ChainCraft: Bridging Causal Discovery and LLM Reasoning for Failure Prediction in Microservices

Yongxin Zhao†, Yongqian Sun†*, Jingyu Wang†, Junhua Kuang†, Shenglin Zhang†
Wenwei Gu†, Boxuan Zhao‡, Li Shi‡, Wei Li‡, Liping Zhang‡, Dan Pei§
† *Nankai University*, ‡ *Alibaba Group*, § *Tsinghua University*

***Abstract*—Failure prediction is essential to the reliability of microservice systems, yet existing approaches rely heavily on labeled data and provide limited interpretability. We present *ChainCraft*, an LLM-based failure prediction framework that bridges metric-driven causal discovery and LLM reasoning. *ChainCraft* first constructs a causal graph from metrics via a causal discovery algorithm, enabling the LLM to reason about anomaly propagation under causal constraints. It then iteratively refines the propagation chain through a closed-loop optimization process to improve reasoning consistency and prediction accuracy. To further enhance prediction, *ChainCraft* incorporates a hybrid text-structure retrieval mechanism to identify relevant historical failure cases as references. Evaluated on a real-world dataset from *Alibaba*, *ChainCraft* achieves an F_1-score of 0.875, improving by 22% over the best baseline. In a three-month industrial deployment at *Alibaba*, OCEs confirm that *ChainCraft* achieves 80.8% accuracy in predicting failures while substantially improving the efficiency of the mitigation workflow compared with prior practice.**

***Index Terms*—failure prediction, microservice systems, service availability**

I. INTRODUCTION

With the widespread adoption of cloud computing and distributed systems, microservice architectures have become a fundamental building block of modern digital infrastructure. However, their highly dynamic nature and complex service dependencies make failures inevitable. Once failures occur, they can cause substantial economic losses and significantly degrade user experience [1], [2]. To ensure service availability and reliability, cloud service providers continuously monitor large volumes of runtime metrics and, upon detecting an alert, analyze the metrics collected within an observation window to identify potential failures at an early stage and enable proactive intervention before alerts escalate into service-level failures [2].

Existing failure prediction approaches primarily extract statistical or temporal features from alerts, metrics, and logs, and employ machine learning classifiers or deep sequential models for prediction [1]–[6]. Despite their effectiveness in specific scenarios, these approaches remain constrained by two fundamental limitations in real-world production environments. First, they rely heavily on labeled failure data for supervised training [1]–[6]. In practice, manually labeling candidate alerts is labor-intensive and error-prone, while the rapid evolution of microservice systems requires models to continuously adapt to new deployment environments, making repeated annotation prohibitively expensive. Second, they provide limited interpretability. Most existing methods output only a failure probability or a ranked list of suspicious components [2], [4]–[6]. Such black-box predictions fail to reveal the underlying anomaly propagation process, preventing operators from understanding why a prediction is made and consequently limiting the practical adoption of these methods in production environments.

Owing to their remarkable semantic understanding and reasoning capabilities, large language models (LLMs) have recently shown great promise in intelligent operations tasks, including failure diagnosis [7]–[9]. Their few-shot and zero-shot learning capabilities substantially reduce the dependence on large-scale labeled failure data, while their natural language generation ability provides intuitive explanations for prediction results. Nevertheless, directly applying LLMs to failure prediction in microservice systems remains challenging.

Challenge 1: Unreliable construction of anomaly propagation chains. In microservice systems, a local anomaly does not necessarily evolve into a service-level failure. Whether it eventually develops into a failure largely depends on how anomalies propagate through service dependencies and affect critical business components. Therefore, identifying anomaly propagation paths is essential for determining whether anomalies evolve into failure-escalation propagation patterns and for improving interpretability. However, LLMs exhibit limited reasoning capability over numerical time-series data, struggle to infer reliable causal relationships among metrics, and are susceptible to hallucinations, often generating propagation edges unsupported by observational evidence [10].

Challenge 2: Underutilization of expert knowledge during LLM reasoning. When investigating abnormal alerts, on-call engineers (OCEs) typically refer to historical incidents with similar symptoms to accelerate troubleshooting [7]. However, such operational knowledge is often scattered across unstructured incident tickets, making it difficult to exploit effectively. Furthermore, failures arising from different propagation patterns may exhibit highly similar surface symptoms despite requiring entirely different remediation strategies. Conventional retrieval-augmented generation (RAG) methods primarily retrieve cases based on textual semantic similarity, making them vulnerable to false matches caused by symptom similarity and limiting their ability to effectively reuse historical operational experience.

*Yongqian Sun is the corresponding author.

To address these challenges, we propose *ChainCraft*, an LLM-based failure prediction framework for microservice systems. To improve the reliability of propagation chain construction, *ChainCraft* first detects metric anomalies and applies the PCMCI algorithm [11] to infer causal relationships among anomalous metrics, producing a directed causal graph that constrains subsequent LLM reasoning. Guided by this graph, the LLM generates candidate anomaly propagation chains and iteratively refines them through a closed-loop optimization process incorporating structural, temporal, and causal-semantic evaluation together with quality comparison, thereby improving the consistency and reliability of the generated propagation chains. To better exploit expert knowledge, *ChainCraft* persistently stores historical propagation chains and symptom representations in a structured case repository with dedicated indexes. During online prediction, the framework jointly considers symptom semantic similarity and propagation graph topology through a hybrid similarity metric, enabling topology-aware case retrieval. Finally, the LLM integrates the retrieved historical experience with the current anomaly propagation chain to produce a comprehensive prediction report containing the failure prediction result, the most relevant historical cases, and recommended remediation actions, thereby supporting OCEs in proactive decision-making.

The main contributions are summarized as follows:

- We propose *ChainCraft*, which, to the best of our knowledge, is the first failure prediction framework for microservice systems that integrates metric-driven causal discovery with LLM reasoning, enabling proactive and interpretable failure prediction.
- We develop a unified prediction pipeline that detects metric anomalies, constructs and iteratively refines anomaly propagation chains, and incorporates expert knowledge through a hybrid text-structure retrieval mechanism, jointly improving prediction accuracy and interpretability.
- We deploy and evaluate *ChainCraft* in a real-world production environment at *Alibaba* across diverse failure scenarios, including database bottlenecks, resource exhaustion, and network anomalies. Experimental results demonstrate that *ChainCraft* consistently outperforms state-of-the-art baselines while providing actionable and interpretable decision support for accelerating incident response. Our code is publicly available at https://github.com/AIOps-Lab-NKU/ChainCraft.

II. BACKGROUND AND RELATED WORK

A. Problem Formulation

Motivated by prior studies [1], [12], we formulate failure prediction as a time-window classification task. Fig. 1 illustrates the problem setting.

Given an alert at time t, we define an observation window $[t-w, t]$ of length w. Alerts are triggered by fluctuations in application performance indicators, such as success rate, response time, and error QPS. Once an alert is raised, the monitoring platform collects metric data within the observation window for subsequent analysis. The objective is to predict

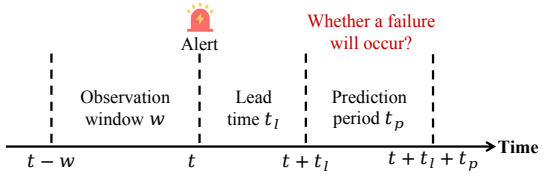

Fig. 1. Problem formulation of failure prediction.

whether the current alert will escalate into a service-level failure during the prediction window $[t+t_l, t+t_l+t_p]$, where t_l is the lead time and t_p is the prediction period. The lead time specifies the minimum advance notice required for OCEs to perform mitigation actions (*e.g.*, master–standby switchover), ensuring sufficient time to prevent service degradation. Based on discussions with OCEs, we set $w = 60$ min, $t_l = 10$ min, and $t_p = 120$ min.

B. Related Work

Failure prediction for large-scale systems has received growing attention in recent years. MING [3] integrates LSTM-based temporal modeling, random forest-based spatial learning, and learning-to-rank for node failure prediction. PreMiSE [13] identifies anomalous deviations using Granger causality and associates them with learned failure signatures. MEPFL [14] exploits system trace logs to predict latent failures in microservice applications. AirAlert [4] models alert dependencies with Bayesian networks and employs gradient boosting for outage prediction. eWarn [1] combines LDA-based feature extraction with multi-instance learning to suppress noisy alerts for incident prediction. MISP [6] integrates multi-modal operational data through parallel and cross-attention encoders for failure probability estimation. To improve model training, SOIL [5] adopts score-conditioned diffusion to synthesize failure samples for mitigating class imbalance, while UPTAKE [2] incorporates a risk estimator to address uncertain positive instances introduced by auto-mitigation.

Despite their effectiveness in modeling statistical associations and structural dependencies, these methods do not explicitly construct interpretable failure propagation chains that capture causal relationships among anomalous metrics. In contrast, our approach leverages LLMs to generate semantically coherent propagation chains and combines them with case-based reasoning to enable accurate failure prediction.

III. APPROACH

A. Overview

As illustrated in Fig. 2, *ChainCraft* comprises three major stages.

Stage 1: Data Preprocessing and Anomaly Detection (§III-B). *ChainCraft* first collects multi-source system metrics and organizes them into three architectural layers. It then employs Prophet [15] to identify anomalous metrics, after which the *Anomaly Profiler* summarizes the detected anomalies into a structured report that characterizes the dominant abnormal metrics and their statistical properties.

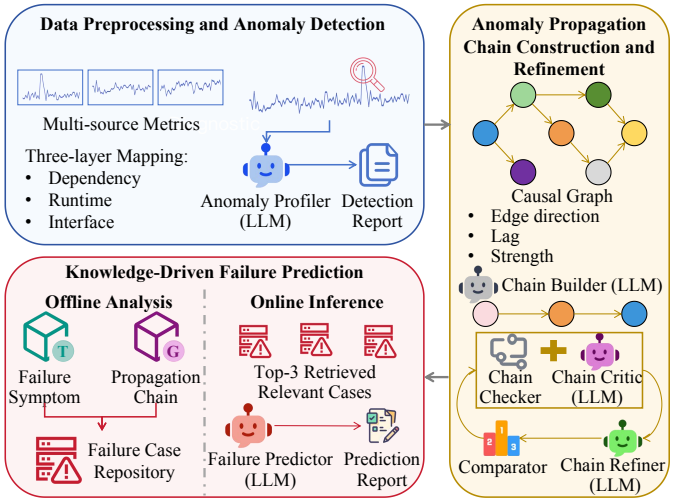

Fig. 2. The framework of *ChainCraft*.

Stage 2: Anomaly Propagation Chain Construction and Refinement (§III-C). Based on the detected anomalies, *ChainCraft* leverages the Peter and Clark Momentary Conditional Independence (PCMCI) causal discovery algorithm [11] to infer a causal graph that captures temporal dependencies among anomalous metrics. The *Chain Builder* subsequently derives candidate anomaly propagation chains from the graph. To improve their causal consistency and completeness, the chains are iteratively refined through a closed-loop process involving rule-based Checkers and the *Chain Critic* auditing, *Chain Refiner* revisions, and deterministic comparison.

Stage 3: Knowledge-Driven Failure Prediction (§III-D). Validated propagation chains, together with their associated symptoms, are incrementally accumulated into a historical failure case repository. During inference, *ChainCraft* retrieves the most relevant historical cases and provides them as contextual knowledge to the *Failure Predictor*, which determines whether the current anomaly is likely to evolve into a failure.

B. Data Preprocessing and Anomaly Detection

The data collection module gathers runtime metrics from multiple sources. Each metric is associated with the system boundary where it is observed, resulting in three architectural layers including the dependency layer for downstream services, the runtime layer for internal execution states, and the interface layer for externally observable service behavior. This organization enables both propagation chain construction and case retrieval to leverage architectural topology as a structural constraint in addition to statistical correlations.

Production monitoring data often contain missing values and invalid measurements introduced by monitoring inconsistencies rather than genuine performance degradation, which can generate spurious anomalies and mislead downstream analysis. We therefore filter invalid values using predefined feasibility constraints and apply linear interpolation only to short isolated gaps to preserve temporal continuity without obscuring real anomalies. Prophet [15] is then applied to each metric, learning normal behavior from three days of historical data while examining only the observation window preceding the alert. Only anomalies detected within this window are retained, and the first anomaly onset time of each metric is recorded for subsequent chain construction. A dual-confidence-interval strategy is adopted, where a wide interval identifies anomalous observations and a narrow interval determines the corresponding onset time.

The detected anomalies are summarized by the *Anomaly Profiler*, which generates a structured report containing the 20 most frequently anomalous metrics. For each metric, the report records the first anomaly onset time, anomaly trend, severity rating, and statistical characteristics within the observation window. The severity rating combines a severity quantile that measures the historical rarity of the observed deviation with a metric sensitivity table derived from OCE knowledge under a conservative strategy. The resulting report integrates statistical evidence with operational semantics to support subsequent propagation chain construction.

C. Anomaly Propagation Chain Construction and Refinement

Existing approaches typically rely on the PC algorithm [16] or Granger causality tests [17] to infer causal relationships among metrics. However, both are susceptible to autocorrelation in high-dimensional time series and often exhibit high false positive rates. *ChainCraft* therefore adopts PCMCI [11] to learn causal relationships among anomalous metrics and construct a causal graph $G = (V, E)$, where each edge $(u, v) \in E$ is associated with a lag estimate τ_{uv} and a causal strength ρ_{uv}. For bidirectional edges, only the direction with the larger absolute causal strength is retained, producing a directed acyclic graph without causal ambiguity.

Based on the anomaly report and causal graph, the *Chain Builder* derives candidate propagation chains. Each chain is represented as a directed path whose nodes correspond to anomalous metrics and whose edges contain relationship descriptors and supporting evidence. The *Chain Builder* further assigns each chain a confidence score and a natural language summary, generating up to three independent failure hypotheses. Chain generation is guided by causal priority computed from the net influence score (out-degree minus in-degree in G), first anomaly onset time, and layer-aware patterns that favor cross-layer propagation.

The generated chains are iteratively refined through deterministic validation and LLM-based reasoning. The Structural Checker verifies graph consistency, including edge completeness, acyclicity, root-node integrity, endpoint coverage, and node uniqueness. The Temporal Checker validates timestamp monotonicity, consistency between observed delays and PCMCI lag estimates τ_{uv}, and that the root node precedes symptom onset. The *Chain Critic* evaluates semantic plausibility beyond rule-based constraints and categorizes detected problems into hard violations and soft issues, each mapped to a structured edit action such as `insert_step` or `replace_edge`.

The *Chain Refiner* performs minimal local edits only at flagged positions. A deterministic comparator then computes

a weighted quality score from structural, temporal, and semantic penalties (with weights 0.4, 0.3, and 0.3, respectively), accepting a revision only when its quality improves. The refinement process terminates once all assertions are satisfied or the iteration budget $T_{\max} = 3$ is reached, and the highest-quality chain is retained.

D. Knowledge-Driven Failure Prediction

Validated propagation chains and their symptom signatures are stored in a structured failure case repository. Each failure case is organized into three chunk types: symptom chunks aggregating anomaly features at the architectural-layer level; chain chunks storing the serialized propagation path with a natural-language description; and remediation chunks containing the resolution actions. Symptom and chain descriptions are encoded into dense vectors and indexed in a vector database.

For a new alert, *ChainCraft* encodes its symptom and chain descriptions into query vectors and retrieves a candidate set of similar historical failure cases using cosine similarity. Each candidate is then re-ranked using a composite similarity function that integrates symptom-level matching with graph-structural chain similarity. The chain similarity is defined as:

$$\begin{aligned}\text{Sim}_{\text{chain}} = \lambda_1 \cdot \text{Sim}_{\text{desc}} + \lambda_2 \cdot \text{Sim}_{\text{node}} + \lambda_3 \cdot \text{Sim}_{\text{edge}} \\ + \lambda_4 \cdot \text{Sim}_{\text{path}} + \lambda_5 \cdot \text{Sim}_{\text{topo}}\end{aligned} \tag{1}$$

where Sim_{desc} is the similarity of chain descriptions, Sim_{node} denotes bidirectional best-match node similarity over shared layer and node type (*i.e.*, root, symptom, or intermediate node) labels, Sim_{edge} is the Jaccard similarity over edge sets, Sim_{path} is the longest common subsequence of layer-type sequences, and Sim_{topo} compares structural features (*i.e.*, node count, edge count, layer coverage, and node-type diversity) of propagation graphs, with corresponding weights of 0.2, 0.25, 0.25, 0.2, and 0.1. The final retrieval score is computed by combining symptom similarity and chain similarity for ranking: $\text{Sim}_{\text{overall}} = w \cdot \text{Sim}_{\text{sym}} + (1 - w) \cdot \text{Sim}_{\text{chain}}$, where the default weight is $w = 0.6$. Sim_{sym} measures the semantic similarity between the symptom descriptions of the query alert and historical cases.

Based on the final reranking scores, the top-3 most relevant historical cases are provided to the *Failure Predictor*, which generates a structured risk assessment including a binary prediction distinguishing potential failures from transient anomalies, a confidence score, an evidence-based reasoning statement grounded in the retrieved cases, and actionable mitigation recommendations. When the similarity between the current alert and the most relevant historical case falls below a predefined sufficiency threshold, the alert is escalated to experienced OCEs for manual validation, and newly confirmed failure cases may be incorporated into the historical repository to support future predictions.

IV. EVALUATION

We evaluate *ChainCraft* using an industrial dataset collected from *Alibaba* to answer the following research questions.

RQ1: How does *ChainCraft* perform compared with baseline methods?
RQ2: How much does each component contribute to the overall performance of *ChainCraft*?
RQ3: How accurate are the generated anomaly propagation chains?

A. Experimental Setup

Dataset. Following the guidance of experienced OCEs, we construct a historical failure case repository using production incidents that satisfy three inclusion criteria: (1) they represent frequently occurring and operationally representative failure scenarios, (2) they contain complete investigation records with documented remediation procedures and fine-grained monitoring metrics, and (3) they are identified by experienced OCEs as the primary references for manual failure prediction, collectively covering the major recurring failure scenarios encountered in daily operations. A total of 13 historical failure cases satisfy these criteria and are included in the repository. To avoid data leakage, the historical repository is disjoint from the evaluation dataset. For evaluation, we collect 100 production alerts spanning e-commerce, search, and ticketing services. Experienced OCEs manually label each alert, confirming that 50 correspond to failures and the remaining 50 are non-failure alerts. All metrics in the dataset are collected at a 1-minute sampling interval.
Baselines. Since labeled production data are scarce, we exclude approaches [1], [2], [4] that rely heavily on supervised training. Existing LLM-based methods are typically scenario-specific, designed for different tasks, and are neither open source nor directly applicable to microservice failure prediction. We therefore compare *ChainCraft* with two representative open-source few-shot reasoning frameworks. CoT [18] decomposes prediction into intermediate reasoning steps, while ReAct [19] alternates reasoning and action to iteratively refine predictions. Similar to COCA [8], both use historical failure cases as external knowledge.
Evaluation Metrics. Following prior work [1], [2], [20], we evaluate precision, recall, and F_1-score. Precision ($\frac{TP}{TP+FP}$) measures the proportion of predicted failures that are correct, recall ($\frac{TP}{TP+FN}$) measures the proportion of actual failures detected in advance, and the F_1-score provides a harmonic mean that balances precision and recall.
Implementation. *ChainCraft* is implemented in Python 3.12 with PyTorch. Experiments are conducted on a Linux server with 32 CPU cores and 64 GB RAM. The *Anomaly Profiler*, *Chain Builder*, *Chain Critic*, *Chain Refiner*, and *Failure Predictor* are accessed through APIs, avoiding complex local deployment while maintaining system performance. We run PCMCI [11] with a maximum time lag of $\tau_{\max} = 3$, a significance level of $\alpha = 0.05$, and partial correlation as the conditional independence test. All hyperparameter weights are empirically set based on the operational experience of OCEs.

B. Overall Performance (RQ1)

Table I shows that *ChainCraft* consistently outperforms CoT [18] and ReAct [19] across both seed LLMs. With

TABLE I
OVERALL PERFORMANCE COMPARISON

Seed LLM	Method	Precision	Recall	F_1-score
DeepSeek-V4	*ChainCraft*	**0.742**	**0.98**	**0.845**
	CoT [18]	0.755	0.74	0.748
	ReAct [19]	0.702	0.8	0.748
	C1	0.632	0.86	0.729
	C2	0.652	0.86	0.741
	C3	0.643	0.9	0.75
GPT-5.4	*ChainCraft*	**0.913**	**0.84**	**0.875**
	CoT [18]	0.756	0.68	0.716
	ReAct [19]	0.729	0.7	0.714
	C1	0.914	0.64	0.753
	C2	0.857	0.72	0.783
	C3	0.841	0.74	0.787

DeepSeek-V4 [21], *ChainCraft* achieves the highest F_1-score of 0.845, representing a 13.0% improvement over both CoT [18] and ReAct [19], while increasing recall to 0.98. Since missing an impending failure is considerably more costly than investigating a false alarm in production environments, recall is the primary metric for evaluating failure prediction systems. The substantially higher recall demonstrates that *ChainCraft* can identify more potential failures without relying solely on generic LLM reasoning. With GPT-5.4 [22], *ChainCraft* further achieves the highest F_1-score of 0.875, improving upon CoT [18] and ReAct [19] by 22.2% and 22.5%, respectively, while maintaining the highest precision of 0.913. These results indicate that combining causal propagation reasoning with historical failure knowledge enables more reliable prediction than prompt-based reasoning alone.

C. Contribution of Key Components (RQ2)

To evaluate the contribution of each component, we perform an ablation study using three variants while preserving the overall functionality of *ChainCraft*. Specifically, **C1** removes PCMCI [11] and allows the LLM to generate propagation chains directly; **C2** removes the refinement module and uses the initial chains without iterative revision; and **C3** removes the structure-aware RAG and relies solely on textual symptoms for case retrieval.

Table I reports the ablation results. Removing PCMCI results in the largest performance degradation, with a substantial drop in F_1-score for both seed LLMs. The corresponding recall also drops substantially, indicating that explicit causal modeling is critical for identifying potential failures rather than overlooking them. Removing the refinement module decreases the F_1-score, suggesting that iterative refinement improves the consistency and completeness of the generated propagation chains. Removing the structure-aware RAG also degrades F_1-score, indicating that incorporating structural information into case retrieval improves the reuse of relevant historical failure knowledge beyond symptom similarity alone. Overall, each component contributes to prediction performance, with PCMCI providing the largest benefit.

D. Anomaly Propagation Chain Accuracy (RQ3)

Beyond predicting failures, *ChainCraft* is expected to generate propagation chains that explain whether anomalies evolve into failure-escalation propagation patterns. Following [23], we conduct a human evaluation on the 49 correctly predicted failure cases produced by *ChainCraft* under DeepSeek-V4 [21]. Two OCEs with at least one year of experience in microservice operations independently assess whether each propagation chain accurately captures anomaly evolution, with disagreements resolved by a senior engineer. Among the 49 cases, 40 propagation chains are judged to be correct, yielding an accuracy of 81.6%. These results indicate that *ChainCraft* can generate reliable propagation chains that facilitate OCEs in understanding failure evolution and identifying effective intervention points.

V. DISCUSSION

A. Deployment

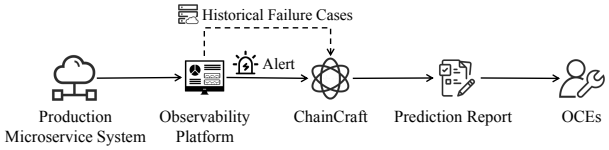

Fig. 3. Deployment architecture of *ChainCraft*.

To evaluate the industrial feasibility of *ChainCraft*, we deploy it at *Alibaba* for three months. As shown in Fig. 3, the system operates in offline and online stages. Offline, labeled historical failures are processed to construct a structured failure case repository with propagation chains and symptom signatures. Online, upon alert triggering, *ChainCraft* generates causal propagation chains from detected anomalies and retrieves structurally similar cases to assess whether an alert may escalate into a service-level failure.

During deployment, OCEs confirm that *ChainCraft* successfully predicts 21 out of 26 failures. The average LLM API cost per case is approximately $0.06, which is negligible in production. With *ChainCraft*, preventive failure mitigation takes about 5 minutes, including approximately 1.5 minutes for *ChainCraft* inference and report generation and the remaining time for OCEs' review and mitigation actions. In contrast, prior workflows require 3–5 senior OCEs and exceed 30 minutes per case. The interpretable propagation chains further help engineers identify critical services, reduce false positives, and accelerate failure mitigation.

B. Case Study

To demonstrate the effectiveness of *ChainCraft* in production, we present a representative case in Fig. 4. When a decline in service success rate triggered an alert, *ChainCraft* detected anomalies whose symptom signatures were similar to historical cases 1 and 2. By further comparing propagation chains, *ChainCraft* identified case 1 as the closest match because its propagation pattern was more consistent with the current event. Based on the retrieved symptoms, propagation chain, and expert knowledge from case 1, the *Failure Predictor* identified a potential failure and recommended that OCEs inspect the database for slow queries or connection pool

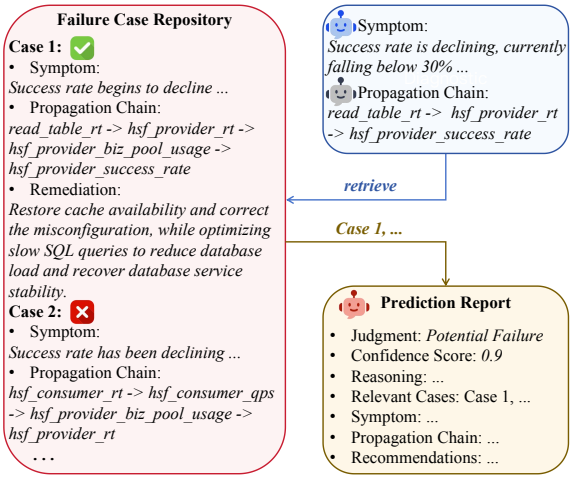

Fig. 4. A case of database slow query failure predicted by *ChainCraft*.

exhaustion while following the historical remediation procedure. These recommendations enabled timely intervention and prevented service disruption.

VI. CONCLUSION

In this paper, we present *ChainCraft*, an LLM-based failure prediction framework that bridges metric-driven causal discovery and LLM reasoning for microservice systems. By combining causal graph construction, closed-loop propagation refinement, and hybrid retrieval of historical failure cases, *ChainCraft* delivers accurate and interpretable failure prediction with reduced reliance on labeled data. Extensive experiments on a real-world industrial dataset demonstrate that *ChainCraft* consistently outperforms existing approaches. Furthermore, its successful deployment in *Alibaba* highlights its practical effectiveness in improving engineers' efficiency for proactive failure prevention, suggesting the promise of causal-guided LLM reasoning for intelligent software operations.

REFERENCES

[1] N. Zhao, J. Chen, Z. Wang, X. Peng, G. Wang, Y. Wu, F. Zhou, Z. Feng, X. Nie, W. Zhang, K. Sui, and D. Pei, "Real-time incident prediction for online service systems," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020, p. 315–326.

[2] H. Li, M. Ma, Y. Liu, P. Zhao, S. Li, Z. Li, M. Chintalapati, Y. Dang, C. Bansal, S. Rajmohan, Q. Lin, and D. Zhang, "Can we trust auto-mitigation? improving cloud failure prediction with uncertain positive learning," in *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, 2024, pp. 499–510.

[3] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J.-G. Lou, C. Li, Y. Wu, R. Yao, M. Chintalapati, and D. Zhang, "Predicting node failure in cloud service systems," in *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018, p. 480–490.

[4] Y. Chen, X. Yang, Q. Lin, H. Zhang, F. Gao, Z. Xu, Y. Dang, D. Zhang, H. Dong, Y. Xu, H. Li, and Y. Kang, "Outage prediction and diagnosis for cloud service systems," in *The World Wide Web Conference*, ser. WWW '19, p. 2659–2665.

[5] C. Duan, F. Yang, P. Zhao, L. Zheng, Y. Dagli, Y. Liu, Q. Lin, and D. Zhang, "Soil: Score conditioned diffusion model for imbalanced cloud failure prediction," in *Companion Proceedings of the ACM Web Conference 2024*, ser. WWW '24, 2024, p. 65–72.

[6] X. Lu, Y. Wang, Y. Fu, Q. Sun, X. Ma, X. Zheng, and C. Zhuo, "Misp: A multimodal-based intelligent server failure prediction model for cloud computing systems," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '24, 2024, p. 5509–5520.

[7] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin, S. Rajmohan, D. Zhang, and T. Xu, "Automatic root cause analysis via large language models for cloud incidents," in *Proceedings of the Nineteenth European Conference on Computer Systems*, ser. EuroSys '24, 2024, p. 674–688.

[8] Y. Li, Y. Wu, J. Liu, Z. Jiang, Z. Chen, G. Yu, and M. Lyu, " COCA: Generative Root Cause Analysis for Distributed Systems with Code Knowledge ," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, ser. ICSE '25, May 2025, pp. 770–770.

[9] C. Pei, Z. Wang, F. Liu, Z. Li, Y. Liu, X. He, R. Kang, T. Zhang, J. Chen, J. Li, G. Xie, and D. Pei, "Flow-of-action: Sop enhanced llm-based multi-agent system for root cause analysis," in *In Companion Proceedings ofthe ACMWeb Conference 2025*, ser. WWW '25, p. 422–431.

[10] Z. Xie, Z. Li, X. He, L. Xu, X. Wen, T. Zhang, J. Chen, R. Shi, and D. Pei, "Chatts: Aligning time series with llms via synthetic data for enhanced understanding and reasoning," *Proc. VLDB Endow.*, vol. 18, no. 8, p. 2385–2398, Apr. 2025.

[11] J. Runge, P. Nowack, M. Kretschmer, S. Flaxman, and D. Sejdinovic, "Detecting and quantifying causal associations in large nonlinear time series datasets," *Science advances*, vol. 5, no. 11, p. eaau4996, 2019.

[12] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Comput. Surv.*, vol. 42, no. 3, Mar. 2010.

[13] L. Mariani, M. Pezzè, O. Riganelli, and R. Xin, "Predicting failures in multi-tier distributed systems," *Journal of Systems and Software*, vol. 161, p. 110464, 2020.

[14] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019, 2019, p. 683–694.

[15] Facebook, "Prophet: Automatic forecasting procedure," 2026. [Online]. Available: https://github.com/facebook/prophet

[16] P. Spirtes and C. Glymour, "An algorithm for fast recovery of sparse causal graphs," *Social science computer review*, vol. 9, no. 1, pp. 62–72, 1991.

[17] C. W. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica: journal of the Econometric Society*, vol. 37, no. 3, pp. 424–438, 1969. [Online]. Available: http://www.jstor.org/stable/1912791

[18] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, 2022, pp. 24 824–24 837.

[19] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[20] Y. Liu, M. Ma, P. Zhao, T. Li, B. Qiao, S. Li, Z. Li, M. Chintalapati, Y. Dang, C. Bansal, S. Rajmohan, Q. Lin, and D. Zhang, "Early bird: Ensuring reliability of cloud systems through early failure prediction," in *2024 IEEE 35th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2024, pp. 49–54.

[21] DeepSeek-AI, A. Xu, B. Lin, B. Xue, B. Wang, B. Xu, B. Wu, B. Zhang, C. Lin, C. Dong *et al.*, "Deepseek-v4: Towards highly efficient million-token context intelligence," 2026. [Online]. Available: https://arxiv.org/abs/2606.19348

[22] A. Singh, A. Fry, A. Perelman, A. Tart, A. Ganesh, A. El-Kishky, A. McLaughlin, A. Low, A. Ostrow, A. Ananthram *et al.*, "Openai gpt-5 system card," 2026. [Online]. Available: https://arxiv.org/abs/2601.03267

[23] S. Zhang, Y. Zhao, X. Xiong, Y. Sun, X. Nie, J. Zhang, F. Wang, X. Zheng, Y. Zhang, and D. Pei, "Illuminating the gray zone: Non-intrusive gray failure localization in server operating systems," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, 2024, p. 126–137.