Bian Que: An Agentic Framework with Flexible Skill Arrangement for Online System Operations

Bochao Liu*, **Zhipeng Qian***, **Yang Zhao***, **Xinyuan Jiang***, **Zihan Liang**, **Yufei Ma**, **Junpeng Zhuang**, **Ben Chen**†, **Shuo Yang**, **Hongen Wan**, **Yao Wu**, **Chenyi Lei**, **Xiao Liang**

Kuaishou Technology
benchen4395@gmail.com

Abstract

Operating and maintaining (O&M) large-scale online engine systems (eg, search, recommendation and advertising) demands substantial human effort for release monitoring, alert response, and root cause analysis. Despite the inherent suitability of LLM-based agents for such operational scenarios, the critical bottleneck impeding their practical deployment lies not in reasoning, but in *orchestration* capability—specifically, the precise selection of relevant *data* (encompassing metrics, logs, and change events) and applicable *knowledge* (including handbook-defined rules and empirically derived practitioner experience) tailored to each individual operational event. Feeding all signals indiscriminately causes dilution and hallucination, while manually curating the event-to-(data, knowledge) mapping is intractable under dozens of daily releases. Here we present BIAN QUE, an agentic operating framework with three contributions: (i) The *unified operational paradigm*, which abstracts routine daily O&M actions into three canonical patterns: release interception, proactive inspection, and alert root cause analysis; (ii) The *flexible Skill Arrangement*, each predefined Skill explicitly defines the requisite data and operational knowledge for each specific context. Such Skills can be automatically generated and updated by LLM agents, and can also be iteratively optimized by on-call engineers via natural language instructions. (iii) The *unified self-evolving mechanism*, where each correction signal enables two parallel evolutionary pathways: distilling event memory into knowledge, and targeted refinement of corresponding Skills. Deployed on the e-commerce search engine of KuaiShou, BIAN QUE reduces alert volume by 75%, achieves 80% root-cause analysis accuracy, cuts mean time to resolution by over 50%, and attains a 99.0% pass rate on offline evaluations. Codes are available at `https://github.com/benchen4395/BianQue_Assistant`.

1 Introduction

Large-scale online engine systems such as search, recommendation, and advertising platforms serve as the fundamental infrastructure of modern Internet services. The operation and maintenance (O&M) of these systems consume considerable engineering resources. Furthermore, underlying modules and services iterate at a high frequency, with dozens of version releases deployed daily. Coping with such rapid iteration requires thorough comprehension of system internal mechanisms and continuous monitoring of runtime status, both of which impose heavy manual workloads. Fortunately, the advances in large language models (LLMs) and autonomous agents (Openclaw [1], Claude Code [2], Harness engineering [3]) have enabled the automation of complex O&M tasks, providing a scalable solution to reduce reliance on manual labor and lower overall operational costs.

*These authors contributed equally. †The corresponding author.

Preprint.

To enable the principled deployment of LLM-based agents, we first abstract the heterogeneous O&M scenarios into a collection of standardized recurring task paradigms. Although daily operational workflows vary across business domains and engineers, practical defensive maintenance actions inherently follow unified behavioral patterns. We summarize typical paradigms and formalize them as the *three lines of defense*: release-period system monitoring, periodic proactive health inspection, and post-alert root cause diagnosis. Existing LLM-based O&M agents [4] mainly focus on the final post-alert troubleshooting, regarding alert triggering as the sole task entry point. By contrast, our framework advances automated release monitoring and proactive inspection *prior to* alert occurrence, relegating alert-driven diagnosis to a fallback measure rather than the primary response.

All three lines of defense share an identical cognitive logic: operators perceive system runtime signals, apply professional operational knowledge, and conduct reasoning to derive diagnostic results or operational decisions. This renders LLM-based agentic frameworks naturally applicable to O&M tasks. However, real-world operational scenarios are highly complex and heterogeneous. We define system runtime signals as *data*, covering time-series metrics, structured logs, change events, traces and business indicators. We also define empirical operational experience for troubleshooting as *knowledge*, including handbook rules, typical failure patterns, and module-specific behavioral norms. Directly feeding all raw data into LLMs hinders the model from capturing truly relevant information. Moreover, operational knowledge is usually undocumented, evolves with system iterations, and varies across business lines, modules and operational contexts. Hence, the core challenge is not reasoning capability, but the contextual selection of appropriate *data* and *knowledge* for each operational event. Such data and knowledge also need post-event incremental updates to accelerate the disposal of future similar incidents. Given the combinatorial complexity of business and scenarios, manually maintaining event-to-data-knowledge mappings is labor-intensive and infeasible.

The above insights yield three key design principles. First, by abstracting O&M workflows into the three lines of defense, we deploy a dedicated agent for each paradigm, individually responsible for release interception, proactive inspection, and alert root cause analysis. Second, given rapid system iteration and dynamic operational contexts, the mapping between operational event-to-(*data*, *knowledge*) requires continuous updating. maintaining it via natural language interfaces, instead of static configurations, ensures interpretability and sustainability. Third, practitioner feedback implicitly reflects the appropriateness of contextual assembly and the value of empirical knowledge to be preserved. A single feedback signal can therefore drive the parallel evolution of both event mappings and domain knowledge, avoiding the need for separate independent update pipelines.

These insights motivate BIAN QUE[1], an agentic O&M framework for online engine systems (Figure 1). Its foundation is a *unified operational paradigm*: the three lines of defense are formalized under a shared structure, each with a specialized agent (sharing the same LLM but differing in scenario knowledge and invoked Skills). Built atop this, *Flexible Skill Arrangement* enables automatic generation and incremental refinement of event-to-(*data*, *knowledge*) mappings, with on-call engineers intervening via natural-language instructions. Closing the loop, a *unified self-evolving mechanism* channels each feedback signal into two parallel pathways—memory-to-knowledge distillation and targeted Skill refinement—realizing co-evolution of the knowledge base and mappings through a single loop. We summarize our contributions as follows:

- We propose a **unified operational paradigm** that abstracts heterogeneous O&M work into three canonical patterns: release interception, proactive inspection, and alert root-cause analysis.
- We design **Flexible Skill Arrangement**, where each Skill specifies the relevant *data* and *knowledge* for each context and can be generated, updated, and corrected through natural-language feedback.
- We introduce a **unified self-evolving mechanism** in which one feedback signal simultaneously drives memory-to-knowledge distillation and targeted Skill refinement, enabling the knowledge base and the context-assembly mapping to co-evolve.
- We evaluate BIAN QUE both online and offline. In online deployment on the e-commerce search engine of a short-video platform serving hundreds of millions of users, BIAN QUE reduces alert volume by 75%, achieves 80% root-cause analysis accuracy, and cuts mean time to resolution by over 50%. In offline evaluation on a curated benchmark of 104 cases, it attains a 99.0% pass rate.

[1]Named after the legendary ancient Chinese physician Bian Que, who diagnosed and prevented illnesses prior to symptom onset—embodying our framework's proactive philosophy.

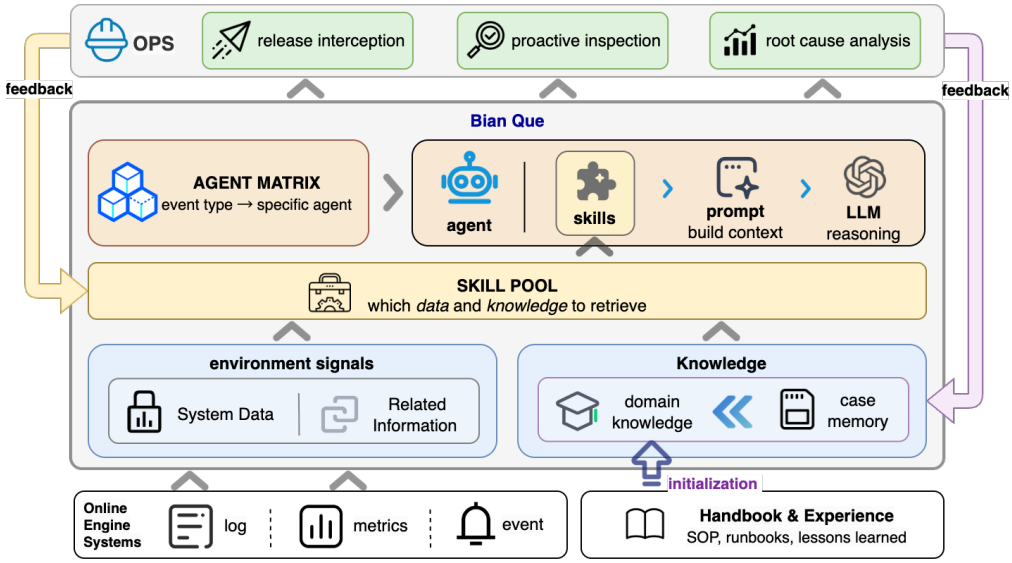

Figure 1: Overview of the BIAN QUE architecture. Operational events from the OPS platform (top) are dispatched to a corresponding Agent, which invokes matched Skills to assemble the relevant *data* (system signals: logs, metrics, change events) and *knowledge* (domain knowledge distilled from case memory, seeded by operational handbooks). The LLM performs reasoning on the assembled information and returns diagnostic results. Practitioner feedback further drives two parallel self-evolution pathways (yellow: Skill refinement; purple: memory-to-knowledge distillation).

2 Method

2.1 Problem Formulation and Architectural Overview

The operational maintenance of large-scale online engine systems (search, recommendation, advertising) gives rise to a stream of *operational events* e, including release deployments, alert triggers, and scheduled inspection requests. For each event, the agent produces a structured output o that includes a decision (e.g., "rollback" or "safe to proceed"), supporting evidence, and recommended follow-up actions. As explained in §1, we identify three canonical patterns—release interception, proactive inspection, and alert root cause analysis—that collectively cover the bulk of the operational workload. All three share a common computational structure:

$$o = f_{\text{LLM}}\big(\mathcal{D}(e),\ \mathcal{K}(e),\ p_s\big), \tag{1}$$

where $\mathcal{D}(e)$ denotes the *data* retrieved for event e, $\mathcal{K}(e)$ denotes the applicable *operational knowledge*, p_s denotes the scenario-specific prompt composition, and f_{LLM} means the large language model performing structured reasoning. The prompt p_s is further decomposed into an agent-level and one or more Skill-level components, which are depicted in Eq. 2.

Figure 1 shows the overall architecture. The three terms of Eq. 1 correspond exactly to the framework modules: the inherent data infrastructure of online engines $\mathcal{D}$, the two-stage knowledge base (§2.4) supplies $\mathcal{K}$, and a general-purpose LLM serves as f_{LLM}. Two key design features are highlighted as follows. First, instead of constructing dedicated monitoring infrastructure, this framework reuses the mature data stack of industrial online engines. The core challenge lies not in signal collection, but in *selecting appropriate signals for individual events*, which is fulfilled by the flexible skill mechanism (Section 2.3). Second, the LLM is deployed as a pluggable reasoning module rather than a task-specific fine-tuned component. Empirical results (§B) show that general-purpose models above roughly 30B scale perform comparably on operational reasoning when given adequate context, so the framework is model-agnostic above this scaling size.

2.2 Agent Matrix and Skill Pool

A monolithic agent cannot efficiently handle the full breadth of online engine maintenance, because the data requirements and knowledge contexts vary drastically across business lines and service

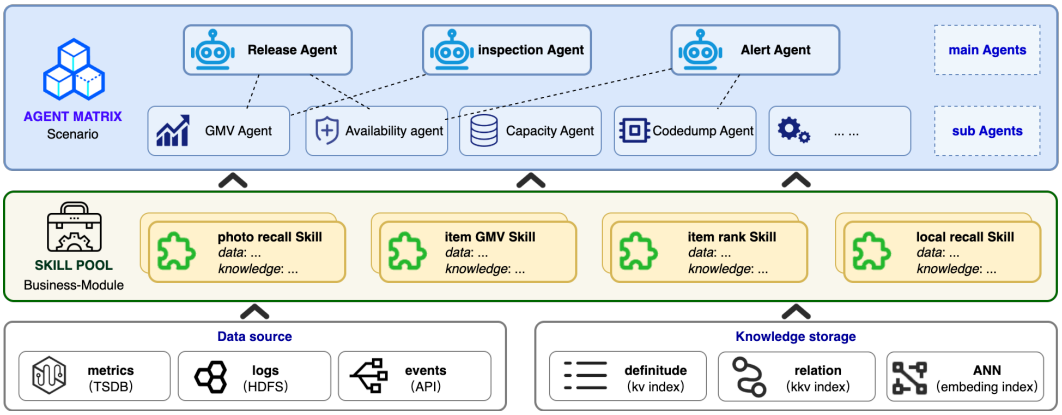

Figure 2: Agent Matrix and Skill Pool. Each Agent (top) implements one canonical pattern; Skills (middle) are business-module orchestration units in a shared pool. Each single execution composes one Agent with matched Skills, which call into the knowledge base and data sources (bottom).

modules. We therefore adopt a matrix design that decomposes the problem along two orthogonal dimensions (shown in Figure 2).

Agents as Scenario Abstractions. Each agent type corresponds to one canonical pattern: the release interception agent implements checkpoint-based evaluation, the inspection agent implements periodic comprehensive analysis, and the alert agent implements event-triggered diagnosis.

Skills as Business-Module Orchestration Units. Each Skill specifies the *data* and *knowledge* needed for a particular combination of business and module. For instance, a Skill for "recommendation recall module availability" declares which metrics, logs, and change events to retrieve, which knowledge-base entries to consult (e.g., "this module's GMV impact is typically small"), and how to reason over both. The internal structure of Skills is detailed in §2.3.

Orthogonal Composition. A single agent execution composes the agent's scenario logic with one or more matched Skills:

$$o = f_{\text{LLM}}\Big(\bigcup_i \mathcal{D}_{s_i}(e),\ \bigcup_i \mathcal{K}_{s_i}(e),\ p_{\text{agent}},\ \{p_{s_i}\}\Big), \quad \{s_i\} = \textsc{Match}(e, \mathcal{S}), \tag{2}$$

where $\mathcal{S}$ is the Skill pool and MATCH selects relevant Skills via keyword matching against event metadata. Here p_s from Eq. 1 is realized as the composition of p_{agent} with the Skill-level prompts $\{p_{s_i}\}$. This orthogonal decomposition allows agents and Skills to iterate independently: agents can improve reasoning strategies without modifying Skills, and Skills can evolve their data and knowledge configurations without changing agent logic.Within the inspection and alert categories, sub-agents are further specialized by metric family (e.g., GMV, capacity, availability, coredump) when a single agent prompt cannot cover the breadth of the category; sub-agents share the scenario logic of their parent and differ only in p_{agent} specialization, and are therefore omitted from the formulation in Eq. 2.

2.3 Flexible Skill Arrangement

The value of the Skill abstraction hinges on providing precise, scenario-appropriate context. Raw data tends to dilute critical signals and induce hallucinations, while overly rigid configurations may lead to information omission. Considering the combinatorial complexity across business lines, modules and operational scenarios, as well as rapid system iteration, manual maintenance of Skill configuration becomes infeasible. The proposed Flexible Skill mechanism tackles this by enabling Skills to be *LLM-generable, LLM-updatable*, and *Human-correctable*.

2.3.1 Skill Structure

Each Skill is a structured document with three fields:

$$\text{Skill} \ = \ \langle \texttt{LoadDataSchema},\ \texttt{Prompt},\ \texttt{Meta} \rangle. \tag{3}$$

`LoadDataSchema` is a JSON specification that declares *what to retrieve*. It lists the data-source calls (e.g., time-series metrics, structured logs, change events, traces) and knowledge-base queries to

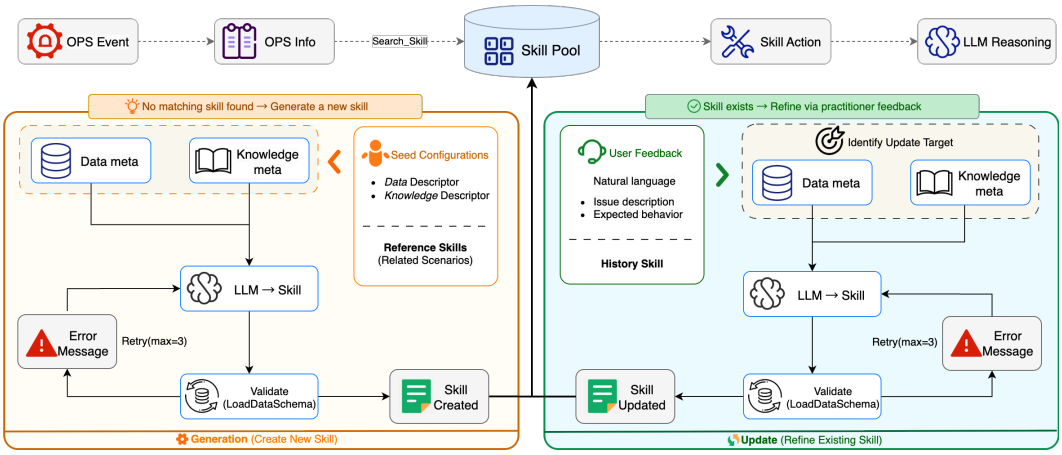

Figure 3: Flexible Skill lifecycle. New Skills are generated from seed configurations with validation and retry. Existing Skills are updated via practitioner feedback through a natural-language interface.

execute, along with their parameters and whether each source is mandatory or optional. `Prompt` is a structured template that defines *how to reason*. It guides the LLM through a multi-step analysis over the retrieved data and knowledge to produce a structured diagnosis. At execution time, the Skill-level `Prompt` composes with the agent-level p_{agent}, and p_{agent} supplies scenario logic (e.g., how to structure a release verdict), while the Skill `Prompt` supplies business-module-specific analysis steps. `Meta` contains name, version, description, and tags used for keyword-based Skill (§2.2).

2.3.2 Skill Lifecycle

The lifecycle consists of two phases (Figure 3): *Generation* creates a new Skill when no matching one exists for a given business-module combination, and *Update* incrementally revises an existing Skill in response to practitioner feedback.

Generation. When a new business-module combination encounters its first event and no matching Skill exists, the Skill framework triggers automated generation. The LLM takes two seed inputs: a *source descriptor* that enumerates the available data sources and knowledge entries along with their semantics, and a *capability descriptor* that specifies which of these sources are relevant to the scenario and how they relate to each other. The generated Skill is then validated by executing its `LoadDataSchema` against the triggering event; if any specified source fails to return the expected fields, the error is fed back to the LLM for regeneration, up to three retries. To facilitate cross-scenario transfer, existing Skills for related scenarios are provided as references during generation.

Update. Once deployed, a Skill may need to evolve as systems change. When a practitioner identifies a gap or an error in the agent's analysis, they describe the issue and expected behavior in natural language through a messaging interface. The framework then revises the appropriate component of the Skill—adjusting `LoadDataSchema` if the agent missed a critical data source or knowledge entry, or adjusting the `Prompt` if the agent retrieved adequate information but reasoned over it incorrectly. All cases follow the same validate-and-retry loop (up to three iterations) and require no code changes.

2.4 Self-Evolving Knowledge and Skill via Unified Feedback

A central design choice of BIAN QUE is that *one* feedback signal simultaneously drives *two* parallel learning pathways: one evolving the knowledge base, one evolving the Skills. Every agent execution produces a case record $c = (e, \{s_i\}, \mathcal{D}, \mathcal{K}, o, f)$, where f is the practitioner's feedback when available. The feedback is dispatched along two pathways:

$$f \longrightarrow \begin{cases} \text{MemoryWrite}(c) \rightarrow \text{KnowledgeDistill}(c) \\ \text{SkillRefine}(\{s_i\},\ f) \end{cases} \tag{4}$$

The knowledge pathway governs the accumulation of distilled operational experience, such as failure patterns and module-specific behavioral norms; the Skill pathway governs which *data* and *knowledge* the agent retrieves and how it reasons over them for each event.

2.4.1 Knowledge Pathway

The knowledge pathway operates in two phases. In the short-term phase, each case record is written into a memory store. Before each reasoning invocation, the agent retrieves the most relevant recent cases as "working memory," enabling it to recognize recurring failure patterns within a recent window. In the long-term phase, each new case also triggers automatic knowledge extraction: the LLM distills reusable insights, such as causal relationships between modules, known failure patterns, and metric interpretation guidelines, into persistent domain knowledge entries. A daily offline process deduplicates and prunes the knowledge base to resolve accumulated contradictions and prevent unbounded growth; corrections take effect immediately through the short-term case memory and need not wait for this cycle. At deployment time, the knowledge base is additionally seeded with a small set of entries imported from operational handbooks, which follow the same indexing scheme and consolidation. To support diverse retrieval needs, the knowledge base employs three complementary indices: a **KV index** for definitive knowledge keyed by business and scenario identifiers; a **KKV index** for relational knowledge, such as the impact of module A on module B for a given metric; and a **vector index** for fuzzy semantic retrieval. Skills address these indices through their `LoadDataSchema` (§2.3), coupling retrieval and knowledge at the query-specification level.

2.4.2 Skill Pathway

The same feedback signal is also analyzed to identify the root cause of any agent error and routed to the corresponding Skill lifecycle operation (§2.3). Two failure modes trigger a Skill **Update**: (1) *inadequate retrieval*, where a critical data source was absent or the wrong knowledge entry was queried, leading to a revision of `LoadDataSchema`; and (2) *flawed reasoning*, where the retrieved information was adequate but the `Prompt` combined it incorrectly, leading to a revision of the `Prompt`. Both operations follow the validation-with-retry procedure of §2.3 and require no code changes. A third case, *incorrect knowledge content*, arises when the retrieved entry itself is wrong or outdated; this falls outside the Skill pathway and is instead handled by the knowledge pathway: corrected knowledge takes effect immediately via short-term case memory, while the long-term entry is pruned or replaced at the next daily consolidation.

2.4.3 Co-Evolution of Knowledge and Skills

The two pathways above are coupled by construction: a richer knowledge base provides more reference entries for newly generated or updated Skills, while higher-quality Skills produce cleaner case records for knowledge distillation. We qualitatively observe this compounding effect in deployment; a controlled quantitative measurement is left to future work, as it requires a longer horizon than our current six-month deployment permits.

3 Experiments

3.1 Experimental Setup

Implementation Details. BIAN QUE is deployed online to operate the e-commerce search engine of the short-video platform, serving hundreds of millions of daily active users. The system comprises dozens of core service modules maintained by a cross-functional team of approximately 100 engineers, with an average of dozens of releases per day. The framework has been in continuous online operation for over six months. The default LLM backbone is Qwen3.5-35B-FP8 deployed on a single NVIDIA Tesla L20 GPU. Skills are stored as structured YAML documents and indexed by keyword tags. The knowledge base uses Redis for KV and KKV indices and a dedicated ANN service for the vector index. The memory store retains a rolling window of recent cases with daily consolidation. All pass@k experiments use temperature sampling at $T = 0.3$; other experiments use greedy decoding unless otherwise noted. For each pass@k evaluation, the Skill directory is reset to a clean seed state before the run, so that Skills from prior evaluation runs do not leak into the current run.

Datasets and Evaluation Metrics. Table 1 summarizes the three patterns and where each is evaluated. We adopt four metrics. **Alert Volume Reduction** measures the relative decrease in alerts requiring human attention after deployment. **RCA Accuracy** is the proportion of root-cause analyses confirmed correct by practitioners. **MTTR** (Mean Time to Resolution) is the average time from alert

Table 1: The three canonical operational patterns and their evaluation in this paper.

Pattern	Target Problem	Core Capability	Evaluated In
Release Interception	Incidents from releases	Single-event detection	Alert vol. reduction
Proactive Inspection	Latent system risks	Multi-event analysis	PASS@k
Alert Root Cause Analysis	Post-alert diagnosis & remediation	Correlation-based attribution	PASS@k; RCA acc.; MTTR

trigger to issue resolution. **PASS@k** runs each event k times and counts it as passed if at least one run produces a correct Skill yielding the ground-truth diagnosis.

3.2 Online Production Deployment Results

Table 2 reports the key operational metrics before and after BIAN QUE deployment over a six-month evaluation period. Relative numbers are normalized to the pre-deployment baseline. The third row (absolute non-actionable alerts) is the compound of rows 1 and 2 and is the most directly interpretable measure of practitioner pager load.

Table 2: Online production deployment results over a six-month window, comparing the month preceding BIAN QUE rollout to the six-month post-rollout period.

Metric	Pre-deployment	Post-deployment
Alert load (upstream and downstream effects):		
Fired alerts (relative)	100%	25% (↓75%)
Non-actionable ratio among fired alerts	80%	15%
Non-actionable alerts, absolute (relative)	100%	∼5% (↓∼95%)
Diagnostic response for fired alerts:		
RCA accuracy	—	80%
Alerts resolved within 5 minutes	—	95% of cases
MTTR (relative)	100%	<50% (↓50%+)

Alert load reduction combines two mechanisms. The 75% drop in fired alerts is primarily driven by release interception and proactive inspection, which resolve problems before they breach alert thresholds. The reduction in non-actionable ratio (80% $\rightarrow$ 15%) results from both upstream and downstream effects: for upstream, the first two lines of defense eliminate unstable releases and slowly-degrading services that previously cascaded into non-actionable alerts; for downstream, the alert RCA agent classifies and suppresses non-actionable alerts rather than paging practitioners. Since the two ratios share the "fired alerts" denominator and that denominator itself shrinks by 75%, they are *not* independent multipliers. The practically relevant quantity is the absolute non-actionable alert volume surfaced to practitioners, which drops to roughly 5% of the baseline ($0.25 \times 0.15/0.80 \approx 0.047$), a roughly 95% reduction in pager noise. We report this compound number in row 3 of Table 2.

Downstream diagnostic quality. For fired alerts that do warrant attention, the alert RCA agent returns a structured diagnosis within 5 minutes in 95% of cases, with an 80% RCA accuracy. This compresses MTTR by over 50%, reflecting not only faster diagnosis but also the reduction in cognitive load on practitioners, who now receive evidence-backed recommendations rather than raw monitoring dashboards. The 80% RCA accuracy, while demonstrating strong practical value, leaves room for improvement; our error analysis reveals that the majority of incorrect diagnoses stem from insufficient domain knowledge for newly deployed services, a limitation that the self-feedback mechanism (§2.4) is designed to address over time and that motivates the ablation studies in §3.4.

3.3 Flexible Skill Mechanism Evaluation

Skill Initialization. For offline evaluation of the Flexible Skill mechanism, we curate 104 real operational events sampled from production logs—44 alerts and 60 inspections—each annotated with a ground-truth diagnosis by senior site reliability engineers (SREs). Table 3 reports pass@k for automated Skill generation across the 104-event dataset. Two findings are worth highlighting. First, the overall pass@1 of 78.8% shows that the majority of Skills are correctly generated on the first attempt, which validates the effectiveness of the seed configuration files (source descriptor and capability descriptor) in providing the LLM with enough scaffolding to produce usable Skills. Second, with a modest retry budget of five attempts, the end-to-end generation pipeline reaches pass@5 of 94.2%, leaving only six cases unresolved. These residual failures concentrate on edge conditions that

Table 3: Skill initialization results (pass@k) on 104 real operational events. "pass@k" denotes the fraction of events for which at least one of k independently sampled Skill generations yields the ground-truth diagnosis.

Scenario	Total	pass@1	pass@2	pass@3	pass@4	pass@5
Alert	44	31 (70.5%)	36 (81.8%)	40 (90.9%)	42 (95.5%)	42 (95.5%)
Inspection	60	51 (85.0%)	52 (86.7%)	54 (90.0%)	55 (91.7%)	56 (93.3%)
Overall	**104**	**82 (78.8%)**	**88 (84.6%)**	**94 (90.4%)**	**97 (93.3%)**	**98 (94.2%)**

Table 4: Human-in-the-loop correction on the six cases that failed at pass@5 during initialization. Here "pass@k" denotes the number of correction rounds attempted.

Scenario	Total	pass@1	pass@2	pass@3	pass@4	pass@5
Alert	2	1 (50.0%)	1 (50.0%)	2 (100.0%)	2 (100.0%)	2 (100.0%)
Inspection	4	2 (50.0%)	3 (75.0%)	3 (75.0%)	3 (75.0%)	3 (75.0%)
Overall	**6**	**3 (50.0%)**	**4 (66.7%)**	**5 (83.3%)**	**5 (83.3%)**	**5 (83.3%)**

Table 5: End-to-end pass@k on all 104 events. Every case is first processed through initialization (up to 5 generation attempts, reaching the 94.2% pass@5 baseline in Table 3); column pass@k then reports the result after k additional correction rounds applied to the still-failing cases.

Scenario	Total	pass@1	pass@2	pass@3	pass@4	pass@5
Alert	44	43 (97.7%)	43 (97.7%)	44 (100.0%)	44 (100.0%)	44 (100.0%)
Inspection	60	58 (96.7%)	59 (98.3%)	59 (98.3%)	59 (98.3%)	59 (98.3%)
Overall	**104**	**101 (97.1%)**	**102 (98.1%)**	**103 (99.0%)**	**103 (99.0%)**	**103 (99.0%)**

involve ambiguous causal attribution or novel business domain knowledge absent from the current Skill pool. They motivate the human-in-the-loop correction mechanism we evaluate next.

Human-in-the-Loop Correction. We asked senior SREs to provide targeted natural-language corrections for the six cases that failed at pass@5 in Table 3 (2 alert and 4 inspection). Table 4 reports pass@k k on these six cases alone, proving the effectiveness of human correction and demonstrating its complementarity to the model's self-correction; Table 5 reports end-to-end pass@k on all 104 events under the combination of model's self-skill-initialization followed by human correction on the failed cases, reflecting the overall system performance. Two findings emerge. First, a single round of human feedback closes most of the remaining gap: overall pass@1 on all 104 events rises from 94.2% to 97.1%, confirming that targeted natural-language correction is a strong and efficient supervision signal. Second, the two scenario types converge at different rates: alert Skills reach 100% by pass@3, while inspection Skills plateau at 98.3% from pass@3 onward, as the residual failures involve higher problem complexity that domain-knowledge iteration alone cannot resolve.

3.4 Ablation Study

We evaluate two ablated variants on the same 104-event dataset used in §3.3, under the same pass@k protocol: 1) STATIC: Skills are manually authored and fixed, disabling LLM-driven generation and update. This isolates the contribution of the Flexible Skill mechanism (§2.3). 2) NOKNOW: No historical knowledge is retrieved during reasoning, disabling both short-term case-memory retrieval and long-term distilled-knowledge retrieval (§2.4). Tables 6 and 7 report pass@k for each variant. Both can be compared directly against the full-framework results in Table 3.

Flexible Skill is the most critical component. As shown in Table 6, replacing Flexible Skills with static configurations causes the largest drop: overall pass@5 falls from 94.2% to 83.7% (-10.5 pp), and pass@1 from 78.8% to 65.3% (-13.5 pp). Alert Skills suffer more than inspection Skills (pass@5 -13.7 pp vs. -8.3 pp), as alert root cause analysis involves more complex attribution logic that hand-authored rules struggle to cover. Crucially, 16.3% of cases remain unresolved even at pass@5 (vs. 5.8% for the full framework), indicating that static rules carry irreducible blind spots rather than a sampling deficit.

Table 6: Ablation on STATIC Skills, where Skills are manually authored without LLM-driven generation or updates, tested on the 104-event dataset.

Scenario	Total	pass@1	pass@2	pass@3	pass@4	pass@5
Alert	44	26 (59.1%)	30 (68.1%)	33 (75.0%)	34 (77.2%)	36 (81.8%)
Inspection	60	42 (70.0%)	45 (75.0%)	46 (76.7%)	51 (85.0%)	51 (85.0%)
Overall	**104**	**67 (65.3%)**	**75 (72.1%)**	**79 (75.9%)**	**85 (81.7%)**	**87 (83.7%)**

Table 7: Ablation on NOKNOW, where both case-memory and distilled-knowledge retrieval are disabled during reasoning, tested on the 104-event dataset.

Scenario	Total	pass@1	pass@2	pass@3	pass@4	pass@5
Alert	44	28 (63.6%)	34 (77.3%)	37 (84.1%)	39 (88.6%)	40 (90.9%)
Inspection	60	46 (76.7%)	46 (76.7%)	48 (80.0%)	49 (81.7%)	50 (83.3%)
Overall	**104**	**74 (71.2%)**	**80 (76.9%)**	**85 (81.7%)**	**88 (84.6%)**	**90 (86.5%)**

Historical knowledge is irreplaceable. As shown in Table 7, disabling knowledge retrieval lowers pass@5 from 94.2% to 86.5% (-7.7 pp), with a nearly identical drop at pass@1 (-7.6 pp). Because the gap between pass@1 and pass@5 is preserved, the loss is systematic rather than stochastic—it *cannot* be recovered by drawing more samples. Without historical knowledge of past incidents and service-specific signal patterns, the LLM falls back on generic priors, and additional retries merely resample from the same under-informed distribution.

Feedback-driven refinement sustains long-term accuracy. We provide two complementary pieces of evidence. **Offline:** as shown in Table 5, on the 104-event dataset, a single round of practitioner feedback lifts overall pass@1 from 94.2% to 97.1%, and three rounds reach 99.0%. **Online:** we deployed the feedback pathway on production traffic for a 13-day controlled experiment (Figure 4). The two arms begin from essentially the same initial state, with comparable day-1 accuracy ($\sim$75–80%), but their trajectories diverge sharply thereafter. Without feedback, the system steadily loses coverage as production traffic drifts: newly emerging alert patterns fall outside the scope of the frozen Skill pool and knowledge base, and accuracy declines almost monotonically from $\sim$75% to $\sim$32% by day 13. With the feedback pathway enabled, the system continuously distills failure cases into new Skills and knowledge, and accuracy remains consistently above 80% throughout the 13 days. The transient dip around day 7,triggered by a burst of novel failure modes,is absorbed within two days as the corresponding Skills are synthesized and deployed, indicating that the feedback loop *actively repairs* coverage gaps rather than merely slowing degradation.

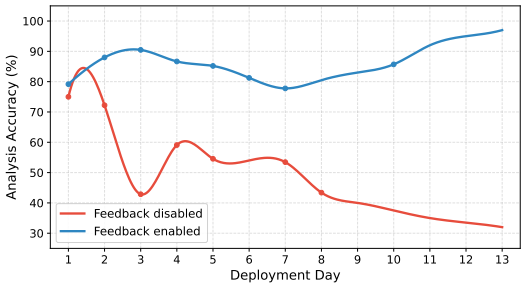

Figure 4: Day-level alert-analysis accuracy on production traffic. Red: feedback disabled; Blue: feedback enabled.

4 Conclusion

We have presented BIAN QUE, an agentic framework for automated intelligent operations of large-scale online engine systems. The framework makes three contributions: a unified operational paradigm that provides complete yet lightweight coverage of the majority of maintenance tasks through three canonical patterns; a Flexible Skill Arrangement mechanism that jointly encapsulates *data* routing and domain *knowledge* and supports LLM-driven Generation and Update; and a unified feedback mechanism that simultaneously drives *knowledge* base evolution and Skill refinement through practitioner signals. Production deployment on a large-scale search engine demonstrates substantial operational improvements: a 75% reduction in fired alerts, a roughly 95% reduction in practitioner-facing non-actionable alerts, 80% RCA accuracy on the alerts that do require attention, and over 50% MTTR compression.

References

[1] OpenClaw. Openclaw. `https://github.com/openclaw/openclaw`, 2026. Open-source personal AI assistant, version 2026.3.8, accessed 2026-03-09.

[2] Anthropic. Claude code overview. `https://code.claude.com/docs/en/overview`, 2026. Official documentation, accessed 2026-03-10.

[3] OpenAI. Harness engineering: leveraging codex in an agent-first world. Engineering blog, 2026. URL `https://openai.com/index/harness-engineering/`. Published: 2026-02-11. Accessed: 2026-03-13.

[4] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. 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*, pages 4966–4974, 2024.

[5] Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Aiwei Liu, Yong Yang, Zhonghai Wu, Xuming Hu, Philip S Yu, and Ying Li. A survey of aiops for failure management in the era of large language models. *arXiv preprint arXiv:2406.11213*, 2024.

[6] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. A survey of aiops methods for failure management. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 12(6):1–45, 2021.

[7] Yinfang Chen, Jiaqi Pan, Jackson Clark, Yiming Su, Noah Zheutlin, Bhavya Bhavya, Rohan Arora, Yu Deng, Saurabh Jha, and Tianyin Xu. Stratus: A multi-agent system for autonomous reliability engineering of modern clouds. *arXiv preprint arXiv:2506.02009*, 2025.

[8] Evelien Riddell, James Riddell, Gengyi Sun, MichaĹ Antkiewicz, and Krzysztof Czarnecki. Stalled, biased, and confused: Uncovering reasoning failures in llms for cloud-based root cause analysis. *arXiv preprint arXiv:2601.22208*, 2026.

[9] Arthur Vitui and Tse-Hsun Chen. Empowering aiops: Leveraging large language models for it operations management. *arXiv preprint arXiv:2501.12461*, 2025.

[10] Mohammad Braei and Sebastian Wagner. Anomaly detection in univariate time-series: A survey on the state-of-the-art. *arXiv preprint arXiv:2004.00433*, 2020.

[11] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, pages 33–40. IEEE, 2017.

[12] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pages 480–490, 2018.

[13] Jacopo Soldani and Antonio Brogi. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)*, 55(3):1–39, 2022.

[14] Siraaj Akhtar, Saad Khan, and Simon Parkinson. Llm-based event log analysis techniques: A survey. *arXiv preprint arXiv:2502.00677*, 2025.

[15] Paulina Toro Isaza, Michael Nidd, Noah Zheutlin, Jae-wook Ahn, Chidansh Amitkumar Bhatt, Yu Deng, Ruchi Mahindru, Martin Franz, Hans Florian, and Salim Roukos. Retrieval augmented generation-based incident resolution recommendation system for it support. *arXiv preprint arXiv:2409.13707*, 2024.

[16] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. Exploring llm-based agents for root cause analysis. In *Companion proceedings of the 32nd ACM international conference on the foundations of software engineering*, pages 208–219, 2024.

[17] Jonathan Pan, Wong Swee Liang, and Yuan Yidi. Raglog: Log anomaly detection using retrieval augmented generation. In *2024 IEEE World Forum on Public Safety Technology (WFPST)*, pages 169–174. IEEE, 2024.

[18] Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Aiwei Liu, Yong Yang, Zhonghai Wu, Xuming Hu, Philip Yu, and Ying Li. A survey of aiops in the era of large language models. *ACM Computing Surveys*, 58(2):1–35, 2025.

[19] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

[20] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36: 68539–68551, 2023.

[21] Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*, 2023.

[22] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

[23] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[24] Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, et al. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*, 2023.

[25] Zihan Liang, Yufei Ma, Ben Chen, Zhipeng Qian, Huangyu Dai, Lingtao Mao, Xuxin Zhang, Chenyi Lei, and Wenwu Ou. Ig-search: Step-level information gain rewards for search-augmented reasoning. *arXiv preprint arXiv:2604.15148*, 2026.

[26] LangChain. The anatomy of an agent harness. Engineering blog, 2026. URL `https://blog.langchain.com/the-anatomy-of-an-agent-harness/`. Published: 2026-03-10. Accessed: 2026-03-12.

[27] Wenhao Wang, Peizhi Niu, Zhao Xu, Zhaoyu Chen, Jian Du, Yaxin Du, Xianghe Pang, Keduan Huang, Yanfeng Wang, Qiang Yan, et al. Mcp-flow: Facilitating llm agents to master real-world, diverse and scaling mcp tools. *arXiv preprint arXiv:2510.24284*, 2025.

[28] Zhuoxuan Jiang, Tianyang Zhang, Haotian Zhang, Yinong Xun, Yang Liu, Dehua Feng, Wen Si, and Shaohua Zhang. Llma4itops: A lightweight llm-based multi-agent framework for it operations and maintenance. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 471–482. Springer, 2025.

[29] Pei Yang, Wanyi Chen, Yuxi Zheng, Xueqian Li, Xiang Li, Haoqin Tu, Jie Xiao, Yifan Pang, Bill Shi, Lynn Ai, et al. Aoi: Turning failed trajectories into training signals for autonomous cloud diagnosis. *arXiv preprint arXiv:2603.03378*, 2026.

[30] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[31] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2023.

[32] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7036–7050, 2024.

[33] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 19724–19731, 2024.

[34] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[35] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[36] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[37] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[38] Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*, 2024.

[39] Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026.

[40] Yaolun Zhang, Yiran Wu, Yijiong Yu, Qingyun Wu, and Huazheng Wang. Live-evo: Online evolution of agentic memory from continuous feedback. *arXiv preprint arXiv:2602.02369*, 2026.

[41] Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, et al. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

A Related Work

Intelligent Operations and AIOps Artificial Intelligence for IT Operations (AIOps) [5] is essential for large-scale online engine systems, empowering search, recommendation, and advertising, ensuring the normal operation of the system and algorithms. Early AIOps research [6, 7, 8, 9] addressed individual sub-problems such as anomaly detection [10], log parsing [11], failure prediction [12], and root cause localization [13] in isolation. More recently, LLM-based approaches have tackled log analysis [14], incident summarization [15], conversational troubleshooting [16, 17], and agent-based diagnosis [4, 18], validating the feasibility of applying LLMs to operational scenarios. However, these works share a common positioning: they assume a well-curated input context and focus on reasoning over it, rather than on assembling that context from raw production signals. BIAN QUE is positioned *one step earlier* in the pipeline, responsible for selecting the right data and knowledge before any diagnostic reasoning happens. A specialized sub-task reasoner like RCAgent [4] can, in principle, be plugged in as the f_{LLM} component of our paradigm (Eq. 1), consuming the $\mathcal{D}(e)$ and $\mathcal{K}(e)$ that our Flexible Skill layer prepares. Our distinctive contributions are accordingly (i) coverage of the operational *lifecycle* under one unified paradigm, and (ii) treating the (*data*, *knowledge*) updating itself, based on both the reasoning and the existing knowledge, as an LLM-generated, continuously evolving artifact.

LLM-based Agents with Evolving Knowledge LLM-based autonomous agents have established core paradigms for tool use and multi-step reasoning [19, 20, 21, 22, 23, 24, 25]. More recent systems such as LangChain [26], Claude Code [2] and OpenClaw [1] further demonstrate how agentic architectures can leverage standardized tool protocols [27] to operate over large-scale heterogeneous tool ecosystems. These frameworks generally assume that the relevant tools and data sources are either pre-specified or discoverable through exploration, an assumption that breaks down in industrial operations where heterogeneous data sources number in the thousands and the right subset varies drastically across business lines, modules, and scenarios [28, 29]. Our Flexible Skill mechanism addresses this by introducing an evolvable abstraction layer in which the data-routing logic itself is LLM-generated and incrementally updated through natural-language interfaces. On the knowledge side, prior work spans RAG [30] and its variants [31, 32], memory-augmented architectures [33, 34, 35, 36], and self-refinement methods [37, 38]. Recent work has further explored self-evolving memory systems that learn to update their own memory operations from feedback [39, 40, 41]. However, conventional RAG treats the knowledge base as static, and most self-evolving mechanisms operate along a single feedback pathway—improving either the knowledge base or the agent behavior, but not both. Our framework introduces a unified feedback mechanism in which each correction signal simultaneously drives memory-to-knowledge distillation and Skill refinement, enabling the two to co-evolve through a shared loop.

Table 8: Comparison of LLM scaling on root-cause analysis (of 36 events). pass@k is reported as correct cases (%).

Model	**pass@1**	**pass@2**	**pass@3**	**pass@4**	**pass@5**
GLM5	28 (77.8%)	30 (83.3%)	32 (88.9%)	33 (91.7%)	35 (97.2%)
DeepSeek-V3.2	28 (77.8%)	31 (86.1%)	33 (91.7%)	35 (97.2%)	36 (100.0%)
Qwen3.5-35B-FP8	26 (72.2%)	29 (80.6%)	32 (88.9%)	34 (94.4%)	34 (94.4%)
Qwen3.5-27B-FP8	23 (63.9%)	25 (69.4%)	29 (80.6%)	30 (83.3%)	32 (88.9%)
Qwen3.5-9B	17 (47.2%)	22 (61.1%)	26 (72.2%)	28 (77.8%)	29 (80.6%)
Qwen3.5-4B	13 (36.1%)	15 (41.7%)	18 (50.0%)	20 (55.6%)	21 (58.3%)
Qwen3.5-0.8B	5 (13.9%)	6 (16.7%)	8 (22.2%)	8 (22.2%)	9 (25.0%)

B LLM Backbone Comparison.

To evaluate sensitivity to the underlying LLM, we compare seven models (of various versions and scalings) on the alert root-cause analysis task with a 36-event subset described in §3.1. Before each evaluation run, the Skill directory is reset to the clean seed state to eliminate cross-run contamination. All models use temperature $T = 0.3$. We find that, frontier models cluster tightly at the top and clearly separate from smaller ones: DeepSeek-V3.2, GLM5, and Qwen3.5-35B achieve pass@1 within a 5.6-point band (77.8%, 77.8%, and 72.2% respectively); DeepSeek-V3.2 is the only model reaching 100% at pass@5. While dropping from 35B to 9B costs 25% of pass@1 (72.2% $\rightarrow$ 47.2%), and further reductions to 4B and 0.8B cause catastrophic degradation (36.1% and 13.9%). Practical

recommendation is to use a backbone of around 35B parameters or larger for online deployment of BIAN QUE; smaller models exhibit substantially less robust behavior on operational reasoning tasks.

C Limitations and Future Work.

Several directions remain open. First, the current framework covers the reasoning and diagnosis stages of operational workflows but does not automate the subsequent execution of remediation actions (e.g., performing rollbacks or scaling operations); extending the framework to closed-loop autonomous remediation is a natural next step. Second, while the Agent Matrix currently employs keyword-based Skill matching, more sophisticated routing mechanisms (e.g., learned embeddings over event metadata) could improve matching accuracy for novel event types. Third, the coordination across multiple agent executions within a single incident remains largely manual; developing orchestration protocols for multi-agent collaboration in complex, cascading failure scenarios presents an interesting research direction. Fourth, directly quantifying the compounding effect between the knowledge pathway and the Skill pathway requires a longer evaluation horizon than our current six-month deployment permits (§2.4.3) and is left to future work. Finally, the residual RCA errors concentrate on newly deployed services for which the knowledge base has not yet accumulated sufficient experience; we expect the self-feedback mechanism to narrow this gap over time, but validating that hypothesis will require a multi-year evaluation.