SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

# From Alerts to Explanations: LLMs as an Interpretation Layer for Production Machine Learning Systems

Rohit Alekar[1]*

[1]Independent Researcher, USA ORCID: 0009-0007-3058-5379

**Abstract**

Production machine learning systems emit rich telemetry, but incident response often remains limited by human interpretation rather than signal availability. Existing observability tools detect many symptomatic deviations, but they rarely connect signals across stack layers into evidence-backed root-cause hypotheses. This paper proposes an LLM-mediated interpretation layer for production ML operations. The layer assembles incident context from telemetry, lineage, code and configuration changes, historical incidents, and runbooks; generates ranked diagnostic hypotheses; and presents evidence-backed recommendations to human operators. We argue that ML systems require an evidence model that differs materially from generic AIOps because failures span feature pipelines, training dynamics, serving behavior, and experiment systems. We propose evaluation methodology including metrics, ground-truth construction, and calibration requirements. This paper presents a reference architecture and research agenda, not an implemented system. We position the interpretation layer as a human-in-the-loop decision-support component, not an autonomous remediation system.

**Keywords:** LLM-Powered Observability, ML Platform Operations, Retrieval-Augmented Diagnosis, Root Cause Analysis, Confidence Calibration, Evaluation Methodology.

## Introduction

The system complexity of production machine learning systems is growing faster than organizations can scale their operational expertise. At internet scale, a single ranking model can depend on hundreds of feature computation pipelines, multiple concurrent distributed training jobs, dozens of model serving deployments, and thousands of overlapping A/B tests. Many symptomatic deviations can now be detected with reasonable fidelity by mature monitoring systems, but detection remains uneven for silent ML failures, delayed-label failures, and distributional failures that only manifest through downstream performance degradation. The interpretation gap is a direct consequence of ML technical debt: entanglement and undeclared dependencies make symptoms observable long before root causes are obvious [1].

Prior work in AIOps has made progress on root cause analysis for generic cloud incidents [3][4], but production ML systems present a distinct class of failures involving data lineage, feature semantics, training-serving skew, and experiment interactions. These cross-layer failure modes require an evidence model that differs materially from cloud infrastructure RCA (see Section 2.2).

We propose an LLM-mediated interpretation layer that converts incident evidence into ranked, evidence-backed diagnostic hypotheses for human operators. This paper presents a reference architecture and research agenda, not an implemented system.

This paper makes four contributions:

1. It defines the interpretation gap in production ML observability.
2. It proposes a reference architecture with four modules: context assembly, reasoning engine, verifier, and output formatter.
3. It articulates why ML diagnosis differs from generic cloud RCA and identifies ML-specific context requirements.

4. It proposes evaluation methodology including metrics, ground-truth construction, and calibration requirements.

## Related Work

### AIOps and LLM-Based Root Cause Analysis

Recent LLM-based AIOps systems have shown that large language models can assist with incident diagnosis in cloud and microservice environments. AIOpsLab provides an evaluation framework for autonomous cloud agents through fault injection, workload generation, and telemetry export [3]. AIOpsLab offers a useful template for evaluating operational agents, but its cloud-infrastructure focus leaves open how to benchmark ML-specific failures. GPT-based RCA has shown promise for cloud service incidents [4], but ML incidents require additional evidence types. SOP-enhanced systems such as Flow-of-Action show that procedural knowledge improves LLM-guided diagnosis [5]; in ML operations, equivalent procedural knowledge includes feature-debugging runbooks, training-failure playbooks, and experiment-analysis workflows. Tool-assisted systems such as TAMO suggest LLMs should invoke domain tools over metrics, traces, and dependency graphs [6]. ML interpretation layers require the same approach with ML-specific tools for feature lineage, drift analysis, and serving-skew analysis. Recent empirical work documents LLM reasoning failures in cloud RCA, including stalling, bias toward common patterns, and confusion under conflicting evidence [15].

### Why ML Diagnosis Is Not Generic RCA

ML incidents require an evidence model that differs materially from generic cloud RCA. Table 1 summarizes the distinction.

**Table 1: Generic Cloud RCA vs. Production ML RCA**

| Generic Cloud RCA | Production ML RCA |
|---|---|
| Service dependency graph | Feature / data / model lineage graph |
| CPU, memory, network anomalies | Feature drift, label drift, training skew, serving skew |
| Root cause: service, container, or node | Root cause may be data, schema, feature logic, model config, or experiment assignment |
| Remediation: restart, rollback, scale | Remediation: data backfill, feature disablement, retraining, experiment pause |
| Ground truth from incident ticket | Ground truth may require model-quality and data-quality validation |
| Failures typically single-layer | Failures often compound across feature, training, serving, and experiment layers |

### ML Observability and ML Technical Debt

Sculley et al. identified entanglement of ML components as a primary source of operational complexity [1]. Shankar and Parameswaran proposed end-to-end observability for deployed ML pipelines with assisted detection, diagnosis, and reaction, and introduced a bolt-on architecture around existing stack tools [7]. Their work is the closest precursor to ours. We build on this vision but focus specifically on LLM-mediated diagnosis, evidence grounding, verifier design, and calibration for generated hypotheses. Interview studies of ML operationalization confirm that production ML failures are also workflow, ownership, monitoring, and coordination probslems [23].

### Retrieval-Augmented Diagnosis, Grounding, and Verification

The interpretation layer generates long-form diagnostic reports grounded in retrieved evidence. RAG evaluation frameworks such as RAGAs suggest that evaluation should separately measure context retrieval quality, evidence faithfulness, and answer usefulness rather than treating diagnosis quality as a single scalar [18]. Groundedness work in retrieval-augmented long-form generation suggests that evaluation should occur at the claim level: a diagnostic report can be partially correct while still containing unsupported causal claims [21]. Self-RAG motivates separating retrieval, generation, and critique into explicit stages, supporting the interpretation layer's architectural separation between context assembly, diagnosis generation, and verification [22]. Sampling-

based hallucination detectors such as SelfCheckGPT suggest that unsupported hypotheses should be unstable across generations, whereas evidence-grounded hypotheses should remain consistent when conditioned on the same incident context [19].

## Confidence Calibration and Trust

Prior work on LLM confidence estimation shows that confidence must be treated as a calibrated prediction problem rather than a free-form expression of certainty [20]. Calibration should use standard metrics such as Brier score and expected calibration error [24]. Dong, Lu, and Zhu conclude that LLM agents with opaque reasoning cannot be trusted, which makes transparent hypothesis ranking, evidence citation, and confidence scoring design requirements [10]. Research on generative AI in clinical decision support confirms that systems are most effective when reasoning is visible and uncertainty is communicated [11].

## Feature Stores and Data Lineage

Feature store implementations, including storage, serving, and lineage tracking, are discussed in applied ML literature [8]. Interpretation quality depends on integration with lineage tracking. Without lineage data, the context assembly module cannot reconstruct the dependency chains needed for root cause hypotheses.

## Self-Healing and Autonomous ML Systems

Rauba et al. identified monitoring, diagnosis, adaptation, and testing as the four components of autonomous ML adaptation [9]. Self-healing ML formalizes the idea that adaptation should be diagnosis-guided rather than reason-agnostic. The interpretation layer can be understood as the diagnostic substrate required before safe adaptation is possible.

**The Observability Stack and Its Limits**
**What Modern Monitoring Provides**
Observability infrastructure covers multiple layers: infrastructure metrics, pipeline orchestration, feature distribution stability (PSI, KS distance, coverage, freshness), training metrics (loss, gradient norms), serving metrics (latency, throughput, prediction drift), and experiment metrics. Table 2 summarizes coverage and gaps.

**Table 2: ML Observability Stack**

| Layer | Tool Example | Signals | Detects? | Interpretive Gap |
|---|---|---|---|---|
| Compute | Prometheus / Grafana | CPU, memory, disk I/O, network | Yes | No causal context |
| Pipeline | Airflow / Prefect | Job status, runtime, dependencies | Yes | No upstream lineage |
| Feature | Feature platform | PSI, KS, coverage, freshness | Yes | No root cause |
| Training | Training platform | Loss, gradient norms, GPU util. | Yes | No cross-layer view |
| Serving | Serving platform | Latency p50/p99, throughput | Yes | No code/data context |
| Experiment | Experimentation platform | Metric delta, significance | Yes | No system correlation |

## The Interpretation Gap

Monitoring tells the engineer what is anomalous, not why. To diagnose a feature drift, an engineer needs upstream table freshness, recent schema changes, transformation code history, correlated drift in sibling features, and magnitude analysis. A threshold-based alert cannot do this reasoning. Sculley et al. observed that changes to data, features, and models entangle and propagate in ways that are hard to predict [1].

## The Interpretation Layer: Architecture Positioning

The interpretation layer sits between monitoring infrastructure and human operators. It is not a dashboard, alert router, or automated remediation agent. Its function is diagnostic: given incident signals plus contextual metadata, it produces ranked, evidence-backed hypotheses that a human can act on.

## Module Design

The layer comprises four modules, summarized in Table 3.

**Table 3: Interpretation Layer Architecture**

| Module | Inputs | Function | Failure Mode | Eval Metric |
|---|---|---|---|---|
| Context Assembly | Logs, metrics, lineage, diffs, configs | Retrieve relevant incident context | Missing / irrelevant evidence | Context precision, recall [18] |
| Reasoning Engine | Assembled context + domain knowledge | Generate ranked hypotheses | Hallucinated root cause | Top-k RCA accuracy |
| Verifier | Hypotheses + evidence | Validate support, flag contradictions | False confidence | Faithfulness [18], contradiction rate |
| Output Formatter | Verified hypotheses | Structured report with citations | Unclear recommendations | Human usefulness score |

**Context assembly** collects signals and metadata: triggering alert, recent code and config changes, pipeline dependency graphs, historical incidents, and documentation. ML observability data differs from document corpora: log data requires domain-specific chunking, time-series metrics require summarization, code diffs must be ranked for relevance, and dependency graphs must be traversed along causal paths. RAG evaluation vocabulary from RAGAs [18] (context precision, context recall, faithfulness, answer relevance) provides useful metrics for this module.

**The reasoning engine** is the LLM, prompted with domain-specific context about ML failure modes. Following self-reflective RAG principles [22], generation and critique should be separated: the reasoning engine generates candidate diagnoses, and the verifier independently assesses evidence support.

## The Verifier Module

The verifier checks whether the reasoning engine's output is supported by the assembled evidence. It implements six strategies:

- Evidence entailment check: does the cited telemetry actually support the hypothesis?
- Contradiction check: does any retrieved evidence contradict the hypothesis?
- Dependency validity check: is the claimed causal path valid in the lineage or dependency graph?
- Temporal ordering check: did the alleged cause precede the observed symptom?
- Counterfactual check: are unaffected sibling features or models consistent with the hypothesis?
- Action safety check: could the recommended remediation cause data loss, model regression, or a wider outage?

SelfCheckGPT suggests an additional strategy: unsupported hypotheses should be unstable across sampled generations, whereas evidence-grounded hypotheses should remain consistent when conditioned on the same incident context [19]. Self-RAG [22] motivates this architectural separation between generation and critique. The verifier can combine programmatic checks (graph traversal for dependency validity, timestamp comparison for temporal ordering) with a lightweight LLM pass for entailment and contradiction. The key constraint is that the verifier should operate on explicit, inspectable evidence rather than the reasoning engine's latent internal state.

**The output formatter** produces a structured report: anomaly summary, ranked root causes with evidence and confidence scores, recommended diagnostic steps, and processed signal inventory.

## Feature Pipeline Observability

### Failure Modes and Interpretive Complexity

Feature pipelines are often among the most opaque operational surfaces in production ML systems, because they sit between data engineering and ML. Failures at any stage can manifest through signals that are spatially and temporally distant from the fault. Consider a PSI alert for a user engagement feature: the engineer must determine whether the change was caused by an upstream data source issue, a transformation code change, a shared upstream data source affecting multiple features, or a genuine behavior shift. Without the interpretation layer, this investigation is manual and can take hours.

### Training-Serving Consistency and Hypothesis Reduction

Some publicly described feature platform designs attempt to reduce training-serving skew by enforcing shared definitions, centralized lineage, or consistent transformation logic. Pinterest has documented a transition from custom per-use-case feature extraction code to a unified feature representation to reduce discrepancies [12]. Such consistency guarantees narrow the hypothesis space but do not eliminate runtime, snapshotting, data freshness, or serving-path causes of skew. Integration with feature store lineage tracking [8] is essential for context assembly.

## Training Job Diagnosis

### The Multi-Dimensional Failure Space

Training job failures span hardware, software, data, and configuration dimensions. Naumov et al. showed that DLRM architectures rely heavily on categorical features and embedding tables, which makes feature schema and categorical input quality operationally important [13]. Distributed training frameworks introduce additional diagnostic surfaces including worker communication, parameter synchronization, and fault tolerance [14].

### Cross-Layer Reasoning

Feature pipeline failures and training job failures are typically monitored separately. The interpretation layer addresses this by pulling from both platforms' monitoring through the context assembly module.

### Synthetic Incident Walk-Through

*The following example is synthetic and does not describe any proprietary system.*

**Trigger:** A feature drift alert fires for user_recent_click_rate_7d. PSI is 0.31 (threshold: 0.25).

**Context Assembly:** Retrieves: (1) upstream table freshness for three source tables (one shows a 14-hour partition delay), (2) a refactoring code commit six hours earlier (review shows no semantic changes), (3) lineage showing two sibling features from the delayed table with elevated PSI while a third sibling from a different table shows no drift, (4) three historical incidents involving the same table, two caused by partition delays.

**Reasoning Engine:** Hypothesis 1 (high confidence): Delayed upstream ingestion. Evidence: partition delay, correlated sibling drift, consistent historical pattern, unaffected third sibling. Hypothesis 2 (low): Code change introduced subtle transformation bug. Evidence: commit landed 6h prior, but no semantic changes. Flagged for human review. Hypothesis 3 (very low): Genuine behavior shift. Drift pattern consistent with staleness, not distribution shape change.

**Verifier:** Confirms temporal ordering (delay preceded spike). Validates dependency path in lineage graph. Counterfactual: third sibling from different table is unaffected, consistent with Hypothesis 1. Flags Hypothesis 2 for human code review. Generation-consistency check: Hypothesis 1 stable across re-generations; Hypothesis 2 varies, suggesting weaker grounding.

**Output:** Recommends checking upstream partition repair status and verifying with data engineering team. States no automated remediation should be taken until delay is confirmed. Recommends human review of code commit as secondary check. A deployed system would need sub-minute latency for this class of incident.

## Evaluation Methodology

### Proposed Metrics

A major gap is the absence of standardized evaluation for ML-specific diagnosis. Drawing on RAG evaluation vocabulary from RAGAs [18], we propose that evaluation should separately measure context retrieval quality, evidence faithfulness, and diagnostic usefulness. Groundedness work in retrieval-augmented long-form generation confirms that evaluation should occur at the claim level: a diagnostic report can be partially correct while containing unsupported causal claims [21]. Table 4 presents our proposed metrics.

**Table 4: Proposed Evaluation Metrics**

| Metric | What It Measures | Priority |
|---|---|---|
| Top-1 RCA accuracy | Whether highest-ranked hypothesis identifies the correct root cause | Primary |
| Top-3 RCA recall | Whether correct root cause appears in ranked hypotheses | Primary |
| Evidence precision | Fraction of cited evidence actually relevant (context precision) | Primary |
| Evidence recall | Fraction of important evidence retrieved (context recall) | Primary |
| Faithfulness | Whether generated claims are grounded in retrieved evidence [18][21] | Primary |
| Unsafe recommendation rate | Frequency of remediation suggestions that would cause additional harm | Safety-critical |
| Calibration error (Brier / ECE) | Whether confidence scores correspond to actual correctness rates [20][24] | Trust-critical |
| Human trust score | Whether operators find output useful and trustworthy (survey) | Adoption-critical |

### Confidence Calibration

LLM self-reported confidence is often poorly calibrated [20]. Calibration should use standard metrics such as Brier score and expected calibration error [24]. We identify the following requirements:

- Confidence should reflect evidence strength and consistency, not the model's internal probability estimate.
- "Model confidence" and "evidence strength" should be reported separately to operators.
- Calibration should be measured at the hypothesis level: among root-cause hypotheses assigned 0.8 confidence, approximately 80% should be correct under postmortem-derived ground truth.
- Historical incident replay provides a natural calibration dataset.
- Every hypothesis should cite specific telemetry, logs, or code diffs. Uncited hypotheses should be flagged as speculative.

## Ground Truth Construction

ML incident ground truth may require multiple levels of validation. Table 5 presents our proposed taxonomy.

**Table 5: Ground Truth Taxonomy for ML Incident Diagnosis**

| Ground Truth Level | Example | Validation Source |
|---|---|---|
| Immediate trigger | Upstream partition delay | Pipeline monitoring logs |
| Proximate cause | Feature freshness violation | Feature platform metrics |
| Systemic cause | Missing data freshness contract between teams | Postmortem / design review |
| Correct remediation | Backfill table, suppress stale feature, pause training | Rollback / change records |

| | | |
|---|---|---|
| Unsafe remediation | Retrain model before data repair | Expert adjudication |

Ground truth should be derived from postmortems, incident tickets, code changes, data backfills, rollback records, and expert adjudication. Because ML incident root causes are often multi-factorial, benchmarks should support multi-label root-cause annotations.

## Benchmark Design

No standardized benchmark exists for ML-specific diagnosis. AIOpsLab provides a useful template through fault injection and telemetry export [3], but no equivalent covers ML-specific failure modes. A benchmark should support both replay mode (all evidence frozen) and interactive mode (the system can request additional evidence through tools such as lineage queries, log filters, metric summaries, and code-diff inspection). Interactive mode connects more directly to tool-using AIOps agents.

## Operational Motivation

In most production ML organizations, the ability to diagnose incidents concentrates in a small number of senior engineers. This creates compounding operational risks: availability risk (incidents persist when key engineers are unavailable), scaling risk (system complexity grows faster than the knowledge pool), and turnover risk (departing engineers take institutional knowledge that is not reconstructed from documentation).

The economic impact extends beyond on-call hours. ML systems are sequentially dependent: feature freshness affects training data quality, which affects model quality, which affects the signal used to train the next generation. Faster incident resolution limits cumulative performance degradation. Concrete operational costs include MTTR increase during key-person absence, engineer-hours spent on manual triage, degraded-model exposure time, delayed experiment decisions, unnecessary retraining or backfills triggered by misdiagnosis, and incident recurrence from incomplete root-cause understanding.

The interpretation layer encodes institutional knowledge into a form any engineer can access. Self-healing ML formalizes the idea that adaptation should be diagnosis-guided [9]; the interpretation layer provides the diagnostic substrate required before safe adaptation is possible.

## Challenges and Limitations

## Hallucination and Diagnostic Reliability

The most serious challenge is LLM hallucination. Recent work documents reasoning failures in cloud-based RCA including stalling, bias, and confusion under conflicting evidence [15]. The verifier module (Section 4.2.1) addresses this through evidence entailment, contradiction, temporal ordering, and generation-consistency checks [19].

## Context Assembly

Vanilla document-oriented RAG is insufficient for ML observability data. Log data, time-series metrics, code diffs, and dependency graphs each require different retrieval and summarization strategies.

## Latency

Latency is implementation-dependent and may range from seconds to minutes depending on model size, context length, retrieval cost, and verification depth. Public engineering accounts from Netflix and Pinterest illustrate the operational importance of developer experience, platform latency, and workflow integration in large-scale ML tooling [16][17].

## Trust and Organizational Dimensions

Utility depends on operator trust, which must be built conservatively. Dong, Lu, and Zhu conclude that opaque reasoning prevents adoption [10]. The layer is a knowledge amplifier, not a knowledge substitute: output quality depends on institutional knowledge encoded through the system's design.

## Limitations of This Work

This paper presents a reference architecture and research agenda, not an implemented system. We do not report accuracy results, latency measurements, or user studies. The synthetic example illustrates the architecture but

does not validate it. Building and evaluating a prototype against real or realistic incident corpora is the most important next step.

## Research Priorities

**First,** the community needs benchmarks for ML-specific diagnosis covering feature pipeline failures, training anomalies, serving regressions, and experiment confounds [3].

**Second,** research is needed on domain-specific context assembly for heterogeneous ML observability data.

**Third,** research is needed on trust calibration: computing and communicating confidence scores, building verification pipelines, and tracking accuracy over time [20].

**Fourth,** systematic comparison of design choices (model size, retrieval method, verifier architecture, confidence scoring) would help converge on effective designs.

## Conclusion

A major bottleneck of production ML operations is interpretation: translating observability signals into root-cause understanding and safe remediation actions. This bottleneck exists because the required institutional knowledge concentrates in a small number of engineers and does not transfer or scale.

We propose that large language models can address this gap as an interpretation layer: an architectural component that assembles incident context from heterogeneous telemetry, generates ranked evidence-backed diagnostic hypotheses, verifies them against assembled evidence, and presents them to human operators for judgment. We have described the architecture, applied it to feature pipeline and training job diagnosis, proposed evaluation methodology with metrics, ground-truth construction, and calibration requirements, and discussed the challenges of safe deployment.

We argue that ML-specific diagnosis should be treated as a first-class systems problem, not an incidental extension of generic AIOps. The interpretation layer provides one architectural framing for that problem and a concrete agenda for future evaluation. ML platforms are a strong starting point because they have well-understood failure modes, high operational stakes, and mature observability infrastructure. Realizing this potential requires advances in domain-specific context assembly, benchmark construction, and trust calibration.

## References

1. D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems," NeurIPS, 2015.
2. M. De la Cruz Cabello et al., "AIOps for Log Anomaly Detection in the Era of LLMs: A Systematic Literature Review," ScienceDirect, 2025.
3. Y. Chen et al., "AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds," Proc. MLSys, vol. 7, 2025.
4. X. Zhang et al., "Automated Root Cause Analysis for Cloud Services Using GPT-4 and In-Context Learning," arXiv:2401.13810, 2024.
5. C. Pei et al., "Flow-of-Action: SOP Enhanced LLM-Based Multi-Agent System for Root Cause Analysis," Companion Proc. ACM Web Conference (WWW), 2025.
6. X. Zhang, Q. Wang, et al., "TAMO: Fine-Grained Root Cause Analysis via Tool-Assisted LLM Agent with Multi-Modality Observation Data in Cloud-Native Systems," arXiv:2504.20462, 2025.
7. S. Shankar and A. G. Parameswaran, "Towards Observability for Production Machine Learning Pipelines," Proc. VLDB Endow., vol. 15, no. 13, pp. 4015-4022, 2022.
8. Z. Yan, "Feature Stores: A Hierarchy of Needs," eugeneyan.com, 2021.
9. P. Rauba et al., "Self-Healing Machine Learning: A Framework for Autonomous Adaptation in Real-World Environments," NeurIPS, 2024.
10. L. Dong, Q. Lu, and L. Zhu, "AgentOps: Enabling Observability of LLM Agents," arXiv:2411.05285, 2024.
11. N. Rajashekar et al., "Generative Artificial Intelligence in Clinical Decision Support: Quantitative and Qualitative Analyses," Yale University, 2025.
12. C. Qian and A. Mudgal, "Handling Online-Offline Discrepancy in Pinterest Ads Ranking System," Pinterest Engineering Blog, 2024.
13. M. Naumov et al., "Deep Learning Recommendation Model for Personalization and Recommendation Systems," arXiv:1906.00091, 2019.

14. M. Li et al., "Scaling Distributed Machine Learning with the Parameter Server," OSDI, 2014.
15. E. Riddell et al., "Stalled, Biased, and Confused: Uncovering Reasoning Failures in LLMs for Cloud-Based Root Cause Analysis," IEEE/ACM FORGE, 2026.
16. Netflix Technology Blog, "Supercharging the ML and AI Development Experience at Netflix," November 2025.
17. Pinterest Engineering, "A Decade of AI Platform at Pinterest," November 2025.
18. S. Es, J. James, L. Espinosa Anke, and S. Schockaert, "RAGAs: Automated Evaluation of Retrieval Augmented Generation," Proc. EACL: System Demonstrations, pp. 150-158, 2024.
19. P. Manakul, A. Liusie, and M. J. F. Gales, "SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models," Proc. EMNLP, 2023.
20. J. Geng, F. Cai, Y. Wang, H. Koeppl, P. Nakov, and I. Gurevych, "A Survey of Confidence Estimation and Calibration in Large Language Models," Proc. NAACL-HLT, pp. 6577-6595, 2024.
21. J. Lee, A. Wettig, and D. Chen, "Groundedness in Retrieval-Augmented Long-Form Generation," Findings of NAACL, 2024.
A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," Proc. ICLR, 2024.
22. S. Shankar, R. Garcia, J. M. Hellerstein, and A. G. Parameswaran, "Operationalizing Machine Learning: An Interview Study," arXiv:2209.09125, 2022.
23. C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On Calibration of Modern Neural Networks," Proc. ICML, 2017.