From Technical Excellence to Practical Adoption: Lessons Learned Building an ML-Enhanced Trace Analysis Tool

Kaveh Shahedi*, Matthew Khouzam†, Heng Li*, Maxime Lamothe*, Foutse Khomh*
*Polytechnique Montréal, Montréal, Canada
†Ericsson AB, Montréal, Canada

***Abstract*—System tracing has become essential for understanding complex software behavior in modern systems, yet sophisticated trace analysis tools face significant adoption gaps in industrial settings. Through a year-long collaboration with Ericsson Montréal, developing TMLL (Trace-Server Machine Learning Library, now in the Eclipse Foundation), we investigated barriers to trace analysis adoption. Contrary to assumptions about complexity or automation needs, practitioners struggled with translating expert knowledge into actionable insights, integrating analysis into their workflows, and trusting automated results that they could not validate. We identified what we called the *Excellence Paradox*: technical excellence can actively impede adoption when conflicting with usability, transparency, and practitioner trust. TMLL addresses this through an adoption-focused design that embeds expert knowledge in interfaces, provides transparent explanations, and enables incremental adoption. Validation through Ericsson's experts' feedback, Eclipse Foundation's integration, and a survey of 40 industry and academic professionals revealed consistent patterns: survey results showed that 77.5% prioritize quality and trust in results over technical sophistication, while 67.5% prefer semi-automated analysis with user control, findings supported by qualitative feedback from industrial collaboration and external peer review. Results validate three core principles: cognitive compatibility, embedded expertise, and transparency-based trust. This challenges conventional capability-focused tool development, demonstrating that sustainable adoption requires reorientation toward adoption-focused design with actionable implications for automated software engineering tools.**

I. Introduction

Software systems operating across distributed architectures and multi-core processors demand sophisticated performance analysis capabilities. System tracing has emerged as a powerful technique for capturing detailed runtime behavior, providing detailed visibility into system execution [1]–[3]. While automated software engineering has revolutionized many aspects of software development, from code generation to testing, the automation of dynamic performance analysis remains a critical gap in industrial practice [4]–[6]. Despite sophisticated tracing tools [1], [7]–[12] and established analytical techniques [13]–[15], practitioners struggle to leverage trace data effectively for performance analysis in industrial settings.

This paper presents our systematic investigation of trace analysis adoption barriers through a year-long industrial collaboration with Ericsson Montréal and the development of TMLL (Trace Server Machine Learning Library)[1]. Rather than a purely technical contribution, we focus on empirical insights about the misalignment between academic assumptions regarding automated tool adoption and the reality faced by practitioners in industrial environments. TMLL's integration into the Eclipse Foundation as an official project provides external validation of our industry-focused approach.

[1]https://github.com/eclipse-tracecompass/tmll

Fig. 1: TMLL workflow: from trace data import to actionable insights through automated analysis selection

Our research journey began with seemingly reasonable hypotheses grounded in conventional automated software engineering wisdom: practitioners were not adopting trace analysis tools because existing tools were too complex, required excessive manual effort, and lacked sufficient automation. We hypothesized that democratizing sophisticated analytical techniques through machine learning would naturally lead to widespread adoption, following established patterns in other automated software engineering domains [16]–[19]. Through sustained industrial collaboration and systematic practitioner engagement, these assumptions proved fundamentally flawed.

Our empirical investigation revealed that real adoption barriers were categorically different from our initial assumptions. Practitioners struggled not with tool complexity per se, but with three distinct automation challenges: (1) translating expert analytical knowledge into actionable automated insights, (2) integrating automated analysis capabilities into existing development workflows without disrupting established practices, and (3) building confidence in automated results that they could otherwise not independently validate. Most significantly, we discovered what we term the *Excellence Paradox:* technical excellence can actively hinder adoption when it comes at the cost of usability, transparency, and practitioner trust.

Through iterative development, continuous stakeholder engagement, and systematic analysis of both successful and failed approaches, our research addresses three key questions:

RQ1*: What specific adoption barriers do practitioners face when using sophisticated trace analysis tools in industrial settings, and how do these barriers manifest in practice?*

RQ2*: Which design principles enable TMLL to achieve sustainable adoption in real-world development workflows within our study context?*

RQ3*: What gaps exist between academic tool development assumptions and industrial adoption requirements, as revealed through sustained practitioner feedback?*

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00286

We validated our understanding through multiple convergent methods: a comprehensive survey of 40 industry practitioners and academic researchers, TMLL's integration into the Eclipse Foundation as an official project, early adoption experiences across multiple users within Ericsson and external organizations, and sustained engagement from practitioners throughout the development process.

Our work provides four novel contributions for ML-enhanced trace analysis tools: (1) **characterization of barrier priorities** (i.e., 77.5% prioritizing trust, 75.0% expertise accessibility, and 67.5% preferring semi-automated over fully automated analysis), providing magnitudes unavailable in prior qualitative studies and revealing that transparency-trust concerns outweigh workflow integration (55.0%), contradicting conventional assumptions; (2) **adoption-focused design methodology** systematically derived from iterative industrial collaboration, demonstrating how to translate barrier understanding into architectural decisions through cognitive compatibility, embedded expertise, and transparency-based trust principles; (3) **empirical validation of academic-industrial gaps** through sustained practitioner feedback, quantifying four critical misalignments between academic tool development assumptions and industrial requirements (guided semi-automation, trust priority, embedded knowledge, workflow augmentation); and (4) **actionable design guidance** validated through Eclipse Foundation integration and cross-organizational adoption, providing concrete patterns for building trace analysis tools that achieve sustained industrial use rather than remaining research prototypes.

II. Background and Related Work

Our work builds upon research spanning three key domains: trace analysis tools and systems, machine learning-based trace and log analysis, and empirical studies of developer tool adoption. We structure this section to provide the necessary background while positioning our contributions within the broader research landscape.

A. Trace Analysis Tools and Systems

System tracing is a fundamental technique for understanding complex software behavior in modern distributed systems. Distributed tracing systems capture the flow of requests across multiple services, enabling performance analysis and debugging of microservice architectures [3], [8]–[11], [14], [15], [20]–[22]. Pioneered by foundational work, including Google's Dapper [1], which established the architectural principles used by virtually all contemporary tracing platforms, and X-Trace [7], which introduced end-to-end tracing across administrative domains with metadata propagation.

Systems like Magpie [23] and Canopy [12] demonstrated the feasibility of automatic request flow extraction and large-scale trace processing, with Canopy processing over 1 billion traces daily in Facebook's production environment. Modern distributed tracing has evolved significantly with systems like DeepFlow [24] and TraceWeaver [25], which eliminate the need for application instrumentation through eBPF-based network monitoring and statistical timing analysis, respectively.

Advanced sampling techniques are critical for managing trace data volume at scale. Traditional uniform random sampling proves inadequate for capturing rare but important events, leading to sophisticated approaches like Sifter [26], which uses machine learning to bias sampling toward edge-case code paths, and retroactive sampling systems [27] that capture detailed traces only when problems are detected. Performance analysis frameworks such as The Mystery Machine [28] have demonstrated automatic causal model construction from component logs, enabling critical path analysis and bottleneck identification in distributed systems.

Compared to these existing systems, TMLL addresses a fundamentally different challenge. While tools like Dapper and Jaeger focus on trace collection and visualization, and systems like DeepFlow automate trace generation, TMLL specifically targets the interpretation gap between raw trace data and actionable performance insights. Unlike Canopy's focus on large-scale trace processing infrastructure, TMLL emphasizes cognitive compatibility and workflow integration to achieve practitioner adoption.

B. Machine Learning-based Trace and Log Analysis

Machine learning applications in trace and log analysis have evolved from simple statistical anomaly detection to sophisticated deep learning approaches. The field encompasses log parsing and preprocessing [29], [30], where automated methods extract structured information from unstructured log data, and anomaly detection techniques that identify unusual system behavior patterns.

Deep learning approaches have shown particular promise for trace analysis. DeepLog [31] pioneered LSTM-based sequence modeling for log anomaly detection with workflow construction for root cause analysis. LogAnomaly [32] extended this using template2vec semantic embeddings combined with LSTM networks to detect both behavioral and quantitative anomalies in unstructured logs. Recent work addresses practical deployment challenges, with LogRobust [33] using ensemble methods and transfer learning for evolving log formats, and comprehensive evaluations [34] revealing significant gaps between academic results and practical deployment scenarios.

Microservice-specific approaches have emerged to handle the unique challenges of distributed trace analysis. ServiceAnomaly [35] combines distributed traces with profiling metrics using Context Propagation Graphs for anomaly detection in microservice systems. For high-performance computing environments, tools like STAT [36] provide scalable debugging through tree-based analysis and process equivalence classes. Root cause analysis has advanced through ensemble ML methods [37] and automated performance debugging techniques [38] that combine multiple algorithmic approaches for comprehensive system analysis.

However, despite technical advances in ML-based analysis, adoption remains limited in industrial settings. Most existing work focuses on algorithmic improvements rather

than addressing practical barriers preventing practitioners from effectively using these sophisticated analytical capabilities. Empirical studies [34] confirm significant challenges in real-world deployment scenarios, including concept drift, training data requirements, and interpretability concerns.

TMLL differs from existing ML-based trace analysis tools in its explicit focus on adoption barriers rather than algorithmic sophistication. While systems like DeepLog and LogAnomaly optimize for detection accuracy using complex deep learning models, TMLL prioritizes explainable results and embedded expertise with cognitive compatibility for existing practitioner workflows. TMLL addresses the gap identified by Le and Zhang [34] between academic ML approaches and practical industrial deployment through its adoption-focused design methodology.

C. Developer Tool Adoption and Usability

Understanding why sophisticated developer tools fail to achieve adoption despite technical excellence is critical for building practical analysis systems. Empirical studies across multiple tool domains reveal consistent adoption barrier patterns that inform TMLL's design.

Johnson et al. [39] investigated why developers abandon static analysis tools despite recognizing their value, finding that trust issues from false positives, poor explainability, workflow disruption, and lack of transparency create fundamental adoption barriers. Their study revealed developers consistently struggled with result understandability, workflow integration, and trust issues from unclear warnings. Critically, developers prefer semi-automated approaches with preview and control over fully automated solutions, paralleling the automation-transparency tension in trace analysis tools.

For trace-based analysis tools, Weninger et al. [40], [41] examined adoption barriers in memory monitoring tools, identifying that complexity barriers, expertise requirements, and lack of embedded guidance prevent practitioners from effectively using trace-based performance tools. Their proposed Guided Exploration method addresses these barriers by automatically detecting patterns, highlighting relevant information, and explaining significance.

These studies establish that adoption barriers stem from human factors rather than technical limitations: tools must provide transparency to build trust [39], embed expertise rather than require training [41], and maintain cognitive compatibility with existing workflows [39], [40]. Our work extends these findings to industrial trace analysis contexts, providing quantitative and qualitative validation and demonstrating their application to ML-enhanced performance analysis tools.

III. Methodology

A. Research Design and Approach

This study employs a participatory action research methodology [42], [43] within an industrial setting to systematically investigate trace analysis adoption barriers and develop evidence-based solutions through iterative tool development. Our approach combines sustained practitioner engagement with empirical development, enabling simultaneous study of adoption challenges and validation of proposed solutions in real industrial contexts. The research follows a mixed-methods design integrating qualitative stakeholder feedback with quantitative validation data, specifically targeting the deployment of intelligent trace analysis tools in production environments.

B. Research Context and Setting

Our research was conducted through collaboration with Ericsson Montréal, a major telecommunications company operating complex distributed systems requiring sophisticated performance analysis. The organization's multi-service architectures with intricate subsystem interactions provided an ideal setting for investigating trace analysis adoption barriers, as performance degradation could have a significant operational and business impact. This industrial context featured characteristics essential for our study: technically sophisticated users, clear performance requirements, substantial trace datasets, and strong organizational incentives for performance optimization.

The core collaboration centered on a 5-member trace analysis team with over 15 years of experience working on tracing for high-performance systems, serving as the primary technical partner. Additionally, we engaged with several product development teams (5-15 members each) responsible for telecommunications infrastructure components requiring sophisticated performance analysis. This industrial partnership provided authentic validation of ML-enhanced trace analysis approaches in production environments.

TMLL was developed as an extension to Trace Compass [44], an established trace-analysis tool trusted by thousands of users, leveraging established protocols for optimized handling of large-scale trace data. As shown in Figure 2, The system architecture consists of five primary layers: (1) client interface layer providing Python-based access, (2) data processing layer implementing intelligent data fetching and preprocessing, (3) base module abstraction layer enabling consistent interfaces, (4) specialized analysis modules implementing machine learning techniques, and (5) visualization and results layer. The tool enables practitioners to import trace files into TMLL's client, create experiments from these traces, and then apply various ML-enhanced analysis modules (anomaly detection, correlation analysis, change point detection, etc.) to gain insights from the trace data. The adoption of TMLL by the Eclipse Foundation as an official project validates the industrial relevance and technical soundness of our approach.

C. Research Timeline and Phases

The research was conducted over approximately 12 months (May 2024 - April 2025), organized into four phases aligned with industrial development cycles: **Phase 1** (Months 1-3): Problem investigation and infrastructure design through stakeholder interviews and proof-of-concept development. **Phase 2** (Months 4-6): Core module development, including trace-based anomaly detection, correlation analysis, and memory leak analysis, with continuous feedback integration. **Phase 3** (Months 7-9): Infrastructure improvements and reliability

enhancements based on early adoption feedback, with early adoption beginning in Month 7. **Phase 4** (Months 10-12): Integration capabilities development (including workflow and deployment capabilities), formal evaluation, and successful Eclipse Foundation submission.

D. Participant Recruitment and Data Collection

Our data collection strategy employed two complementary approaches: sustained practitioner engagement throughout development and systematic validation using a structured survey, designed to capture both depth and breadth of industrial adoption patterns.

Development Phase Engagement: Throughout TMLL development, we maintained continuous engagement with approximately 25-30 practitioners using purposive sampling [45] within Ericsson and extended professional networks. Participants represented diverse industrial roles, including software developers, system performance specialists, DevOps engineers, and academic researchers. This sustained engagement provided iterative feedback on design decisions, adoption barriers, and usage patterns through biweekly progress presentations, monthly formal meetings, and semi-structured informal discussions focused on practical deployment challenges.

Validation Survey Design and Execution: To validate insights beyond our development context, we designed a mixed-methods survey with 15 Likert-scale questions measuring adoption preferences and barriers, and 8 open-ended questions exploring practitioner experiences and priorities. We distributed the survey through organizational networks, professional contacts, and academic communities, using snowball sampling [46] to reach practitioners across organizations. The survey recruited 40 practitioners from multiple companies and academic researchers, providing validation across diverse industrial contexts.

E. Tool Development and Analysis

TMLL implements a modular architecture designed to support incremental adoption in industrial environments. The base module abstraction provides consistent interfaces for data processing, analysis execution, and result visualization, enabling practitioners to integrate capabilities progressively without disrupting existing workflows. Our development methodology emphasized rapid prototyping with continuous validation through feature-driven development and comprehensive documentation suitable for enterprise deployment.

The iterative development approach enabled continuous refinement based on practitioner feedback, with each development cycle validated through direct usage in Ericsson's production environment. This industrial validation approach ensures that design decisions reflect real-world deployment constraints rather than theoretical considerations.

F. Research Hypotheses

Based on our preliminary investigation and existing literature on developer tool adoption [16], [17], we formulated three core hypotheses about adoption-focused tool design for intelligent trace analysis systems:

$\mathcal{H}1$ *(Cognitive Compatibility):* Tools that augment existing practitioner mental models achieve higher adoption than those requiring new analytical paradigms.

$\mathcal{H}2$ Practitioners prefer tools that embed expert knowledge in interfaces rather than expose analytical capabilities for manual configuration.

$\mathcal{H}3$ *(Transparency-Trust Relationship):* Explainable automated ML-enhanced analysis achieves higher practitioner trust and adoption than black-box sophisticated algorithms.

These hypotheses guide our design decisions and evaluation methodology, providing testable predictions about the relationship between interface design choices and practitioner adoption patterns in industrial trace analysis environments.

IV. Adoption Challenges in Trace Analysis

Our empirical investigation through sustained practitioner engagement across 5 teams at Ericsson, with 12 observing and 12 months of development, identified three fundamental barriers preventing trace analysis tool adoption in industrial practice. These challenges represent systematic failures in how traditional tools approach the gap between analytical capabilities and practitioner needs in production environments, ultimately impacting the productivity and effectiveness of trace analysis in telecommunications and other enterprise domains.

A. Expertise Distribution and Knowledge Transfer Barriers

Effective trace analysis requires specialized knowledge to interpret patterns, correlate indicators, and distinguish normal system behavior from problematic conditions [13], [30], [47]. In industrial environments, this expertise is concentrated among a few practitioners (Nearly none of the teams have dedicated performance specialists), creating significant bottlenecks when teams need trace analysis capabilities for production incident response and performance optimization. These expertise barriers are well-documented in trace-based analysis tools, where novice users struggle with specialized terminology and non-obvious analysis workflows [40], [41].

Most practitioners can collect traces using standard tools (FTrace, DTrace, LTTng, perf) but struggle to extract actionable insights from raw data. Our observational studies at Ericsson revealed significant interpretation gaps: practitioners could identify obvious anomalies (CPU spikes $> 90\%$) but missed subtle patterns indicating developing problems that could lead to production failures. Expert analysts consistently recognized meaningful correlations that non-experts failed to detect, even when provided with identical visualization tools [47], [48]. For example, experts identified 85% of performance regression cases that non-experts missed entirely during controlled evaluation sessions. This expertise gap in pattern recognition reflects broader challenges in trace-based performance monitoring, where identifying significant information in large datasets requires substantial experience [40].

The business impact of these expertise bottlenecks extends beyond immediate technical challenges. Teams without dedicated performance specialists face increased mean time to resolution for performance issues, higher operational costs,

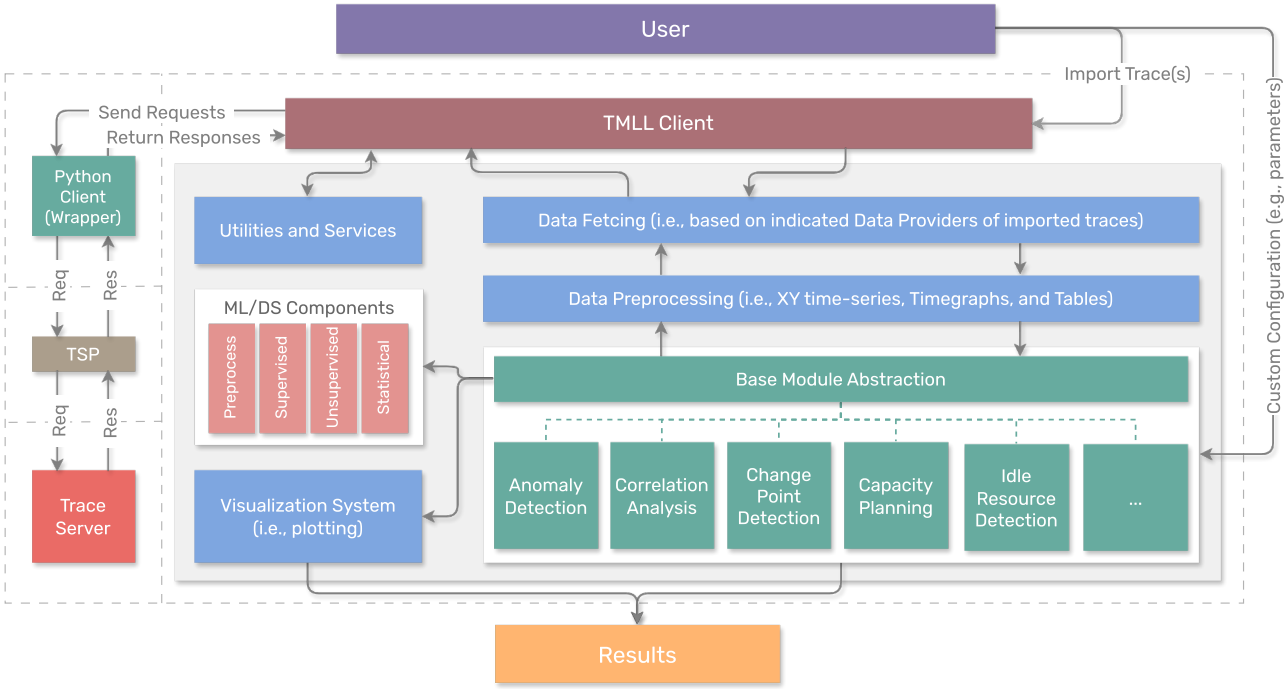

Fig. 2: TMLL's architecture: data processing, analysis modules, and visualization components with client-server integration

and reduced system reliability. In telecommunications environments like Ericsson's, where system performance directly affects service quality and customer satisfaction, these expertise gaps represent significant organizational risk.

B. Scale, Complexity, and Workflow Integration Challenges

Modern distributed systems generate massive trace volumes with hundreds of simultaneous metrics across multiple subsystems [12], [14], [49], [50]. Production systems at Ericsson typically generate between 100,000 and 150,000 events per second across thousands of concurrent processes, requiring correlation of temporal patterns across CPU usage, memory allocation, I/O operations, network traffic, and application-level events. Manual correlation analysis becomes intractable beyond simple scenarios involving around 50 metrics over 15-minute time windows.

Practitioners reported spending significant time analyzing curated datasets with numerous metrics, only to conclude they couldn't determine whether observed patterns represented problems or normal system variation. The combinatorial explosion of potential correlations ($N \times (N-1)/2$ combinations for N metrics) overwhelms human analytical capacity, particularly when temporal offsets and non-linear relationships must be considered [48], [51]. For instance, real-world traces with 50 simultaneous metrics create 1,225 potential pairwise correlations, far exceeding human processing capabilities.

Performance issues often manifest through subtle interactions across multiple trace dimensions rather than obvious single-metric anomalies [28], [31], [49], [52]. Performance regressions typically involve a sensible increase across multiple metrics rather than dramatic spikes in individual measurements. Practitioners struggle to maintain awareness of multi-dimensional patterns while examining individual metric trends.

Existing trace analysis tools exacerbate these challenges by requiring practitioners to learn new analytical paradigms rather than fitting into existing mental models. Practitioners conceptualize performance problems in terms of system-level behaviors ("application becomes slow", "memory usage grows over time") rather than statistical abstractions (correlation coefficients, anomaly scores) [13], [30], [47], yet traditional tools present results in statistical terms requiring translation into actionable insights. This mismatch between statistical outputs and domain-relevant explanations represents a fundamental usability barrier observed across developer analysis tools [39].

Interviews with more than 40 practitioners revealed consistent preferences for tools that *"tell me what's wrong and what to do about it"* rather than *"show me the data and let me figure it out"*. This represents a fundamental mismatch between tool capabilities and practitioner cognitive models. Integration failures occur not due to technical incompatibility but because tools don't align with how practitioners conceptualize performance problems [17], [53], [54]. Successful integration requires workflow compatibility rather than just technical interoperability [16], [17], with empirical studies showing that workflow disruption is a primary reason developers abandon technically sophisticated tools [39].

Traditional trace analysis requires practitioners to switch between familiar debugging contexts (application logs, system

monitors) and unfamiliar analytical contexts (statistical software, specialized visualization tools). This context switching introduces considerable time overhead per analysis session and reduces practitioner willingness to engage with trace analysis regularly, ultimately limiting the effectiveness of performance monitoring and optimization efforts.

C. Empirical Evidence from Industrial Deployment

Our field studies at Ericsson provided quantitative evidence for these challenges in production. Teams with dedicated performance specialists resolved most of the performance issues within hours, while teams without specialists required at least 2-3x of time on average for similar issues. This performance gap translates directly to operational costs and service availability impact in telecommunications environments.

Tool adoption patterns suggest that sophisticated analytical platforms tend to have a lower adoption rate compared to simpler visualization tools, which remain in use despite their limited analytical capabilities. This data confirms that adoption barriers extend beyond technical sophistication to fundamental usability and workflow integration issues that affect long-term tool viability in industrial settings.

Our survey of 40 practitioners confirmed these challenges across diverse organizational contexts beyond Ericsson. Most significantly, 77.5% identified "quality and trust in results" as a primary adoption barrier, followed closely by 75.0% citing "learning curve and required expertise," while "integration with existing tools" ranked third at 55.0%. These findings align with broader patterns observed in developer tool adoption: trust issues from opaque or unreliable results consistently drive tool abandonment [39], while steep learning curves prevent practitioners from adopting sophisticated analysis capabilities [40]. This ranking validates that adoption barriers extend beyond technical capabilities to fundamental transparency and trust issues, with practitioners prioritizing result reliability over workflow compatibility across different industrial domains. The elevation of trust concerns above integration requirements reinforces our core finding that technical excellence without explainability actively hinders adoption.

V. TMLL: Adoption-Focused Architecture for Industrial Deployment

This section presents TMLL's architecture as a systematic response to the three adoption barriers identified in Section IV: expertise distribution barriers, scale and workflow integration challenges, and transparency-trust concerns. Rather than optimizing for algorithmic sophistication, TMLL prioritizes cognitive compatibility, workflow integration, and embedded expertise to achieve sustained adoption, with each architectural layer explicitly addressing specific barriers from our collaboration with Ericsson.

A. Adoption-Driven Architectural Philosophy

TMLL's architecture embodies what we term **adoption-first design**, where each layer and component is designed to minimize adoption barriers rather than maximize technical capabilities. Figure 3 illustrates the fundamental shift from technical excellence to adoption-focused design that enabled TMLL's successful engagements. This philosophy emerged from observing that practitioners consistently abandoned technically sophisticated tools in favor of simple, familiar approaches that fit their existing mental models and workflows.

The architecture addresses three adoption requirements: **expertise accessibility** (by minimizing the need for specialized knowledge), **cognitive compatibility** (by fitting existing practitioner mental models), and **transparency-based trust** (through explainable analysis). These requirements drove fundamental architectural decisions, including embedded expertise in interfaces, transparent explanation generation, and modular capability introduction that enables incremental adoption without workflow disruption.

B. Three-Layer Architecture Addressing Adoption Barriers

TMLL implements a three-layer architecture where each layer specifically addresses the adoption barriers identified in Section IV. The architecture maintains technical rigor necessary for production trace analysis while prioritizing practitioner accessibility and workflow compatibility over algorithmic sophistication. Table I summarizes how each architectural layer addresses specific adoption barriers.

TABLE I: Mapping architectural layers to adoption barriers

Architectural Layer	**Addresses Barrier(s)**
Data Processing	*Expertise Distribution* (eliminates data cleaning expertise requirements)
Analysis Engine	*Expertise Distribution* (embeds analytical expertise) & *Transparency-Trust* (provides explainable analysis)
Interface & Interpretation	*Workflow Integration* (maintains cognitive compatibility) & *Transparency-Trust* (generates domain-relevant explanations)

1) Standardized Data Processing Layer: Addressing Expertise Distribution Barriers: TMLL's foundational layer (i.e., Trace Server) directly addresses the expertise distribution barrier by automatically transforming heterogeneous trace formats (e.g., FTrace, DTrace, LTTng, custom formats) into a standardized representation (i.e., Pandas data frames), removing the need for practitioner data cleaning expertise that has traditionally limited trace analysis to specialists.

The "Data Preprocessing" class implements comprehensive error recovery mechanisms for real-world trace data quality issues, automatically handling missing values (i.e., over than 60% of "unacalibrated" traces contain gaps such as lost events; after tuning, tracers will not have temporal gaps but may have informational gaps), format inconsistencies (i.e., fields assigned with a:=b, a=b, a:b, a=[b,c] and timestamps from multiple timezones stored without geolocation), and temporal alignment problems across concurrent data streams.

This layer eliminates the technical infrastructure expertise barrier commonly observed across many organizations, includ-

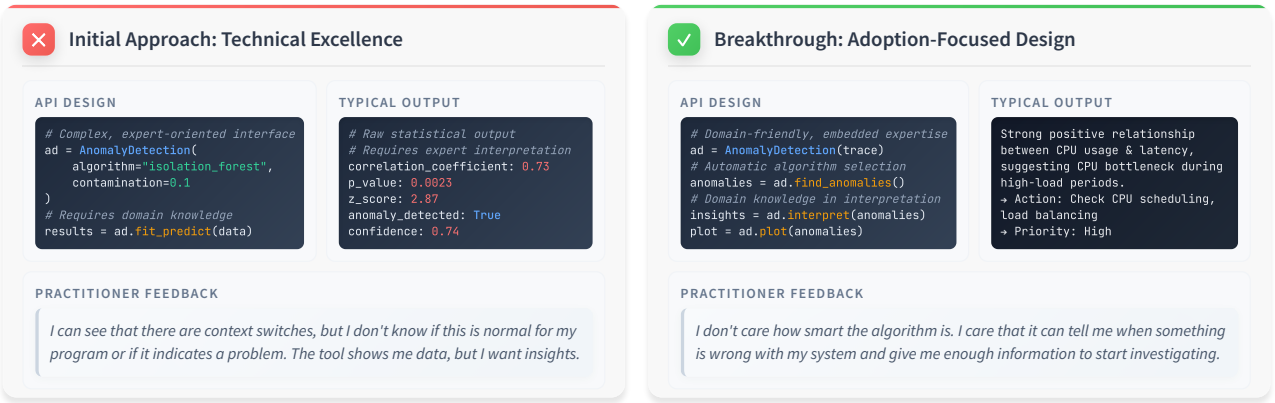

Fig. 3: The Excellence Paradox: When Technical Sophistication Hinders Adoption. TMLL's design philosophy prioritizes practitioner cognitive compatibility over algorithmic sophistication, demonstrated through our development at Ericsson.

ing Ericsson: practitioners often abandon trace analysis when faced with complex data preprocessing requirements. Performance optimizations include streaming processing for traces up to terabytes of data and parallelization [44], ensuring that the abstraction layer doesn't compromise the performance requirements of production environments. This design principle (i.e., hiding complexity without sacrificing capability) proved essential for adoption in technical environments where both ease of use and production-scale performance are mandatory.

2) Analysis Engine Layer: Embedding Expertise and Enabling Transparency: This layer addresses both the expertise distribution barrier and transparency-trust concerns by hiding complex statistics behind one uniform API while providing explainable results. Practitioners can "click and learn" instead of "tune and pray," democratizing sophisticated analysis without requiring expert knowledge. Its six built-in modules cover the trace questions teams hit most often:

- **Anomaly Detection**: scans any desired metric, picks an appropriate detector on the fly (e.g., z-score, IQR, Isolation Forest), and finds suspicious timestamps worth a closer look.
- **Memory Leak Detection**: follows pointer lifetimes (e.g., `malloc`/`free`) and memory growth trends to flag leaks (e.g., `malloc` without `free`) and finds original call sites.
- **Correlation Analysis**: maps which metrics rise or fall together (and in what order) to speed causal reasoning.
- **Change Point Detection**: spots hidden shifts in system behavior by voting across multiple metrics aggregately.
- **Capacity Planning**: projects metrics' upcoming behavior (e.g., CPU, memory, disk, network), warns when defined thresholds will be crossed, and suggests scaling actions.
- **Idle Resource Identification**: detects long idle stretches of metrics (e.g., underutilized memory, imbalance CPU core workloads) and recommends consolidation or re-balancing.

Figures 4 and 5 illustrate representative examples of TMLL's analysis output, demonstrating how the tool translates complex analytical results into actionable insights with clear visualizations and domain-relevant recommendations.

TMLL's standardized, extensible architecture proved essential during deployment, where all modules extend a shared abstract base class to ensure consistent data access, processing, analysis, and output. This design allows teams to create tailored analyses without compromising usability or workflow integration. By presenting results in domain-relevant, actionable terms through a unified interface, TMLL eliminates the need for practitioners to learn different configurations, supporting scalable adoption across diverse teams.

3) Interface and Interpretation Layer: Ensuring Workflow Integration and Transparency: The presentation layer addresses workflow integration challenges and transparency-trust concerns by maintaining cognitive compatibility with existing performance engineering practices [13], [47]. The "Visualization" and "Document Generation" classes translate analytical results into actionable insights that align with practitioner mental models rather than requiring adoption of new visualization paradigms, while providing the transparency necessary to build trust in automated analysis.

This layer emerged from the critical insight that adoption failures often result from interface mismatches rather than algorithmic limitations. Practitioners at Ericsson consistently preferred tools that augmented their existing debugging approaches over comprehensive analytical platforms requiring new paradigms. Results include domain-relevant explanations, confidence assessments, and recommended actions rather than statistical outputs, bridging the gap between sophisticated analysis and practical decision-making while building trust through transparency.

The interface supports various visualization formats (e.g., XY line graphs, heatmaps, box plots) and exports to standard formats for integration with existing toolchains, ensuring that TMLL enhances rather than replaces established workflows. This integration capability proved essential for sustained adoption, as practitioners could gradually incorporate TMLL's insights into their familiar debugging processes.

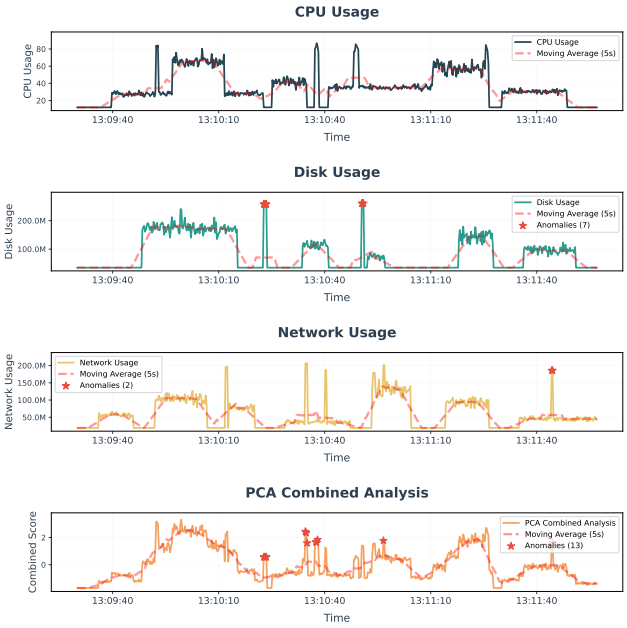

Fig. 4: Multi-metric anomaly detection output showing CPU, disk, and network usage analysis with identified anomalies and combined PCA scoring

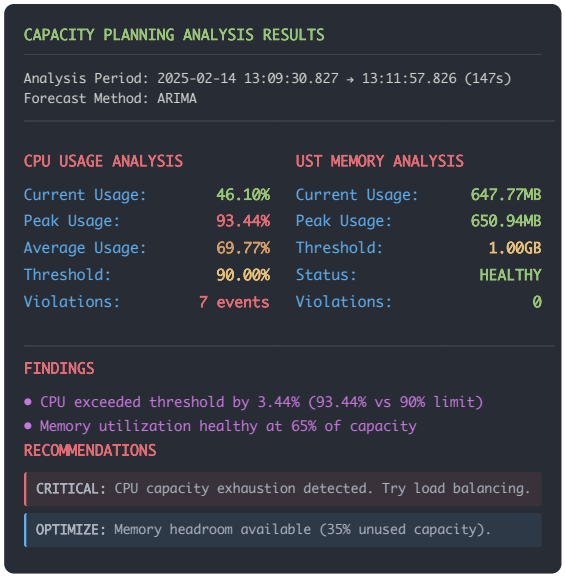

Fig. 5: TMLL capacity planning analysis results with threshold violations, resource utilization assessment, and actionable recommendations

C. Production-Scale Implementation and Deployment

TMLL's implementation emphasizes production readiness and enterprise adoption through a carefully designed technology stack and deployment architecture. The system is built in Python using established scientific computing libraries including NumPy [55], SciPy [56], and pandas [57] for numerical analysis, with Matplotlib [58] and Seaborn [59] providing visualization capabilities. Machine learning functionality is supported through scikit-learn [60], TensorFlow [61], and Statsmodels [62], while the framework accommodates custom library integration to meet specialized analytical requirements.

The deployment architecture scales from individual developer workstations to high-performance computing clusters, enabling flexible usage patterns that range from personal debugging sessions to enterprise-wide performance monitoring. This architectural flexibility proved critical during Ericsson's adoption, where TMLL supported both individual practitioner workflows and large-scale organizational analytics initiatives.

Central to TMLL's extensibility is the Base Module abstraction, which provides a standardized interface for custom analytical development [63]. This design enables practitioners to implement domain-specific analyses within TMLL's infrastructure while maintaining consistency across data access patterns, result formatting, and visualization generation. The module system supports multiple analysis types through unified interfaces, allowing users to extend TMLL's capabilities without requiring them to learn new architectural paradigms.

The technology stack deliberately avoids exotic dependencies that could complicate enterprise deployment, instead prioritizing proven libraries that balance analytical power with operational simplicity [16], [17]. This approach ensures that TMLL can be deployed across diverse enterprise environments while maintaining the analytical sophistication required for complex performance analysis tasks.

VI. Validation and Results

We validated TMLL's adoption-focused design through systematic evaluation combining iterative development feedback from industrial collaboration at Ericsson and multiple external organizations, quantitative survey data from diverse organizations, and external peer review through Eclipse Foundation acceptance.

A. Validation Methodology

Our validation approach assessed TMLL's design principles across diverse contexts through three complementary methods: sustained development collaboration with Ericsson Montréal practitioners, cross-organizational survey validation, and external peer review.

1) Development Collaboration at Ericsson: The primary validation occurred through sustained development collaboration at Ericsson Montréal over 12 months, involving iterative feedback from cross-functional development teams spanning diverse engineering domains, including radio baseband, DevOps, AI, and infrastructure teams. Rather than post-deployment metrics, validation came through iterative design cycles where Ericsson practitioners provided feedback on each development phase, enabling us to validate design decisions in real-time based on practitioner responses to prototypes, feature demonstrations, and hands-on evaluation sessions.

TABLE II: Practitioner Priorities vs. Initial Research Assumptions Survey Analysis (n=40)

Capability/Feature	% Valued	Key Insight
Most Valued Capabilities		
Anomaly Detection	77.5%	Strong preference for actionable insights over algorithmic sophistication
Root Cause Analysis	75.0%	Contextual interpretation valued more than correlation detection
Data in Python-ready Formats	42.5%	Infrastructure important but secondary to analytical insights
Automation vs. Control Preferences		
Fully Automated Analysis	20.0%	Overwhelming preference for guided analysis with transparency over black-box automation
Semi-automated with User Control	67.5%	
Primary Adoption Factors (Ranked by Priority)		
Quality and Trust in Results	77.5%	Explainability and reliability now top priority for adoption
Learning Curve and Expertise Required	75.0%	Critical need for embedded expertise, not simplified tools
Integration with Existing Tools	55.0%	Workflow compatibility important but secondary to trust
Documentation and Support	42.5%	Essential infrastructure for enterprise adoption
Feature Importance (Rated 4-5/5)		
Automated Data Preprocessing	100.0%	Universal agreement: infrastructure work as high-value innovation
Interactive Visualizations	100.0%	Exploration and discovery more valued than automated conclusions
Anomaly Detection Capabilities	100.0%	Automated pattern recognition essential for practical adoption
Root Cause Analysis Features	100.0%	Diagnostic capabilities critical for production environments
Extensibility for Custom Modules	97.5%	Near-universal need for adaptation to local requirements
Adoption Likelihood		
Likely/Very Likely to Try TMLL	85.0%	Strong validation of adoption-focused design approach

2) Cross-Organizational Survey: To validate insights beyond Ericsson's organizational context, we developed a mixed-methods instrument with 15 quantitative ratings (5-point Likert scales) and 8 qualitative feedback questions structured around our three research questions. The survey instrument included attention checks and consistency verification [63]. We recruited 40 practitioners through organizational networks, professional contacts, and academic communities using snowball sampling [46]. Participants included software developers (55.0%, n=22), researchers (12.5%, n=5), team managers/leads (7.5%, n=3), DevOps/SRE engineers (5.0%, n=2), students (12.5%, n=5), performance engineers (2.5%, n=1), data scientists (2.5%, n=1), and trace product managers (2.5%, n=1) across different organizations, providing validation across different organizational contexts and technical domains.

B. Validation of Adoption Barriers

RQ1*: What specific adoption barriers do practitioners face when using sophisticated trace analysis tools in industrial settings, and how do these barriers manifest in practice?*

1) Barrier Priorities Across Organizations: Survey results in Table II revealed the relative importance of different adoption barriers. Most significantly, **quality and trust in results** emerged as the top adoption priority (77.5%, n=31), followed closely by **manageable learning curves** and required expertise (75.0%, n=30), while **integration with existing tools** ranked third (55.0%, n=22). This prioritization provides validation that the barriers identified at Ericsson reflect broader industrial challenges. The elevation of trust and expertise concerns above workflow integration demonstrates that transparency and knowledge accessibility represent more fundamental adoption barriers than initially assumed.

2) Manifestation in Development Practice: The iterative feedback process at Ericsson validated how these barriers manifest in practice. Practitioners consistently preferred demonstration scenarios where TMLL provided immediate actionable insights over those requiring interpretation of statistical outputs. Comments like *"I want the insights that an expert would have, but I don't want to become an expert myself"* directly confirmed the expertise distribution barrier.

Early prototypes focusing on algorithmic sophistication received lukewarm responses, while interfaces emphasizing embedded expertise and domain-relevant interpretation generated strong positive feedback. This pattern emerged across multiple feedback sessions and validated our shift from capability-focused to adoption-focused design, confirming that technical excellence alone was insufficient for practitioner acceptance.

Answer to RQ1: Practitioners face three validated adoption barriers that manifest across organizations: (1) **expertise accessibility** (75.0% prioritize manageable learning curves), requiring embedded domain knowledge rather than expert training; (2) **transparency and trust** (77.5% prioritize quality and trust in results), demanding explainable results they can validate against domain knowledge; and (3) **workflow integration** (55.0% value integration with existing tools), needing cognitive compatibility with existing practices. These barriers

manifest through rejection of technically sophisticated but opaque tools in favor of simpler, more transparent alternatives.

C. Validation of Design Principles

***RQ2**: Which design principles enable TMLL to achieve sustainable adoption in real-world development workflows within our study context?*

Our validation tested three core hypotheses about adoption-focused design through systematic evaluation across development collaboration and cross-organizational survey data.

1) $\mathcal{H}1$: Cognitive Compatibility - VALIDATED: Both Ericsson's development feedback and survey results (55.0% valuing integration with existing tools) confirmed that practitioners prefer tools augmenting existing mental models over those requiring new analytical paradigms. The elevation of quality and trust in results to the top adoption priority (77.5%) demonstrates that cognitive compatibility extends beyond workflow integration to encompass result interpretability. Ericsson practitioners emphasized that tools should "fit well into existing debugging practices" without disrupting established responsibilities. Survey comments consistently emphasized preference for tools that "augment existing practices" rather than "require learning new approaches."

2) $\mathcal{H}2$: Embedded Expertise - VALIDATED: Development collaboration feedback and survey validation (67.5% preferring guided over black-box approaches, 100% valuing automated preprocessing and intelligent analysis capabilities) strongly confirmed our embedded expertise approach. The preference for semi-automated analysis (67.5% vs. 20.0% for fully automated) demonstrates practitioners' desire for expert-level capabilities without requiring expert-level knowledge.

As illustrated in Table II, universal agreement emerged on core infrastructure capabilities: automated preprocessing (100%), interactive visualizations (100%), anomaly detection (100%), and root cause analysis (100%) were rated as highly valuable (4-5/5), with extensibility achieving near-universal approval (97.5%). The shift in capability preferences further validates our design approach: anomaly detection (77.5%) and root cause analysis (75.0%) were most valued for providing actionable insights rather than algorithmic sophistication. Comments emphasized wanting "contextual interpretation, not just correlation detection."

3) $\mathcal{H}3$: Transparency-Based Trust - VALIDATED: Strong preference for semi-automated approaches (67.5%) over fully automated solutions (20.0%) demonstrates the critical importance of transparency for building trust. Most significantly, quality and trust in results emerged as the primary adoption factor (77.5%), the highest-ranked concern among practitioners. Development collaboration feedback consistently emphasized understanding "how the tool reached its conclusions", validating that transparency serves as essential infrastructure for tool adoption in expert domains.

Answer to RQ2: Three validated design principles enable sustainable adoption: (1) **Cognitive compatibility** (77.5% prioritize interpretable results), augmenting existing mental models rather than requiring new paradigms; (2) **Embedded expertise** (67.5% prefer guided analysis, 100% value automated capabilities), democratizing sophisticated analysis through intelligent automation and familiar interfaces; and (3) **Transparency-based trust** (77.5% prioritize quality and trust), providing explainable results that practitioners can validate. These principles proved essential across diverse organizational contexts and technical domains.

D. Validation of Academic-Industrial Gaps

***RQ3**: What gaps exist between academic tool development assumptions and industrial adoption requirements, as revealed through sustained practitioner feedback?*

Our systematic validation revealed four critical gaps between academic assumptions about tool development and industrial adoption realities.

1) Automation Assumptions vs. Control Requirements: Our initial hypothesis assumed practitioners needed more automation to handle complexity. Validation revealed the opposite: 67.5% (n=27) preferred semi-automated analysis with user control over fully automated solutions (20.0%, n=8). This contradicts the common academic assumption that "more automation is always better" and demonstrates that practitioners value transparency and agency over algorithmic sophistication. Development collaboration feedback confirmed this pattern through practitioner reactions to different interface approaches.

2) Capability Focus vs. Trust Priority: Academic tool development typically emphasizes technical capabilities and algorithmic advancement. However, quality and trust in results (77.5%) emerged as the top adoption factor, significantly higher than technical sophistication or advanced features. Development feedback demonstrated that interpretability provides essential trust infrastructure for analysis tools, enabling users to validate conclusions against domain knowledge.

3) Expert Training vs. Knowledge Embedding: Academic approaches often assume practitioners need training to use sophisticated tools effectively. Survey data (75.0% prioritize manageable learning curves) and development feedback confirmed that practitioners prefer tools embedding expert knowledge rather than facilitating expert knowledge acquisition. This validates the democratization of sophisticated capabilities through embedded expertise rather than training programs or documentation.

4) Workflow Replacement vs. Augmentation: Academic tool development often proposes comprehensive new analytical frameworks. Validation showed practitioners consistently prefer tools that "augment existing practices" rather than "require learning new approaches." While workflow integration ranked third (55.0%), the overwhelming preference for transparent, guided analysis over black-box automation demonstrates that successful enterprise tools should enhance existing mental models rather than requiring paradigm shifts.

Answer to RQ3: Four validated gaps exist between academic assumptions and industrial requirements: (1) practitioners prefer **guided semi-automation** (67.5%) over full automation (20.0%), contradicting "more automation is better"; (2)

trust and transparency (77.5%) outweigh technical sophistication as adoption drivers; (3) practitioners favor **embedded expertise** (75.0% prioritize low learning curves) over expert training; and (4) tools must **augment existing workflows** rather than replace them, despite potential technical superiority. These gaps explain why technically excellent academic tools often fail to achieve industrial adoption.

E. External Validation and Key Findings

The Eclipse Foundation's adoption of TMLL as an official project[2] provided rigorous external validation through peer review by industry experts and academic researchers working on performance analysis tools. Review feedback confirmed broader applicability: *"The focus on practitioner adoption rather than algorithmic sophistication addresses a real gap in performance tooling"* and *"The emphasis on interpretability and workflow integration reflects challenges we've seen across multiple organizations."* This demonstrates that TMLL's adoption-focused design principles extend beyond telecommunications to other enterprise domains.

VII. Threats to Validity

Internal Validity: Participants were recruited through convenience sampling from existing professional networks, potentially limiting perspective diversity and introducing selection bias toward practitioners interested in analytical tools. Practitioner feedback during structured sessions may have been influenced by demonstration contexts and social desirability bias rather than reflecting natural usage patterns.

External Validity: Primary development occurred within a single telecommunications organization, potentially limiting generalizability to other organizational contexts and technical domains. Findings may be specific to telecommunications and high-performance computing environments, requiring validation in domains like web development or embedded systems where trace analysis challenges differ.

Construct Validity: We lacked comprehensive quantitative adoption metrics due to organizational policies, relying primarily on qualitative indicators (satisfaction, feedback, continued usage) rather than objective usage analytics. Our definition of "successful adoption" may be influenced by TMLL's developmental state rather than inherent adoption barriers. Additionally, we did not conduct a comparative evaluation against existing trace analysis tools, limiting our ability to definitively attribute adoption success to specific design principles versus other factors such as organizational support or technology choices.

Reliability: Some insights relied on informal feedback and observational notes that may affect reproducibility. Qualitative data analysis involved subjective interpretation of practitioner feedback, potentially introducing researcher bias in pattern identification and insight generation.

[2]https://projects.eclipse.org/projects/tools.tracecompass.tmll

VIII. Conclusion

This work reports our systematic investigation of trace analysis adoption barriers through a year-long industrial collaboration with Ericsson Montréal and the development of TMLL, revealing the *Excellence Paradox*: technical excellence can hinder adoption when it comes at the cost of usability and trust. Through development collaboration, cross-organizational survey validation (n=40), and external peer review leading to Eclipse Foundation adoption, we validated three key design principles: (1) practitioners prefer embedded expert knowledge over expertise development requirements, (2) cognitive compatibility outweighs technical sophistication for adoption, and (3) transparency and user control build trust in intelligent analysis tools. Our findings provide actionable guidance: prioritize workflow integration over algorithmic advancement, implement modular architectures supporting incremental adoption, ensure analytical transparency, and incorporate adoption metrics in evaluation frameworks. TMLL's Eclipse Foundation integration and strong practitioner interest (85% willing to try) demonstrate that adoption-focused design can achieve both technical rigor and practical utility, with implications extending beyond trace analysis to other expert domains requiring sophisticated automated tools.

References

[1] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," tech. rep., Google Technical Report, 2010.

[2] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "An industrial survey on microservice tracing and analysis," *Empirical Software Engineering*, vol. 27, no. 4, pp. 1–45, 2022.

[3] M. Gebai and M. R. Dagenais, "Survey and analysis of kernel and userspace tracers on linux: Design, implementation, and overhead," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–33, 2018.

[4] Y. Zhao, X. Xia, D. Lo, J. Grundy, and T. Zimmermann, "Large language models for software engineering: A systematic literature review," *arXiv preprint arXiv:2308.10620*, 2023.

[5] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2022.

[6] G. Fraser and A. Arcuri, "A large-scale evaluation of automated unit test generation using evosuite," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 2, pp. 1–42, 2014.

[7] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "X-trace: A pervasive network tracing framework," in *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, pp. 271–284, USENIX Association, 2007.

[8] M. Desnoyers and M. R. Dagenais, "The lttng tracer: A low impact performance and behavior monitor for gnu/linux," in *OLS (Ottawa Linux Symposium)*, vol. 2006, pp. 209–224, Citeseer, 2006.

[9] N. Kim, "uftrace: Function graph tracer for c/c++/rust."

[10] "Perf." https://perf.wiki.kernel.org/. Linux, Perf Wiki.

[11] "gperftools." https://github.com/gperftools/gperftools. Google, GitHub repository.

[12] J. Kaldor, J. Mace, M. Bejda, E. Gao, W. Kuropatwa, J. O'Neill, K. W. Ong, B. Schaller, P. Shan, B. Viscomi, V. Venkataraman, K. Veeraraghavan, and Y. J. Song, "Canopy: An end-to-end performance tracing and analysis system," in *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 34–50, ACM, 2017.

[13] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 684–702, 2009.

[14] D. Ardelean, A. Diwan, and C. Erdman, "Performance analysis of cloud applications," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pp. 405–417, 2018.

[15] L. Huang and T. Zhu, "tprof: Performance profiling via structural aggregation and automated analysis of distributed systems traces," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 76–91, 2021.

[16] Z. S. Li, N. N. Arony, A. M. Awon, D. Damian, and B. Xu, "Ai tool use and adoption in software development by individuals and organizations: A grounded theory study," in *Proceedings of the International Conference on Software Engineering*, ACM, 2024.

[17] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989.

[18] A. Memon, Z. Gao, B. Nguyen, S. Dhanda, E. Nickell, R. Siemborski, and J. Micco, "Taming google-scale continuous testing," in *Proc. of the 39th Int. Conf. on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pp. 233–242, 2017.

[19] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Proc. of the 31st IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*, pp. 426–437, 2016.

[20] J. Fenlason and R. Stallman, "Gnu gprof," *GNU Binutils. Available online: http://www. gnu. org/software/binutils (accessed on 21 April 2018)*, 1988.

[21] K. Shahedi, H. Li, M. Lamothe, and F. Khomh, "Tracing optimization for performance modeling and regression detection," *ACM Transactions on Software Engineering and Methodology*, 2023.

[22] N. Ezzati-Jivan, H. Daoud, and M. R. Dagenais, "Debugging of performance degradation in distributed requests handling using multilevel trace analysis," *Wireless Communications and Mobile Computing*, vol. 2021, pp. 1–17, 2021.

[23] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation*, pp. 91–110, USENIX Association, 2004.

[24] J. Shen, H. Zhang, Y. Xiang, X. Shi, X. Li, Y. Shen, Z. Zhang, Y. Wu, X. Yin, J. Wang, M. Xu, Y. Li, J. Yin, J. Song, Z. Li, and R. Nie, "Network-centric distributed tracing with deepflow: Troubleshooting your microservices in zero code," in *Proceedings of the ACM SIGCOMM 2023 Conference*, pp. 420–437, ACM, 2023.

[25] S. Ashok, V. Harsh, B. Godfrey, R. Mittal, S. Parthasarathy, and L. Shwartz, "Traceweaver: Distributed request tracing for microservices without application modification," in *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 420–437, ACM, 2024.

[26] P. Las-Casas, G. Papakerashvili, V. Anand, and J. Mace, "Sifter: Scalable sampling for distributed traces, without feature engineering," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 312–324, ACM, 2019.

[27] L. Zhang, Z. Xie, V. Anand, Y. Vigfusson, and J. Mace, "The benefit of hindsight: Tracing edge-cases in distributed systems," in *Proceedings of the 20th USENIX Conference on Networked Systems Design and Implementation*, pp. 321–339, USENIX Association, 2023.

[28] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch, "The mystery machine: End-to-end performance analysis of large-scale internet services," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, pp. 217–231, USENIX Association, 2014.

[29] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, pp. 121–130, IEEE Press, 2019.

[30] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 60–70, ACM, 2018.

[31] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1285–1298, ACM, 2017.

[32] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 4739–4745, AAAI Press, 2019.

[33] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J.-G. Lou, M. Chintalapati, F. Gao, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 807–817, ACM, 2019.

[34] V. H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?," in *Proceedings of the 44th International Conference on Software Engineering*, pp. 1–12, ACM, 2022.

[35] M. Panahandeh and A. Hamou-Lhadj, "Serviceanomaly: An anomaly detection approach in microservices using distributed traces and profiling metrics," *Information and Software Technology*, vol. 164, p. 107308, 2023.

[36] D. C. Arnold, D. H. Ahn, B. R. de Supinski, G. L. Lee, B. P. Miller, and M. Schulz, "Stack trace analysis for large scale debugging," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium*, pp. 1–8, IEEE, 2007.

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

[38] X. Liu, Q. Guan, B. Wu, P.-H. Lin, W.-Y. Chen, M. Chabbi, and J. Zhan, "Automatic performance debugging of spmd-style parallel programs," *Journal of Parallel and Distributed Computing*, vol. 73, no. 6, pp. 747–763, 2013.

[39] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in *Proc. of the 35th Int. Conf. on Software Engineering (ICSE)*, pp. 672–681, 2013.

[40] M. Weninger, P. Grünbacher, E. Gander, and A. Schörgenhumer, "Evaluating an interactive memory analysis tool: Findings from a cognitive walkthrough and a user study," *Proceedings of the ACM on Human-Computer Interaction*, vol. 4, no. EICS, pp. 1–37, 2020.

[41] M. Weninger, E. Gander, and H. Mössenböck, "Guided exploration: A method for guiding novice users in interactive memory monitoring tools," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. EICS, pp. 1–34, 2021.

[42] R. M. Davison, M. G. Martinsons, and N. Kock, "Principles of canonical action research," *Information Systems Journal*, vol. 14, no. 1, pp. 65–86, 2004.

[43] S.-T. Siew, A. W. Yeo, and T. Zaman, "Participatory action research in software development: Indigenous knowledge management systems case study," in *Human-Computer Interaction. Human-Centred Design Approaches, Methods, Tools, and Environments*, Lecture Notes in Computer Science, pp. 470–479, Springer Berlin Heidelberg, 2013.

[44] Eclipse Foundation, "Eclipse trace compass." https://eclipse.dev/tracecompass/, 2025. Open source application for performance and reliability analysis through trace and log analysis.

[45] L. A. Palinkas, S. M. Horwitz, C. A. Green, J. P. Wisdom, N. Duan, and K. Hoagwood, "Purposeful sampling for qualitative data collection and analysis in mixed method implementation research," *Administration and Policy in Mental Health and Mental Health Services Research*, vol. 42, no. 5, pp. 533–544, 2015.

[46] P. Biernacki and D. Waldorf, "Snowball sampling: problems and techniques of chain referral sampling," *Sociological Methods & Research*, vol. 10, no. 2, pp. 141–163, 1981.

[47] N. Pennington, "Stimulus structures and mental representations in expert comprehension of computer programs," *Cognitive Psychology*, vol. 19, no. 3, pp. 295–341, 1987.

[48] L. J. Gonçales, K. Farias, and B. C. Silva, "Measuring the cognitive load of software developers: An extended systematic mapping study," *Information and Software Technology*, vol. 131, p. 106486, 2021.

[49] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 243–260, 2021.

[50] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 3–18, 2019.

[51] J. Sweller, "Cognitive load during problem solving: Effects on learning," *Cognitive Science*, vol. 12, no. 2, pp. 257–285, 1988.

[52] L. Liao, J. Chen, H. Li, Y. Zeng, W. Shang, C. Sporea, A. Toma, and S. Sajedi, "Locating performance regression root causes in the field operations of web-based systems: An experience report," *IEEE Transactions on Software Engineering*, vol. 48, no. 12, pp. 4986–5006, 2021.

[53] M. Bano and D. Zowghi, "A systematic review on the relationship between user involvement and system success," *Information and Software Technology*, vol. 58, pp. 148–169, 2015.

[54] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, "Lessons from building static analysis tools at google," *Communications of the ACM*, vol. 61, no. 4, pp. 58–66, 2018.

[55] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with numpy," *Nature*, vol. 585, pp. 357–362, 2020.

[56] E. Jones, T. Oliphant, P. Peterson, *et al.*, "Scipy: Open source scientific tools for python." http://www.scipy.org/, 2001–.

[57] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, 2010.

[58] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[59] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021.

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[61] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 265–283, 2016.

[62] S. Seabold and J. Perktold, "statsmodels: Econometric and statistical modeling with python," in *Proceedings of the 9th Python in Science Conference*, 2010.

[63] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.