AURORALABS

Solution Brief

Line-of-Code Intelligence for Compiled Binary Files: Bringing Observability-Level Insights to Static Analysis

Published Q1 2025

Executive Summary

- Organizations building performance-critical software (such as networking infrastructure or automotive systems) face strict demands for reliability, speed, and cost-effectiveness.
- Traditional static analysis tools and real-time observability platforms each provide partial coverage, but a gap remains in identifying subtle hardware- and runtime-related performance issues before software is deployed.
- LOCI bridges this divide by delivering real-time observability insights directly from compiled binaries, leveraging Aurora Labs' Large Code Language Model (LCLM),
- With LOCI, teams can address issues proactively, reduce mean time to resolution (MTTR), and accelerate continuous delivery.

The Challenge:

When Performance is Mission-Critical, Existing Observability Tools Fall Short

What is performance-critical software?

Software performance and reliability are always important. But for certain categories of software, they become absolutely critical. If a timetracking app goes down, a few thousand users get frustrated; the same outage at a data center can impact hundreds of companies and millions of end users. In some cases – such as the software powering driver-assist software in modern cars - even a millisecond-level delay can put lives at risk.

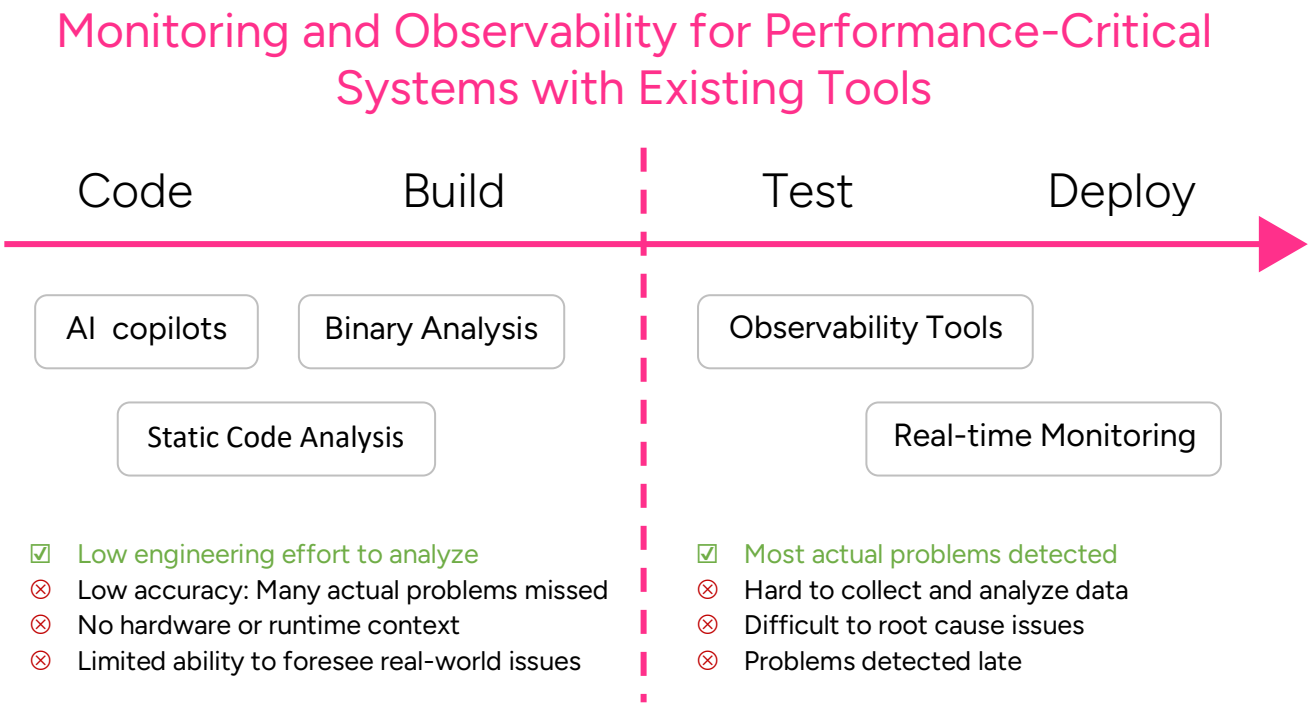
Edge, embedded, and IoT software are often performance-critical.
Examples include software that runs on networking infrastructure, sensors, or industrial control systems. This type of code has several common characteristics:

- It runs on-device, often in unpredictable conditions (rather than in the public cloud)
- Software performance is closely coupled with the hardware it runs on
- Code is written in a compiled language such as C or C++

In these cases, performance is mission-critical, and prolonged downtime can be an existential risk to the business. Development and SRE teams apply multiple layers of testing, monitoring, and control to ensure a consistently high level of performance and reliability. And yet, as we explain next, the reality is that current solutions struggle to catch problems quickly enough in the development lifecycle.

Understanding the observability dilemma

Currently, there are two types of solutions used to identify and resolve performance issues: pre-build checks (based on static analysis of code or binaries) and post-deployment observability. Neither approach fully addresses the need for early, actionable insights that can prevent performance degradations before they impact production.



Pre-build: static analysis and AI copilots

This category of tools analyzes source code, without executing it. **Static code analysis** scans source code to identify potential issues like memory leaks, buffer overflows, and inefficient algorithms. **Binary analysis** takes this a step further by running similar tests on compiled code. In recent years, teams have also started using **LLM-based copilots**, trained on vast repositories of code, to provide real-time suggestions and catch common programming antipatterns.

> Static analysis tools provide rapid insight at minimal time and cost - but they rarely tell the full story of code's real-world performance.

All of these techniques are valuable for **identifying basic structural and algorithmic issues** early in the development process. They are capable of catching code issues that will ultimately have an impact on reliability and performance - such as outdated libraries, inefficient algorithms, potential memory leaks, and security vulnerabilities. What's more, these tools have the advantage of being **easy to run**, as they do not require the code to be deployed in production (i.e., on the actual hardware) or in a production-like environment; and they provide near-instant insights to developers, moments after uploading the relevant codebase or repo.

However, **this comes at the expense of realism and context.** At the end of the day, the performance that matters is the compiled binary, running in real-world conditions on the actual hardware it is intended to run on. Tools that look strictly at code lack this context, and thus will be unable to predict many of the real-world issues which can degrade performance. For example, they cannot tell you how code will perform when dealing with hardware-specific issues like CPU throttling, memory constraints, or temperature-related degradation.

Post-build: Observability and Real-Time Monitoring Tools

Observability tools are used to collect logs, metrics, and traces in real-world production or test environments. They can help reveal how the software actually behaves under various workloads and conditions, identify outages, and provide the data foundation for further investigation or incident response.

> Observability offers comprehensive monitoring but requires a significant operational lift and can only catch problems after the fact.

The advantage of observability is that it **deals with real data**, collected during runtime in the field. This means developers have an accurate picture of **what** is happening - i.e., when a performance degradation occurs. However, when it comes to understanding the **why** and actually resolving the issues, things get trickier:

- **Data collection:** Observability tools collect massive amounts of telemetry, which is often expensive to store and complicated to analyze. And despite the deluge of logs they generate, they can still miss the important parts. Observability tools will often employ interval-based sampling to reduce the event stream to manageable levels, which can lead to crucial data being left out.
- **Root-causing:** At runtime, many things can go wrong. Root causing the issue to the code level - and especially to a specific part of the code - requires significant time and effort. Developers need to sift through telemetry, logs, and user reports to isolate root causes. They then have to test different hypotheses to try and identify which lines of code are responsible for the issue, further delaying the actual fix.
- **Problems caught late in the process**: When issues such as latency spikes, memory leaks, or CPU overutilization surface only after deployment, they can cause greater operational disruption and require 'all hands on deck' for large-scale patching or rollback. This can delay release cycles, degrade user experience, or have a significant commercial impact.
- **Missing hardware context**: Popular observability tools are general-purpose and not designed for the specific quirks of performance-critical systems, such as the impact of running software on different chipsets or SOC (System-on-Chip) cores.

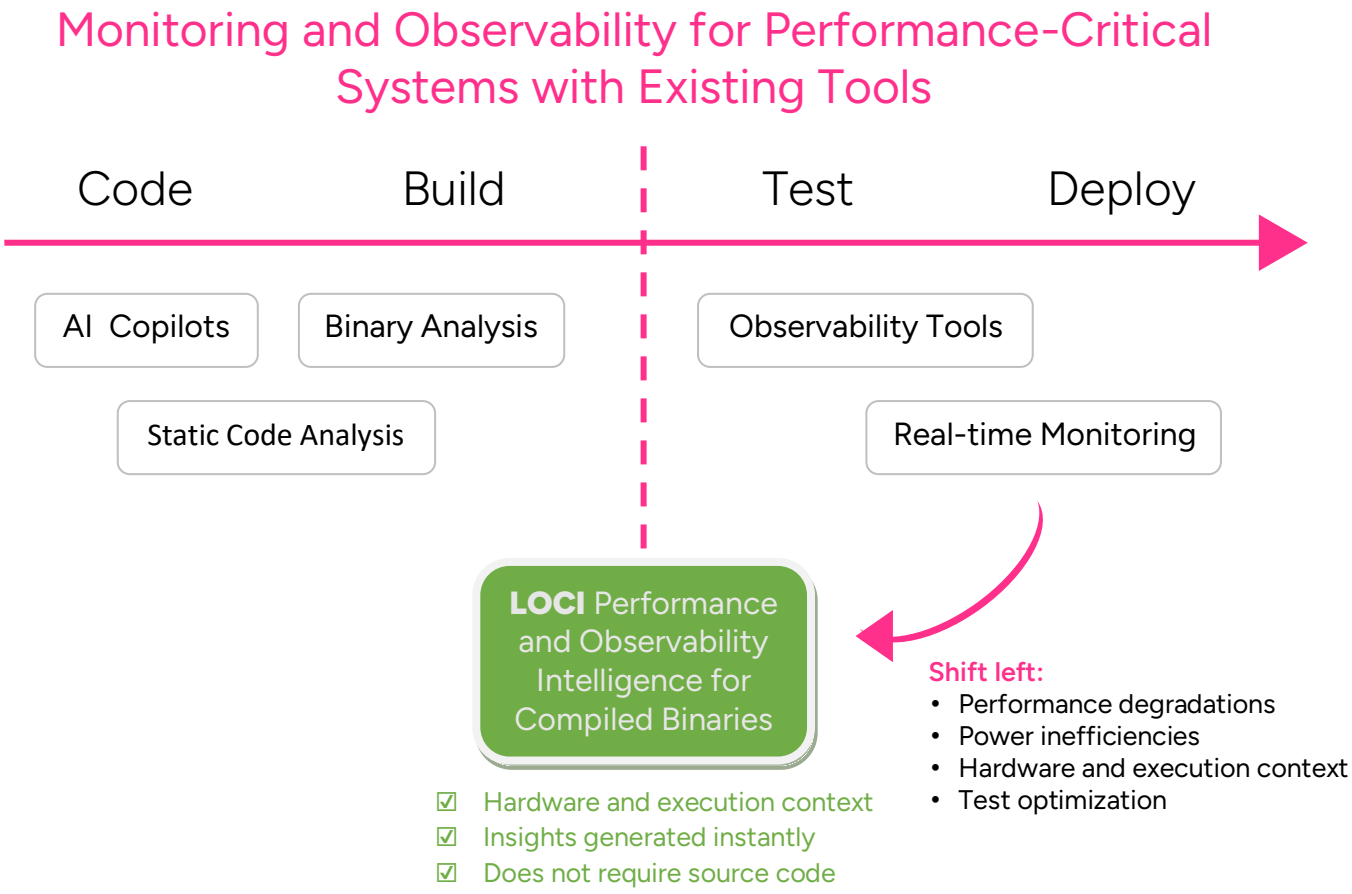
The end result is a significantly higher Mean Time To Resolution (MTTR) for performance issues. Late detection, complicated troubleshooting, and high costs of log analysis all combine to slow development velocity and increase the risks of unreliable software being deployed in mission-critical contexts.

The Solution:

Bridging the Gap Between Static and Dynamic Analysis with AI-Based Performance Insights for Compiled Binaries

How, then, can you bridge the gap between rapid insights delivered by static analysis and the comprehensive, realistic coverage delivered by observability tools? Advancements in AI offer a way forward.

Lines of Code Intelligence (LOCI) by Aurora Labs introduces a new layer of reliability and observability at the level of compiled binaries. Trained on years of data generated by mission-critical safety projects, LOCI is an AI model that provides observability-level insights on static binary files - instantly, and without conducting lengthy testing processes in order to collect data from the field.



LOCI goes beyond static analysis and understands the context of the software behavior within a given functional flow, detecting anomalies and deviations from expected behavior before the software is deployed to production.

By analyzing the binary based on a massive corpus of low-level code deployed on different hardware configurations, LOCI offers a far more realistic simulation compared to traditional static analysis; at the same time, it eliminates the complexity of observability at the edge by providing instant insights with no engineering overhead. As we shall explain below, the insights it generates both help prevent problems before they occur, and significantly simplify efforts to root cause issues at runtime.

Examples of insights that only LOCI can provide

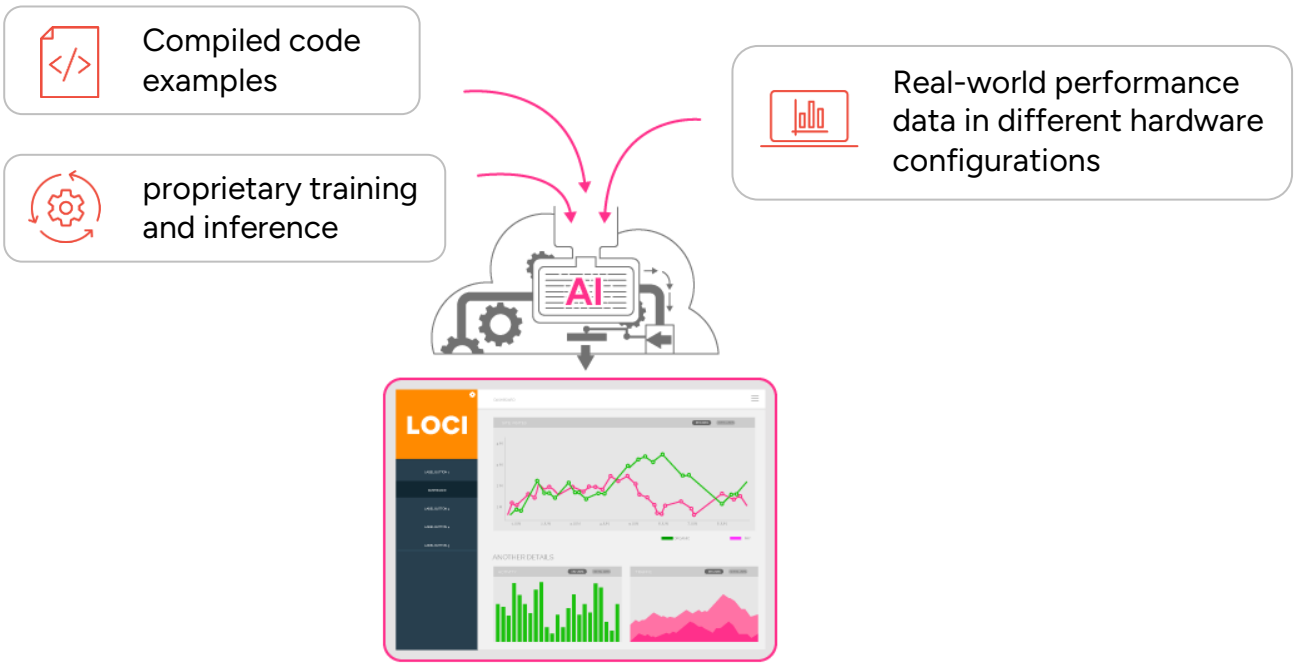
- How specific code sections and functions impact hardware, Energy, temperature and performance.
- Performance changes between different software versions, without source code.
- Predict power-hungry code and functions, pre-deployment

A quick look under the hood

LOCI is built on Aurora Labs' proprietary Large Code Language Model (LCLM) – a vertical language model that is specifically designed for compiled binaries, and which runs fully securely and without requiring the original source code. Unlike general-purpose Large Language Models (LLMs), the Aurora LCLM delivers superior, efficient, and accurate binary analysis and detection of software behavior changes, offering deep contextual insights into system-wide impacts.

LOCI was trained and optimized on binary files and years of real-world performance data, allowing it to accurately analyze complex software performance based on the binary alone and without requiring the source code. Based on a single software artefact, LOCI can predict performance issues at 97% accuracy.

LOCI by Aurora Labs: Proprietary Language Model for Binary Analysis



AURORALABS

Compared to most popular LLMs, LOCI's vocabulary is more productive and efficient because it is x1000 smaller. It was designed with Aurora Labs' homegrown tokenizers and GPU-efficient methodologies, which keeps inference costs low, even for larger artifacts.

Key Capabilities: Shifting Observability Left

- **Full system behavior analysis:** LOCI provides comprehensive analysis of the entire software system by analyzing compiled binaries at the symbol level, without requiring access to source code.
- **Performance degradation prediction:** LOCI can predict time and performance degradation across different software versions, allowing for accurate root cause analysis. This increases system reliability and quality, ensuring functionality uptime and availability for users.
- **Granular telemetry**: Provides breakdown of specific function calls, allowing engineers to pinpoint issues to the line-of-code level, based only on a single binary file.
- **Contextualized system overview**: LOCI can instantly map the hardware and software components that an executable is meant to run on, providing accessible visual insights.
- **Hardware-specific analysis**: Understands differences between CPU-bound versus hardware-accelerated operation to enable workload placement.

Engineering efficiency benefits

- **Observability-level insights before the observability hassle**

  LOCI enables teams to identify potential performance issues immediately after build, rather than waiting for problems to surface in testing or production – helping teams address problems earlier in the development cycle, when fixes are less costly and disruptive. This includes insights about system behavior, performance bottlenecks, and power consumption patterns that would previously only be available through extensive runtime monitoring.

- **Reducing cloud infrastructure and testing costs**

  LOCI significantly reduces cloud infrastructure costs across multiple dimensions. Teams spend less time running and re-running automated tests, and require fewer debug-deploy-test iteration cycles – which translates into reduced CPU usage in CI/CD pipelines and test environments. Additionally, by minimizing the number of problematic deployments that need to be rolled back and redeployed, LOCI helps organizations avoid the compound cloud costs associated with maintaining parallel environments and running extensive post-deployment diagnostics.

- **Performance analysis when source code is not available**

  Engineering teams often work with third-party software components (such as automotive OEMs or chip manufacturers) where source code isn't available. LOCI enables teams to analyze performance characteristics and potential issues using only the compiled binary files. This helps to pinpoint performance issues in complex systems, and can help allocate responsibilities for solving problems between partnered companies.

- **Test strategy optimization**

  Based on behavioral analysis, LOCI indicates which functions to focus on during unit and system tests. This increases the coverage of system test scenarios and enhances functional quality.

Optimizing Safety-Critical Software Testing for a Global Manufacturing Company

Challenge

A Tier-1 automotive supplier faced significant challenges with lengthy testing cycles for their Electric Power Steering (EPS) controller. This led to

- Delayed product development timelines
- Increased engineering costs
- Product recalls

Solution

LOCI was integrated directly into the supplier's development pipeline, which is deployed in a private cloud on Amazon Web Services. LOCI analyzes the compiled binaries after each build, providing immediate insights about potential performance issues and test coverage gaps, along with early defect detection.

Results

- Engineering efficiency: Significant reduction in manual review requirements through automated analysis
- Test coverage: Achieved 100% functional coverage, up from partial coverage in the previous workflow
- Quality: Early detection of potential defects before deployment to production systems
- Development velocity: Reduced testing cycles by catching issues immediately after build

---

Enabling Real-Time Performance Monitoring at the Silicon Level

Challenge

A leading semiconductor manufacturer needed to detect real-time anomalies on their hardware, while ensuring minimal impact on system resources and computing power.

Solution

LOCI was integrated directly with the manufacturer's Performance Processing Unit (PPU) software, allowing developers to detect timing deviations at the line-of-code level and correlate performance with hardware temperature.

Results

- Minimal resource impact: The monitoring solution consumed only 0.3% of PPU resources
- Hardware insights: Successfully identified performance degradation patterns correlated with temperature increases

AURORALABS

Insights Speak Louder Than Words: See LOCI in Action

LOCI does not require any agents or access to your internal systems. You simply upload a binary file to receive specific contextual insights.

Currently, only C and C++ software running on ARM / AURIX processors is supported.

For a free demo of LOCI's capabilities on your actual artifacts, contact Aurora Labs at info@auroralabs.com.email us: info@auroralabs.com

www.loci-dev.com

www.auroralabs.com

Follow us