TopoEvo: A Topology-Aware Self-Evolving Multi-Agent Framework for Root Cause Analysis in Microservices

***Abstract*—Root cause analysis (RCA) in microservices is challenging due to (i) noisy and heterogeneous multimodal observability (metrics, logs, traces), (ii) cascading failure propagation that amplifies downstream symptoms, and (iii) non-stationary topology drift induced by autoscaling and rolling updates. Recent LLM-based RCA agents can generate tool-grounded explanations, yet they often remain topology-agnostic and suffer from *symptom-amplification bias*, misattributing the root cause to salient downstream victims. We propose TopoEvo, a topology-aware self-evolving multi-agent framework that couples graph representation learning with structured, topology-constrained reasoning. TopoEvo first introduces *Metric-orthogonal Multimodal Alignment* (MOMA), which decomposes metric embeddings into complementary subspaces and contrastively aligns logs and traces to reduce modality redundancy and sparsity, yielding stable node representations for graph encoding. It then applies *Vector Quantization* (VQ) to discretize topology-enhanced states into auditable *symptom tokens* with a symptom lexicon, enabling reliable retrieval and token-level evidence grounding. On top of these discrete topology cues, TopoEvo performs a multi-agent *Hypothesis–Evidence–Test* (HET) workflow to explicitly verify propagation-consistent explanations and separate initiating anomalies from amplified downstream symptoms. Finally, a *Self-Evolving Mechanism* refreshes hierarchical incident memory and performs conservative test-time adaptation with high-confidence pseudo-labels to maintain robustness under drift.**

We evaluate TopoEvo on both a public AIOps benchmark and a real-world production incident dataset. Compared to the state-of-the-art baselines, TopoEvo achieves absolute improvements of up to 3.44% in root cause localization accuracy, while remarkably boosting fault-type classification performance by 4.39% to 16.81% across diverse datasets.

***Index Terms*—Root Cause Analysis, Microservices, Multi-Agent Systems, AIOps, LLM Agent**

I. Introduction

Microservice architectures have become a dominant paradigm for building large-scale cloud applications due to their flexibility, support for independent deployment, and rapid iteration. [1]–[3] However, this architectural shift also makes modern systems increasingly complex to operate: failures rarely stay local, and subtle issues can quickly propagate along service dependencies, producing amplified and misleading symptoms downstream [4], [5]. As a result, *root cause analysis* (RCA)—identifying the initiating faulty entity and its fault type—has become a critical capability for maintaining service reliability and meeting strict QoS/SLA requirements.

Despite substantial progress, existing RCA solutions still face three practical limitations in real microservice deployments. **First**, microservices generate *multimodal observability* (metrics, logs, traces), yet many approaches underutilize this richness or rely on simplistic fusion [6]–[8]. In particular, heterogeneous modalities differ in frequency, sparsity, noise patterns, and missingness, making *effective cross-modal alignment* non-trivial; naive concatenation often yields unstable representations and spurious correlations. **Second**, with the rise of LLMs, multi-agent diagnosis has emerged as a promising direction for RCA, enabling tool-grounded inspection and human-readable explanations. However, most agent-based pipelines remain *topology-agnostic*: they struggle to internalize microservice dependency constraints and therefore tend to over-trust the most salient downstream symptoms (symptom amplification), leading to inefficient investigation and frequent misattribution [9]–[12]. **Third**, microservice environments are inherently non-stationary: autoscaling, rolling updates, and evolving call graphs introduce distribution shifts and out-of-distribution (OOD) behaviors. Static RCA models and fixed reasoning heuristics often degrade sharply under such drift, lacking a reliable mechanism to refresh knowledge and adapt [13], [14].

To address these challenges, we propose **TopoEvo**, a topology-aware, reasoning-enhanced, and self-evolving framework for joint *microservice root cause localization* and *fault type classification*. TopoEvo systematically connects representation learning with topology-constrained reasoning: (1) a *metric-anchored orthogonal multimodal alignment* module constructs stable shared subspaces for heterogeneous signals, mitigating modality sparsity and redundancy; (2) a *topology-aware enhancement* module discretizes topology-aware states via vector quantization (VQ) and builds a *symptom vocabulary*, turning opaque vectors into compact, retrievable, and auditable evidence units for reasoning; (3) a *reasoning-enhanced multi-agent* workflow follows an explicit hypothesis–evidence–test loop under topology constraints, reducing symptom-amplification errors and improving diagnostic efficiency; and (4) a *self-evolving mechanism* continuously refreshes incident knowledge and cautiously adapts the encoder under drift, improving robustness to OOD configurations.

Our main contributions are summarized as follows:

- **Metric-anchored orthogonal multimodal alignment.** We propose a metric-centric alignment scheme that decomposes metric embeddings into **orthogonal** subspaces and aligns logs and traces to complementary components via contrastive learning, mitigating modality sparsity and

reducing redundant correlations in multimodal observability.

- **Topology-aware symptom tokenization for agent reasoning.** We introduce a VQ-based discretization of topology-aware graph states and construct a *symptom vocabulary* that maps discrete codes to compact, auditable symptom tokens, enabling topology-constrained reasoning and reducing symptom-amplification misattribution in LLM-based RCA.
- **Reasoning-enhanced multi-agent RCA.** We propose a hypothesis–evidence–test (HET) multi-agent workflow that explicitly models fault propagation paths and performs tool-grounded evidence verification, yielding more reliable and explainable RCA under noisy multimodal telemetry.
- **Self-evolving adaptation under drift.** We introduce a self-evolving mechanism that refreshes incident knowledge and continuously adapts the graph encoder and alignment objectives under distribution shift, improving robustness to dynamic microservice configurations and out-of-distribution (OOD) incidents.

II. Preliminaries and Motivation

A. Observability of Microservice

Microservice observability refers to the capability of inferring internal system states and diagnosing runtime issues from external signals emitted by distributed services. In practice, observability data are commonly summarized as three pillars: *metrics*, *logs*, and *traces*.

Metrics are time-stamped numerical measurements (e.g., QPS, latency percentiles, error rates, CPU/memory usage) that provide compact and continuous views of service health and resource consumption.

Logs are discrete event records produced by services and infrastructure components, typically containing semi-structured messages, levels, and contextual fields; they capture rich semantic clues about failures but are often noisy and heterogeneous.

Traces record end-to-end request executions across services, usually organized as a trace graph of *spans* with parent–child and causal relationships; they expose cross-service propagation paths and fine-grained latency breakdowns. Given an incident time window, we denote the multimodal observability of node v as $\mathcal{O}_v = \{\mathbf{x}_v^{metric}, \mathbf{x}_v^{log}, \mathbf{x}_v^{trace}\}$, where $\mathbf{x}_v^{metric}$ is a metric time-series segment, $\mathbf{x}_v^{(log}$ is a log snippet/set, and $\mathbf{x}_v^{trace}$ is the trace-derived feature set [15], [16].

B. Vector Quantization & Codebook.

Vector Quantization (VQ) maps continuous embeddings into a finite set of discrete prototypes, enabling compact and symbolic representations [18]. Given a feature vector $\mathbf{z} \in \mathbb{R}^d$, VQ assigns it to the nearest entry in a learnable codebook $\mathcal{C} = \{\mathbf{c}_k\}_{k=1}^K$ by

$$k^* = \arg\min_{k \in [K]} \|\mathbf{z} - \mathbf{c}_k\|_2, \qquad \mathrm{VQ}(\mathbf{z}) = \mathbf{c}_{k^*}. \tag{1}$$

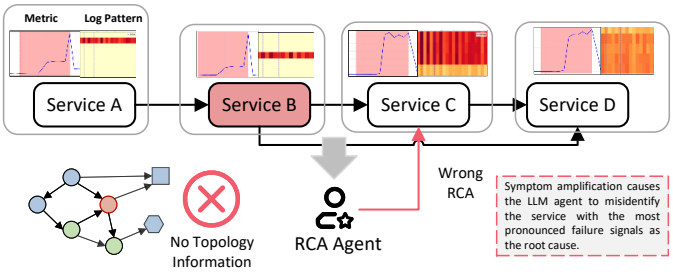

Fig. 1. Illustration of symptom-amplification bias.

The codebook can be viewed as a dictionary of representative patterns that summarize frequently occurring structures in the embedding space. During training, VQ modules are typically optimized with a reconstruction-style objective (e.g., VQ-VAE), which pulls codebook entries toward the assigned vectors while encouraging encoder outputs to commit to selected prototypes. A common formulation is

$$\mathcal{L}_{\mathrm{VQ}} = \underbrace{\|\mathrm{sg}[\mathbf{z}] - \mathbf{c}_{k^*}\|_2^2}_{\text{codebook loss}} + \beta \underbrace{\|\mathbf{z} - \mathrm{sg}[\mathbf{c}_{k^*}]\|_2^2}_{\text{commitment loss}}, \tag{2}$$

where $\mathrm{sg}[\cdot]$ denotes the stop-gradient operator and β controls the strength of the commitment term. By discretizing dense vectors into code indices, VQ yields interpretable "tokens" and can reduce sensitivity to noise, which is useful when exposing learned latent patterns to downstream reasoning modules.

C. Motivation

Microservice incidents are rarely isolated. A fault triggered at an upstream component often propagates along the dependency graph, gradually amplifying observable symptoms (e.g., retries, queueing, and timeouts) at downstream services. This propagation nature makes RCA fundamentally a *topology-conditioned* inference problem: the most anomalous node is not necessarily the initiating cause, and correct diagnosis requires reasoning over multi-level entities (node–service–pod) and their interactions.

1) Motivation 1: Topology-unaware LLM-based RCA suffers from symptom-amplification bias. Recent LLM-based RCA systems have shown promising capabilities in tool use and explanation generation, yet most of them remain largely *topology-agnostic*. They typically ingest multimodal observations and produce conclusions by pattern matching or narrative reasoning, without enforcing structural constraints from the microservice dependency graph. This omission leads to a systematic failure mode: *symptom-amplification bias*(shown in Fig. 1). When latency and error signals accumulate along a call chain, downstream services may exhibit the strongest symptoms, so an unconstrained reasoning process tends to select the most salient downstream node as the root cause, even though it is merely a victim of upstream propagation.

This challenge is exacerbated by the *multi-level* nature of microservice systems. A root cause may originate from a pod-level resource contention, a service-level misconfiguration, or

a node-level failure, while the observable "peak anomaly" may appear at a different level. Therefore, simply providing an LLM with raw logs/traces/metrics is insufficient: the model needs an explicit mechanism to *perceive* and *operate on* hierarchical topology evidence. This motivates TopoEvo to couple a GAT-based localizer with topology-aware vector quantization (VQ) and a symptom lexicon, transforming dense topology-aware states into discrete, retrievable symptom tokens. These tokens serve as compact evidence units that allow the reasoning layer to stay anchored to propagation structure, explicitly separating *initiating anomalies* from *amplified downstream symptoms*.

2) Motivation 2: Dynamic topology requires reasoning-enhanced adaptation under OOD drift. Microservice environments are highly non-stationary. Autoscaling continuously adds/removes pods, rolling updates change service versions, and configuration changes alter call patterns and dependency edges. Such dynamics induce distribution shift in both graph topology and observability, causing static RCA models to degrade—even when the fault semantics remain similar. In practice, the same fault type may manifest differently after a topology change, and previously reliable patterns can become out-of-distribution (OOD).

This motivates an RCA design that is robust to topology drift. On one hand, LLM-based reasoning provides a natural advantage: it can test hypotheses, reconcile incomplete evidence, and generalize beyond exact pattern matches. On the other hand, reasoning alone is not enough for sustained performance: the system must *retain* validated diagnostic knowledge and *adapt* to new topologies quickly. Therefore, TopoEvo introduces a reasoning-enhanced multi-agent workflow (hypothesis–evidence–test) to explicitly verify causal explanations under topology constraints, and a self-evolving mechanism that (1) refreshes hierarchical incident memory and (2) performs conservative test-time adaptation using high-confidence pseudo-labels. Together, these mechanisms enable TopoEvo to leverage strong OOD reasoning while continuously aligning its encoder and knowledge base with evolving service dependencies.

III. Methodology

As shown in Fig.2, TopoEvo consists of 5 components that support joint microservice root cause localization and fault type classification: (1) data process and dependency graph construction (2) metric-orthogonal multimodal alignment on a fine-grained dependency graph, (3) topology-aware enhancement via vector quantization and symptom vocabulary, (4) reasoning-enhanced multi-agent diagnosis via hypothesis–evidence–test, and (5) a self-evolving mechanism using hierarchical memory and test-time adaptation.

A. Data Process and Dependency Graph Construction

1) multimodal data preprocessing: For each entity $v \in V$, metrics, logs, and traces are encoded into modality embeddings.

- **Metric** signals are represented as a normalized multivariate time series $\mathbf{M} \in \mathbb{R}^{L\times D}$ and segmented into overlapping temporal patches (width w, stride s) to capture local dynamics. Each patch is encoded by a TCN and projected by an MLP to obtain the metric embedding $\mathbf{x}_v^{\text{metric}} \in \mathbb{R}^{E_m}$.
- **Traces** are parsed into span relations and aggregated into window-level statistics for each entity (e.g., latency, error rate, call frequency) after normalization. A 1D dilated CNN followed by an MLP produces the trace embedding $\mathbf{x}_v^{\text{trace}}$.
- **Logs** are parsed into templates using Drain3, and each window is represented by a PF-IDF vector that reweights template frequency by inverse document frequency across windows. This PF-IDF representation is passed through a lightweight projector to obtain the log embedding $\mathbf{x}_v^{\text{log}}$.

2) Construction of a fine-grained service dependency graph: Although the microservice system intrinsically contains entities at three granularities (node, service, pod), we flatten it and construct a *homogeneous* directed graph $G = (V, E)$ for unified representation learning. A type function $\tau(v) \in \{\text{NODE}, \text{SERVICE}, \text{POD}\}$ is retained merely as a categorical feature to record the semantic role of each node. The edge set E encodes connectivity derived from both interaction/propagation relations (e.g., service-to-service or pod-to-pod calls) and structural relations (e.g., pod-to-service membership and pod-to-node placement). In this homogeneous formulation, the graph provides unified *structural constraints* for message passing across all entity levels, while multimodal observability and type information are injected through node features.

B. Metric-Orthogonal Multimodal Alignment on a Fine-grained Dependency Graph

1) Orthogonal regularization and contrastive alignment: TopoEvo anchors alignment on metrics, which are continuous, high-frequency, and consistently available, providing a stable reference under noisy or missing observability from logs and traces. This metric-centric alignment reduces ambiguity from sparse modalities and yields a well-posed shared space for cross-entity comparison.

To prevent collapse where logs and traces align to the same metric factors, we introduce an orthogonal decomposition of metric features into two complementary components, dedicated to log-consistent and trace-consistent alignment, respectively, thereby preserving information while reducing redundancy in downstream graph reasoning.

Given metric embeddings $\mathbf{x}_v^{\text{metric}}$, two components are produced as

$$\mathbf{u}_v = \mathbf{W}_u \mathbf{x}_v^{\text{metric}}, \qquad \mathbf{v}_v = \mathbf{W}_v \mathbf{x}_v^{\text{metric}}, \tag{3}$$

where $\mathbf{u}_v$ is intended to capture metric factors most consistent with log evidence, while $\mathbf{v}_v$ captures complementary factors most consistent with trace evidence. To encourage

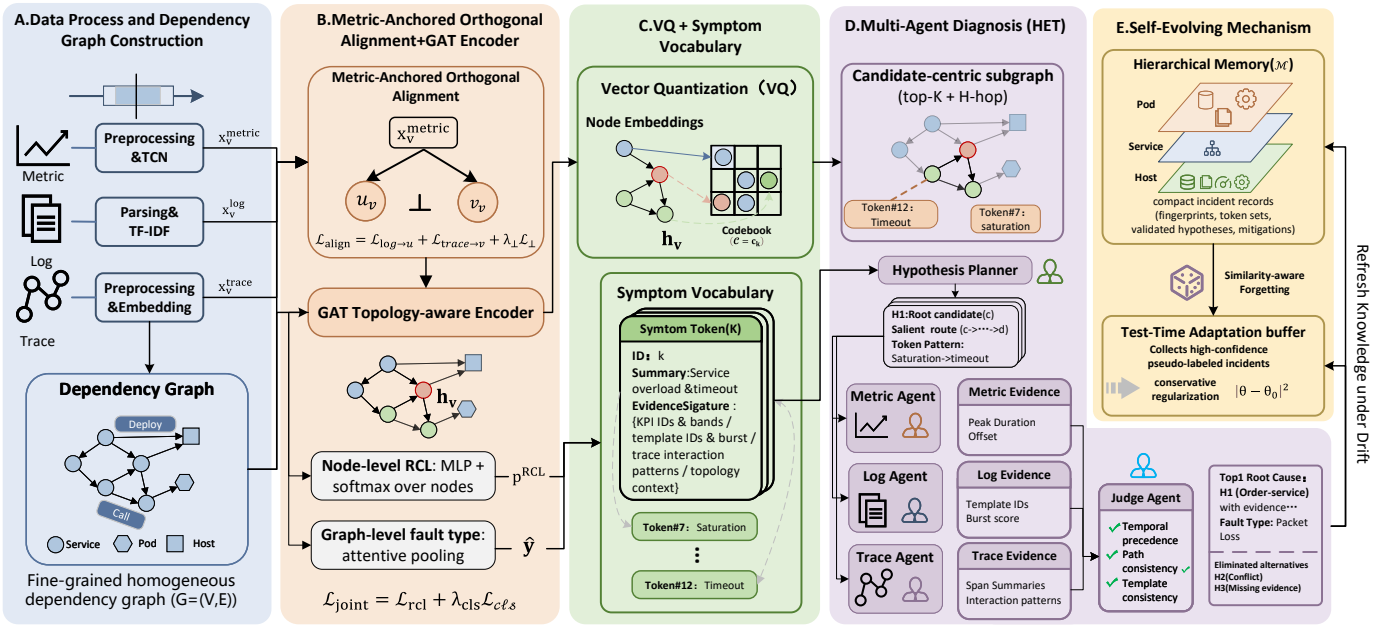

Fig. 2. The overview of TopoEvo.

these components to span different subspaces, an orthogonality regularizer is imposed:

$$\mathcal{L}_{\perp} = \sum_{v \in \mathcal{B}} \left(\frac{\mathbf{u}_v^{\top} \mathbf{v}_v}{\|\mathbf{u}_v\|_2 \, \|\mathbf{v}_v\|_2 + \epsilon} \right)^2 . \tag{4}$$

Modality alignment is performed with InfoNCE over a mini-batch $\mathcal{B}$:

$$\mathcal{L}_{\text{nce}}(\mathbf{a}, \mathbf{b}) = -\sum_{v \in \mathcal{B}} \log \frac{\exp(\text{sim}(\mathbf{a}_v, \mathbf{b}_v)/\tau)}{\sum_{v' \in \mathcal{B}} \exp(\text{sim}(\mathbf{a}_v, \mathbf{b}_{v'})/\tau)}, \tag{5}$$

yielding $\mathcal{L}_{\log \leftrightarrow u} = \mathcal{L}_{\text{nce}}(\mathbf{x}^{\log}, \mathbf{u})$ and $\mathcal{L}_{\text{trace} \leftrightarrow v} = \mathcal{L}_{\text{nce}}(\mathbf{x}^{\text{trace}}, \mathbf{v})$. Overall, the alignment objective combines contrastive alignment and orthogonality:

$$\mathcal{L}_{\text{align}} = \mathcal{L}_{\log \leftrightarrow u} + \mathcal{L}_{\text{trace} \leftrightarrow v} + \lambda_{\perp} \mathcal{L}_{\perp}. \tag{6}$$

To improve training stability and mitigate oscillations, we pretrain the alignment module first using the alignment loss $\mathcal{L}_{\text{align}}$ before proceeding to end-to-end optimization.

2) GAT-based topology-aware representation learning for root cause analysis: The node input feature is obtained by fusing modality embeddings

$$\mathbf{x}_v = \psi_n\left([\mathbf{x}_v^{\text{metric}};\ \mathbf{x}_v^{\log};\ \mathbf{x}_v^{\text{trace}}]\right). \tag{7}$$

A GAT encoder is applied on G. Let $\mathbf{h}_v^{(0)} = \mathbf{x}_v$. For each layer ℓ and edge $(j \to i)$,

$$e_{ij}^{(\ell)} = \text{LeakyReLU}\left(\mathbf{W}^{(\ell)} \mathbf{h}_i^{(\ell)} \ \|\ \mathbf{W}^{(\ell)} \mathbf{h}_j^{(\ell)}\right), \tag{8}$$

$$\alpha_{ij}^{(\ell)} = \frac{\exp(e_{ij}^{(\ell)})}{\sum_{j' \in \mathcal{N}(i)} \exp(e_{ij'}^{(\ell)})}. \tag{9}$$

$$\mathbf{h}_i^{(\ell+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(\ell)} \mathbf{W}^{(\ell)} \mathbf{h}_j^{(\ell)} \right). \tag{10}$$

The final topology-aware representation is $\mathbf{h}_v = \mathbf{h}_v^{(L)}$.

For *Root-cause localization*, we apply an MLP to produce per-entity probabilities:

$$\mathbf{p}^{\text{rcl}} = \text{softmax}(\text{MLP}(\mathbf{h})), \tag{11}$$

where $\mathbf{h}$ stacks $\mathbf{h}_v$ for all $v \in V$ and $\mathbf{p}_v^{\text{rca}}$ is the predicted root-cause probability of entity v. The objective of root cause localization is

$$\mathcal{L}_{\text{rcl}} = -\frac{1}{|\mathcal{D}|} \sum_{(G,y) \in \mathcal{D}} \sum_{v \in V} \mathbb{I}[i = y] \log p_v^{\text{rcl}}(G). \tag{12}$$

where $|\mathcal{D}|$ is the dataset size.

For *fault-type classification*, we first derive a graph-level representation via attentive pooling and then predict the fault-type distribution by an MLP followed by softmax:

$$\mathbf{g} = \sum_{v \in V} \alpha_v \mathbf{h}_v, \qquad \hat{\mathbf{y}} = \text{softmax}(\text{MLP}(\mathbf{g})), \tag{13}$$

where α_v is produced by a shallow scoring function on $\mathbf{h}_v$ and normalized over V.

Let $y^{\text{cls}} \in \{1, \ldots, C\}$ be the ground-truth fault type. The objective of fault-type classification is

$$\mathcal{L}_{\text{cls}} = -\frac{1}{|\mathcal{D}|} \sum_{(G,y^{\text{cls}}) \in \mathcal{D}} \sum_{c=1}^{C} \mathbb{I}[c = y^{\text{cls}}] \log \hat{y}_c. \tag{14}$$

The joint objective is

$$\mathcal{L}_{\text{joint}} = \mathcal{L}_{\text{rcl}} + \lambda_{\text{cls}} \mathcal{L}_{\text{cls}}. \tag{15}$$

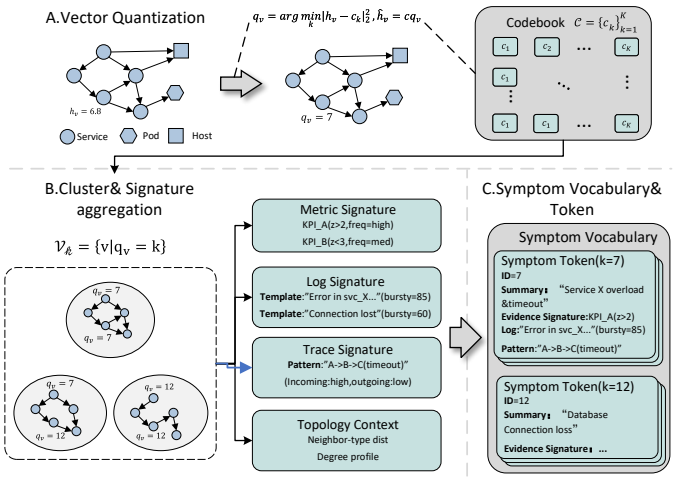

Fig. 3. Illustration of vector quantization and symptom vocabulary construction.

C. Topology-Aware Enhancement via Vector Quantization and Symptom Vocabulary

As shown in Fig. 3, This component is central to TopoEvo: it converts dense graph representations into discrete, retrievable, and auditable **symptom tokens**, enabling the reasoning process to operate on compact evidence rather than opaque vectors.

1) Vector quantization of topology-aware states: Vector quantization maps topology-aware node representations $\mathbf{h}_v$ to a finite codebook $\mathcal{C} = \{\mathbf{c}_k\}_{k=1}^{K}$. For each node v, the nearest code index and its quantized representation are:

$$q_v = \arg\min_k \|\mathbf{h}_v - \mathbf{c}_k\|_2^2, \qquad \hat{\mathbf{h}}_v = \mathbf{c}_{q_v}. \tag{16}$$

The codebook is learned with the standard VQ objective using stop-gradient $\mathrm{sg}[\cdot]$:

$$\mathcal{L}_{\mathrm{vq}} = \sum_v \left\|\mathrm{sg}[\mathbf{h}_v] - \hat{\mathbf{h}}_v\right\|_2^2 + \beta \sum_v \left\|\mathbf{h}_v - \mathrm{sg}[\hat{\mathbf{h}}_v]\right\|_2^2. \tag{17}$$

Quantization acts as a bottleneck that suppresses incident-specific noise, encourages clustering of recurring failure manifestations, and provides stable discrete indices for retrieval.

2) Symptom vocabulary construction: A **symptom vocabulary** translates each discrete code into an interpretable evidence unit. After quantization, each node is assigned to a code q_v, and nodes mapped to a code k form a cluster $\mathcal{V}_k = \{v \mid q_v = k\}$. Each code is treated as a reusable *symptom prototype*; instead of storing opaque vectors, TopoEvo attaches a compact descriptor that summarizes what tends to be abnormal when nodes fall into this cluster and how such abnormalities manifest in the system topology.

For each code k, a cluster-conditioned signature is estimated by aggregating observability evidence of nodes in $\mathcal{V}_k$ across training incidents. On the **metric** side, dominant KPI patterns are identified by ranking KPI dimensions using deviation statistics within $\mathcal{V}_k$ (e.g., typical z-score percentiles and frequency of crossing anomaly thresholds); the vocabulary keeps only a small set of representative KPIs together with typical magnitude bands to remain concise and robust. On the **log** side, raw messages are compressed into templates (e.g., Drain-style parsing) and the descriptor records templates that frequently co-occur with the code together with burstiness indicators, yielding stable textual anchors despite surface-form variability. On the **trace** side, propagation-relevant evidence is captured by recording recurring abnormal interaction patterns associated with the cluster, allowing the vocabulary to reflect whether anomalies are typically "incoming" (victim-like) or "outgoing" (source-like) along propagation. Finally, lightweight topology context such as neighbor-type distributions and degree profiles is attached to ground the token in the node's structural role.

Each code k is mapped to a structured symptom token,

$$\begin{aligned}\textsc{SymptomToken}(k) = \{&\mathrm{ID} = k,\ \textsc{Summary}(k),\\ &\textsc{EvidenceSignature}(k)\}\end{aligned} \tag{18}$$

where SUMMARY is a short natural-language descriptor distilled from dominant cluster-conditioned patterns, and EVIDENCESIGNATURE stores compact, verifiable pointers to the underlying statistics (KPI identifiers and value bands, template identifiers and burst scores, and interaction-pattern descriptors). This symptom vocabulary makes latent evidence auditable and provides stable retrieval keys across incidents, enabling topology-scoped token querying without exposing high-dimensional embeddings.

D. Reasoning-Enhanced Multi-Agent Diagnosis via Hypothesis–Evidence–Test

Instead of asking a single model to read all signals and directly output a root cause, TopoEvo decomposes diagnosis into 5 specialized roles that communicate through structured artifacts: a **hypothesis planner** (main prompt is shown in Fig. 4) proposes candidate explanations under topology constraints, modality-specific agents (**metric agent, log agent, trace agent**) collect verifiable evidence, and a **judge agent** performs constraint-based verification and explicitly eliminates strong alternatives. Discrete symptom tokens from the lexicon provide compact, consistent context for these agents, while topology saliency focuses attention on propagation-relevant regions.

1) Shared context: Candidate-centric subgraph and tokenized evidence. A candidate-centric subgraph is constructed to keep the workflow focused. TopoEvo selects the top-K candidate nodes according to the root-cause score $s(v)$ and extracts an H-hop induced subgraph around them. All subsequent reasoning and tool queries are restricted to this subgraph, while nodes outside it are masked to prevent distraction by weakly related services. Within this region, each node is mapped to a small set of symptom tokens by querying the symptom vocabulary using its VQ code q_v. Together with the graph structure, the tokenized subgraph forms a shared, compact context that is passed to all agents.

You are the Hypothesis Planner (HP) in TopoEvo, a topology-constrained multi-agent RCA framework for microservices.
Your job is NOT to decide the final root cause. Your job is to convert a ranked candidate list into a small set of TESTABLE, TOPOLOGY-CONSISTENT hypotheses and a modality-specific query plan.

Core principles:
1) Topology constraints first: every hypothesis must respect reachability and directionality in the dependency graph (causal propagation follows directed edges).
2) Symptom-amplification awareness: the most anomalous downstream node is often a victim. You MUST include at least one strong "victim-as-cause" alternative hypothesis to be eliminated later by the judge.
3) Multi-level consistency: entities can be Node / Service / Pod. State the hypothesized root level and use membership/placement edges when relevant.
4) Verifiability: each hypothesis must specify expected supporting evidence AND refuting checks, and a concrete query plan for Metric/Log/Trace agents.
5) Budgeted output: produce at most H_max hypotheses, deduplicate near-duplicates, and keep each hypothesis concise.

Do NOT hallucinate tools or new entities. Use only the provided graph, candidates, topology evidence, symptom tokens, and (optional) memory hits.
Return ONLY valid JSON that matches the required output schema. No extra prose.

Fig. 4. Main prompt of Hypothesis Planner.

2) Hypothesis generation, evidence acquisition, and verification: **Hypothesis generation** is *structure-prior-driven* rather than free-form. Starting from the top-K candidates, the planner uses saliency to highlight likely propagation routes, preferring paths that connect a candidate to downstream symptomatic nodes through high-saliency neighborhoods. Symptom tokens provide a compact description of what is abnormal at each node, allowing the planner to form hypotheses that are concrete enough to verify (e.g., "candidate c exhibits a saturation-like token; downstream nodes exhibit timeout-like tokens along a salient chain"). To keep the hypothesis set small and diverse, near-duplicate hypotheses are merged by sharing route prefixes or dominant token patterns. **Evidence acquisition** follows the planner's query plan and is intentionally *tool-grounded*. Instead of returning narrative explanations, modality agents return only structured evidence tied to entities in the candidate-centric subgraph, including timestamped onset estimates, magnitude bands, and template IDs or span summaries that can be checked later. This design prevents the workflow from being dominated by persuasive but unverifiable text: every claim used by the judge must be backed by cached evidence.

Verification makes the reasoning step explicit. For each hypothesis, the judge checks temporal precedence (whether the proposed root cause becomes abnormal no later than downstream symptoms within a slack), path consistency (whether evidence is located along the hypothesized route and consistent with the dependency graph), and template consistency (whether the collected evidence matches the intended failure template under strict/relaxed criteria to handle partial observability). The outcome is summarized with supporting evidence, conflicting evidence, and missing-but-expected evidence, making the result auditable.

3) Decision with explicit alternatives: The final output is not only a top-1 root cause, but also an explicit elimination of strong alternatives. TopoEvo reports the 4 most competitive alternative hypotheses and explains why they are rejected, attributing rejection to either *evidence conflict* (clear contradictions under the above constraints) or *missing evidence* (key signatures required by the route/template are absent). This "decision with alternatives" directly exposes the benefit of topology grounding and symptom-token querying: conclusions are tied to verifiable evidence under explicit structural constraints, rather than produced as unchecked narratives.

E. Self-Evolving Mechanism: Hierarchical Memory and Test-Time Adaptation

This component supports continual effectiveness under non-stationary systems by refreshing incident knowledge and cautiously adapting the encoder when reliable new supervision emerges.

1) Hierarchical incident memory with stochastic forgetting: A persistent memory $\mathcal{M}$ stores solved incidents as compact records consisting of candidate-centric topology fingerprints, symptom-token sets, validated hypotheses with key evidence, and mitigation outcomes. Updates follow a hierarchical policy: pod-level patterns are refreshed more aggressively than service- and node-level patterns, stabilizing slow-changing infrastructure knowledge while tracking fast-changing deployment dynamics.

To prevent redundancy collapse, similarity-aware stochastic forgetting is applied when inserting a new incident representation $\mathbf{u}_{\text{new}}$. Let $\text{sim}(\mathbf{u}_{\text{new}}, \mathbf{u}_i)$ denote cosine similarity and $\mathcal{N}_\tau$ be the set of memories above threshold τ. If $\mathcal{N}_\tau \neq \emptyset$, one similar item is forgotten with probability proportional to similarity:

$$p(i \mid \mathcal{N}_\tau) = \frac{\exp(\text{sim}(\mathbf{u}_{\text{new}}, \mathbf{u}_i)/\gamma)}{\sum_{j \in \mathcal{N}_\tau} \exp(\text{sim}(\mathbf{u}_{\text{new}}, \mathbf{u}_j)/\gamma)}. \tag{19}$$

2) Test-time adaptation with high-confidence pseudo-labels: During deployment, high-confidence diagnoses (strong support, low missingness, and consistent topology constraints) are treated as pseudo-labeled samples and accumulated in a buffer $\mathcal{B}_{\text{tta}}$. Once the buffer reaches a batch size, the encoder is updated with a conservative objective:

$$\mathcal{L}_{\text{tta}} = \mathcal{L}_{\text{rca}}^{\text{pseudo}} + \lambda_{\text{reg}} \|\theta - \theta_0\|_2^2, \tag{20}$$

where θ_0 are pre-deployment parameters and the regularizer mitigates catastrophic drift.

At inference time, the same encoder produces topology-aware representations and symptom tokens, while memory refresh and test-time adaptation enable continual robustness under evolving microservice behavior.

IV. Experiments

A. Experiment Setup

1) Dataset and Preprocessing: Dataset A is a large-scale public benchmark released by the AIOps Challenge [3]. It

TABLE I
EXPERIMENTAL RESULTS OF DIFFERENT APPROACHES ON ROOT CAUSE LOCALIZATION AND FAULT CLASSIFICATION. BEST RESULTS ARE BOLDED, AND SECOND-BEST RESULTS ARE UNDERLINED.

Data	Approach	Root Cause Localization			Fault Types Classification					
		Acc@1	Acc@3	Acc@5	MiPr	MaPr	MiRe	MaRe	MiF1	MaF1
A_s	HolisticRCA	65.62%	76.33%	81.69%	0.6355	0.7130	0.7728	0.8037	0.6975	0.7556
	Eadro	13.58%	37.03%	45.68%	0.1887	0.0770	0.1235	0.1046	0.1493	0.0887
	TAMO	71.87%	82.14%	88.83%	**0.7164**	0.6671	0.6486	0.6149	0.6808	0.6399
	Nezha	52.27%	80.68%	82.95%	–	–	–	–	–	–
	RCAgent	22.10%	28.40%	30.25%	–	–	–	–	–	–
	mABC	34.19%	42.13%	44.51%	–	–	–	–	–	–
	TopoEvo	**73.10%**	**83.05%**	**89.20%**	0.7052	**0.7248**	**0.7815**	**0.8112**	**0.7414**	**0.7656**
A_p	HolisticRCA	65.62%	80.80%	86.60%	0.5913	**0.6682**	0.7534	0.7598	0.6626	0.7111
	Eadro	9.38%	13.13%	15.63%	0.3901	0.3344	0.4665	0.4792	0.4249	0.3939
	TAMO	64.28%	80.36%	**87.50%**	0.7182	0.6597	0.6840	0.6642	0.7007	0.6619
	Nezha	44.18%	59.30%	65.11%	–	–	–	–	–	–
	RCAgent	21.70%	27.95%	30.40%	–	–	–	–	–	–
	mABC	33.80%	41.70%	44.20%	–	–	–	–	–	–
	TopoEvo	**66.10%**	**81.20%**	87.30%	**0.7325**	0.6668	**0.7588**	**0.7625**	**0.7454**	**0.7114**
A_n	HolisticRCA	73.66%	75.89%	79.01%	0.6219	0.6437	0.7612	0.7332	0.6845	0.6855
	Eadro	17.18%	28.13%	42.19%	0.5426	0.6003	0.7969	0.8145	0.6456	0.6912
	TAMO	**84.37%**	**91.96%**	97.77%	0.8718	**0.8869**	0.8947	0.8681	0.8831	0.8774
	Nezha	22.38%	29.85%	31.34%	–	–	–	–	–	–
	RCAgent	22.60%	29.10%	31.10%	–	–	–	–	–	–
	mABC	34.50%	42.60%	45.00%	–	–	–	–	–	–
	TopoEvo	84.10%	91.70%	**98.40%**	**0.8805**	0.8840	**0.9021**	**0.8720**	**0.8912**	**0.8780**
B	HolisticRCA	61.11%	75.00%	77.78%	0.5000	0.5936	**0.8056**	**0.8056**	0.6170	0.6835
	Eadro	40.62%	56.25%	71.87%	0.4167	0.5211	0.6250	0.6516	0.5000	0.5791
	TAMO	72.22%	**80.55%**	86.11%	**0.8500**	**0.8750**	0.5313	0.5682	0.6539	0.6890
	Nezha	49.50%	63.20%	71.40%	–	–	–	–	–	–
	RCAgent	24.80%	31.60%	35.10%	–	–	–	–	–	–
	mABC	36.90%	45.20%	49.30%	–	–	–	–	–	–
	TopoEvo	**75.66%**	79.80%	**86.80%**	0.8420	0.8685	0.8030	0.8024	**0.8220**	**0.8341**

is collected via controlled fault injection on a real-world deployed microservice system, *HipsterShop2*. The dataset provides multimodal observability signals, including *metrics, logs, and traces*. HipsterShop2 is deployed on a dynamic Kubernetes (K8s) cluster with 10 services, each replicated by 4 pods (40 pods in total), and the pods are dynamically scheduled across 6 nodes. The benchmark covers 15 fault types in total: 9 service/pod-level faults (in the K8s container context) and 6 node-level faults, including sudden memory pressure, disk space exhaustion, disk I/O anomalies, CPU pressure, and gradual CPU slowdown.

Dataset B is a real-world dataset collected from a production **Project Management Platform** operated by an Electric Power Information enterprise. Unlike Dataset A, the incidents in Dataset B are captured under *real operating conditions* rather than injected failures. The platform contains 12 microservices and 48 pods, and the dataset records multimodal observability (metrics, logs, and traces) during real incidents. Faults in Dataset B span 5 categories: CPU hog, memory leak, network delay, packet loss, and disk payload overload.

2) Baselines: To comprehensively evaluate **TopoEvo**, we compare against representative RCA approaches covering multimodal learning, graph-based localization, and LLM/agent-based diagnosis. **Nezha** [37] jointly encodes metrics, logs, and traces into a shared space with contrastive alignment and performs graph-based reasoning to localize causally relevant services under noisy/partial observability. **Eadro** [38] is an end-to-end multi-task framework that jointly learns anomaly detection and root-cause localization by modeling intra-service behaviors and inter-service dependencies from KPIs, logs, and traces. **HolisticRCA** [39] performs holistic RCA in cloud-native systems by standardizing heterogeneous observability data and reasoning over service dependencies to identify root causes across diverse failure scenarios. **TAMO** [40] is

a tool-assisted LLM-agent framework for fine-grained RCA that integrates multimodal alignment and model tools (e.g., localization/classification tools) to support root-cause analysis in cloud-native systems. **RCAgent** [9] is an autonomous, tool-augmented LLM agent for practical cloud RCA, which iteratively collects evidence from observability tools and synthesizes a diagnosis. **mABC** [11] is a blockchain-inspired multi-agent collaboration framework that reduces hallucination via decentralized voting and prevents non-terminating loops with a step-bounded standardized workflow.

3) Metrics: We evaluate *root cause localization* and *fault type classification* For localization, we report Top-K accuracy (ACC@1/3/5) and mean reciprocal rank (MRR). For fault-type classification, we report micro-/macro-precision, micro-/macro-recall, and micro-/macro-F1 (MIPR/MAPR, MIRE/MARE, MIF1/MAF1). For efficiency, we measure end-to-end diagnosis latency (wall-clock time per incident, including agent/tool calls when applicable). For explanation quality, we report a **topology faithfulness** score, which checks whether the cited propagation paths are valid in the dependency graph G and whether they overlap with topology-salient (high-weight) edges used by TopoEvo.

B. Implementation

All experiments are conducted on a server equipped with an NVIDIA A100 80GB GPU and 256GB RAM. The graph encoder adopts a two-layer GAT architecture with 8 attention heads. Training is performed using the Adam optimizer with a learning rate of 0.001. For the vector quantization module, the codebook size K is set to 128.

For all LLM-based components, GPT-4o-2024-11-20 is used as the backbone model to ensure consistent reasoning ability across different settings.

C. RQ1: Effectiveness of root cause localization and fault type classification

1) Results: Table I summarizes the results on root cause localization and fault type classification. Overall, **TopoEvo achieves the most competitive and consistent performance across datasets**, obtaining the best results on most metrics and remaining close to the top performer elsewhere.

For **root cause localization**, TopoEvo performs best on the **service level**, outperforming TAMO by $1.23/0.91/0.37$ percentage points on Acc@1/3/5. On the pod level, it achieves the best Acc@1 and Acc@3 (**66.10%/81.20%**) and ranks second on Acc@5 (87.30%), only slightly below TAMO. On the node level and Dataset B, although TAMO is stronger on Acc@1/3, TopoEvo consistently ranks second and achieves the best Acc@5, indicating stronger *top-k coverage* of true root causes.

For **fault type classification**, TopoEvo shows clear advantages in balanced prediction quality, especially on F1. On A_s, it achieves the best MaPr, MiRe, MaRe, MiF1, and MaF1; on A_p and A_n, it again leads on most recall and F1 metrics while remaining competitive on precision. On B, although TopoEvo is not the best on individual precision or recall metrics, it achieves the highest **MiF1** and **MaF1**, demonstrating stronger overall class discrimination.

These results suggest that TopoEvo not only improves localization quality, but also provides more reliable downstream diagnosis. This advantage comes from topology-aware multimodal representation learning and the HET-based multi-agent reasoning process, which together improve candidate quality and reduce spurious decisions.

TABLE II
ABLATIONS ON TOPOEVO COMPONENTS. WE REPORT ROOT-CAUSE LOCALIZATION ACCURACY (AC@1/3/5). BEST RESULTS ARE BOLDED, AND SECOND-BEST ARE UNDERLINED.

Approach	A_s			A_p		
	Acc@1	Acc@3	Acc@5	Acc@1	Acc@3	Acc@5
Full	**73.10%**	**83.05%**	**89.20%**	**66.10%**	**81.20%**	**87.30%**
w/o MOMA	63.10%	74.05%	80.20%	56.10%	72.20%	78.30%
w/o VQ	61.10%	72.05%	79.20%	54.10%	70.20%	77.30%
w/o HET	57.10%	68.05%	75.20%	50.10%	66.20%	73.30%
w/o SEM	65.10%	75.05%	81.20%	58.10%	73.20%	79.30%
Approach	A_n			B		
	Acc@1	Acc@3	Acc@5	Acc@1	Acc@3	Acc@5
Full	**84.10%**	**91.70%**	**98.40%**	**75.66%**	**79.80%**	**86.80%**
w/o MOMA	74.10%	82.70%	89.40%	61.90%	70.80%	77.80%
w/o VQ	72.10%	80.70%	88.40%	59.90%	68.80%	76.80%
w/o HET	68.10%	76.70%	84.40%	55.90%	64.80%	72.80%
w/o SEM	76.10%	83.70%	90.40%	63.90%	71.80%	78.80%

Abbrev: MOMA = Metric-Orthogonal Multimodal Alignment; VQ = Vector Quantization; HET = Hypothesis–Evidence–Test; SEM = Self-Evolving Mechanism.

D. RQ2: Ablation Study

Table II reports ablations of TopoEvo on root-cause localization (AC@1/3/5) across four subsets (A_s, A_p, A_n, and B). Overall, removing any key component consistently degrades performance.

Impact of Hypothesis–Evidence–Test (HET). We use Disabling the HET reasoning loop yields the largest drop, confirming that TopoEvo is not merely a one-shot aggregation of signals but a verification-driven diagnosis pipeline. In particular, w/o HET reduces AC@1 by 15–16 points on all subsets, with similar degradation on AC@3/AC@5. This suggests that explicit hypothesis testing and alternative exclusion are essential to avoid symptom-root confusion when anomalies propagate.

Impact of Metric-Orthogonal Multimodal Alignment (MOMA). Removing MOMA also causes substantial performance loss (typically 10 points on AC@1), indicating that metric-centered orthogonal alignment effectively reduces modality mismatch and stabilizes node representations before GAT encoding. For example, on A_p the AC@1 drops from 66.10% to 56.10%, and on A_n from 84.10% to 74.10%, demonstrating its importance under correlated and noisy observability.

Impact of Vector Quantization (VQ). Disabling VQ leads to a clear performance decrease across all subsets (about 12 points on AC@1 in most cases), showing that discretizing node states into symptom tokens provides robust, low-entropy evidence for downstream reasoning. Besides improving ranking

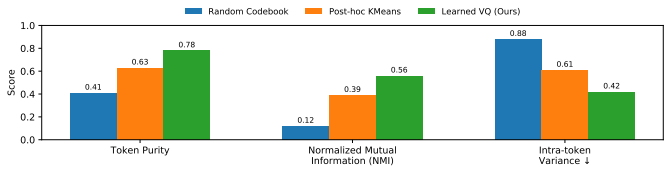

Fig. 5. Parameter sensitivity analysis. K denotes the VQ partition (codebook size) parameter.

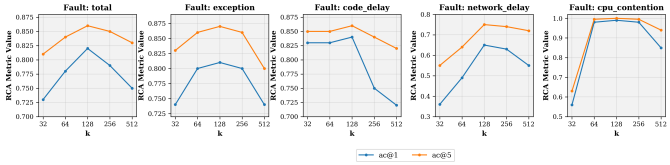

Fig. 6. Parameter sensitivity analysis. K denotes the VQ partition (codebook size) parameter.

accuracy (e.g., $73.10\% \rightarrow 61.10\%$ on A_s), VQ also supports more consistent topology-grounded explanations by enabling auditable token-level references.

Impact of the Self-Evolving Mechanism (SEM). Finally, removing the self-evolving mechanism produces a smaller but still notable drop (≥ 8 points on AC@1), reflecting the benefit of retrieval priors and adaptation under non-stationarity. For instance, w/o SEM decreases AC@1 from 66.10% to 58.10% on A_p and from 71.90% to 63.90% on B. This confirms that continual experience reuse and adaptation help maintain RCA robustness as system behavior shifts.

E. RQ3: Does VQ Learn a Better Symptom Vocabulary?

To verify whether VQ contributes more than performance gain, we evaluate the quality of the learned symptom vocabulary. We compare **Learned VQ** with two baselines built on the same frozen topology-aware encoder: **Random Codebook** and **Post-hoc KMeans**. All methods use the same codebook size $K = 128$.

We measure vocabulary quality using **Token Purity**, **Normalized Mutual Information (NMI)** and **Intra-token Variance**. The first three metrics evaluate semantic alignment between token assignments and ground-truth fault categories, while the latter two characterize cluster compactness and separability.

As shown in Fig. 5, **Learned VQ consistently outperforms both baselines on all metrics**: it achieves higher Token Purity and Normalized Mutual Information, while yielding lower Intra-token Variance. These results show that VQ learns more compact and fault-consistent discrete units than heuristic or post-hoc discretization.

Overall, the learned codebook forms a compact and discriminative symptom vocabulary, which provides more stable evidence units for retrieval and HET-based reasoning.

F. RQ4: Parameter Sensitivity

We study the sensitivity of TopoEvo to the VQ codebook size K, which controls the granularity of discretizing topology-aware states into symptom tokens. Fig. 6 reports root-cause localization performance under different $K \in \{32, 64, 128, 256, 512\}$, including overall results and representative fault types.

1) Overall trend: Across both ACC@1 and ACC@5, performance improves as K increases from 32 to 128, and then degrades when K becomes larger (256/512). The best overall performance is achieved at $K = 128$, indicating that a *moderate* codebook size provides the most effective discretization for token-based reasoning.

2) Why too small K hurts: When K is small (e.g., 32/64), the codebook becomes overly coarse and forces heterogeneous failure manifestations to share the same discrete code. This causes *prototype collision*: distinct symptoms (e.g., saturation-like vs. timeout-like patterns) are compressed into a single token, blurring causal cues and weakening downstream hypothesis verification. As a consequence, the symptom lexicon becomes less discriminative and the planner/judge receive ambiguous evidence, especially for faults with overlapping surface symptoms.

3) Why too large K hurts: When K is large (256/512), the discretization becomes overly fine-grained, causing subtle noise and incident-specific variations to be encoded as separate codes while splitting recurring symptoms into many near-duplicate tokens. This over-segmentation leads to three main issues:

- **Sparse code usage and unstable symptom vocabulary.** Larger K leaves fewer samples per code, making code-level signatures noisy and reducing retrieval consistency.
- **Weaker generalization under drift.** Fine-grained codes are more sensitive to topology and traffic shifts, which harms cross-incident transfer under OOD conditions.
- **Fragmented evidence for verification.** Similar symptoms may be mapped to multiple tokens, making propagation patterns less coherent and weakening hypothesis confidence.

G. RQ5: Case Study

a) Scenario and observation: Fig. 7 presents an incident in a microservice dependency graph where an upstream overload in **payment-service** (pod **P0**) propagates to **user-service** (pod **U2**) and further triggers timeouts at **gateway-service** (pod **G1**). A GAT-based root-cause localizer (RCL) assigns the highest root-cause score to **user-service** ($S(\mathbf{U2}) = 0.51$), while the true root cause **payment-service/P0** is only ranked second ($S(\mathbf{P0}) = 0.42$). This mismatch is a typical *symptom-amplification bias*: downstream services may exhibit stronger and more visible symptoms (e.g., retry bursts and timeouts), misleading symptom-driven rankers.

b) Why GAT misranks the root cause: In this incident, the most salient anomaly manifests at **user-service** as a bursty retry/timeout pattern, which yields stronger multimodal

evidence aggregated at node **U2**. Meanwhile, the true cause **P0** primarily shows early-stage resource saturation (CPU stress) that may be less visually dominant at the service level. As a result, a purely score-based ranking tends to select the node with the largest observed symptom magnitude rather than the node that *causally initiates* the propagation.

c) TopoEvo: topology evidence → hypothesis planning → tool-grounded verification. TopoEvo corrects this failure mode by converting "score ranking" into "topology-constrained causal adjudication". Starting from the GAT candidate list (Top-1: **U2**, Top-2: **P0**), TopoEvo constructs a compact *Hierarchical Evidence Trace* (HET) package that explicitly aligns symptoms with topology across levels: *service path* **payment** → **user** → **gateway**, *pod path* **P0** → **U2** → **G1**, and *token trace* **Saturation** → **retry burst** → **Timeout**. These structured traces are summarized into discrete symptom tokens (e.g., Token #7: saturation/overload; Token #12: timeout), which serve as interpretable evidence units for the reasoning layer.

Given the evidence package, the *Hypothesis Planner* generates explicit, topology-consistent alternatives: **H1**: **payment-service/P0** overload → timeout propagation to **gateway**, **H2**: **gateway-service/g1** timeout is the primary root (alternative). Unlike direct ranking, hypotheses are required to respect causal direction and propagation reachability in the dependency graph.

Next, TopoEvo launches tool-grounded agents to verify each hypothesis using multimodal signals: (1) a metric agent checks onset time and threshold crossing (CPU/latency bands); (2) a log agent checks template IDs and burst scores(retry bursts at **U2**); (3) a trace agent verifies abnormal span chains and their directionality along **P0**→**U2**→**G1**. A judge module then applies a checklist-style adjudication: *temporal precedence* (cause precedes effect), *path consistency* (evidence aligns with a valid propagation path), and *template consistency* (log/trace patterns match the hypothesized fault type, under strict/relaxed matching). In this case, the evidence supports H1: saturation at **P0** occurs earlier and lies on the unique propagation path to the downstream timeouts, while H2 fails temporal/path constraints because **g1** symptoms are downstream effects without upstream initiating evidence. Therefore, TopoEvo **accepts H1** and **rejects H2**, yielding the final decision: **Root Cause**: **payment-service/P0**; **Fault Type**: CPU stress, while eliminating alternatives such as **gateway-service/G1** and **order-service/G2**.

V. Conclusion

We presented **TopoEvo**, a topology-aware self-evolving multi-agent framework for microservice RCA under noisy multimodal observability and non-stationary topology drift. TopoEvo tightly connects representation learning and structured reasoning: (1) *Metric-Orthogonal Multimodal Alignment* stabilizes cross-modal fusion by aligning logs/traces to complementary metric subspaces; (3) *VQ-based symptom tokenization* converts dense topology-aware states into compact, retrievable, and auditable symptom evidence; (3)

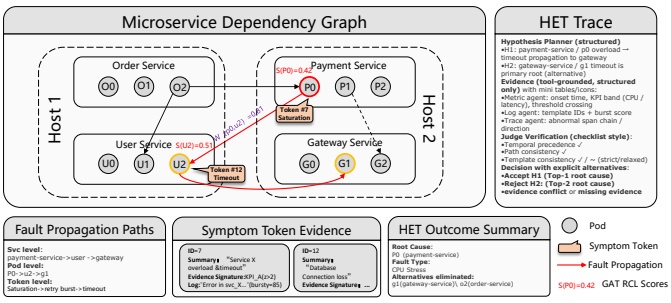

Fig. 7. In case study experiments, We injected a **CPU Stress** fault into the Payment Service, and the fault propagated along the path P0–U2–G1.

a *Hypothesis–Evidence–Test* multi-agent workflow performs tool-grounded verification under explicit topology constraints to mitigate symptom-amplification misattribution; and (4) a *Self-Evolving Mechanism* maintains robustness via hierarchical memory refresh and conservative pseudo-label adaptation.

Experiments on both injected-fault benchmarks and real production incidents demonstrate that TopoEvo achieves strong and consistent gains in root-cause localization and fault-type classification across granularities, and ablation results further verify that each module contributes materially, with HET-based verification yielding the largest benefit. In future work, we plan to (1) strengthen causal validation beyond topology-consistent verification (e.g., intervention-aware checks), (2) design safer and more cost-aware continual adaptation policies under drift, and (3) extend evaluation to more heterogeneous microservice platforms and observability conditions.

VI. Related Work

Microservice RCA has been widely studied under metrics/logs/traces, ranging from classical survey/benchmark efforts [1]–[3] to learning-based multimodal localization frameworks that fuse heterogeneous telemetry for fine-grained diagnosis [6], [37]–[39]. Another line models propagation and causality via event/causal graphs or dynamic causal inference to improve robustness under cascading failures and limited observability [4], [5], [22], [28], [29]. Recently, LLM/agent-based RCA has emerged, leveraging tool use and iterative evidence gathering for explainable diagnosis [9]–[11], [34]. Compared to purely learned localizers, LLM agents offer stronger semantic abstraction over logs and the ability to follow SOP-like workflows, but they can be brittle under noisy tool outputs and may drift into ungrounded narratives without explicit structural constraints [9], [10]. This motivates topology- and evidence-constrained reasoning that couples graph-based localization with structured prompts and verification, mitigating symptom-amplification and improving reliability in real deployments.

References

[1] Soldani, J., Brogi, A.: Anomaly Detection and Failure Root Cause Analysis in (Micro)Service-Based Cloud Applications: A Survey. Journal of Systems and Software **178**, 110964 (2021)

[2] Wang, T., Qi, G.: A Comprehensive Survey on Root Cause Analysis in (Micro) Services: Methodologies, Challenges, and Trends. arXiv preprint arXiv:2408.00803 (2024)

[3] L. Pham, H. Zhang, H. Ha, F. Salim, and X. Zhang, "RCAEval: A Benchmark for Root Cause Analysis of Microservice Systems with Telemetry Data," in *Companion Proceedings of the ACM Web Conference 2025 (WWW Companion)*, pp. 777–780, 2025.

[4] Z. Yao, C. Pei, X. Nie, et al., "Chain-of-Event: Interpretable Root Cause Analysis for Microservices through Automatically Learning Weighted Event Causal Graph," in *Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE)*, 2024.

[5] Pan, X., Yu, Y., Zhang, H.: DyCause: Dynamic Causal Inference for Root Cause Analysis. In: Proceedings of the 45th International Conference on Software Engineering (ICSE), pp. 435–446 (2023)

[6] Wu, G., Liu, D., Zhang, H.: MMRCA: Multimodal Root Cause Analysis via Fusion of Logs, Metrics and Traces. In: IEEE International Conference on Services Computing (SCC), pp. 110–120 (2021)

[7] Li, X., Zhang, H., Pham, L., et al.: MRCA: Metric-Level Root Cause Analysis for Microservices via Multi-Modal Data. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1115–1127 (2024)

[8] Lou, J., Yu, H., Zhang, M., et al.: Unimodal is Not Enough: A Benchmark and Unified Framework for Multimodal Anomaly Detection in Logs, Metrics, and Traces. In: Proceedings of the ACM Web Conference (WWW), pp. 2841–2852 (2023)

[9] Z. Wang, Z. Liu, Y. Zhao, et al., "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 (CIKM)*, 2024.

[10] 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 *Companion Proceedings of the ACM Web Conference 2025 (WWW Companion)*, pp. 422–431, 2025.

[11] W. Zhang, H. Guo, J. Yang, Z. Tian, Y. Zhang, C. Yan, Z. Li, T. Li, X. Shi, L. Zheng, and B. Zhang, "mABC: Multi-Agent Blockchain-inspired Collaboration for Root Cause Analysis in Micro-Services Architecture," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4017–4033, 2024.

[12] P. Tang, S. Tang, H. Pu, Z. Miao, and Z. Wang, "MicroRCA-Agent: Microservice Root Cause Analysis Method Based on Large Language Model Agents," *arXiv preprint arXiv:2509.15635*, 2025.

[13] L. Zheng, Z. Chen, and H. Chen, "Online Multi-modal Root Cause Identification in Microservice Systems," in *2025 IEEE International Conference on Big Data (BigData)*, pp. 1–8, 2025.

[14] Phan, X., Li, M., Zhang, Q.: MicroCERCL: Root Cause Localization in Cloud-Edge Microservice Environments. arXiv preprint arXiv:2406.13604 (2024)

[15] Li, Z., Chen, J., Jiao, R., et al.: Practical Root Cause Localization for Microservice Systems via Trace Analysis. In: IEEE International Conference on Cloud Computing (CLOUD), pp. 25–34 (2021)

[16] Jha, S., et al.: Localizing and Explaining Faults in Microservices Using Distributed Traces and Logs. In: IEEE International Conference on Cloud Computing (CLOUD), pp. 53–63 (2022)

[17] Li, Z., Zhang, H., Wang, C.: MicroIRC: Instance-Level Root Cause Localization for Microservice Systems. Journal of Systems and Software **206**, 111520 (2023)

[18] Chillarege, R., Bhandari, I., Chaar, J., et al.: Orthogonal Defect Classification—A Concept for In-Process Measurements. IEEE Trans. Softw. Eng. **18**(11), 943–956 (1992)

[19] Nguyen, H.A., Tan, J., Gu, X., et al.: PAL: Performance Anomaly Localization for Cloud Systems. In: Proceedings of the 17th ACM SIGKDD, pp. 1230–1238 (2011)

[20] Jia, Z., He, J., Xu, X., et al.: Fault Localization for Microservice Systems via Graph Neural Network Learning. In: IEEE International Conference on Services Computing (SCC), pp. 218–225 (2020)

[21] Yang, Y., Zhou, M., Yang, S., et al.: Unsupervised Root Cause Analysis for Microservice Systems via Spatio-Temporal Graph Modeling. In: IEEE/ACM International Conference on Software Engineering (ICSE), pp. 156–166 (2020)

[22] Guo, Y., Wang, J., Li, Y., et al.: CARR: A Causal-Aware Neural Approach for Root Cause Localization in Cloud Systems. In: Proceedings of the 28th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), pp. 3922–3930 (2022)

[23] Lin, Y., Fang, Y., Zhang, H., et al.: Causal Tracing for Root Cause Localization in Microservices. In: IEEE International Symposium on Software Reliability Engineering (ISSRE), pp. 33–44 (2022)

[24] Wang, H., Wu, Z., Jiang, H., et al.: Groot: An Event-Graph-Based Approach for Root Cause Analysis in Industrial Settings. arXiv preprint arXiv:2108.00344 (2021)

[25] A. Ikram, S. Chakraborty, S. Mitra, S. K. Saini, S. Bagchi, and M. Kocaoglu, "Root Cause Analysis of Failures in Microservices through Causal Discovery," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[26] Chakraborty, S., et al.: Root Cause Analysis of Failures in Microservices through Causal Structure Discovery. In: Advances in Neural Information Processing Systems (NeurIPS) (2022)

[27] Zhou, Y., Chen, T., Liu, Z., et al.: LatentScope: Unsupervised Root Cause Analysis with Limited Observability. In: ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), pp. 1–13 (2024)

[28] Z. Xie, S. Zhang, Y. Geng, X. Nie, Z. Yao, L. Xu, Y. Sun, W. Li, and D. Pei, "Microservice Root Cause Analysis With Limited Observability Through Intervention Recognition in the Latent Space," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2024.

[29] L. Pham, H. Ha, and H. Zhang, "BARO: Robust Root Cause Analysis for Microservices via Multivariate Bayesian Online Change Point Detection," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 2214–2237, 2024.

[30] Neumann, A.: Causality Based Instant Root Cause Analysis for Microservices Failure. In: NCTA Conference Proceedings, pp. 140–152 (2024)

[31] Pham, L., Ha, H., Zhang, H.: Root Cause Analysis for Microservice Systems Based on Causal Inference: How Far Are We? arXiv preprint arXiv:2408.13729 (2024)

[32] DoWhy Contributors: Root Cause Analysis of Latencies in a Microservice Architecture. DoWhy Example Notebook (2024), https://www.pywhy.org

[33] Zhang, H., Pham, L., Liu, Y.: Intelligent Root Cause Localization in MicroService Systems. In: ACM Symposium on Cloud Computing (SoCC) (2025)

[34] L. Wang, C. Zhang, R. Ding, Y. Xu, Q. Chen, W. Zou, Q. Chen, M. Zhang, X.-C. Gao, H. Fan, S. Rajmohan, Q. Lin, and D. Zhang, "Root Cause Analysis for Microservice Systems via Hierarchical Reinforcement Learning from Human Feedback," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 5116–5125, 2023.

[35] L. Zhang, T. Jia, K. Wang, W. Hong, C. Duan, M. He, and Y. Li, "Adaptive Root Cause Localization for Microservice Systems with Multi-Agent Recursion-of-Thought," *arXiv preprint arXiv:2508.20370*, 2025.

[36] L. Zhang, T. Jia, Y. Zhai, L. Pan, C. Duan, M. He, M. Jia, and Y. Li, "Agentic Memory Enhanced Recursive Reasoning for Root Cause Localization in Microservices," *arXiv preprint arXiv:2601.02732*, 2026.

[37] G. Yu, P. Chen, Y. Li, et al., "Nezha: Interpretable Fine-Grained Root Causes Analysis for Microservices on Multi-modal Observability Data," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2023.

[38] C. Lee, T. Yang, Z. Chen, Y. Su, and M. R. Lyu, "Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-source Data," in *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*, pp. 1750–1762, 2023.

[39] Y. Han, Q. Du, Y. Huang, P. Li, X. Shi, J. Wu, P. Fang, F. Tian, and C. He, "Holistic Root Cause Analysis for Failures in Cloud-Native Systems Through Observability Data," *IEEE Transactions on Services Computing*, vol. 17, no. 6, pp. 3789–3802, 2024.

[40] Zhang X, Wang Q, Li M, et al. TAMO: Fine-Grained Root Cause Analysis via Tool-Assisted LLM Agent with Multi-Modality Observation Data in Cloud-Native Systems[J]. IEEE Transactions on Services Computing, 2025, 18(6): 4221-4233.