ByteDance

Proactive Change Risk Detection in Production Cloud Systems: ByteDance's Experience

Jinyang Liu, Yichen Li, Tieying Zhang*, Binbin Chen,

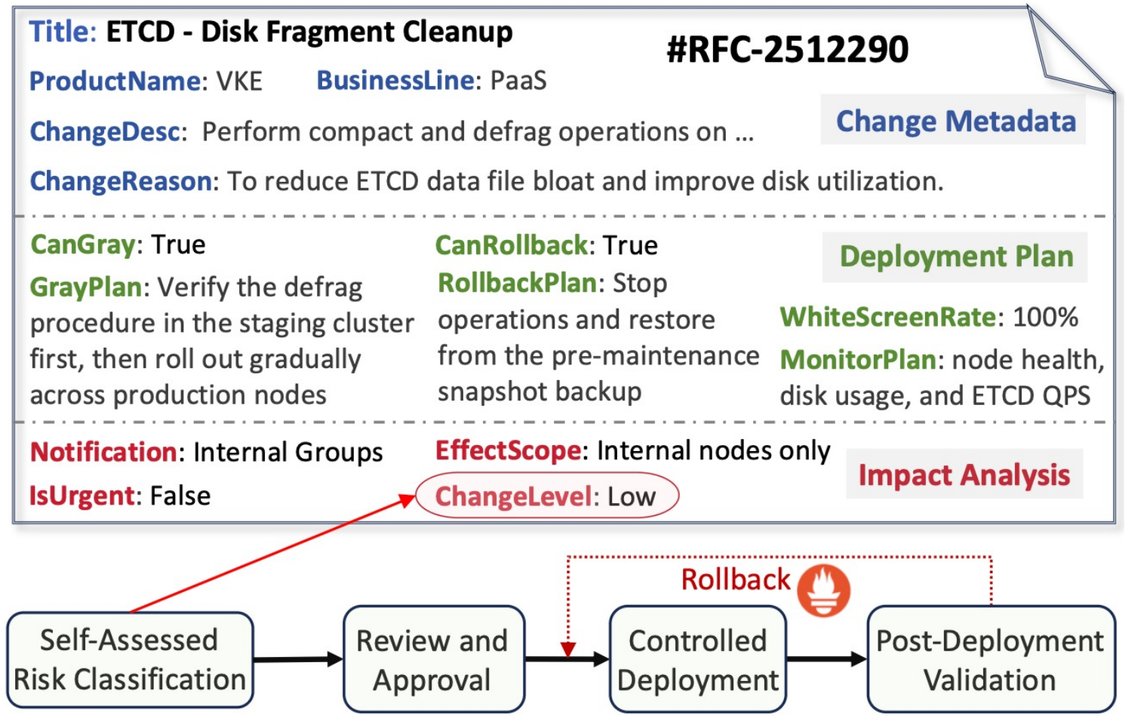
Xiao He, Zhihan Jiang, Haipeng Zhang, Gang Wu, Yi Li

ByteBrain Team

EuroSys '26, Edinburgh, Scotland | April 27–30, 2026

THE VELOCITY-STABILITY PARADOX

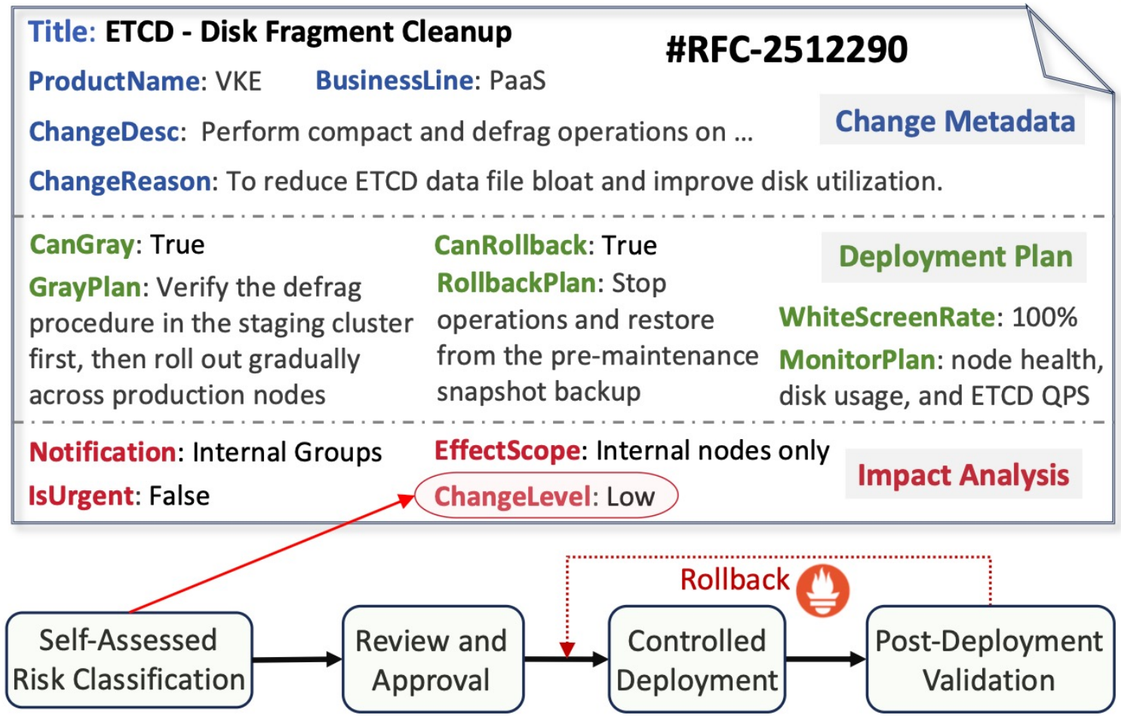
- Modern cloud services require **thousands of daily changes** for innovation
- Each change (deployments, configs, infrastructure patches) introduces **risk**
- Post-mortems from Google, AWS, Azure identify changes as the **MAIN cause of incidents**



A Sample RFC (request for change) from Volcano Engine

A single problematic change can cascade into widespread disruption.

THE TIERED REVIEW PIPELINE



RFC Lifecycle in Volcano Engine's CMP

Engineers self-assess risk as **Low / Medium / High**.

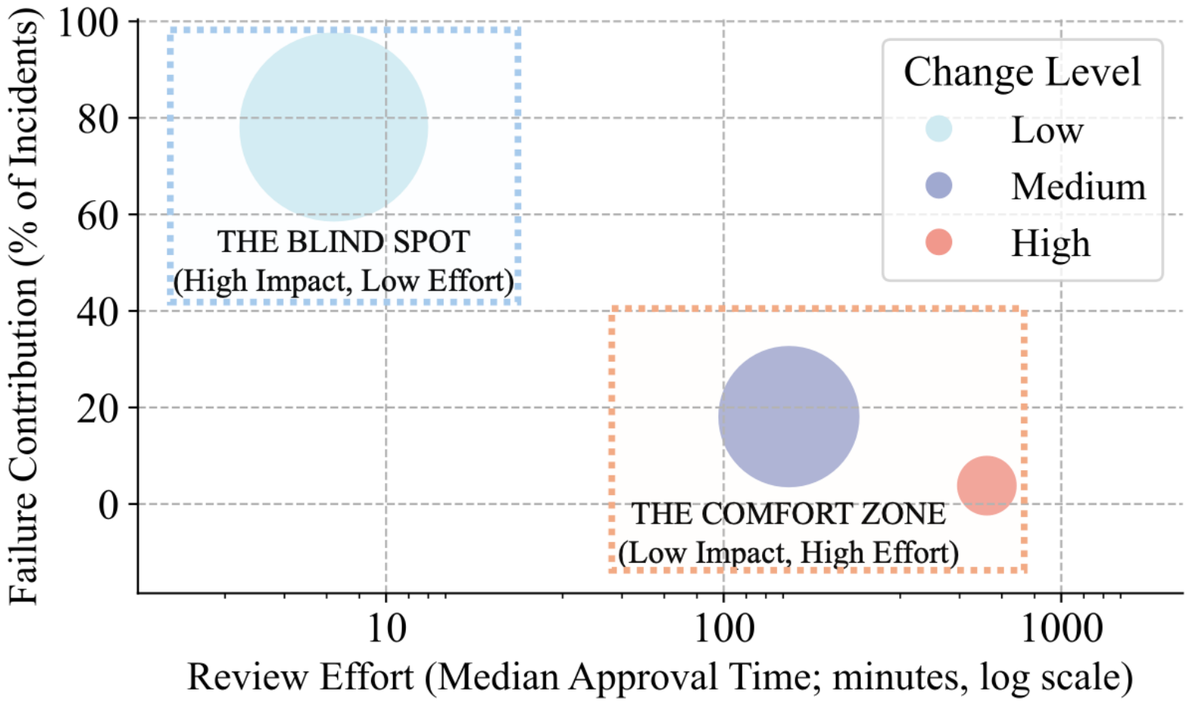
This classification determines the entire review path:

7 min median review for **low-risk**

602 min (10+ hrs) for **high-risk**

Risk classification shapes every step: who reviews, how long, and what gates apply.

THE BLIND SPOT



Bubble size ∝ total RFC volume. Low-risk changes dominate incident volume.

The Blind Spot

Low-risk changes cause **78.1%** of incidents with only minimal review effort.

The Comfort Zone

High-risk changes consume 10+ hours of review but cause only **3.8%** of incidents.

THE 78.1% PROBLEM

78.1%

of change-induced incidents originate from **low-risk changes**

3.8%

from high-risk changes (despite 10+ hrs review each)

*The long tail of **routine** changes becomes the largest aggregate source of risk.*

THE OBVIOUS NEXT STEP: AUTOMATION

With such a clear blind spot, the natural response is to build an **automated system** that can flag underestimated changes before they reach production.

Rule-Based Policies

Hard-coded rules from expert committees to block known risky patterns

ML Classifiers

Gradient boosted models trained on structured RFC metadata and text embeddings

Both approaches failed — but each failure taught us something critical.

STATIC RULES: SHALLOW AND STALE

What Rules CAN Do

- Block changes during holidays or maintenance windows
- Require additional reviewers for high-risk self-assessments
- Enforce white-screen rate thresholds on services

What Rules CANNOT Do

- **Superficial:** completely blind to unstructured text where engineers describe actual intent
- **Static:** only protects against known failure modes; slow manual updates after each new incident type

Rules verify structure. They cannot understand context.

ML CLASSIFIERS: ACCURATE BUT OPAQUE

Table 1: Strawman Escalation Detectors

Model	Precision	Recall	F1-Score
Structured-Field	50.0%	52.9%	51.4%
Hybrid Semantic	59.5%	68.5%	63.8%

Key Findings

+12.4% F1 with text embeddings

But explanations point to *embedding_dimension_247*

SREs rejected it immediately

> *"If a system cannot explain why it flags a change, it becomes an uninterpretable tool that people cannot trust or learn from."*

INTERPRETABILITY IS NOT OPTIONAL

In operational settings, engineers need to **understand** each alert, **verify** its cause, and **learn** from it.

This led us to a core principle:

Interpretability must be built in from the start.

> *Risk assessments should provide clear, human-readable justifications based on real operational evidence.*

A DIFFERENT KIND OF SYSTEM

Rules lack context. ML lacks trust. What we need is a system that

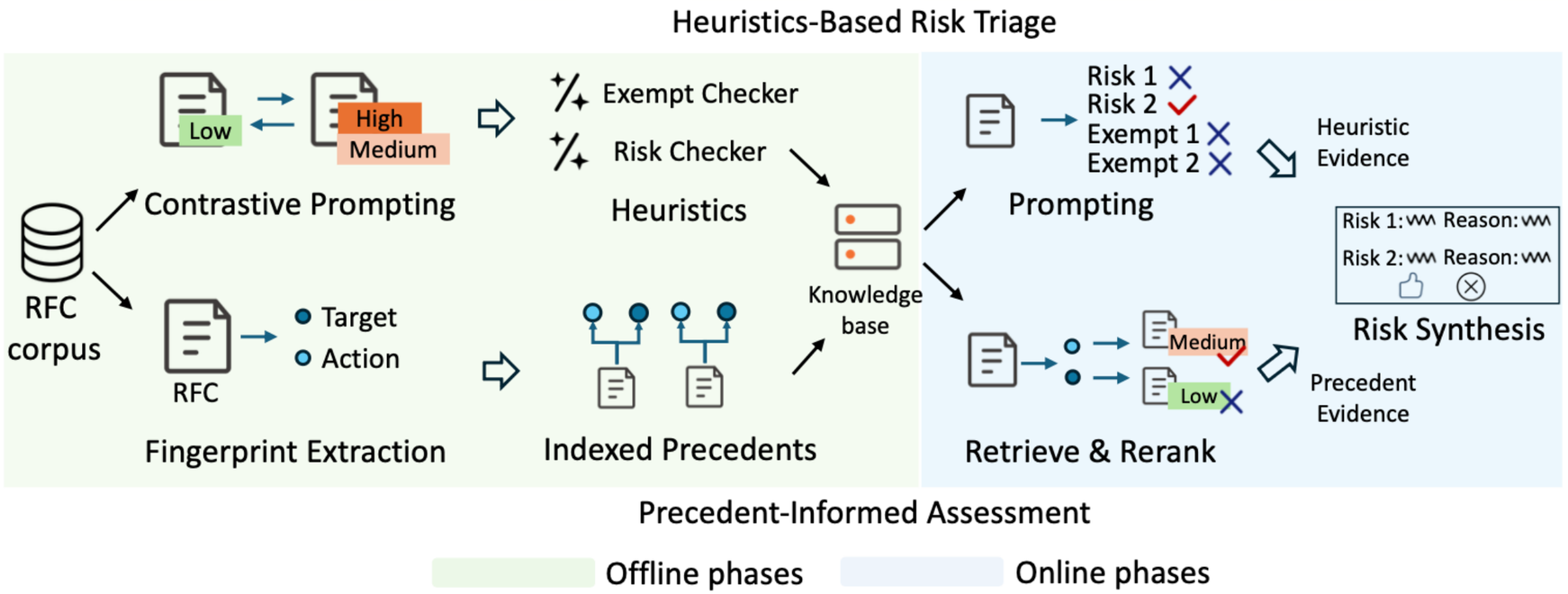
combines the reasoning power of LLMs with the credibility of historical operational evidence.

1. **Every assessment grounded in explicit evidence:** no black-box scores
2. **Learn from historical changes:** transform operational data into structured knowledge
3. **Human-readable warnings**: reviewers must understand, verify, and learn
4. **Improve continuously via feedback**: a self-correcting knowledge base

This is Aegis.

CORE PHILOSOPHY: INTERPRETABLE BY DESIGN

Every assessment grounded in explicit evidence



- **Human-readable warnings** — not binary scores or numbers

Two complementary knowledge forms:

- Generalized heuristics — recurring patterns from historical RFCs
- Specific precedent cases — analogous historical incidents

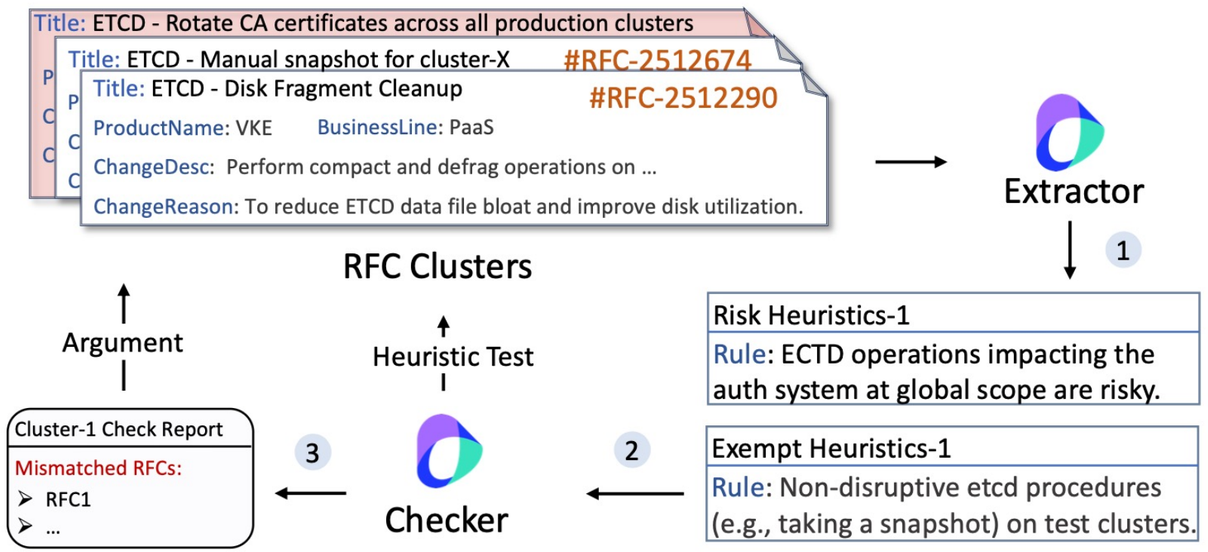
TWO PILLARS: HEURISTICS + PRECEDENTS

PILLAR 1: Heuristics

Learn recurring patterns from historical RFCs

Contrast high/medium-risk vs. low-risk changes

Risk heuristics flag failures; Exemption heuristics filter safe routines



Contrastive prompting generates heuristics; Extractor-Checker loop validates them

TWO PILLARS: HEURISTICS + PRECEDENTS

PILLAR 2: Precedents

Retrieve analogous historical incidents

Operational similarity via (Action, Target) fingerprints

Context-aware warnings grounded in concrete cases

Challenge: Standard semantic search retrieves topic-similar but operationally different changes

Solution: Reduce each RFC to a structured (Action, Target) fingerprint

The minimal schema maximizes recall; LLM reranking ensures precision

TWO PILLARS: HEURISTICS + PRECEDENTS

PILLAR 1: Heuristics

Learn recurring patterns from historical RFCs

Contrast high/medium-risk vs. low-risk changes

Risk heuristics flag failures; Exemption heuristics filter safe routines

PILLAR 2: Precedents

Retrieve analogous historical incidents

Operational similarity via (Action, Target) fingerprints

Context-aware warnings grounded in concrete cases

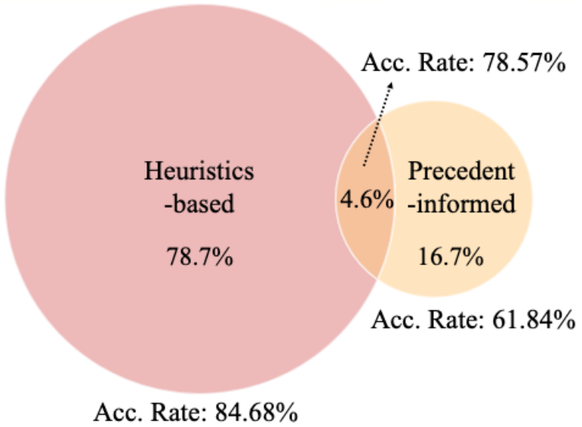
Risk Synthesis integrates both into a single, focused recommendation

DEPLOYMENT RESULTS: 75% ACCEPTANCE

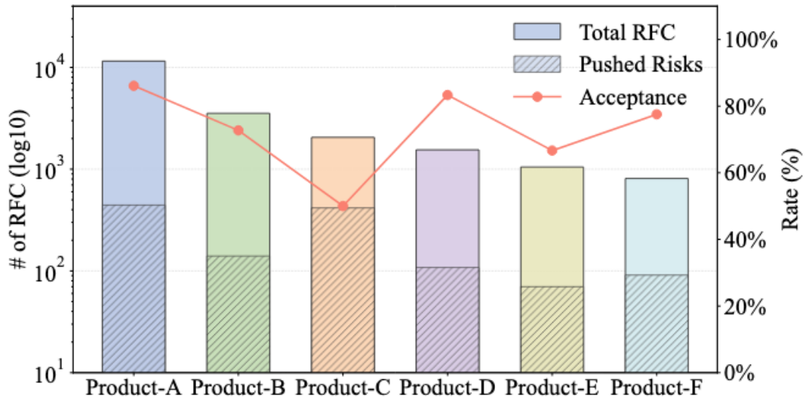
75.03%
Overall Acceptance Rate

84.68%
Heuristics Alerts Accepted

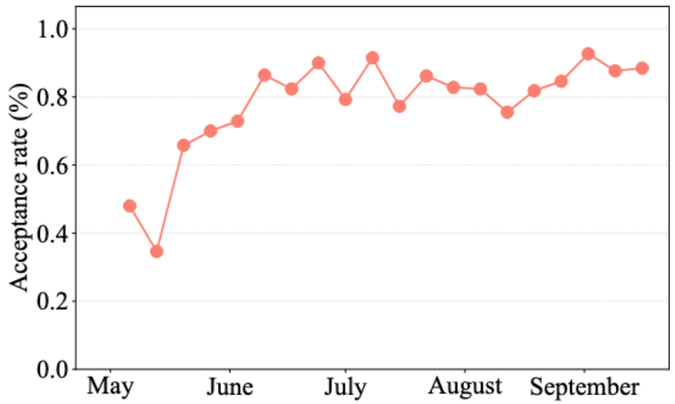
61.84%
Precedent Alerts Accepted



(a) Path-wise Escalations — Volume Share & Acceptance Breakdown



(b) Per-Product Scale — Total RFCs, Escalation (Risk) Rate & Acceptance



(c) Monthly Acceptance Rate (May–Sep)

Fig. 5: Five-month production deployment results across Volcano Engine

ESCAPED INCIDENTS & REJECTION ANALYSIS

Escape Analysis

~75% Execution-level

Runtime factors unpredictable from RFC content — resource contention, race conditions, environment drift

~25% Plan-level

Risks identifiable in hindsight from the RFC plan; each escape prompts addition of a new heuristic to the knowledge base

Rejection Analysis (~25% overall)

~65% Operational evolution

Previously risky operations automated or hardened since heuristic creation — knowledge staleness, not system error

~35% LLM misinterpretation

e.g., UI display changes confused with database schema modifications — ambiguous RFC descriptions

Both feed the feedback-driven refinement loop

WHEN IT WORKED: A NEAR-MISS

May, 2025

Stress Test on CBR Service — Flagged as Low-Risk

1. An engineer submitted a **stress test on the CBR service** as low-risk, with EBS rate-limited
2. **Aegis found a direct precedent:** a similar stress test previously escalated to high-risk
3. A **senior architect** was auto-invited; they recognized the **EBS dependency risk**
4. The change was **safely postponed** — a potential incident was prevented

> *"Aegis drives collaboration to prevent incidents. By identifying the risk and engaging the right experts, it helped transform a dangerous change into a safe operational decision."*

Outcome: Incident Prevented

FIVE LESSONS LEARNED

01 **Augment human judgment, not replace it.**

Real challenge: spotting the few risky changes mistakenly treated as low-risk. Targeted escalation shifts human attention where it matters most.

02 **Integration design determines adoption.**

Non-blocking integration, interactive feedback, and easy override were as crucial as detection accuracy. Engineers engage when systems enhance their workflow.

03 **Fragmented knowledge is both challenge and opportunity.**

Turning scattered operational signals into structured knowledge via heuristics + precedents produces context-aware assessments.

04 **Prefer liberal flagging over confidence thresholds.**

LLM confidence is poorly calibrated for operational risk. Flag liberally, refine via feedback — a self-improving system.

05 **Knowledge freshness is the hardest challenge.**

Rapidly evolving services outpace the KB. Lag between operational evolution and KB adaptation is the primary source of false positives.

WHAT REMAINS UNSOLVED

1 **Plan vs. Execution Boundary**

Aegis assesses RFC content but cannot detect runtime failures like resource contention or race conditions

2 **Proactive Knowledge Drift**

Rejection feedback lags behind fast-evolving services. Proactive pattern-shift detection remains unsolved

3 **Cold-Start for New Services**

New services or change types lack historical data for heuristic extraction and precedent retrieval

4 **LLM Reasoning Reliability**

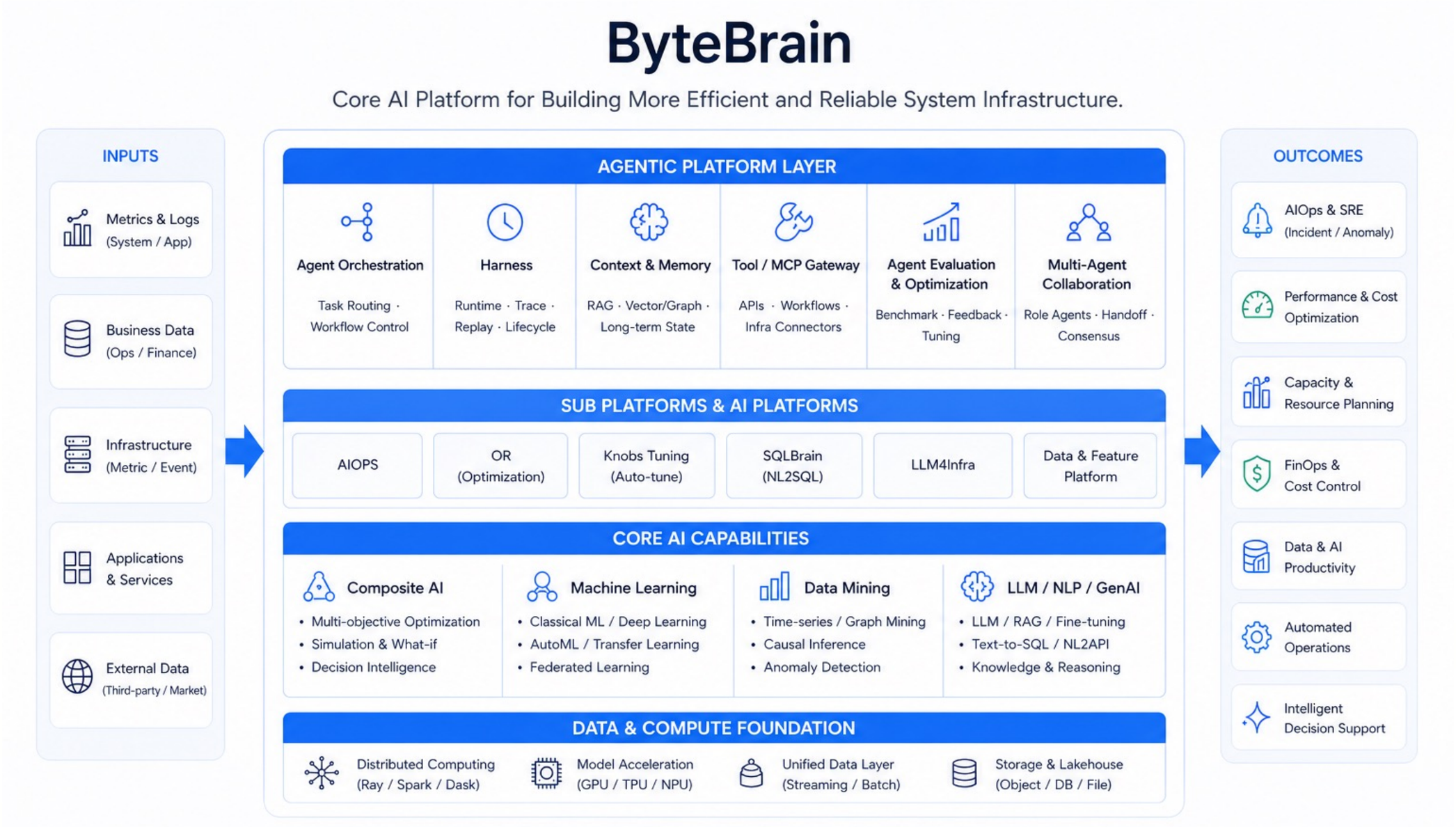
Reducing hallucinations and misinterpretations without relying on poorly calibrated confidence scores

5 **Cross-Organization Validation**

Our evaluation is within one organization. Shared benchmarks and transferable heuristics are open problems

*We believe these problems are **not unique**— and we look forward to the community's insights.*

AI for Infra Platform - *ByteBrain Team*



THANK YOU

Questions & Discussion