International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(20s) 2026 pp. 741-751
https://doi.org/10.70917/ijcisim-2026-5217
Article

LLM Prompt Interfaces as Execution Contracts: Reducing Unsafe Tool Invocation Across the Incident Lifecycle in AIOps

Niharika Gupta

Microsoft, USA

Abstract: Natural language interfaces enable flexible interaction with LLM-based AIOps platforms but introduce reliability risks when user input lacks a verified mapping to a supported intent, parameter schema, or policy envelope. Unconstrained prompts can trigger hallucinated procedures, incorrect tool invocation, parameter-level violations such as missing service identifiers or malformed time windows, and silent partial execution that evades post-mortem detection. Guided discovery exposes system capabilities via BM25-plus-embedding hybrid retrieval over a versioned intent catalog, matching exact operational tokens alongside paraphrastic variation. Soft enforcement applies a two-stage slot filling pipeline — deterministic parsers for structured identifiers combined with LLM-based entity extraction — validated against JSON-Schema definitions, converting validation failures into actionable repair prompts rather than hard rejections. Dynamic recommendation maintains lightweight dialog state across incident lifecycle stages and employs a rules-plus-learned ranker to re-rank intents, pre-populate parameters, and suppress sequencing or policy violations contextually. The reference architecture integrates these mechanisms with margin-based confidence gating that enforces stricter thresholds for destructive operations, server-side RBAC and environment policy gates evaluated before tool invocation, and an allowlisted execution registry that rejects hallucinated tool calls. Structured telemetry emitted at each pipeline stage — retrieval scores, extraction accuracy, validation error distributions, policy denial rates, and correction loop metrics — enables governance, drift monitoring, and continuous catalog improvement. This model demonstrates that a meaningful proportion of perceived LLM shortcomings in production AIOps can be mitigated through UX-level constraints that align operator requests with the system's executable capability set. In production systems such as incident monitoring copilots, conversational agents invoke runbooks, diagnostic queries, or remediation workflows through downstream tool interfaces. Prompt-level ambiguity may therefore propagate into incorrect execution plans, such as mis-scoped tenant queries, environment-mismatched remediation, or sequencing violations in mitigation workflows. The reliability of an operational agent is thus determined not solely by model reasoning but by the safety of the interface layer mediating prompt-to-execution translation.

Keywords: Constrained Prompt UX, AIOps Reliability, Intent Catalog Validation, Guided Prompt Discovery, Policy-Gated Execution

1. Introduction

AIOps platforms based on LLM assistants are regularly evaluated on model capability alone, but a meaningful proportion of production failures originate at the interface edge: the transformation of ambiguous natural language into a definite execution plan over workflows, APIs, and runbooks. The conditions under which these assistants operate are characterized by continuous outage risk and growing cost pressure. The Uptime Institute's 2024 annual resiliency survey showed that 55% of operators experienced an outage in the last three years, down from 60% in 2022 and 69% in 2021, confirming that outages remain an operational reality across most organizations [1]. On an international scale, the Uptime Institute estimates approximately 10–20 high-profile IT downtime and data center incidents annually that produce severe repercussions across business workflows, customer experience, and regulatory reputation [1]. These figures define the operating conditions within which conversational AI assistants must function

— high-stakes, time-pressured environments where the cost of misinterpreting a prompt or incorrectly activating a workflow is not user frustration but downtime, financial loss, and reputational harm.

A critical and underexamined failure path runs directly from prompt ambiguity to customer impact through a sequence of compounding execution errors. Prompt ambiguity produces an invalid or misscoped execution plan — an intent resolution that selects the wrong workflow, omits required parameters, or violates sequencing constraints. An invalid execution plan drives incorrect tool invocation — triggering runbooks against the wrong environment, executing diagnostic queries scoped to the wrong tenant, or initiating remediation before qualification is complete. Incorrect tool invocation in a production AIOps system translates directly into customer impact: service disruption, data exposure, prolonged incident duration, or regulatory non-compliance. This causal chain — prompt ambiguity → invalid execution plan → tool invocation risk → customer impact — is not a theoretical failure path; it is the operational mechanism by which interface-layer failures become production incidents. The constrained prompt UX model is designed to interrupt this chain at its earliest point, at the interface layer, before ambiguity propagates into execution.

In practice, the immediate interaction layer of an AIOps assistant is a weakly typed, implicitly conditioned informal programming interface whose failure behavior is unspecified. Empirical studies on automated root cause analysis with LLM agents have shown that the reliability of operational task execution at scale is bounded not only by model reasoning but equally by the integrity of the interface layer translating prompts into execution plans [2]. This paper therefore restructures prompt UX as an engineering problem in interface design and systems reliability — formalizing supported intents as versioned workflow selectors, verifying inputs against explicit schemas, and instrumenting the interaction pipeline to prevent invalid or unsafe executions before they reach downstream routing and tool layers.

2. Free-Form Prompt Failure Modes

When users engage with an AIOps assistant using unstructured natural language, various types of failures are consistently apparent, and understanding these failure modes is the precondition for making effective constraints. These failures are not merely hypothetical issues; they propagate into lower reasoning and execution layers, increasing in magnitude and complicating root-cause analysis significantly. The practical effects of this propagation can be highlighted by recent industry statistics on outage preventability: a study conducted by the Uptime Institute in the annual outage analysis of 2025 revealed that four out of five (approximately 80 percent) respondents claimed that their last serious outage could have been prevented with more proper management, processes, and configuration [3]. This observation is directly applicable to constrained prompt UX since the failure types it targets — out-of-catalog intent, schema violation, over-broad intent mapping, workflow mismatch, grounding gap, and policy violation — are inherently failures of process, configuration, and management at the interface level.

In this model, an intent functions as a workflow selector: resolving an intent does not directly invoke a single tool or API endpoint but instead selects a defined operational workflow — a sequenced set of runbooks, diagnostic queries, or remediation procedures — that governs how downstream tool calls are composed and ordered. Tool invocation is therefore a consequence of workflow selection, not a direct output of intent matching. This distinction is operationally significant: a mis-resolved intent does not merely call the wrong function; it activates the wrong workflow, which may include multiple incorrect or unsafe execution steps across dependent systems.

Guided discovery ensures that operators do not initiate requests that lack any executable workflow support in the catalog. Schema validation guarantees that the parameters required to instantiate a workflow safely are present, correctly typed, and internally consistent before the workflow is activated. RBAC, environment constraints, and change-management approvals are server-side controls imposed by policy gates, which cannot be circumvented through conversational phrasing. Collectively, these mechanisms address precisely the type of avoidable failures that Uptime data identifies as the leading cause of major outages in the industry.

This argument is further supported by the trend in human procedural breakdown. In the Uptime Institute 2025 outage analysis, the proportion of outages attributable to failure to follow established procedures grew by 10 percentage points relative to 2020 results [3]. This is a substantial increase, indicating that although infrastructure technology has improved, the human-process interface remains a growing weakness — one that makes operators experiencing cognitive load during incident response increasingly likely to deviate from established procedures. Constrained prompt UX specifically addresses this dynamic: soft enforcement steers operators toward validated intents and pre-populated parameters rather than relying on memory or ad-hoc phrasing, and policy gates enforce workflow sequencing constraints (e.g., diagnostic workflows must complete before remediation workflows are unlocked) regardless of how the operator phrases the request. Procedural knowledge becomes embedded in the interface itself, removing dependence on individual operator recall under high-pressure conditions.

Empirical studies of large language models operating over large API surfaces have demonstrated that the reliability of execution decreases as the tool collection and API endpoint count grows, and that structured function signatures with explicit parameter schemas substantially reduce hallucinated invocations compared to free-form selection [4]. In the constrained prompt UX model, this finding applies at the workflow level: because each intent maps to a registered, versioned workflow rather than an ad-hoc tool call, the attack surface for hallucination and mismatch is systematically bounded. The most frequent failure modes — missing environment specifiers, absent service identifiers, malformed time windows, and incorrect workflow selection — are intercepted and surfaced by the validation layer rather than manifesting as undetected or unexpected behavior during or after execution. These are not model-level reasoning failures requiring a more capable model; they are interface-level failures requiring better-constrained inputs, and the workflow-selector model of intent resolution addresses them at their source.

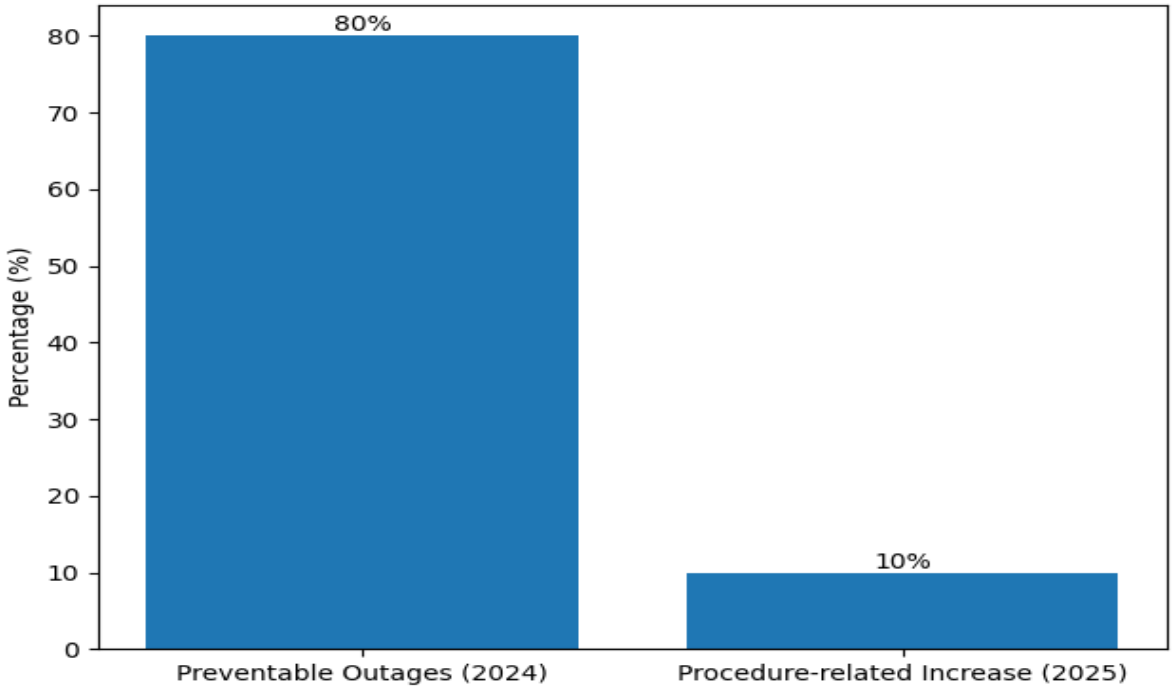

Fig. 1: Outage Trends and Preventability Metrics [3, 4]

3. Constrained Prompt UX Model

The constrained prompt UX model handles the above-named failure modes via three interlocking mechanisms: guided discovery, soft enforcement, and dynamic recommendation, which together help to turn the prompt layer into a structured, validated, and policy-checked interface.

3.1 Guided Prompt Discovery

Prompts that are supported are presented through a hybrid retrieval layer on top of a prompt/runbook registry. The registry contains intent definitions, natural-language descriptions, sample utterances, and executable metadata such as tool bindings, required parameters, and validation rules. User input is incorporated and compared with this catalog in two complementary ways: lexical scoring of exact operational tokens, e.g., service names, incident identifiers, and environment labels, and embedding similarity of paraphrases and semantically equivalent requests. Experiments of hybrid retrieval-augmented generation as applied to enterprise knowledge systems have shown that joint dense and sparse retrieval techniques outperform single methods either individually or together in a uniformly better recall and precision rate than either dense or sparse methods in a domain-specific case where both precise terminology and varying natural language formulations are present [5]. This observation is the direct basis of the architectural decision to combine BM25-style keyword matching with dense vector similarity in the prompt discovery layer. The lexical component is necessary to make sure that the operational identifier service names, region codes, and incident keys are an exact match, and the embedding component inherits the paraphrastic variation, which is unavoidable when operators state the same intent in different words under time pressure. The implication is a prioritized list of candidate intent and consumer-induced confidence scores, allowing the system to collapse the ambiguous text to the execution layer and instead present the operator with a refined list of actionable options.

Mechanism	Function	Key Technique	Outcome
Guided Prompt Discovery	Expose supported intents to operators	Hybrid retrieval (BM25 + dense embedding) over prompt/runbook registry	Ranked candidate intent set with confidence scores
Soft Enforcement	Converge nonconforming input toward valid execution requests	Nearest-neighbor intent surfacing, structured slot filling, JSON-Schema validation with repair prompts	Iterative refinement preserving conversational flow
Dynamic Prompt Recommendation	Context-aware suggestion of next-best actions	Rules + learned ranker hybrid using dialog state, telemetry, and policy constraints	Pre-populated parameters, suppressed unsafe actions, reduced cognitive load

Table 1. Constrained Prompt UX Model—Mechanism Summary [5, 6]

3.2 Soft Enforcement

Instead of rejecting nonconforming input outright and interrupting the flow of conversation, the system uses assistive constraints to bring the system to a valid execution request. The soft enforcement loop reads the next-closest entity recognition and parameter extraction intents to the catalog and verifies the output payload against the JSON-Schema definition of the intent. Validation errors are transformed into usable, user-friendly prompts indicating missing fields, accepted values, and range limits, which allow the process to be refined through multiple improvements without breaking the conversational flow. Studies of user confidence in conversational AI systems has demonstrated that openness regarding system capability and constrainment, such as direct expression of system ability and failure to perform a specific request, as mentioned, are the main factors that drive long-term user involvement and adjustments in trust perception [6]. This is the main principle of soft enforcement: instead of generating a nonspecific refusal or, even worse, simply continuing to function with a half-baked or inaccurate execution, the system will give well-structured, specific feedback that will enable the operator to know the difference between what he or she requested and what the system needs. The iteration refinement cycle retains the merits of conversational interaction and validates that schema and policy-valid requests make it to the execution layer.

3.3 Dynamic Prompt Recommendation

Recommendations are actively updated depending on the context of the conversation and the state of the system. A lightweight dialog state tracks the current incident or ticket identifier, selected service and environment, active lifecycle phase, current time window, and the history of previous workflows invoked and their results. This state is employed to re-rank candidate intents based on contextual relevance, pre-populate known parameters to minimize operator intervention, and suppress actions that do not satisfy policy or sequencing requirements.

A central organizing principle of dynamic recommendation in AIOps is that the incident lifecycle itself defines a natural sequencing contract over allowed action classes. Rather than treating all registered workflows as equally available at any point in a conversation, the recommender constrains the visible and executable intent set to those appropriate for the current lifecycle phase. The following table illustrates this phase-gated action model:

Lifecycle Phase	Allowed Action Class
Detection	Read-only diagnostics
Qualification	Scoped RCA queries
Mitigation	Change-approved remediation
Recovery	Validation checks
Post-mortem	Report generation

Table 2. AIOps Incident Lifecycle Phases and Allowed Workflow Action Classes

This structure ensures, for example, that remediation workflows are not surfaced or executable during the Detection phase, and that change-approved actions require the system to confirm that Qualification has been completed and its outputs recorded in dialog state. The lifecycle phase is inferred from the current incident record, ticketing system context, or explicit operator declaration, and is updated as the incident progresses. Suppression of out-of-phase actions is enforced at both the recommendation layer — preventing the intent from appearing as a suggestion — and the policy gate layer — rejecting execution even if the operator bypasses discovery and enters the intent directly.

The recommender is modeled as a rules-plus-learned ranker hybrid. The rules layer encodes operational sequencing constraints and policy requirements, including the lifecycle phase boundaries described above. The learned ranker is trained on interaction telemetry, including clickthrough rates, successful execution rates, and frequency of correction loops, enabling the system to improve phase-appropriate suggestions over time as operational patterns accumulate. Context-sensitive recommendation transforms the assistant from a passive command executor into a proactive operational partner that anticipates the operator's next action based on the current state of the incident and the boundaries of safe, policy-compliant execution.

4. Reference Architecture and Implementation

4.1 End-to-End Pipeline

The constrained prompt UX pipeline is structured as a series of phases, each with clearly defined inputs, outputs, and failure semantics. The pipeline begins with natural language prompt ingestion, traverses intent retrieval and ranking, proceeds through intent selection with parameter extraction and schema validation, and terminates in a multi-gate execution model before any workflow or tool is activated.

A central architectural commitment of this pipeline is that intent resolution does not directly invoke a tool or API endpoint. Instead, a resolved intent instantiates a **Diagnostic Workflow DAG** — a directed acyclic graph of sequenced operational steps comprising runbook activations, diagnostic queries, and conditional remediation branches. Execution of this DAG is then subject to three sequential enforcement gates before any remediation action is permitted:

Intent Resolution

↓

Diagnostic Workflow DAG

↓

Policy Gate

(RBAC, change-management, rate limits)

↓

Environment Gate

(environment match, tenant scope, region boundary)

↓

Remediation Gate

(dry-run confirmation, blast radius check, idempotency key)

↓

Workflow Execution

Each gate is an independent enforcement point evaluated server-side, prior to the activation of the next DAG stage. A denial at any gate produces a structured, actionable response — identifying the gate, the specific rule or constraint violated, and the resolution path — rather than a silent failure or generic error. This architecture ensures that the path from a resolved intent to a remediation action passes through three distinct safety checkpoints, each targeting a different category of execution risk.

Studies on structured output generation and validation in tool-using language agents have established that applying machine-verifiable output contracts to tool invocations — combining typed parameter structures, tool libraries, and deterministic post-processing — substantially reduces execution errors compared to uncontracted generation [7]. The three-gate DAG model extends this principle from individual tool calls to full workflow execution, applying sequential contractual enforcement at the policy, environment, and remediation boundaries.

Pipeline Stage	Input	Output	Failure Handling
Intent Retrieval + Ranking	Natural language prompt	Ranked candidate intents with confidence scores	Low confidence triggers selection mode (top-k display)
Parameter Extraction + Schema Validation	Selected intent + user input	Validated JSON payload against intent schema	Structured repair prompts for missing or invalid fields
Diagnostic Workflow DAG Instantiation	Validated intent and parameters	Executable DAG of sequenced operational steps	Unresolvable DAG triggers catalog escalation path
Policy Gate	DAG + resolved parameters	Allow or deny with rule identifier	Denial with actionable explanation and escalation path
Environment Gate	Policy-approved DAG	Scoped execution context	Scope mismatch triggers environment clarification prompt
Remediation Gate	Environment-scoped DAG	Confirmed execution plan with idempotency key	Dry-run abandonment logged; destructive action

			requires explicit confirmation
Workflow Execution	Gate-approved payload	Execution result with structured status	Typed failure codes (auth, transient, dependency) fed into refinement loop

Table 3. Reference Architecture—Pipeline Stage Summary [7, 8]

4.2 Intent Catalog and Data Model

Each supported prompt is modeled as an intent record where identifiers are stable and metadata is executable. A base record consists of an intent identifier, display name, description, sample utterances, required and optional parameters defined as machine-readable schemas with types, regular expression patterns, and cross-field constraints, and an execution binding containing the workflow DAG reference, operation version, preconditions, postconditions, and explicit versioning for both the intent and its schema. The recommendation component takes the user query, conversation history, and available prompts as input and outputs recommendations under a rigid output contract — a JSON array of title-prompt pairs — that is deterministically parsed and validated against the known registry, ensuring the model selects from enumerated workflow choices rather than generating novel commands.

4.3 Parameter Extraction, Validation, and Policy Gating

Parameter extraction in the constrained prompt UX pipeline is performed through a structured, multi-stage slot-filling architecture in which extraction responsibility is partitioned by identifier type and reliability requirement.

Deterministic Identifier Extraction Layer

The first stage is a deterministic identifier extraction layer that operates prior to any LLM-based or hybrid retrieval processing. This layer applies rule-based and pattern-matched parsers to extract structured identifiers whose format is fully enumerable and verifiable without model inference. Extracted identifier classes include incident and ticket keys, timestamps and time window expressions, environment enumerations such as production, staging, and development designators, region and availability zone codes, service identifiers, and tenant or account scope qualifiers. Because these identifiers carry direct operational consequence — an incorrect environment label or a malformed time window can scope a workflow to the wrong system boundary — deterministic extraction ensures that their values are captured with full fidelity and are not subject to the paraphrastic variation or generalization behavior of LLM-based extraction. Identifiers successfully extracted at this stage are locked as validated slot values and passed forward as ground-truth inputs to subsequent pipeline stages, bypassing re-extraction.

LLM-Based Entity Extraction and Hybrid Retrieval

For free-form and semantically variable entities that cannot be reliably captured by deterministic parsers — such as symptom descriptions, affected component references expressed in natural language, or operator-stated urgency qualifiers — an LLM-based extraction step generates a candidate JSON object representing the remaining unfilled slots. This extraction operates over the residual prompt content after deterministic slots have been resolved, reducing the surface area over which the model must reason and limiting the opportunity for hallucinated or misscoped values to enter the payload.

Schema Validation and Policy Gating

The combined output of deterministic and LLM-based extraction is validated against the JSON-Schema definition of the selected intent. Validation errors are translated into structured, user-friendly repair prompts that identify missing fields, accepted value ranges, and format requirements, enabling iterative refinement without breaking conversational flow. In cases of ambiguous fields where the action is destructive, the system favors explicit disambiguation over best-guess completion.

Prior to any tool invocation or workflow activation, a policy gate evaluates the resolved intent and parameter payload against RBAC policies, environment policies, rate limits, and change-management requirements. Policy-guided enforcement is implemented as a server-side architectural constraint evaluated independently of model output, ensuring that policy compliance cannot be bypassed through conversational phrasing or model-generated parameter values [8]. An allowlisted execution registry rejects hallucinated workflow references and unsafe function calls, and destructive operations require dry-run support, explicit operator confirmation, and idempotency keys for safe retry handling.

4.4 Observability, Telemetry, and Safety

Structured events are produced at every pipeline stage - retrieval scores, intent selection confidence, chosen versions of the extraction schema, validation errors, policy choices, tool call latency and status, and execution results - that governance dashboards, drift monitoring, and specified improvements to the catalog. The pipeline considers user text untrusted, system and tool instructions are not mixed with user content, the model is forbidden to produce tool names, input screening is used to identify known injections, and suspicious sessions are placed in safe mode with read-only intents and other checks.

5. Discussion

5.1 Coverage, Flexibility, and Calibration

A catalog-based strategy is more accurate but may be restrictive in cases of new circumstances. The teams are supposed to anticipate the long tail of out-of-catalog requests and use it as a signal of the product, prioritizing them into new intents and versioning the catalog in a manner such that addition does not disrupt current behavior. The cornerstone of a safe operation is confidence gating: during periods when the system is not sure about similar intents, this means that it is better to fall back to selection and clarification rather than to do nothing. The risk should be risk-conscious thresholds needed to allow read-only intentions with less certainty, gating with great severity actions that are destructive, and thresholds should be recalibrated with telemetry since the distribution of retrieval and working vocabulary changes over time. There is much money at stake when it comes to the financial aspect of making this calibration accurate. The report by Uptime Institute published in 2023 on their annual outage analysis shows that over two-thirds of outages incurred over 100,000 USD, which confirms the financial importance of mitigating invalid or unsound executions by means of schema validation and policy gating has more than just visible financial benefits (it improves user experiences) but also direct operational savings [9]. When one miscategorized intent or missing environment parameter can cause an undesired production change, and the median cost of the outage it causes is greater than six figures, the investment in constrained UX catalog curation, schema design, the calibration of confidence threshold, and the maintenance of policy gates is a high-payoff reliability investment.

5.2 Failure Semantics, Trust, and Evaluation

The primary benefit of constrained UX is that failures are made explicit and categorical — "missing parameter," "policy denied," or "workflow unavailable" — rather than manifesting as silent conversational breakdown or undetected partial execution. These failure signals should be consistently accompanied by actionable next steps, such as parameter recommendations, runbook links, or escalation paths, which increase operator trust and minimize time spent debugging the assistant itself.

Assessment must be approached carefully. Offline measures such as retrieval hit rate and schema validity rate indicate pipeline health but do not confirm operational value when the catalog is not grounded in real operational tasks. Task completion rates alone can conceal execution quality problems when policy refusals are not measured, correction loops are not counted, or operator confirmation steps are treated as friction rather than safety signals. Metrics must therefore be decomposed by risk level and lifecycle phase to accurately reflect whether the constrained UX is producing safer, more reliable outcomes rather than merely higher throughput.

The following metrics are proposed as the primary evaluation framework for a deployed constrained prompt UX system, each selected because it surfaces a distinct category of execution reliability failure:

Metric	Why It Matters
Successful execution rate	Intent correctness
Rollback rate	Execution misalignment
Dry-run abandonment rate	Operator distrust
Policy denial override rate	Catalog gap
Post-execution RCA correction	Remediation misfire

Table 4. Evaluation Metrics for Constrained Prompt UX in Production AIOps Systems

Successful execution rate measures whether resolved intents produce the intended workflow outcome, serving as the primary signal of intent correctness across the catalog. Rollback rate identifies cases where execution completed but produced an incorrect or unsafe system state, indicating misalignment between the resolved workflow and the operator's actual intent. Dry-run abandonment rate captures instances where operators initiated a dry-run preview but declined to proceed, which, when elevated, signals operator distrust in the system's proposed execution plan rather than a model or schema failure. Policy denial override rate tracks how frequently operators seek to bypass policy gates, functioning as a leading indicator of catalog gaps where legitimate operational needs lack a supported, policy-compliant workflow. Post-execution RCA correction rate measures how often a completed remediation workflow is followed by a root cause analysis correction, indicating that the selected workflow addressed symptoms rather than the underlying fault.

These metrics are expected to be reported in two categories — completion and denial rates, and success and rollback rates — separated by risk level and lifecycle phase. The telemetry structure must be aligned to industry severity standards so that the constrained UX demonstrably shifts incident distributions toward lower severity outcomes. In an overview of Uptime Institute outage results, 76% of the 768 operators surveyed reported that their outages caused little or no impact [10]. This distribution provides a substantive baseline: the aim of constrained prompt UX telemetry is to increase the proportion of incidents resolved at low severity by intercepting interface-layer errors before they propagate to execution, and to provide early warning when telemetry distributions shift toward higher severity, signaling catalog drift, policy gap accumulation, or emerging failure patterns not yet represented in the intent registry.

5.3 Organizational Adoption and Future Directions

The interface contract in which prompts are treated implies ownership: the teams should determine the identity of the intent author, the identity of the schema and policy change reviews, and emergency overrides. Effective deployments are similar to API governance—change control, versioning, depreciation policy, audit logs, and UX iteration through operator feedback. Future directions, such as learning how to better disambiguate interaction data based on the interaction data, automatically suggesting new intents based on repeated out-of-catalog interaction by clustering and generating templates, and more closely integrating with change management, such as automatically estimating blast radii and generating rollback plans without modifying the typed policy-checked interaction model, have been proposed. The overall direction is towards timely interfaces that are no longer constrained but in a continuous self-improving way: through structured telemetry to detect coverage holes, calibration drift, and new patterns of operation, and feed this information back into catalog expansion and threshold adjustment in a closed-loop governance cycle.

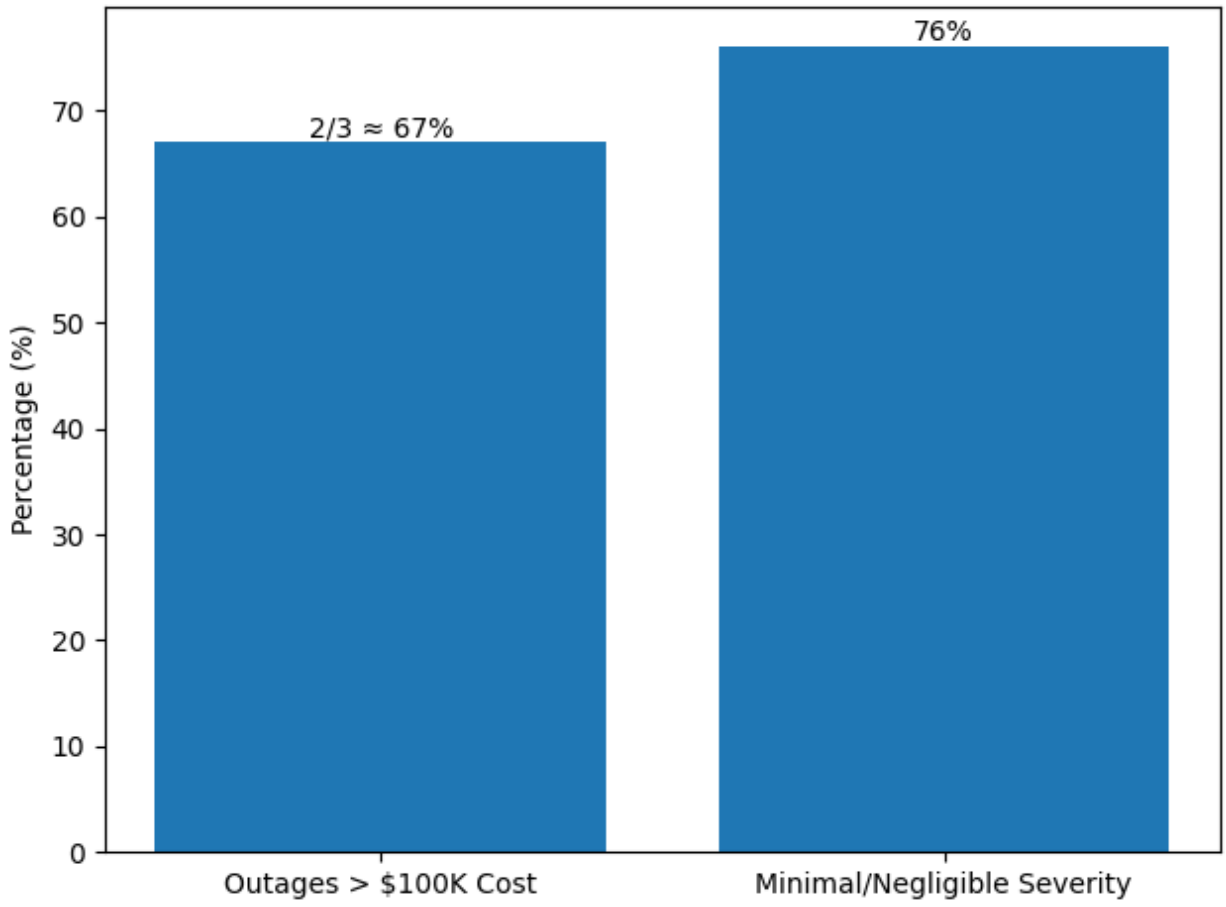

Fig. 2: Outage Cost and Severity Distribution [9, 10]

6. Conclusion

Constrained prompt UX is not a constraint placed on conversational AI but a reliability mechanism related to AIOps production systems. The organizations can mitigate the free-form failure modes that contribute a large portion of the LLM-related production incidents by handling the prompt layer as a typed interface contract instead of an unconstrained text field. The three components of this model, guided discovery, soft enforcement, and dynamic recommendation, work together to guide the operators towards being supported by proven system intents and maintain the conversational freedom that makes natural language interfaces attractive in operational environments. Guided discovery makes sure that the operators have an opportunity to find the capabilities that the system supports using hybrid retrieval (lexical precision and semantic flexibility). Soft enforcement makes requests based on executable schemas without interrupting conversational flow, turning validation errors into repair prompts that can be edited to refine the request instead of being rejected. Dynamic recommendation makes sure that context-sensitive recommendations ease the cognitive load in incident response at high pressure, pre-populating parameters and inhibiting action that is prohibited under policy or sequence. The reference architecture, which comprises hybrid retrieval, confidence-gated intent selection, schema-validated parameter extraction, policy-checked execution, and structured telemetry at all levels, offers a tangible, reproducible implementation trajectory to any organization wishing to realize the deployment of the LLM-assisted operations on the scale of an enterprise. More importantly, this structured telemetry generated by every stage of the pipeline allows evaluating it, governance thereof, and constant improvement of the catalog in a continuous cycle that improves as time goes on. The larger point here is that the focus on model capability predominantly existing is not enough to make operational AI reliable. Much of what is typically referred to as the shortcomings in the production deployments of what is often termed as the LLM are interface design

failures—failures to limit the action space, failures to validate inputs, and failures to make system boundaries visible to the user. Limited timely UX serves as the interface and operational AI reliability with human-computer interaction and provides a philosophical way to build an artificial intelligence that is not merely competent but also responsible, auditable, and reliable enough in practice.

REFERENCES

1. Uptime Institute, "Executive summary Uptime Intelligence report: Annual outage analysis 2024," 2024. [Online]. Available: https://datacenter.uptimeinstitute.com/rs/711-RIA-145/images/2024.Resiliency.Survey.ExecSum.pdf
2. Diego Clerissi et al., "DBInputs: Exploiting Persistent Data to Improve Automated GUI Testing," IEEE Transactions on Software Engineering, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10624678
3. Uptime Institute, "Uptime Announces Annual Outage Analysis Report 2025," 2025. [Online]. Available: https://uptimeinstitute.com/about-ui/press-releases/uptime-announces-annual-outage-analysis-report-2025
4. Kuppani Sathish et al., "Robust Design of Floor Cleaning Robot with Smart Power Handling Abilities Based on Solar Assisted Renewable Energy Sources," 2024 5th International Conference on Electronics and Sustainable Communication Systems (ICESC), 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10690142
5. Jintao Wan et al., "Lane-Changing Tracking Control of Automated Vehicle Platoon Based on MA-DDPG and Adaptive MPC," IEEE Access, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10478923
6. Yao Wei et al., "BuilDiff: 3D Building Shape Generation using Single-Image Conditional Point Cloud Diffusion Models," 2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10350742
7. Mingda Hu et al., "Accelerating Federated Learning With Model Segmentation for Edge Networks," IEEE Transactions on Green Communications and Networking, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10587213
8. Anya Sharma et al., "Policy-Guided LLM Agents for Automated Remediation in DevSecOps Workflows," ResearchGate, 2025. [Online]. Available: https://www.researchgate.net/publication/400668524_Policy-Guided_LLM_Agents_for_Automated_Remediation_in_DevSecOps_Workflows
9. Andy Lawrence and Lenny Simon, "Annual outages analysis 2023," Uptime Institute, 2023. [Online]. Available: https://datacenter.uptimeinstitute.com/rs/711-RIA-145/images/AnnualOutageAnalysis2023.03092023.pdf
10. Jason Ma, "Uptime Institute: Outages in 2024 less frequent and severe but more expensive," Data Center Dynamics, 2025. [Online]. Available: https://www.datacenterdynamics.com/en/news/uptime-institute-outages-in-2024-less-frequent-and-severe-but-more-expensive/