[19] X. Zhang *et al.*, "Robust log-based anomaly detection on unstable log data," *Foundations of Software Engineering*, Aug. 2019, doi: https://doi.org/10.1145/3338906.3338931.

[20] C. Zhou, P. He, C. Zeng, and J. Ma, "Software defect prediction with semantic and structural information of codes based on Graph Neural Networks," *Information and Software Technology*, vol. 152, p. 107057, Dec. 2022, doi: https://doi.org/10.1016/j.infsof.2022.107057.

[21] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "CodeSearchNet Challenge: Evaluating the State of Semantic Code Search," *arXiv.org*, 2019. https://arxiv.org/abs/1909.09436 (accessed Dec. 30, 2025).

[22] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, Jun. 2022, doi: https://doi.org/10.1145/3520312.3534862.

[23] Z. Gao, X. Xia, D. Lo, J. Grundy, X. Zhang, and Z. Xing, "I Know What You Are Searching for: Code Snippet Recommendation from Stack Overflow Posts," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, pp. 1–42, Jul. 2022, doi: https://doi.org/10.1145/3550150.

[24] A. Sattar and Davide Bacciu, "Graph Neural Network for Context-Aware Recommendation," *Neural Processing Letters*, vol. 55, no. 5, pp. 5357–5376, Jun. 2022, doi: https://doi.org/10.1007/s11063-022-10917-3.

[25] M. Chen, G. Li, C. Ma, J. Li, and H. Fu, "Repo4QA: Answering Coding Questions via Dense Retrieval on GitHub Repositories," *CityUHK Scholars*, pp. 1580–1592, Oct. 2022, doi: https://hdl.handle.net/2031/e1a368b6-d387-47ee-a754-8fab968bc911.

[26] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning," *Proceedings of the 44th International Conference on Software Engineering*, May 2022, doi: https://doi.org/10.1145/3510003.3510155.

[27] X. Jin and F. Servant, "Which builds are really safe to skip? Maximizing failure observation for build selection in continuous integration," *Journal of Systems and Software*, vol. 188, p. 111292, Jun. 2022, doi: https://doi.org/10.1016/j.jss.2022.111292.

[28] Adha Hrusto, Per Runeson, and M. C. Ohlsson, "Autonomous Monitors for Detecting Failures Early and Reporting Interpretable Alerts in Cloud Operations," *Lund University Publications (Lund University)*, pp. 47–57, Apr. 2024, doi: https://doi.org/10.1145/3639477.3639712.

[29] S.-K. Wang, S.-P. Ma, G.-H. Lai, and C.-H. Chao, "ChatOps for microservice systems: A low-code approach using service composition and large language models," *Future Generation Computer Systems*, vol. 161, pp. 518–530, Dec. 2024, doi: https://doi.org/10.1016/j.future.2024.07.029.