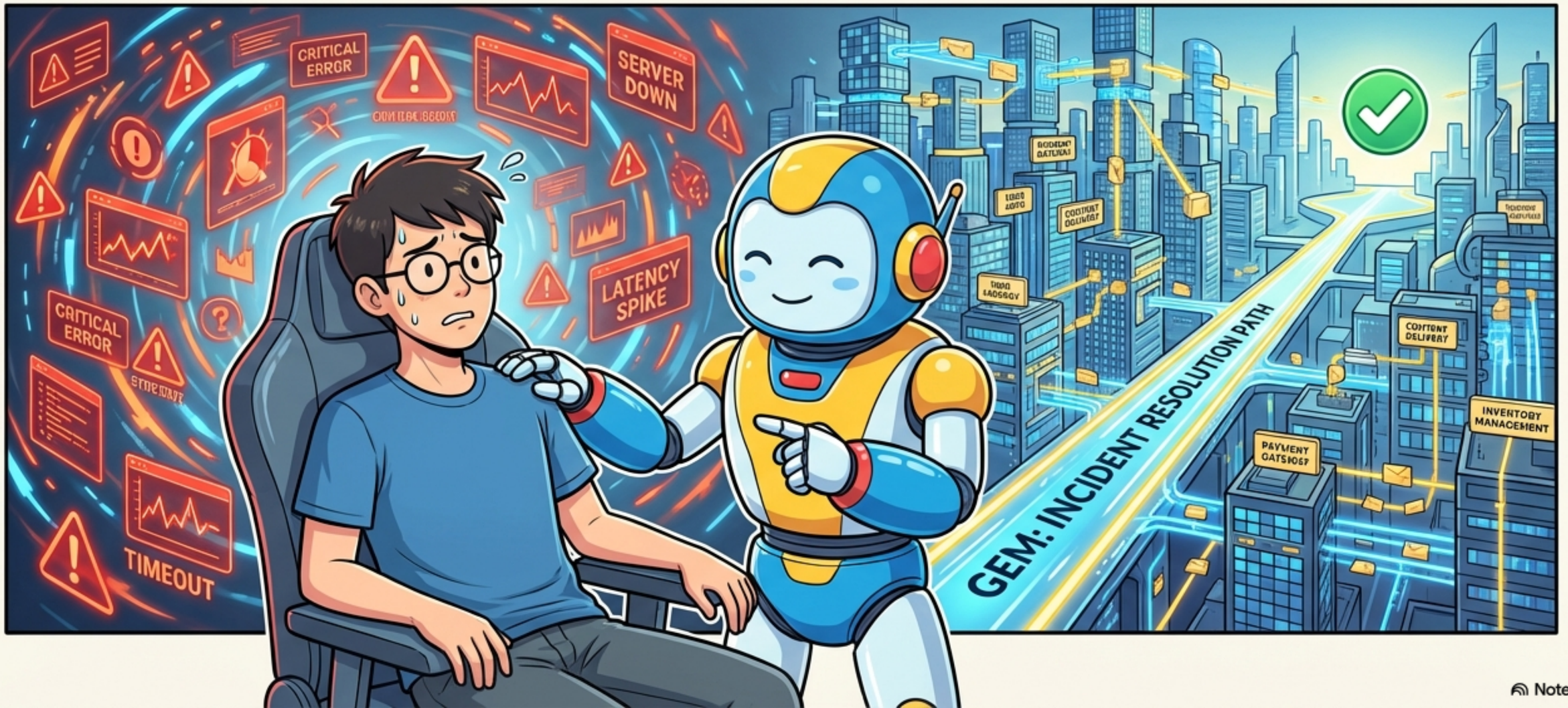
Taming the Incident Storm

How GEM brings order to complex, large-scale online systems.



Our Adventure Map

Lost in the Incident Storm

- The overwhelming scale of modern systems.
- Why traditional tools fall short.

A Gadget from the Future

- Introducing GEM and its core idea: Impact Subgraphs.
- The Four-Step Workflow for incident management.

Inside the Gadget - How GEM Works

- Smart Incident Detection.
- Pinpointing the Root Cause.
- A system that learns and evolves.

Taming the Storm

- Quantitative results and performance benchmarks.
- Real-world success stories from the WeChat deployment.

Modern Systems Have Become Impossible Cities

On-call engineers face three relentless challenges as systems evolve:

1. System Scale Evolution

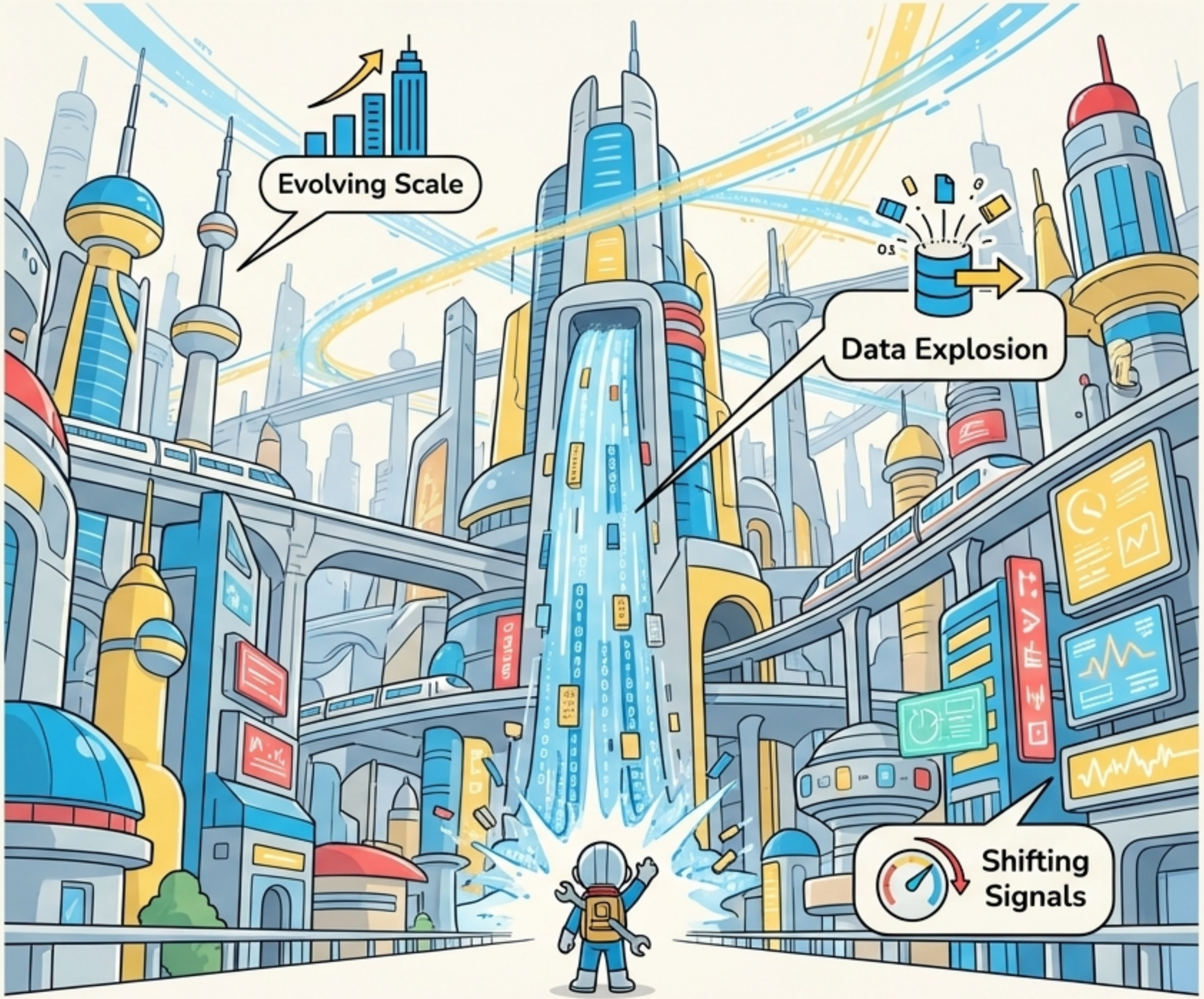
Systems grow constantly. The WeChat backend, for example, evolved from ~3,000 services to over **21,871 services** with **85,966 call relationships**. Whole-system modeling becomes impractical.

2. Data Volume Evolution

The amount of telemetry data is overwhelming. WeChat generates over **500TB** of call-relationship data *per day*. Persisting all of this for analysis is not feasible, and valuable incident records are scarce.

3. Diagnosis Telemetry Evolution

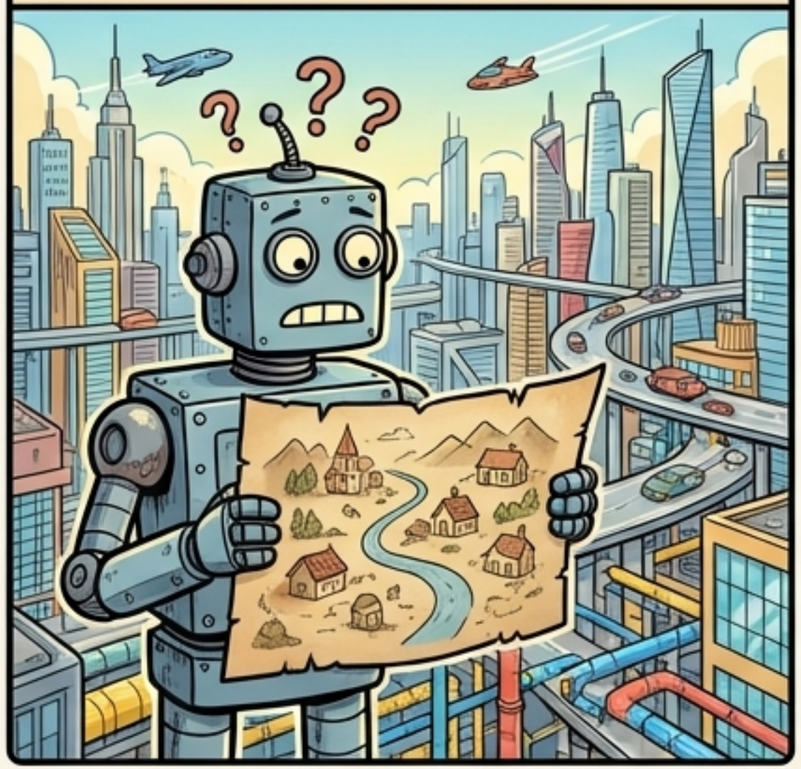
The useful signals for diagnosis change over time. Manually selecting the right telemetry in advance is a losing battle.



Why Traditional Tools Can’t Keep Up

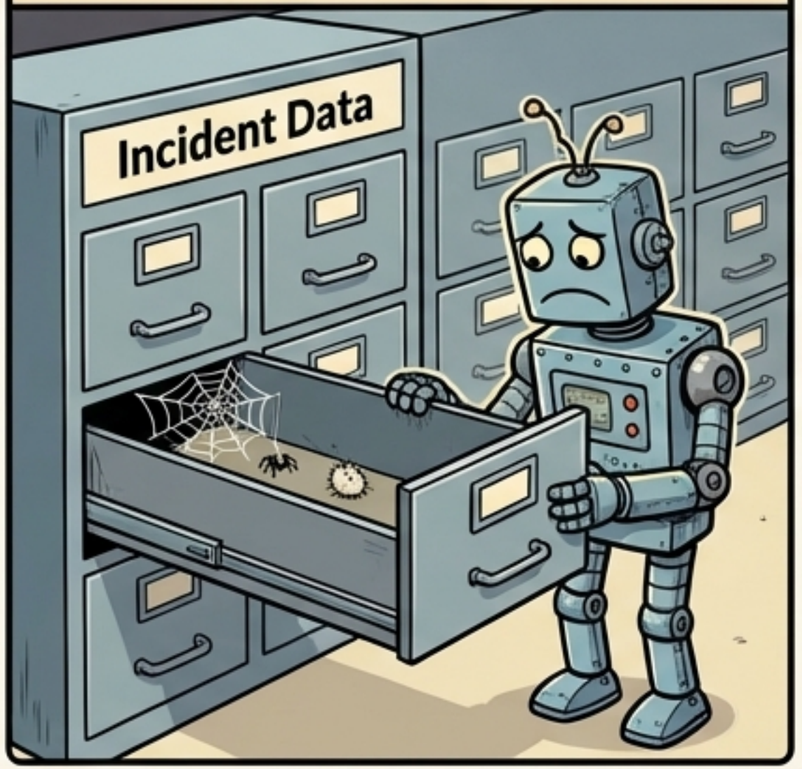
Prior methods often fail in the real world because they are built on over-optimistic assumptions:

Assumption #1: “The system will always be small and static.”



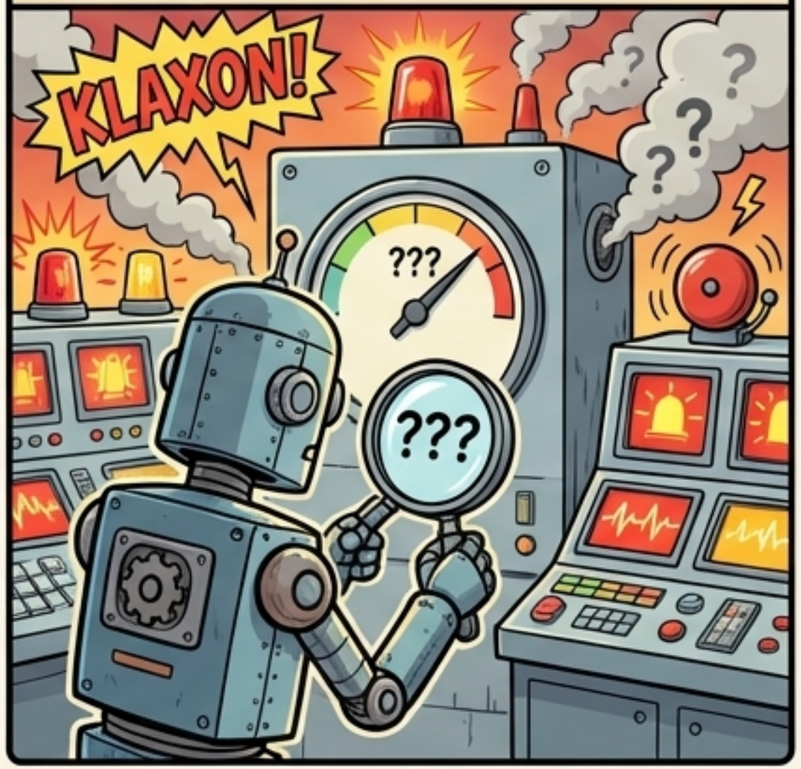
Reality: They rely on whole-system modeling, which breaks in large-scale, evolving systems where historical snapshots are too expensive to keep.

Assumption #2: “A rich dataset of labeled incidents is always ready.”



Reality: Incident records are scarce and generated slowly. You can't wait for a perfect dataset; you need a solution that can work from day one.

Assumption #3: “We know all the useful telemetry in advance.”



Reality: The principle of “Garbage in, Garbage out” applies. If operators select the wrong telemetry, the model fails. The useful signals must be discovered over time.

A Helper from the Future: Meet GEM

Instead of wrestling with the entire system, GEM **focuses only** on the **active problem area**.

The Breakthrough: GEM treats **Issue Impact Subgraphs** as "first-class citizens."

What this means: We refine raw telemetry data into compact, expressive graph representations of an issue *as early as possible*.

- These subgraphs are dynamically created, passed between processes, and persisted for analysis.
- They are the central unit of work, not whole-system snapshots.

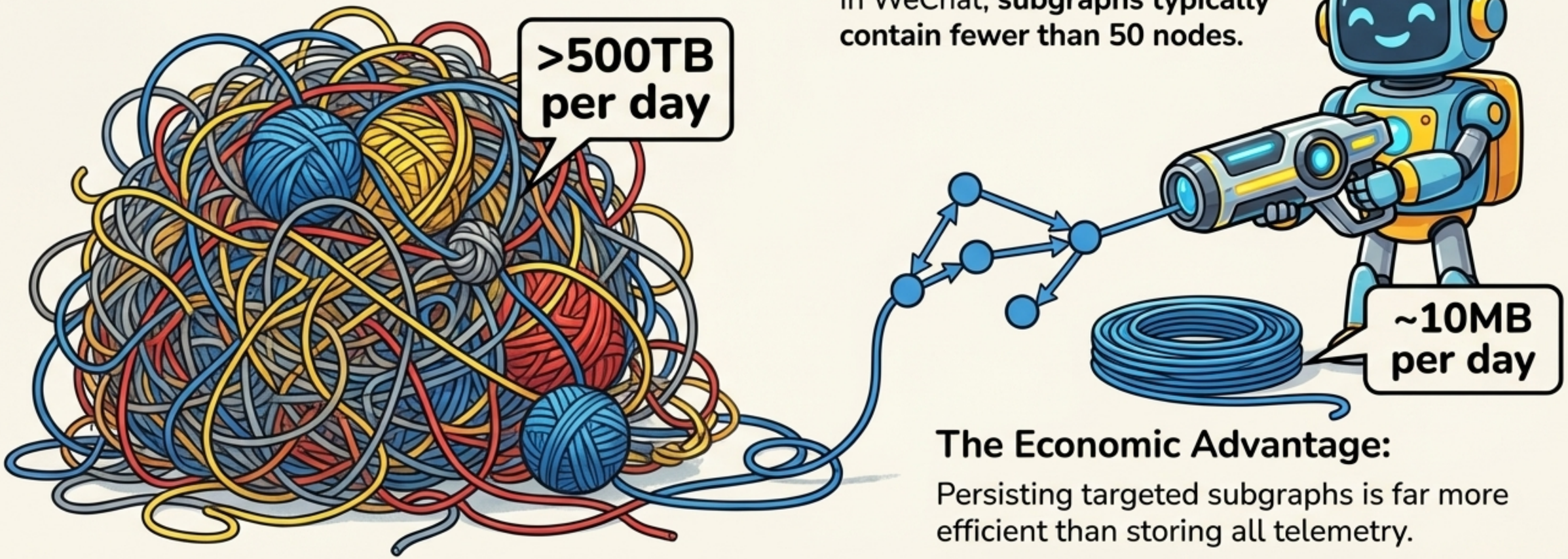
The Gadget: A Lens to Isolate the Problem

How it works:

GEM sifts through millions of anomalous signals and groups them into distinct issues based on their temporal patterns and connectivity.

The Result:

A small, focused **Impact Subgraph** that represents the true scope of a single issue. Failures rarely cascade throughout the entire system. In WeChat, **subgraphs typically contain fewer than 50 nodes.**

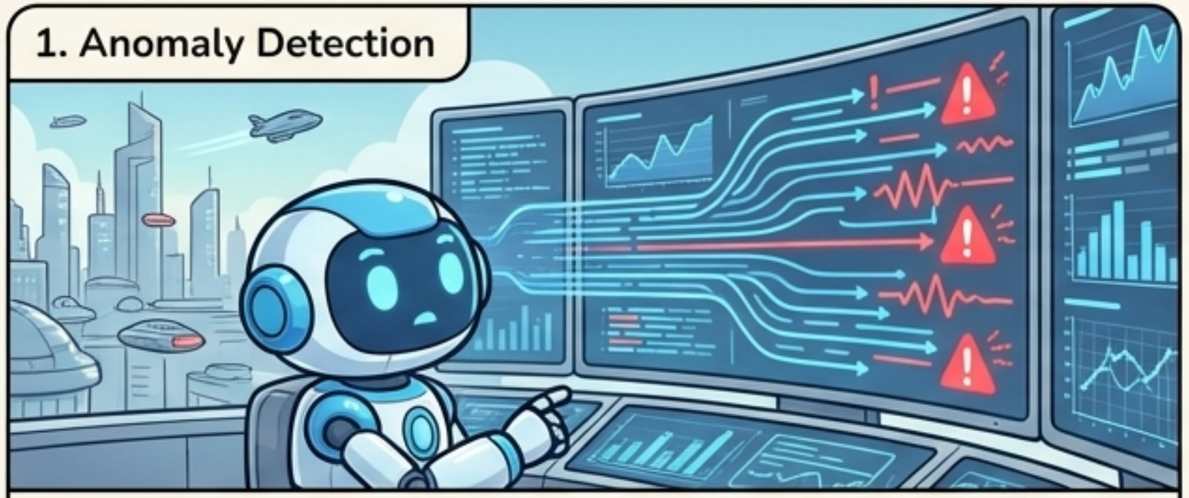


The Economic Advantage:

Persisting targeted subgraphs is far more efficient than storing all telemetry.

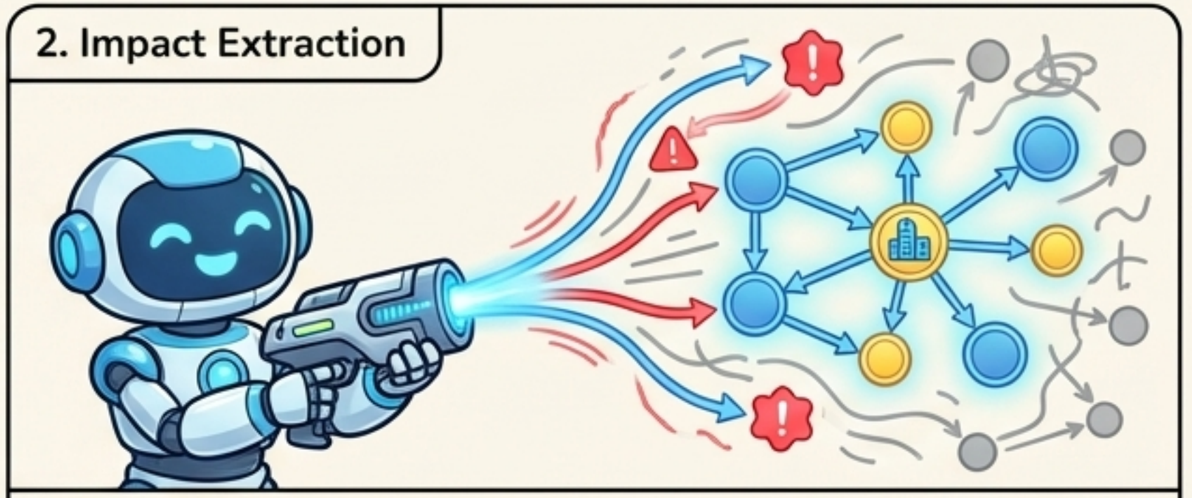
The Four Steps to Clarity

1. Anomaly Detection



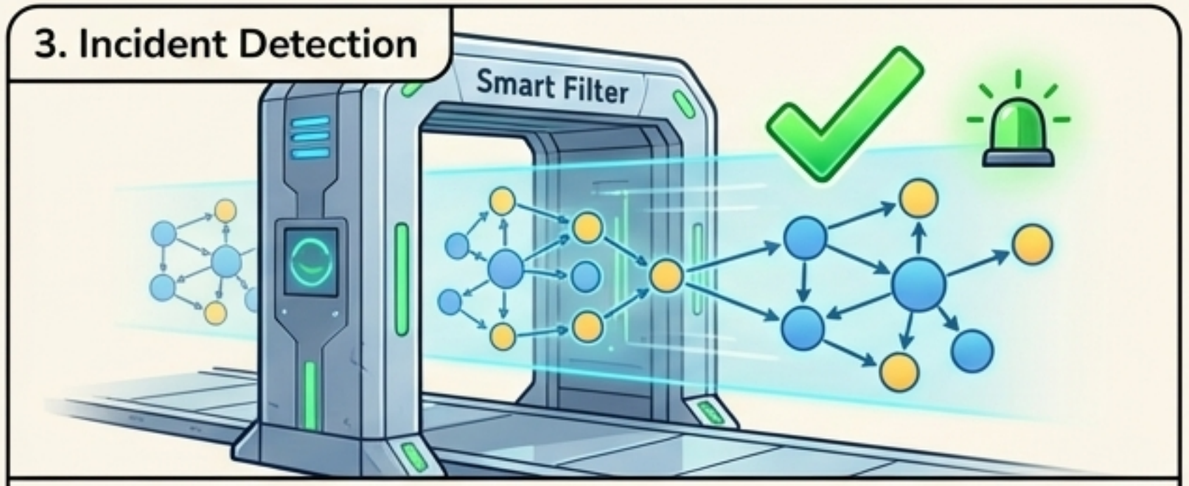
First, GEM identifies abnormal signals in real-time by comparing current telemetry to the same time yesterday (day-by-day difference).

2. Impact Extraction



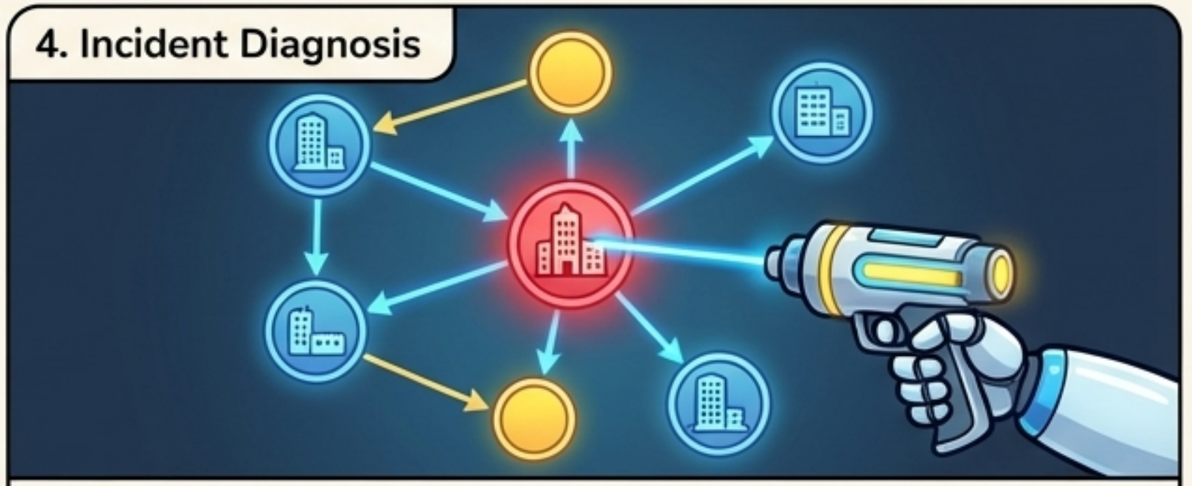
Next, related anomalies are aggregated into a coherent "Issue Impact Subgraph," filtering out all the noise.

3. Incident Detection



A graph neural network then analyzes the subgraph to determine if the issue is a true, user-impacting incident that needs immediate attention.

4. Incident Diagnosis



Finally, for confirmed incidents, an evolutionary PageRank algorithm diagnoses the root cause entity within the subgraph.

Deep Dive: Is This *Really* an Incident?

To determine an issue's severity, GEM's detection model analyzes the subgraph from two critical perspectives, just like an experienced engineer.

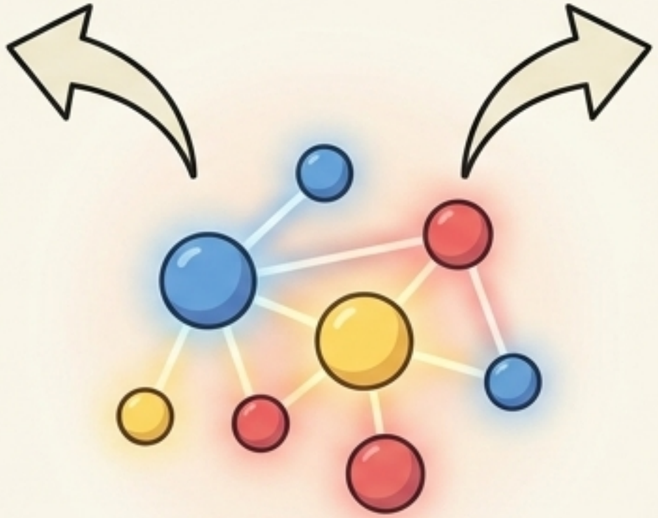
1. Affected Service Attributes (What is affected?)

Represented as **Node Features (X)** in the graph.

Information Used:

- Importance level, provided functionality, service role, fault tolerance strategy. These are pulled from the CMDB.

Example:
An issue affecting a critical "payment processing" service is more severe than one affecting an internal administrative tool.



Node Features Table

Abnormal Service	Fault Tolerance	Function	Importance Level	Service Role
frontend	Failover	Shopping	Important	appsvr
checkout	Failover	Shopping	Important	appsvr
cart	Failover	Shopping	Important	appsvr
...	...	...	...	...

X_{cart}: [0, 1, 0, ..., 0, 1, 0, ..., 1, 0, 0, ..., 0, 1 ..., 0, ...]

One-hot Encoding

2. Issue Symptoms (How badly is it affected?)

Represented as **Edge Features (E)** and **Global Features (G).**

Information Used:

- Failure Counts (FC), FC normalized by call count, and FC relative to historical baselines.

Example:
A high failure rate on a high-traffic service call is a strong indicator of an incident.

Edge Features Table

Abnormal call-relationship	Failure Count	Call Count	Failure Count (in Previous Day)
frontend->checkout	[..., 12]	[0]	[3]
frontend->cart	[..., 15]	97	192
checkout->cart	[..., 159	192	245]
...	...	...	...

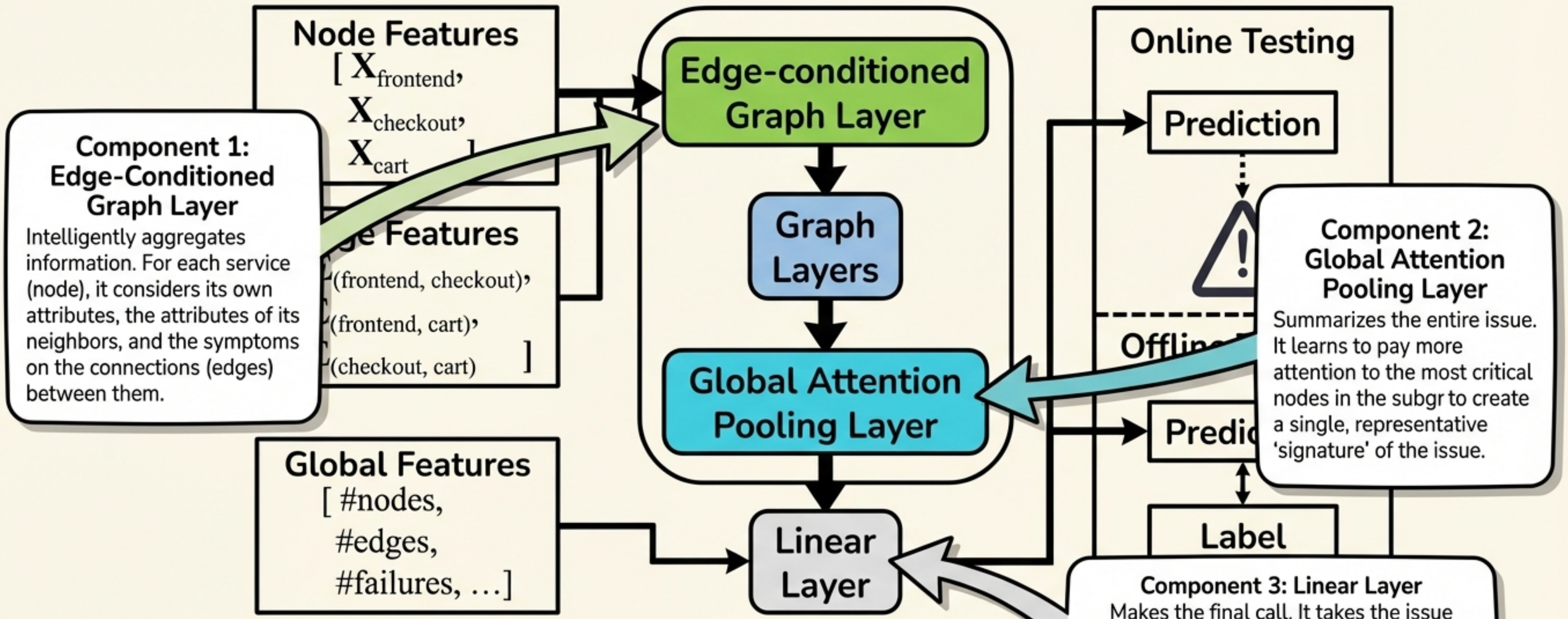
$$E_{(checkout, cart)}: \left[\frac{(245+\epsilon)}{(245+192+159)/3+\epsilon'}, ..., \frac{(245+\epsilon)}{(245+\epsilon')}, ..., \frac{(245+\epsilon)}{(0+\epsilon)}, ...\right], \epsilon=1$$

Compare Failure Count with its preceding values

Compare it with Call Count

Compare it with its value in previous day

The Detector's Brain: An Edge-Conditioned Graph Network



Key Takeaway: This model is designed to handle varying subgraph sizes and captures the complex interplay between *what* is affected and *how* it is affected.

Deep Dive: Pinpointing the Root Cause

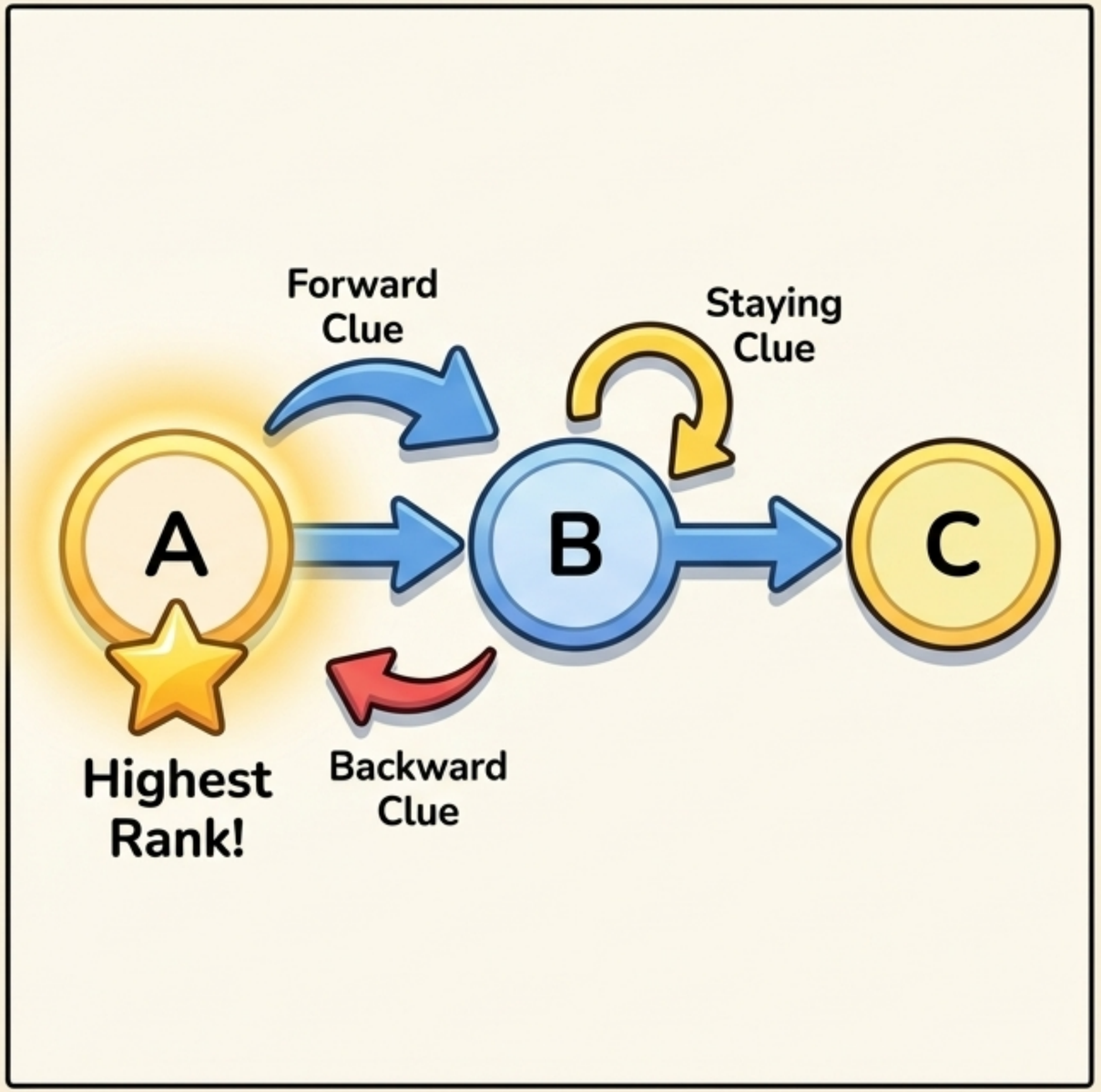
The Method: GEM simulates an operator's investigation using an **evolutionary PageRank algorithm**. The entity with the highest final "rank" is the most likely root cause.

How it Works: The algorithm navigates the impact subgraph, following **"Diagnosis Clues"** derived from telemetry data. The stronger the clue, the more likely the algorithm is to move in that direction.

Key Diagnosis Clues:

- **Forward Clues:** Move towards entities with more severe symptoms.
- **Staying Clues:** Remain at an entity if it is the epicenter of the symptoms.
- **Backward Clues:** Revisit a source entity if its siblings are also showing symptoms.

The Result: By balancing these clues, GEM identifies the entity that is the most probable origin of the fault propagation.

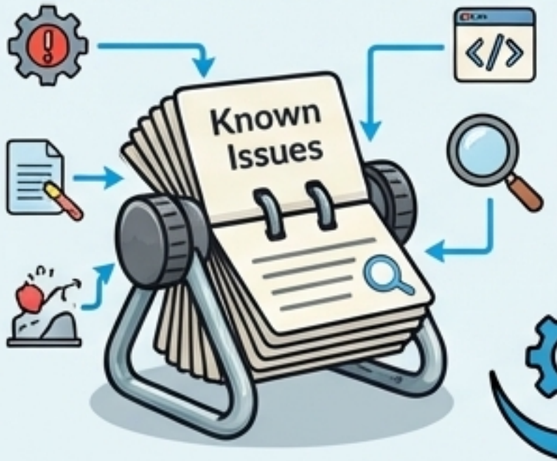


A Gadget That Gets Smarter Over Time

GEM is a **lifelong learning** framework. It's designed to work from day one and improve with every incident.

For Incident Detection

Cold Start



When incident data is scarce, GEM uses **Retrieval Augmented Detection**. It classifies new issues by comparing their "signature" to the nearest neighbors of known past incidents and non-incidents.

Continual Learning

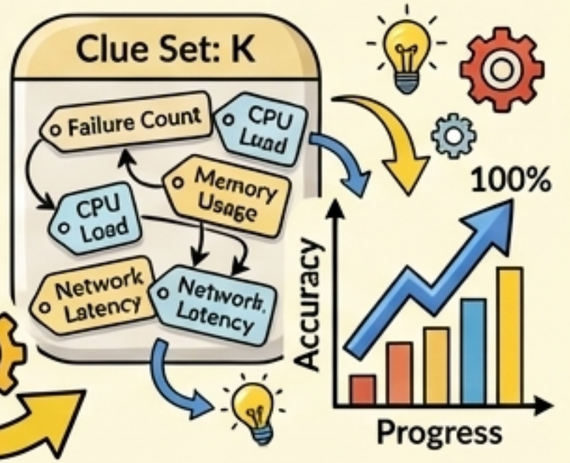
As more labeled data becomes available, the graph neural network is periodically retrained (e.g., every two weeks) to capture new patterns and improve accuracy.

For Incident Diagnosis

Cold Start

The model starts simple, using only **Failure Count** (FC) as the primary diagnosis clue.

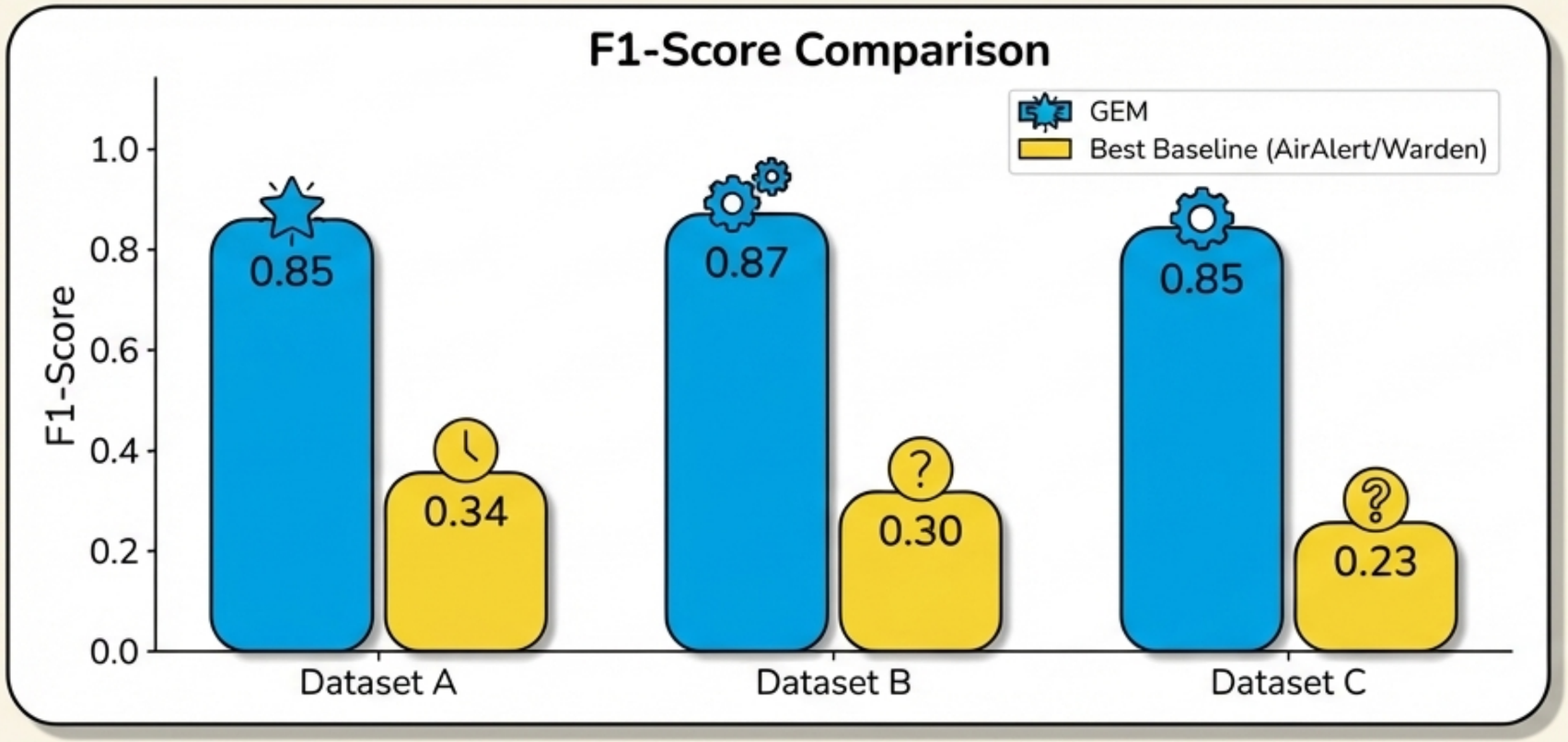
Continual Optimization



When a diagnosis is wrong, GEM automatically learns. It identifies the telemetry that would have best explained the true root cause and adds it to its set of clues (**K**). It then re-optimizes the strength of all clues (**α**) to improve future performance.

The Proof: Superior Incident Detection

GEM dramatically outperforms state-of-the-art baselines, improving the F1-Score by **150% to 270%**.



Cold Start Performance

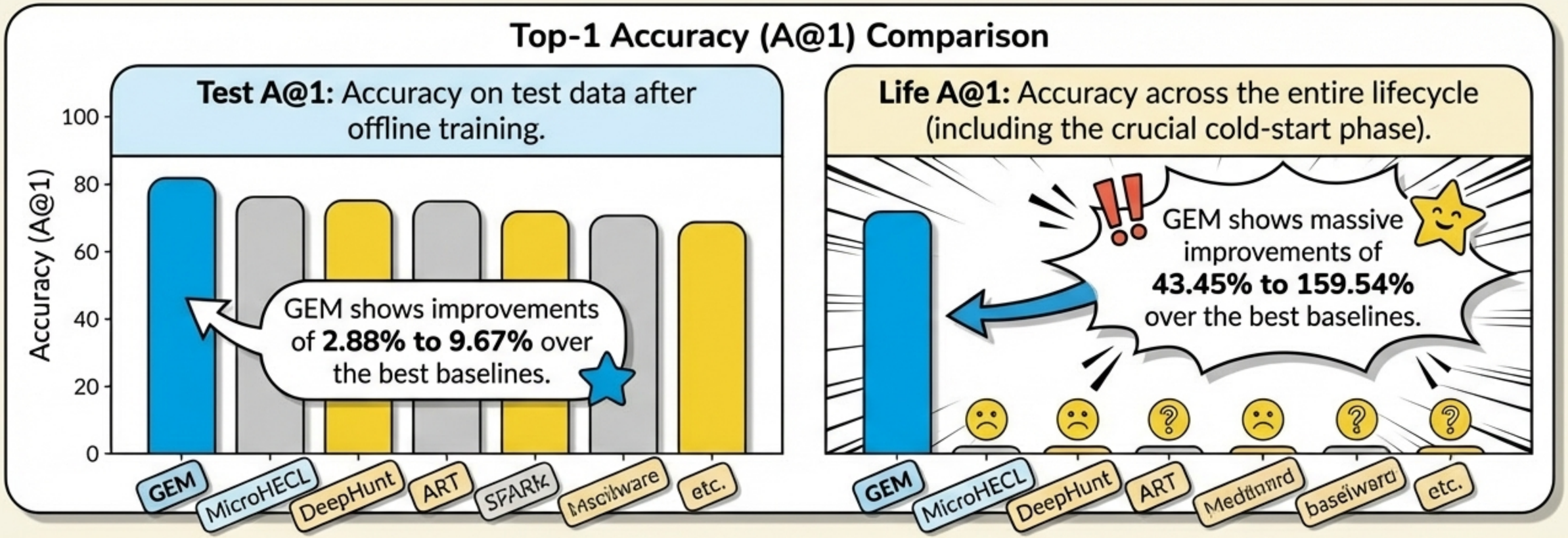
Even when starting with very limited data, GEM achieves **a strong F1-Score of 0.75**.

Traditional methods can't even start until a large dataset is collected.

By focusing on the targeted feature space within impact subgraphs, GEM achieves a level of accuracy that whole-system approaches cannot match.

The Proof: Unmatched Root Cause Diagnosis

GEM consistently achieves superior root cause localization accuracy, especially when considering the entire model lifecycle.



GEM's ability to cold start and learn online provides huge practical value. It delivers high accuracy from day one without requiring extensive, pre-collected training data, a major hurdle for other methods.

Saving the Day at WeChat

GEM has been in production at WeChat since July 2020.

Operator Feedback

"Before this approach, our team was constantly overwhelmed by alert storms... Now, we can focus on what truly matters."
— **Leader of the Operator Team**

"What impresses me most... is its ability to detect incidents before my configured alerting rules. In one particularly critical case, it identified a potential overload issue nearly 15 minutes before our configured alerts."
— **On-call Operator**

Quantified Impact

95%

Reduction in insignificant issues, filtering out the noise.

96%

Reduction in the root cause search space (average incident subgraph size is 26, only 1 is the root cause).

88%

of incidents were detected by GEM before an operator manually reported them.

Case Studies from the Field

Case I: The Hidden Overload

Problem

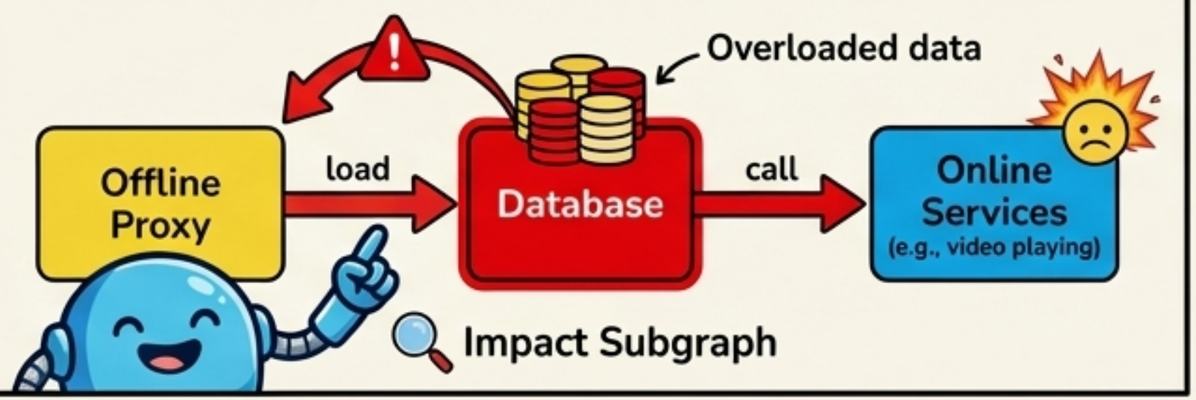
An offline proxy service overloaded a core database, affecting online services like video playing.

GEM's Action

GEM extracted an impact subgraph that clearly connected the offline proxy, the database, and the affected online services.

Result

The root cause was diagnosed as the database service. Operators quickly stopped the data loading operation to restore the system.



Case II: The Sneaky Program Bug

Problem

A bug in the error handling logic of a "retrieve service" caused it to crash when receiving bad input, impacting the system's recommendation function.

GEM's Action

GEM's impact subgraph pinpointed the crashing service and its dependencies.

Result

Operators identified the root cause, rolled back the input, and restarted the service to recover the system.