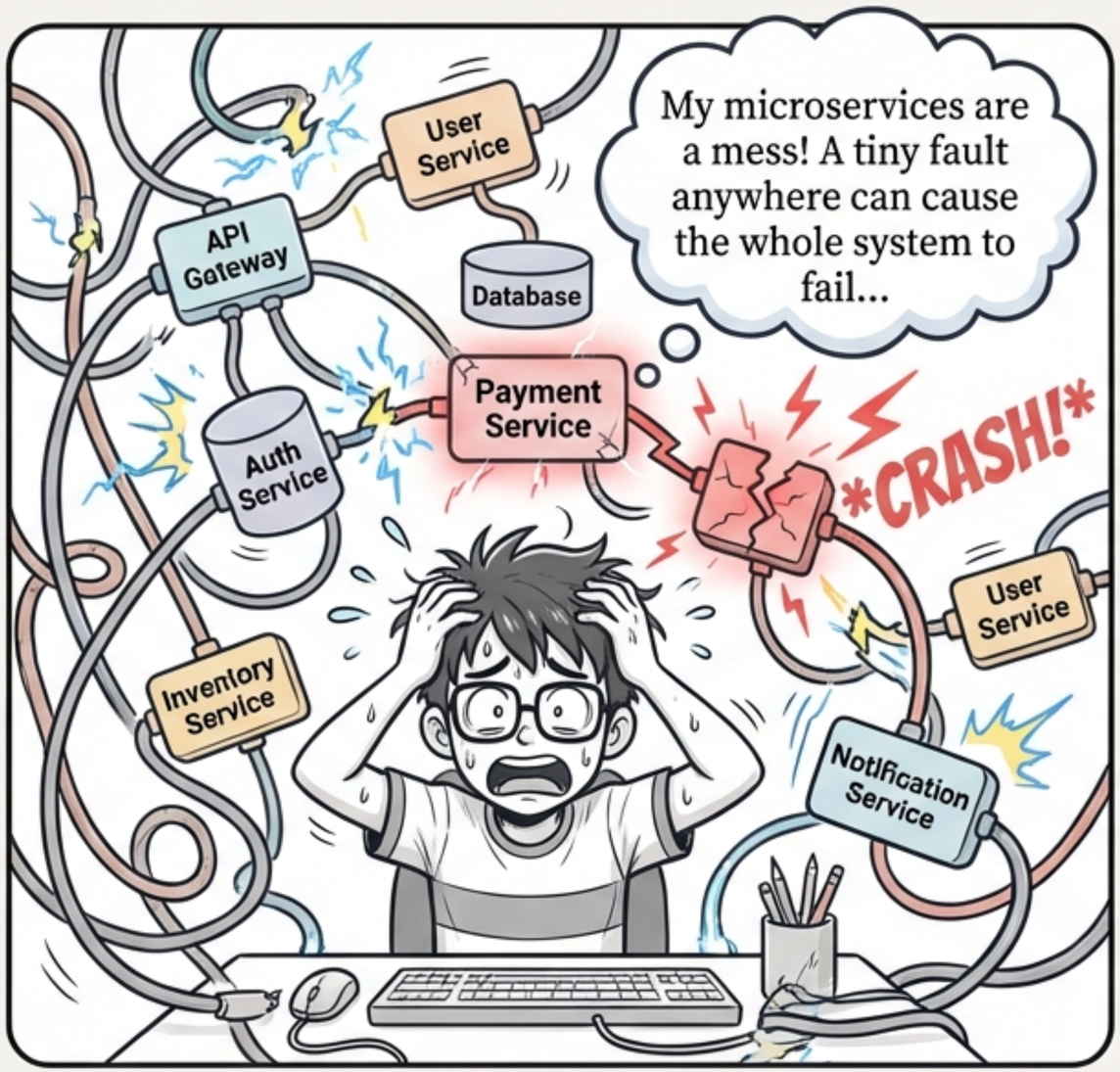
ChaosEater

ChaosEater: Building Resilient Software with a Little Help from the Future.

An LLM-Powered Agent that Fully Automates Chaos Engineering.

Based on the paper: *LLM-Powered Fully Automated Chaos Engineering: Towards Enabling Anyone to Build Resilient Software Systems at Low Cost* by Kikuta, Ikeuchi, & Tajiri (NTT, Inc.).

Modern systems are powerful, but incredibly fragile.

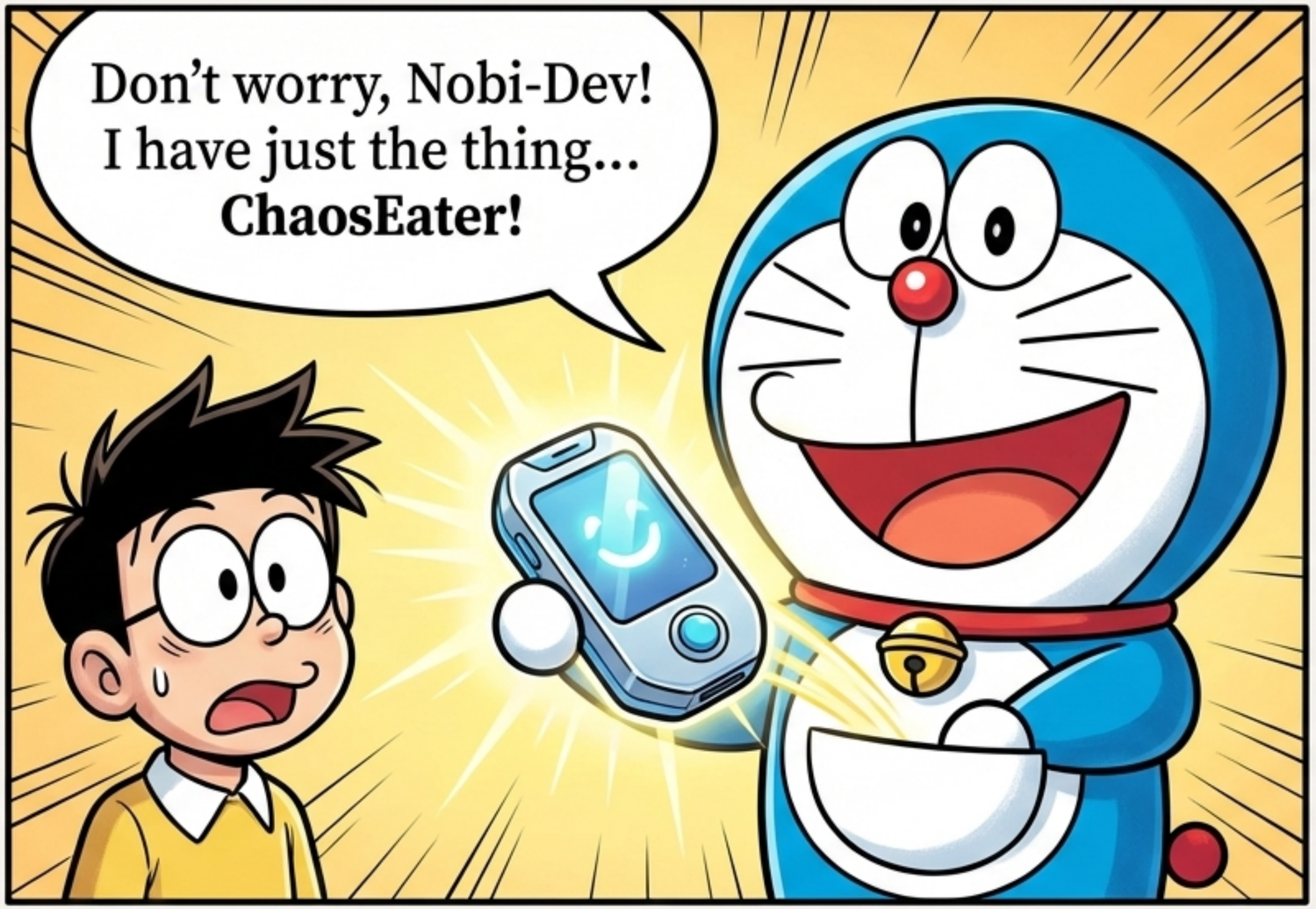


Modern software is built on complex distributed systems, where “even minor faults may lead to unpredictable and chaotic behavior.”

Chaos Engineering (CE) is the solution: intentionally injecting faults to find and fix weaknesses before they cause real failures.

The Problem: The key parts of Chaos Engineering—like forming a hypothesis and improving the system—are still manual. This process is “labor-intensive and require[s] multi-domain expertise,” making it slow and expensive.

What if you had a tool that could do it all for you?



Introducing **ChaosEater**:

An LLM-based system that automates the *entire* Chaos Engineering cycle.

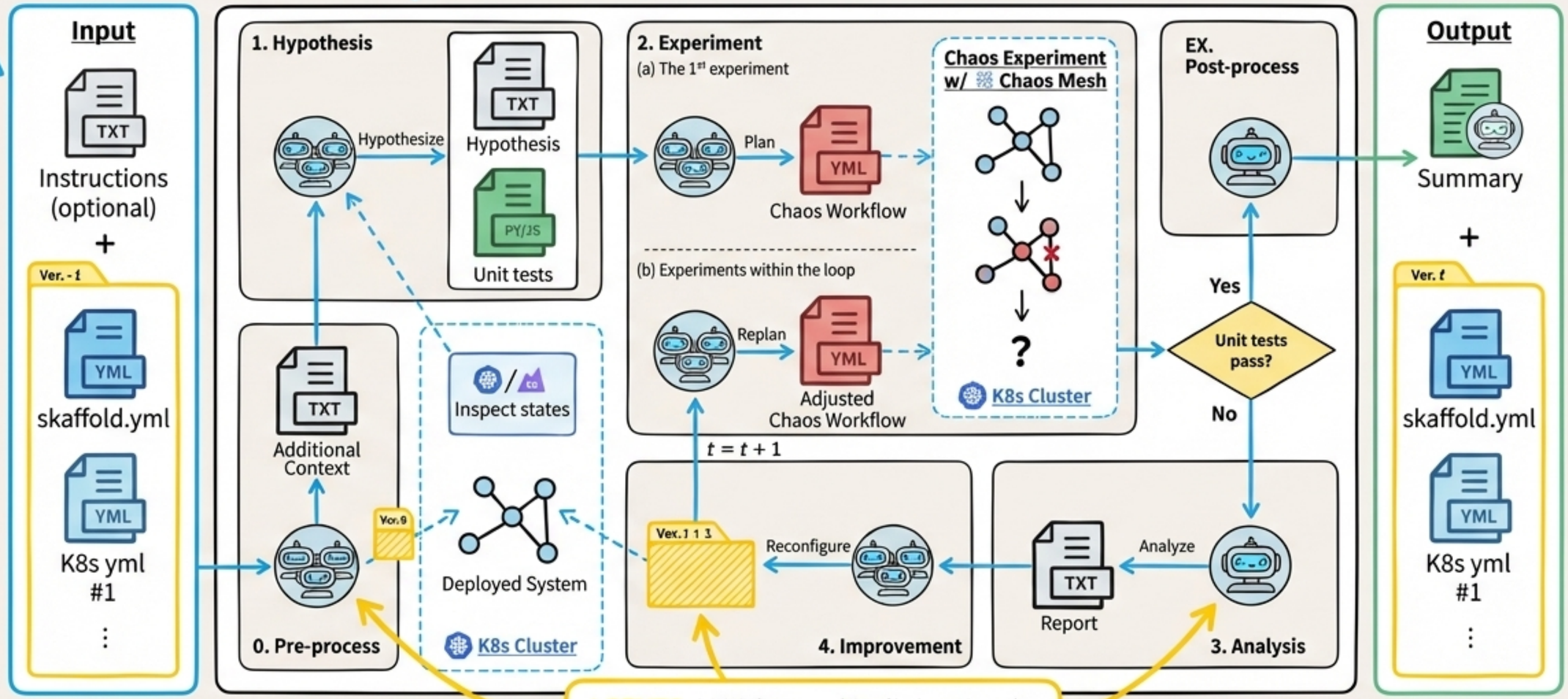
- It uses an “agentic workflow” where multiple LLM agents collaborate to complete the CE cycle for systems built on Kubernetes.
- Its goal is to “enable anyone to build resilient systems at low cost.”
- It works through software engineering tasks: requirement definition, code generation, testing, and debugging, all done automatically.

Here's the blueprint for how the ChaosEater gadget works.

INPUT: You provide your system's Kubernetes (K8s) files.

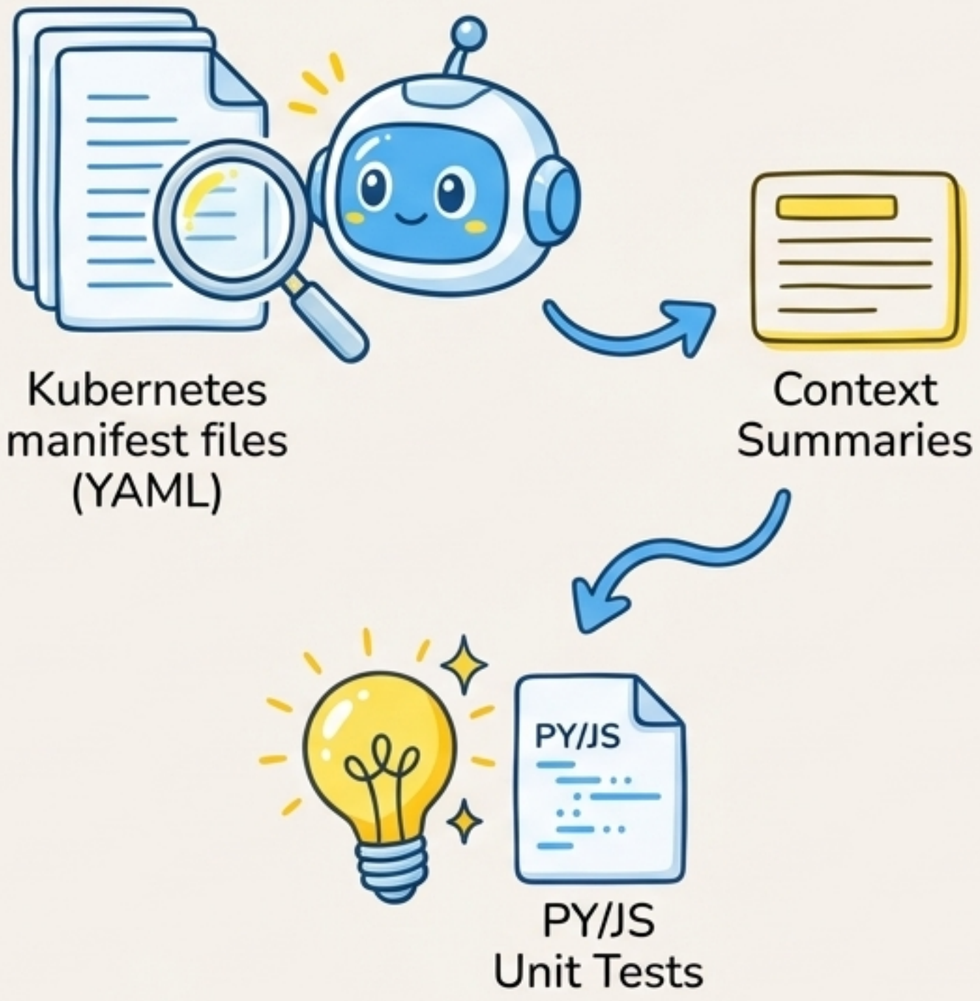
PROCESS: ChaosEater runs a six-phase cycle: Pre-process, Hypothesis, Experiment, Analysis, Improvement, and Post-process.

OUTPUT: It delivers a summary report and your new, improved K8s files—now more resilient!



AGENTS: LLM Agents (the little robots) are assigned specific tasks in each phase.

Step 1: The gadget scans your system and forms a hypothesis.



Phase 0: Pre-processing

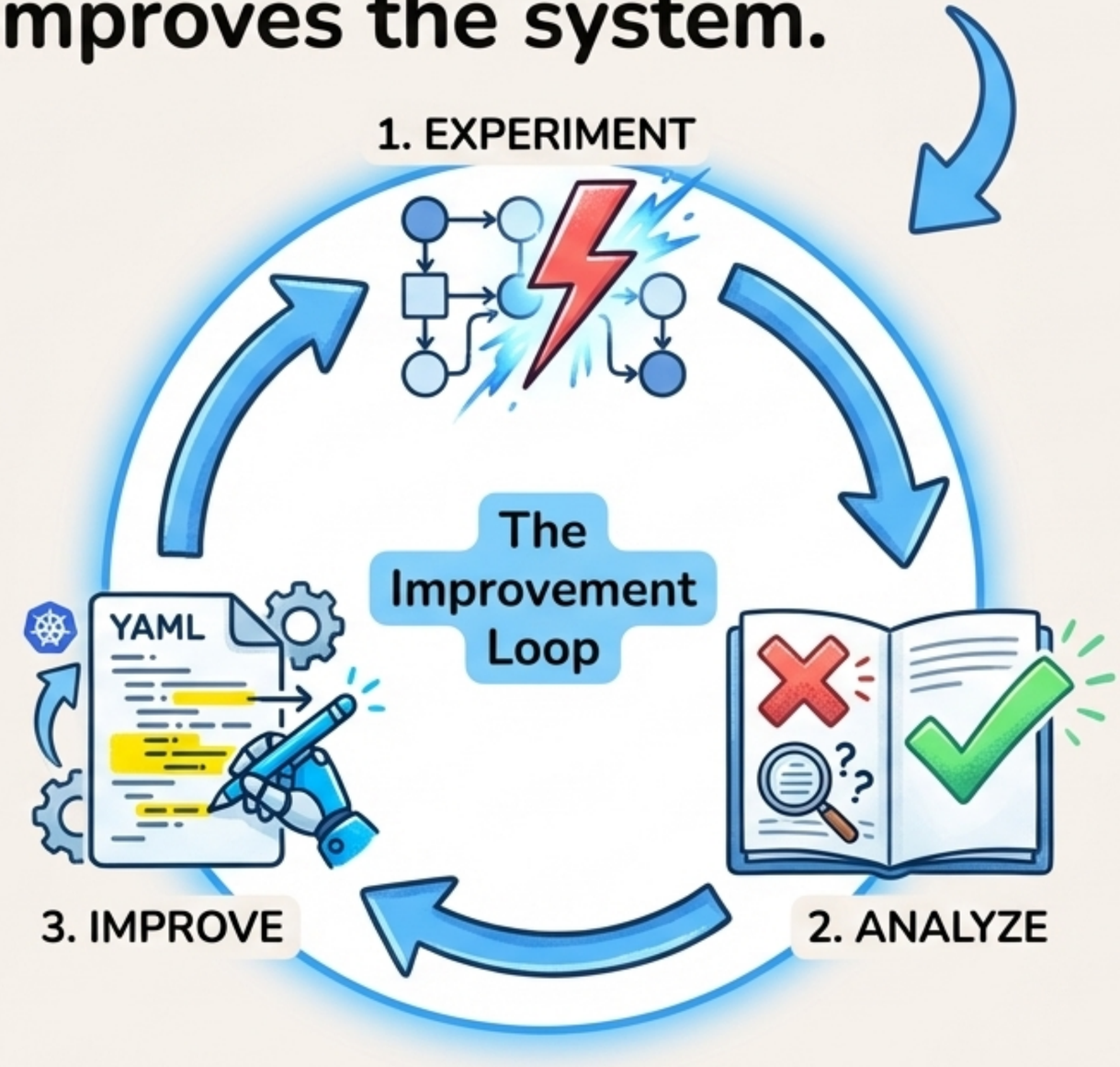
The gadget deploys your system and has LLM agents analyze it to understand its components and potential resilience issues.

Phase 1: Hypothesis

Here, an agent identifies the system's normal, healthy behavior (e.g., number of active pods, response time) to **Define Steady State**. Then, an agent proposes a **Define Failure scenario** (e.g., a traffic surge) and selects faults from Chaos Mesh to simulate it.

A key innovation is Validation as Code (VaC). An LLM agent writes a unit test script to automatically and transparently validate if the system's steady state is maintained during the experiment. The hypothesis is simple: "all VaC scripts pass, even when the defined faults are injected."

Step 2: It runs the experiment and automatically improves the system.



Phase 2: Experiment

An LLM agent plans the chaos experiment and generates a Chaos Mesh workflow file. The experiment is executed automatically.

Phase 3: Analysis

If any VaC script fails, an LLM agent analyzes the logs, identifies the root cause, and generates a report.

Phase 4: Improvement

Based on the analysis, an agent reconfigures the K8s manifests to fix the weakness (e.g., increasing redundancy).

The Improvement Loop

ChaosEater repeats the Experiment, Analysis, and Improvement phases until all VaC scripts pass and the hypothesis is satisfied.

Step 3: You get a resilient system and a full report.

Extra Phase: Post-processing:
Once the CE cycle is complete, an LLM agent writes a final summary of the entire process.

- Your Deliverables:
  1. **Summary Report:** Details the user's input, the hypothesis tested, the weaknesses found, and the improvements made.
  2. **Reconfigured Manifests:** The updated Kubernetes files, now hardened against the tested failure scenario, ready for deployment.

But does it actually work? Let’s test it on real systems.

To evaluate ChaosEater, we ran it on two systems with intentionally configured weaknesses. The only instruction given to the AI was to keep experiments under one minute.

Case 1: NGINX

A minimal system (2 K8s manifests).

⚠ `restartPolicy` was set to `Never`, meaning if the server pod failed, it would stay down, causing an outage.

Case 2: SOCKSHOP

A large-scale e-commerce system (29 manifests).

⚠ The `front-end` service had only one replica, creating a single point of failure.

The results are in: It’s fast, affordable, and stable.

NGINX

Time: 11 minutes
API Cost: $0.21

SOCKSHOP

Time: 25 minutes
API Cost: $0.84

Single CE Cycle, using gpt-4o

Key Takeaway

Even with the number of resources increasing by more than ten times from NGINX to SOCKSHOP, the cost increase **remains minimal**, demonstrating low costs even for large-scale systems.

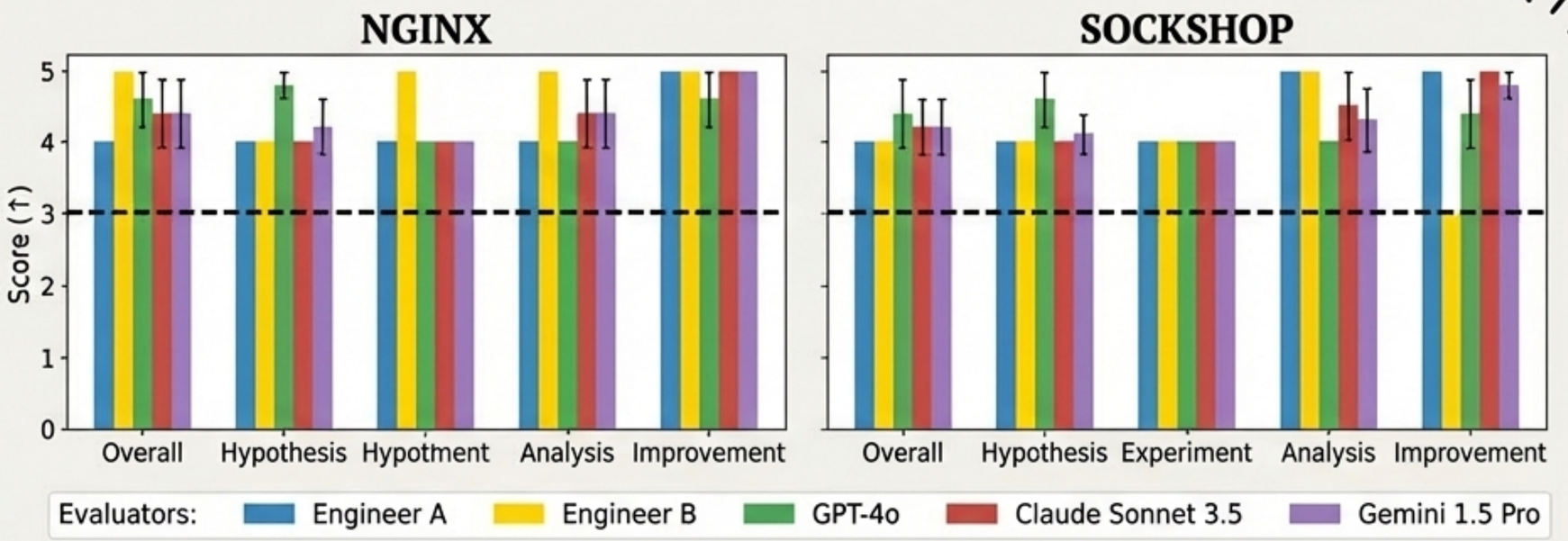
Stability

ChaosEater successfully completed the CE cycle in all 5 runs for both systems and correctly reconfigured NGINX (5/5 times) and SOCKSHOP (4/5 times).

Human engineers and other AIs agree: The results are solid.

The completed CE cycles were evaluated on a 5-point scale (where 3+ is a positive rating).

The Evaluators: Two external human engineers and three different LLMs (GPT-4o, Claude Sonnet 3.5, Gemini 1.5 Pro).



The Verdict

"All evaluators rated every phase above the threshold for a positive rating for both systems, demonstrating that ChaosEater completed reasonable single CE cycles."

The Specifics

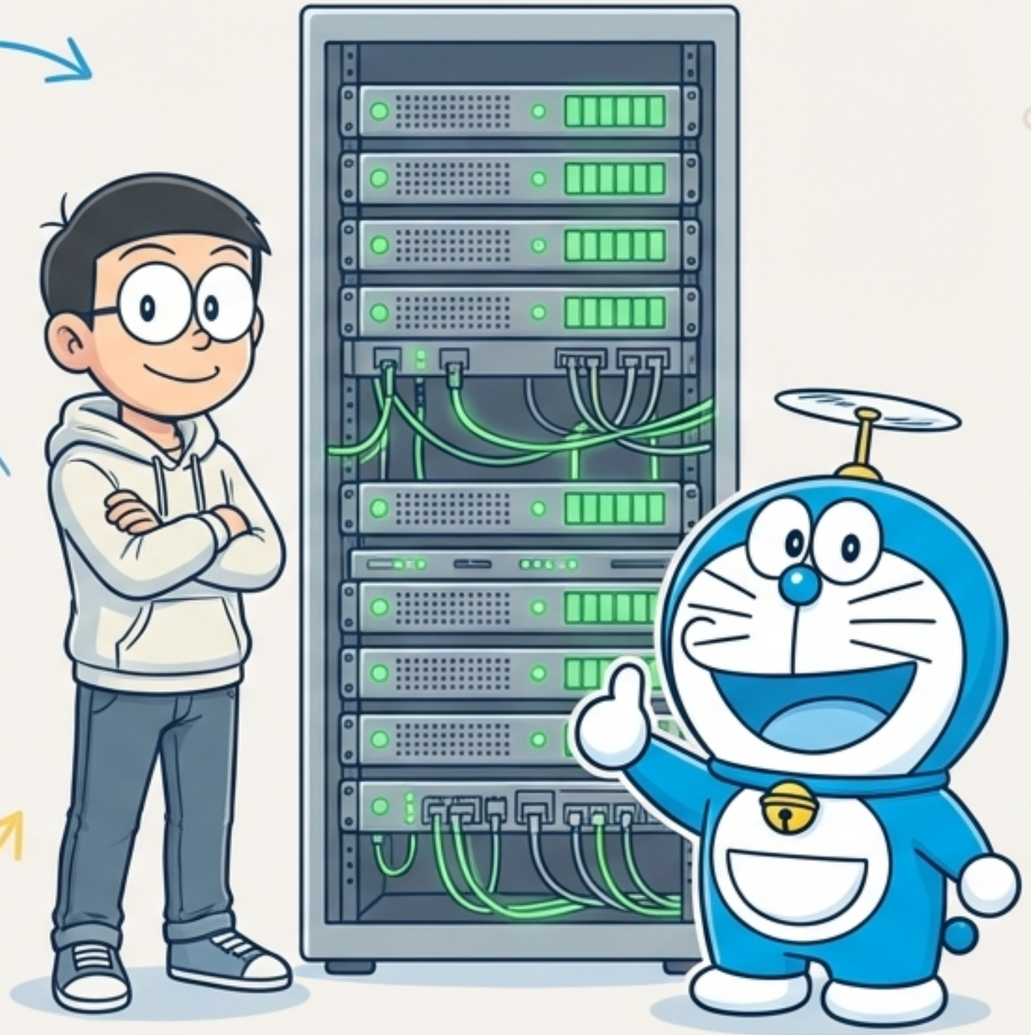
In the evaluated runs, ChaosEater correctly defined Pod availability as a steady state, hypothesized failures like cyberattacks, and "successfully identified the issues of `restartPolicy` and the number of replicas, and solved them."

Chaos Engineering is now for everyone.

A Key Breakthrough: “We are the first to implement an LLM-based system that **automates** the ***entire*** CE cycle.”

This provides concrete “**evidence for the feasibility** of a new direction: **the full automation of system resilience** improvement.”

This suggests “a future where **anyone** can build **resilient systems** at **low cost**.”



What's next? Upgrading ChaosEater for tougher challenges.

The current ChaosEater has limitations, which point to exciting future work:

Future Upgrade 1 (Safety First)

Control the impact of faults in production environments, safeguarded by higher-level monitoring.

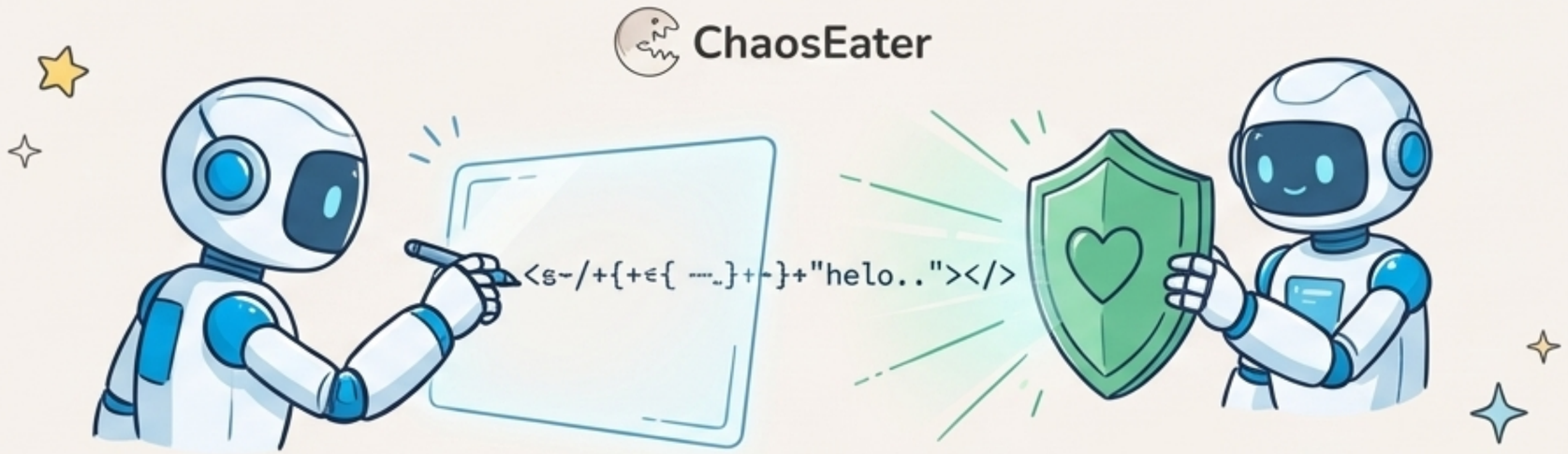
Future Upgrade 2 (Full Stack Resilience)

Expand beyond K8s manifests to also reconfigure application code (Python), frontend code (JS, HTML), and infrastructure code (Terraform).

Future Upgrade 3 (Smarter Exploration)

Enable 'long-horizon vulnerability exploration' where agents learn from previous cycles to find deeper, more complex issues over time.

The future of resilience isn’t manual. It’s automated.



"As the automatic generation of software applications by LLMs has become widespread recently, such automation is *be*coming even more important for ensuring the resilience of the generated applications."
ChaosEater is Open Source": "We release all the resources of ChaosEater. This release provides a concrete foundation for subsequent works in the new direction."

Explore the project on GitHub: https://github.com/ntt-dkiku/chaos-eater