VIGIL: Towards Edge-Extended Agentic AI for Enterprise IT Support

Sarthak Ahuja* **Neda Kordjazi*** **Evren Yortucboylu*** **Vishaal Kapoor** **Mariam Dundua**
Yiming Li **Derek Ho** **Vaibhavi Padala** **Jennifer Whitted** **Rebecca Steinert**
Amazon Engine, AI Center of Excellence
{sarahuja,nedakord,yortuc,vishaalk,madundua}@amazon.com
{yimingll,derekjho,vapadala,jenwhitt,rsteinrt}@amazon.com
*Equal contribution

Abstract

Enterprise IT support is constrained by heterogeneous devices, evolving policies, and long-tail failure modes that are difficult to resolve centrally. We present VIGIL, an edge-extended agentic AI system that deploys desktop-resident agents to perform situated diagnosis, retrieval over enterprise knowledge, and policy-governed remediation directly on user devices with explicit consent and end-to-end observability. In a 10-week pilot of VIGIL's operational loop on 100 resource-constrained endpoints, VIGIL reduces interaction rounds by 39%, achieves at least 4× faster diagnosis, and supports self-service resolution in 82% of matched cases. Users report excellent usability, high trust, and low cognitive workload across four validated instruments, with qualitative feedback highlighting transparency as critical for trust. Notably, users rated the system higher when no historical matches were available, suggesting on-device diagnosis provides value independent of knowledge base coverage. This pilot establishes safety and observability foundations for fleet-wide continuous improvement.

1 Introduction

Large enterprise IT ecosystems comprise vast fleets of heterogeneous devices, networks, and policies operating under partial observability (Li et al., 2021). Failures often emerge from subtle local interactions rather than clear global faults, making diagnosis reactive and slow. Operational knowledge remains fragmented, and support workflows continue to rely on centralized human intervention and static playbooks, with AI chatbots offering limited post-incident assistance (Reinhard et al., 2024).

In parallel, AI systems are evolving beyond conversational assistants toward agentic, computer-use models capable of acting within software environments through terminal interfaces and operational tools (Anthropic; Kiro, 2025; Shen et al., 2024; Zhuang et al., 2023). Enterprise platforms are beginning to incorporate such capabilities into structured IT service workflows (Zhang, 2025; ServiceNow and Accenture, 2025), typically through centrally orchestrated agents within predefined automation boundaries rather than distributed, on-device execution across heterogeneous endpoints. Executing bounded actions directly on edge devices can enable lower-latency remediation, access to fine-grained local context, and improved resilience under partial connectivity, but such distributed autonomy requires mechanisms that prevent out-of-bounds behavior.

VIGIL explores this opportunity by bringing agentic and computer-use capabilities into enterprise IT through an emerging edge-extended architecture that combines on-device autonomy with cloud coordination and continuous improvement. At its core, VIGIL relies on language understanding, retrieval-augmented generation, and natural language reasoning to interpret user-reported issues, ground diagnosis in enterprise knowledge, and communicate remediation steps transparently. VIGIL treats each endpoint as an intelligent participant running specialized agents for diagnosis, knowledge access, and remediation, while cloud services provide model inference, fleet-level observability, and structured escalation to human operators.

Each interaction produces structured experience signals that form the basis for future learning at the context and prompt levels without requiring model retraining. This paper makes the following contributions:

- A distributed architecture for enterprise IT support in which endpoints perform situated diagnosis and remediation under cloud-coordinated governance and observability
- An edge-device controller that integrates on-

device diagnostic and remediation agents with enterprise knowledge access, observability, and structured escalation to human operators

- A proof-of-value pilot evaluating the edge controller component on real IT support scenarios, providing early evidence on efficiency, usability, and user trust.

2 Related Work

LLM-enhanced IT operations. Recent work demonstrates that large language models can enhance AIOps by structuring logs and incidents for improved detection and decision-making (Vitui and Chen, 2025), and surveys document both the promise of such techniques and the persistence of centralized, human-mediated workflows (de la Cruz Cabello et al., 2025). Agent-oriented efforts explore partial autonomy in detection, classification, and remediation (Zota et al., 2025), while evaluation frameworks probe the reliability of LLM-based operational agents (Chen et al., 2025). However, these systems retain centralized architectures in which endpoints serve as passive telemetry sources rather than active participants in diagnosis and resolution.

Agentic and tool-augmented systems. The broader agentic AI landscape has advanced rapidly, with surveys characterizing the design space of autonomous agents and their orchestration patterns (Acharya et al., 2025; Sapkota et al., 2025). Tool-augmented language models enable structured interaction with external systems through learned or specified tool interfaces (Schick et al., 2023; Patil et al., 2023), and standardized protocols such as MCP (Model Context Protocol Contributors) and A2A (Agent2Agent Working Group, 2025) provide interoperability across agent boundaries. Retrieval-augmented generation grounds model outputs in curated knowledge (Gao et al., 2023). VIGIL builds on these capabilities but is the first to integrate them into a governed, edge-deployed IT operations system.

Edge and distributed AI. Recent proposals advocate distributing intelligence across edge devices rather than concentrating it in centralized services (Luo et al., 2025; Tallam, 2025). While these works establish architectural principles for multi-LLM edge systems, they remain largely theoretical and do not address the governance, observability, and safety constraints required for enterprise IT. VIGIL addresses this gap by operating under deterministic policy control.

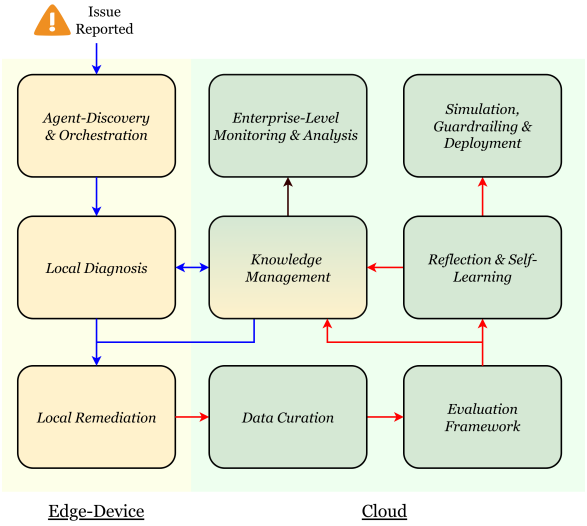

Figure 1: Operational and self-improvement loops within VIGIL. Device-level cognition, the focus of this paper, executes locally on endpoints (shown in **blue**), while cloud-level reflection, designed for future activation, refines policies, memory, and prompts over time (shown in **red**). Monitoring at the fleet level operates in parallel across both loops (shown in **black**).

3 VIGIL Vision and Architecture

VIGIL is organized around two tightly coupled control loops that together define how intelligence, action, and adaptation are distributed across the enterprise (depicted in figure 1).

Operational loop. The operational loop governs real-time diagnosis and remediation directly on endpoint devices, and is the primary focus of this paper. On each endpoint, the loop follows a structured diagnose, retrieve, and remediate workflow: when anomalies are detected through telemetry, scheduled checks, or user-reported issues, the system collects evidence, retrieves relevant enterprise knowledge, and executes bounded remediation actions. All steps are logged as structured traces for downstream analysis and monitoring. Execution is resilient to partial connectivity, allowing devices to continue troubleshooting using cached knowledge and locally enforced policies even when cloud services are unavailable.

Self-improvement loop. The self-improvement loop operates over structured experience produced by the operational loop. Rather than retraining foundation models, it refines the contextual scaffolding that shapes agent behavior, including mem-

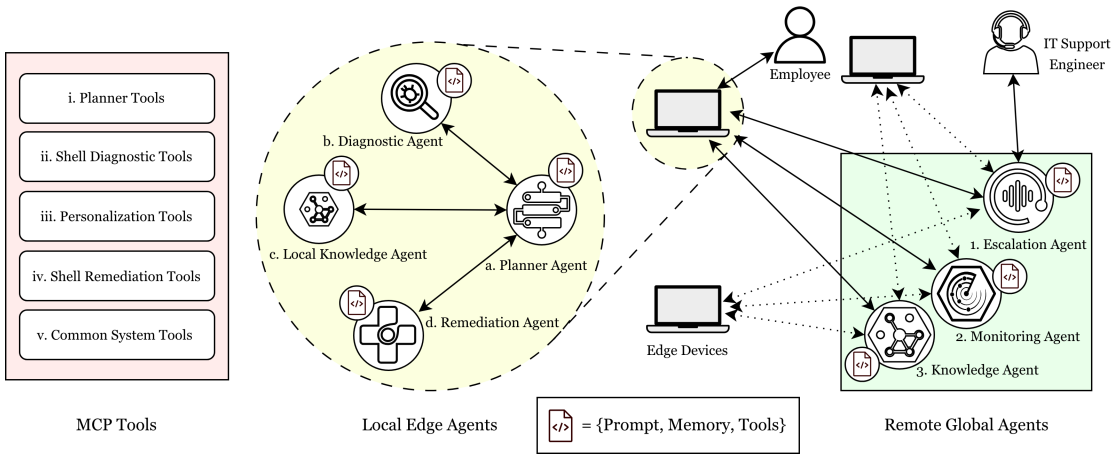

Figure 2: VIGIL's agentic architecture integrating local edge agents and remote cloud agents. Edge agents execute diagnosis and remediation on-device, while cloud agents provide coordination, governance, and observability across the fleet.

ory (Ouyang et al., 2025), retrieval strategies (Gao et al., 2023), prompts and workflows (Agrawal et al., 2025), and governance policies (Santurkar et al., 2023). This loop reflects the broader VIGIL vision and is left as future work, to be activated once sufficient safety and observability guarantees are established.

3.1 Agent Roles

VIGIL realizes the operational loop through a small set of coordinated agents running on each endpoint (figure 2). These agents are tightly integrated components of a single control system, sharing state and operating over constrained, auditable tool interfaces (Yao et al., 2023b; Schick et al., 2023; Patil et al., 2023; Wang et al., 2023).

Planner Agent. The planner agent orchestrates the local workflow. It decomposes troubleshooting goals into diagnostic steps and remediation plans, employing structured reasoning methods such as ReAct-style deliberation and multi-step reasoning (Yao et al., 2023c,a). The planner reasons over device context, episodic history, and enterprise policies, delegating evidence collection and actuation to specialized agents while keeping decision logic explicit and governable (Agent2Agent Working Group, 2025).

Diagnosis Agent. The diagnosis agent gathers structured evidence about local system state using tools exposed through the Model Context Protocol (MCP). Tool usage is constrained by purpose, preconditions, and safety annotations, consistent with modern tool-augmented language model systems (Schick et al., 2023; Patil et al., 2023). Probe selection is informed by uncertainty-aware debugging and information-gain-driven exploration (Kirsch et al., 2022; Agrawal et al., 2024; Gupta et al., 2023; Moshkovich and Zeltyn, 2025). The output is a structured diagnostic profile.

Knowledge Agent. The knowledge agent manages contextual information used throughout the loop. Locally, it maintains compact episodic memory capturing device-specific history. Remotely, it interfaces with curated enterprise knowledge and federated summaries from other devices. Together, these form a hybrid memory substrate integrating episodic, semantic, and causal structure (Anokhin et al., 2025; Cai et al., 2025; Fiorini et al., 2025).

Remediation Agent. The remediation agent executes bounded, reversible interventions such as service restarts, configuration changes, or rollbacks. Actions are validated against deterministic safety and compliance constraints, gated by user consent. Execution follows stepwise verification and self-correction paradigms drawn from recent work on automated repair and safe actuation (Bouzenia et al., 2025; Santurkar et al., 2023; Madaan et al., 2023; Zhu et al., 2023). All actions produce auditable traces.

While execution remains local, the VIGIL design includes cloud-based agents for fleet-level coordination, observability, and governance, enabling cross-device pattern discovery (Laptev et al., 2023; Yu et al., 2022) and structured escalation to human operators (Miller, 2019).

4 Implemented Edge System

VIGIL's deployed edge layer is implemented as a standalone desktop application built on top of Strands (Amazon Web Services, 2024b) and Bedrock (Amazon Web Services, 2024a) that executes diagnosis, reasoning, and remediation directly on endpoint devices.

Upon a user-reported issue, the system first invokes a bounded set of operating-system-specific diagnostic tools exposed through the Model Context Protocol (MCP) (Model Context Protocol Contributors) to collect evidence and produce a structured diagnostic profile (refer Table 1. Based on this profile, the system performs retrieval-augmented reasoning over two enterprise-curated sources: a knowledge base of IT portal articles and a repository of previously resolved cases authored by support engineers. Retrieved artifacts are assembled into a grounded context package, from which a stepwise remediation plan is synthesized and executed locally under policy control and continuous verification.

Tool	Purpose
`system_uptime`	Retrieves device uptime to identify reboots
`security_updates`	Detects pending enterprise security updates
`cpu_process`	Retrieves top running processes by CPU usage
`disk_usage`	Reports disk usage across all mount points.
`network_status`	Retrieves network connectivity status

Table 1: Illustrative diagnostic tools used in the diagnose–retrieve–remediate loop.

All state-changing actions are mediated locally by a deterministic policy engine based on Open Policy Agent (OPA) (Open Policy Agent Contributors, 2024). Generated commands are evaluated against declarative policies and classified into one of three tiers: allow, warn, or deny. Low-risk actions execute automatically; moderate-risk actions require explicit user consent with a natural-language explanation of purpose and impact; and high-risk actions are blocked. Remediation is executed incrementally with verification after each step; if health signals regress, the system adapts its plan or escalates, and where possible selects reversible actions to enable rollback.

5 Evaluation

We conducted a proof-of-value (PoV) pilot to evaluate VIGIL under real enterprise operating conditions. The evaluation combines LLM-based automated assessment of operational effectiveness with direct human evaluation through validated user experience instruments.

5.1 Deployment Setting

The pilot was conducted over a 10-week period across 100 enterprise endpoints running a single hardware and software configuration (Windows-based HP G8 devices). This device category was selected because it is among the highest drivers of IT support contacts in the organization: the devices are resource-constrained and frequently exhibit issues related to the proprietary enterprise software stack required on each machine. Participants were instructed to use VIGIL as their first point of contact for IT issues they would otherwise escalate to human support, and all interaction traces were recorded for subsequent analysis.

5.2 Operational Trace Analysis

To quantify VIGIL's impact relative to conventional IT support, we performed a retrospective case-matching analysis against a centralized graph repository (CGR) containing over 60,000 pre-resolved IT support interactions authored by human support engineers. Each VIGIL session was matched to historically similar CGR cases using a two-step process: semantic similarity via dense embeddings (threshold 0.55), followed by language-model verification of full case details (confidence $\geq$ 7/10). This yielded 826 confirmed matches spanning 60 of 153 VIGIL sessions (39%). For matched cases, we compared interaction efficiency and response time, assessed response quality across five dimensions (issue understanding, root cause accuracy, relevance, actionability, completeness) using automated evaluation, and estimated self-service potential.

5.3 User Experience Evaluation

At the conclusion of the pilot, participants completed a structured questionnaire comprising four validated instruments:

- **System Usability Scale (SUS)** (Brooke et al., 1996): a 10-item Likert scale providing a composite usability score (0–100), with an established industry average of 68.

Metric	SHIELD	Human Support
Interaction rounds (median)	11 cycles	18 turns*
Diagnosis time (median)	36.5 s	$\geq$2.7 min†
Response quality (median)	8.0 / 10	—
Self-service potential	82% (high + medium confidence)	
Tool success rate	95.3% across 1,586 calls	

Table 2: Operational metrics on matched cases (n=60). *Human support rounds reflect full conversational turns between user and support engineer. Interaction rounds were reduced by 39% (83% of cases). †Conservative lower bound assuming 10% of median contact duration (26.5 min) reflects active work; realistic estimates yield 4–17$\times$ speedup.

- **NASA Task Load Index (NASA-TLX)** (Hart and Staveland, 1988): a six-dimensional workload assessment covering mental demand, physical demand, temporal demand, performance, effort, and frustration.
- **Trust in Automation** (Jian et al., 2000): a scale measuring user confidence in automated system decision-making and reliability.
- **Technology Acceptance Model (TAM)** (Davis, 1989): items assessing perceived usefulness, perceived ease of use, and behavioral intention to adopt.

All reverse-scored items were handled according to each instrument's standard scoring procedure. Of the 100 pilot participants, 23 completed the full survey. Respondents also provided optional open-ended feedback on their experience.

6 Results and Discussion

We report results along two complementary axes: operational effectiveness of the edge component measured through trace analysis against historical IT support records, and user experience assessed through validated survey instruments.

6.1 Operational Effectiveness

Table 2 summarizes key metrics from the case-matching analysis described in Section 5. Of 153 VIGIL sessions recorded during the pilot, 60 (39%) were matched to historically similar cases in the centralized graph repository.

Instrument	Score
SUS	86.2 ± 15.7
Trust in Automation	4.29 / 5.0
TAM	4.41 / 5.0
NASA-TLX	2.53 / 7.0

Table 3: User experience scores across four validated instruments (n=23). SUS industry average is 68.0.

Efficiency. VIGIL required a median of 11 autonomous reasoning cycles to produce a complete diagnosis, compared to 18 back-and-forth exchanges between users and human agents for matched cases, a 39% reduction observed in 50 of 60 matched pairs. VIGIL's cycles are internal (tool invocation, reasoning, verification) and require no user interaction beyond the initial problem statement. Median diagnosis time was 36.5 seconds; for example, when a user reported persistent application crashes, the diagnosis agent identified memory pressure from a background process, retrieved a matching resolution, and executed a targeted service restart within this window. Even conservatively assuming only 10% of recorded human contact time (median 26.5 minutes) is active diagnostic work, VIGIL is at least 4$\times$ faster.

Response quality. Automated evaluation across five dimensions- issue understanding (8.5), response relevance (8.1), actionability (7.8), completeness (7.4), and root cause accuracy (7.1), yielded a median overall score of 8.0 / 10, with 81% of sessions rated Good or Excellent. Resolution confidence analysis estimated that 82% of cases could be resolved through self-service (39% high confidence, 43% medium), suggesting that the majority of issues handled by VIGIL would not require human escalation. Root cause accuracy was the lowest-scoring dimension, indicating an area for targeted improvement through richer diagnostic tooling and knowledge coverage.

Operational reliability. Across all 153 sessions, VIGIL invoked diagnostic tools in 95.4% of cases with a 95.3% tool execution success rate over 1,586 total calls, confirming that the MCP-based tool interface operates reliably under real enterprise conditions.

6.2 User Experience

Table 3 and Figure 4 summarize responses from 23 participants who completed the post-pilot questionnaire.

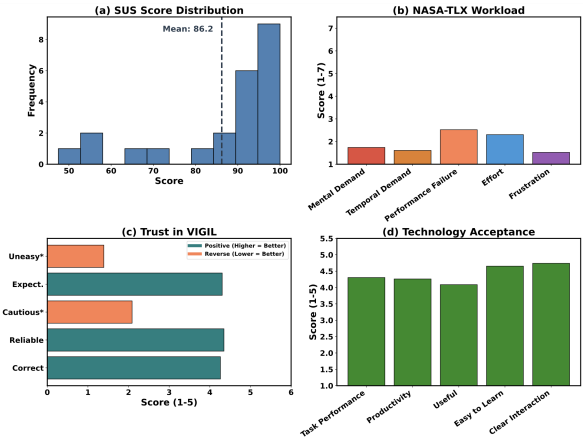

Figure 3: User experience survey: (a) SUS score distribution, (b) NASA-TLX workload, (c) user trust (bottom-left) and (d) technology acceptance

As shown in Figure 3, the SUS score of 86.2 places VIGIL 18 points above the industry average of 68.0, in the Excellent tier corresponding to the 90th percentile of evaluated systems (Brooke et al., 1996).High trust (4.29/5.0) and technology acceptance (4.41/5.0) scores indicate that users found VIGIL's recommendations reliable and believed it improved their productivity. The low NASA-TLX workload score (2.53/7.0) confirms that the system reduces rather than adds cognitive burden, a common concern with AI assistance tools.

Situated diagnosis exceeds retrieval alone. A stratified analysis revealed that the 16 respondents who experienced sessions without matching historical cases rated VIGIL higher across all instruments (SUS 89.2, Trust 4.44, TAM 4.55, TLX 2.29) compared to the overall population. This suggests that VIGIL's on-device diagnostic capabilities and transparent reasoning provide value independent of knowledge base coverage, supporting the architectural decision to invest in situated diagnosis rather than relying solely on retrieval.

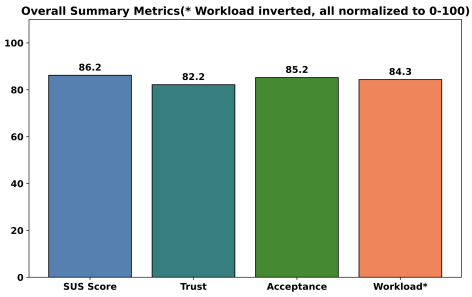

Figure 4: Normalized summary of the user experience survey across the four validated instruments (n=23)

Qualitative feedback. Open-ended responses (n=8) highlighted three themes: (1) users valued the transparency of VIGIL's actions and reasoning as critical for trust; (2) several requested elevated permissions for autonomous task completion, indicating demand for broader actuation scope; and (3) multiple respondents requested continued access and wider rollout.

6.3 Limitations

Several factors qualify these results. The case-matching analysis covers only 39% of VIGIL sessions; unmatched cases may differ systematically in complexity. Response quality was assessed via automated LLM-based evaluation rather than human expert judgment. The survey response rate (23 of 100 participants) introduces potential self-selection bias. Finally, the pilot was conducted on a single device category; generalization to heterogeneous fleets remains to be validated.

7 Conclusion

We presented VIGIL, an edge-extended agentic AI architecture for enterprise IT support, and evaluated its on-device diagnosis and remediation component through a real-world pilot. Across 100 resource-constrained endpoints over 10 weeks, the system reduced interaction rounds by 39%, achieved at least 4× faster diagnosis, and demonstrated high response quality, user trust, and low cognitive workload. Notably, users rated the system higher when no historical matches were available, indicating that on-device reasoning adds value beyond knowledge base lookup.

By colocating reasoning and actuation with the environment where failures occur, VIGIL reduces latency, preserves fine-grained context, and enables resilient operation under partial connectivity. At the same time, distributing autonomy to heterogeneous endpoints introduces architectural challenges in governance, monitoring, and deciding when local adaptations should generalize across the fleet. Although VIGIL is designed to support improvement through context refinement and prompt optimization rather than model retraining, these mechanisms were not exercised in the current deployment. Overall, our findings suggest that distributing agentic intelligence to the edge under deterministic policy control offers a viable architectural direction for scalable, governed autonomy in enterprise IT operations.

Ethical Considerations

All 100 pilot participants provided informed consent prior to enrollment. Interaction traces collected during the pilot were anonymized before analysis, with no personally identifiable information retained. The system's policy engine ensures that all state-changing actions on user devices are governed by deterministic safety constraints, with moderate-risk actions requiring explicit user approval and high-risk actions blocked unconditionally. No actions are executed without user awareness, and all agent reasoning and actions are logged transparently for auditability.

References

Deepak Bhaskar Acharya, Karthigeyan Kuppan, and B Divya. 2025. Agentic ai: Autonomous intelligence for complex goals–a comprehensive survey. *IEEE Access*.

Agent2Agent Working Group. 2025. Agent2Agent (A2A) Protocol Specification, Version 1.0. https://a2a-protocol.org/latest/specification/. Open standard for interoperability between AI agents.

Aditi Agrawal, Xuechen Li, Caiming Xiong, and Aditi Raghunathan. 2024. Deep uncertainty quantification for decision-making with large language models. In *International Conference on Learning Representations*.

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*.

Amazon Web Services. 2024a. Amazon bedrock documentation. https://docs.aws.amazon.com/bedrock/. Accessed: 2026-02-11.

Amazon Web Services. 2024b. Strands agents documentation. https://strandsagents.com/latest/documentation/docs/. Accessed: 2026-02-11.

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Mikhail Burtsev, and Evgeny Burnaev. 2025. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. In *Proceedings of IJCAI 2025*.

Anthropic. Claude code overview. https://code.claude.com/docs/en/overview. Accessed: 2026-02-08.

Ilias Bouzenia, Marcel Böhme, Cristian Cadar, and 1 others. 2025. An autonomous, llm-based agent for program repair. In *ICSE*.

John Brooke and 1 others. 1996. Sus-a quick and dirty usability scale. volume 189, pages 4–7. London, England.

Linyue Cai, Chaojia Yu, Yongqi Kang, Yu Fu, Heng Zhang, and Yong Zhao. 2025. Practices, opportunities and challenges in the fusion of knowledge graphs and large language models. *Frontiers in Computer Science*.

Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. 2025. Aiopslab: A holistic framework to evaluate ai agents for enabling autonomous clouds. *arXiv preprint arXiv:2501.06706*.

Fred D Davis. 1989. Perceived usefulness, perceived ease of use and user acceptance of information technology. *MIS quarterly*.

M. de la Cruz Cabello and 1 others. 2025. Aiops in the era of large language models. *ACM Computing Surveys*.

Sandro Rama Fiorini, Leonardo G Azevedo, Raphael M Thiago, Valesca M de Sousa, Anton B Labate, and Viviane Torres da Silva. 2025. Episodic memory in agentic frameworks: Suggesting next tasks. *arXiv preprint arXiv:2511.17775*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Pranay Gupta, Anirudh Suresh, Damanpreet Singh, and 1 others. 2023. Llm-debugger: Autonomous debugging with tool-integrated large language models. *arXiv preprint arXiv:2311.07480*.

Sandra G Hart and Lowell E Staveland. 1988. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier.

Jiun-Yin Jian, Ann M Bisantz, and Colin G Drury. 2000. Foundations for an empirically determined scale of trust in automated systems. *International journal of cognitive ergonomics*, 4(1):53–71.

Kiro. 2025. Bring kiro agents to your terminal with kiro cli. https://kiro.dev/blog/introducing-kiro-cli/. Accessed: 2026-02-08.

Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. 2022. Minimizing epistemic uncertainty in deep learning. *Advances in Neural Information Processing Systems*.

Nikolay Laptev and 1 others. 2023. Multivariate time-series anomaly detection with llm-augmented forecasting. *arXiv preprint arXiv:2310.06745*.

Liqun Li, Xu Zhang, Xin Zhao, Hongyu Zhang, Yu Kang, Pu Zhao, Bo Qiao, Shilin He, Pochian Lee, Jeffrey Sun, Feng Gao, Li Yang, Qingwei Lin, Saravanakumar Rajmohan, Zhangwei Xu, and Dongmei Zhang. 2021. Fighting the fog of war: Automated incident detection for cloud systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 131–146. USENIX Association.

Haoxiang Luo, Yinqiu Liu, Ruichen Zhang, Jiacheng Wang, Gang Sun, Dusit Niyato, Hongfang Yu, Zehui Xiong, Xianbin Wang, and Xuemin Shen. 2025. Toward edge general intelligence with multiple-large language model (multi-llm): architecture, trust, and orchestration. *IEEE Transactions on Cognitive Communications and Networking*.

Anirudh Madaan, Shuyan Liu, Amir Yazdanbakhsh, Xinyun Chen, Xi Victoria Lin, and Yuandong Zhou. 2023. Self-refine: Iterative refinement with large language models. In *Advances in Neural Information Processing Systems*.

Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*.

Model Context Protocol Contributors. Model context protocol. `https://modelcontextprotocol.io`. Accessed: 2026-02-08.

Dany Moshkovich and Sergey Zeltyn. 2025. Taming uncertainty via automation: Observing, analyzing, and optimizing agentic ai systems. *arXiv preprint arXiv:2507.11277*.

Open Policy Agent Contributors. 2024. Open policy agent. `https://openpolicyagent.org/`. Accessed: 2026-02-11.

Sherry Ouyang and 1 others. 2025. Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*. Introduces the ReasoningBank memory framework.

Shishir G. Patil, Eric Wallace Hu, and 1 others. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Philipp Reinhard and 1 others. 2024. Generative ai in customer support services: A framework for augmenting the routines of frontline service employees. `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4862940`. Accessed: 2026-02-08.

Shashank Santurkar, Esin Durmus, Faisal Ladhak, Percy Liang, and Tatsunori Hashimoto. 2023. Whose opinions do language models reflect? In *International Conference on Machine Learning*.

Ranjan Sapkota, Konstantinos I Roumeliotis, and Manoj Karkee. 2025. Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *arXiv preprint arXiv:2505.10468*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.

ServiceNow and Accenture. 2025. Bring ai to every corner of your business with servicenow and accenture. White paper. Accessed 2026-02-11.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2024. Taskbench: Benchmarking large language models for task automation. In *Advances in Neural Information Processing Systems*, volume 37. NeurIPS 2024 Datasets and Benchmarks Track.

Krti Tallam. 2025. From autonomous agents to integrated systems, a new paradigm: Orchestrated distributed intelligence. *arXiv preprint arXiv:2503.13754*.

Arthur Vitui and Tse-Hsun Chen. 2025. Empowering aiops: Leveraging large language models for it operations management. *arXiv preprint arXiv:2501.12461*.

Weizhe Wang, Lili Yu, Jason Weston, and Sainbayar Sukhbaatar. 2023. Augmenting language models with long-term memory. In *Advances in Neural Information Processing Systems*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023c. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Jing Yu and 1 others. 2022. Gdn: Graph neural networks for multivariate time-series anomaly detection. *AAAI*.

John Zhang. 2025. Agentic ai (ai agent) development guidelines and use cases (hands-on experience). ServiceNow Community (Blog). Accessed 2026-02-11.

Liang Zhu, Zhe Ye, Yingjun Li, and Jian Sun. 2023. Repairagent: Fixing software bugs with llm-based autonomous agents. *arXiv preprint arXiv:2307.10779*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. In *Advances in Neural Information Processing Systems*, volume 36. NeurIPS 2023 Datasets and Benchmarks Track.

Răzvan Daniel Zota, Corneliu Bărbulescu, and Radu Constantinescu. 2025. A practical approach to defining a framework for developing an agentic aiops system. *Electronics*, 14(9):1775.

8 Appendix

8.1 Technology Stack

Layer	Technology
Agent Framework	AWS Strands (Python)
LLM Inference	Claude Opus 4.1
Inter-Agent Protocol	Agent2Agent (A2A)
Packaging	PyInstaller
Observability	OpenTelemetry tracing
Policy Enforcement	Open Policy Agent (OPA)
Authentication	Enterprise SSO

Table 4: Core implementation components of VIGIL

8.2 Example Execution Summaries

After each session, we convert the execution trace into human-readable summaries. A few examples are provided below.

8.2.1 No Issue Case

User Request: "How can I make my computer faster?"

VIGIL Execution Trace:
1. Invoked system diagnostics:
- CPU utilization: 9%
- Disk utilization: 0.6%
- No abnormal processes detected
2. Evaluated performance indicators against policy thresholds.
3. Determined system operating within normal parameters.
4. No remediation actions executed.

Resolution: VIGIL concluded that the system was performing normally and avoided unnecessary troubleshooting or state-changing actions. This is considered a successful resolution because it correctly diagnosed "no issue" rather than applying unnecessary fixes.

8.2.2 Failure Case

User Request: "my headset mic is not working"

VIGIL Execution Trace:
1. Opened Sound Control Panel.
2. Launched built-in Audio Troubleshooter.
3. Attempted to restart audio service (failed – requires admin privileges).
4. Detected that the headset was not recognized by the operating system.

Resolution:
VIGIL determined that the issue could not be resolved remotely.
It generated a structured escalation path including:
- Manual physical troubleshooting steps.
- Request for administrative access to reinstall drivers.
- Recommendation to visit IT vending machine for hardware replacement.

8.2.3 Success Case

User Request: "My laptop doesn't respond after waking from sleep and requires a hard reboot."

VIGIL Execution Trace:
Diagnosis Phase:
1. Invoked system uptime check to detect abnormal reboot patterns.
2. Retrieved relevant enterprise knowledge base articles.
3. Verified hardware specifications, driver versions, and pending updates.
Remediation Phase:
4. Synthesized stepwise remediation plan grounded in retrieved guidance.
5. Executed bounded shell commands to adjust power management settings.
6. Disabled fast startup configuration.
7. Updated advanced power configuration parameters.
8. Re-validated system state after each step.
Execution Summary:
- Total shell commands executed: 11.
- Successful command rate: 90.9%.
- All actions executed under policy constraints with user consent.

Resolution:
VIGIL systematically diagnosed a Windows sleep/wake configuration issue.
It applied targeted power-management adjustments under policy control.
Continuous verification ensured safe execution and avoided unnecessary or high-risk interventions.

8.3 Evaluation Prompts

```
==========================
EVALUATION PROMPT A: RESPONSE QUALITY
==========================
You are an expert IT support quality evaluator. Evaluate the following
automated IT assistant (VIGIL) response.
» User Issue
{user_input}
» VIGIL's Diagnostic Process
Tools used: {tools_used}
Total cycles: {total_cycles}
Duration: {duration} seconds
Execution trace:
{trace_summary}
» VIGIL's Final Response
{response}
—
» Evaluation Criteria
Please evaluate the response on the following criteria (1-10 scale,
where 10 is excellent):
1. Issue Understanding: Did VIGIL correctly understand what the user
was asking about?
2. Root Cause Accuracy: Did VIGIL identify the correct root cause or
relevant factors?
3. Response Relevance: Is the response directly relevant to the user's
issue?
4. Actionability: Are the suggested steps clear, specific, and
actionable?
5. Completeness: Does the response address all aspects of the issue?
Also assess:
6. Resolution Confidence: How confident are you that following VIGIL's
guidance would resolve the issue?
- "high": Very likely to resolve
- "medium": Likely to help but may need follow-up
- "low": Unlikely to fully resolve
- "unable": VIGIL could not provide a resolution
Output: JSON with scores and assessment.
```

Prompt A. LLM-as-judge prompt used to score response quality for VIGIL interactions.

```
==========================
EVALUATION PROMPT B: CASE SIMILARITY MATCHING
==========================
You are an IT support case analyst. Compare the following two IT
support cases and determine if they are similar in terms of:
1. Issue type (the fundamental problem being addressed)
2. Root cause (the underlying cause of the issue)
3. Resolution actions (the steps taken to resolve, if available)
» VIGIL Case (Automated IT Assistant)
User Issue:
{user_input}
Final Response:
{response}
» CGR Case (Human Agent IT Support)
Issue Description:
{issue_description}
Chat Transcript (excerpt):
{chat_transcript}
Resolution Summary:
{resolution_Summary}
Root Cause:
{resolution_RootCause}
Resolution Steps:
{resolution_ResolutionAgent}
» Your Task
Analyze both cases and provide your assessment in the following JSON
```

```
format:
{
"is_similar": true/false,
"confidence": 1-10,
"issue_type_match": true/false,
"root_cause_match": true/false,
"resolution_match": true/false,
"reasoning": "Brief explanation of your assessment"
}
```

Prompt B. Prompt used to assess similarity between VIGIL-resolved cases and matched human support cases.

8.4 Example OPA Policies

```
==========================
DENY (Blocked - Never Execute)
==========================
1. Command: format C:
Description: Attempts to format the C: drive, erasing the system disk
including the OS, user files, and installed programs. Catastrophically
destructive and irreversible.
2. Command:
deny[msg] {
contains(input.command, "net user")
contains(input.command, "/add")
msg := "Blocked: User creation command detected"
}
Description: Blocks creation of new user accounts that could establish
unauthorized persistent access.
3. Command:
deny[msg] {
lower_cmd := lower(input.command)
contains(lower_cmd, "netsh")
contains(lower_cmd, "firewall")
contains(lower_cmd, "delete")
msg := "Blocked: Firewall rule deletion command detected"
}
Description: Prevents deletion of firewall rules that protect the system
from network threats. Captures syntactic variations to avoid bypass.
==========================
ALLOW (Execute Immediately)
==========================
1. Command: Get-Process
Description: Lists running processes with CPU/memory usage.
Read-only diagnostic command.
2. Command: systeminfo
Description: Displays system configuration details including hardware,
OS version, and network settings. Read-only diagnostic command.
==========================
WARN (Requires User Approval)
==========================
1. Command: Stop-Process -Name chrome
Description: Terminates Chrome and may close unsaved work.
Requires user confirmation.
2. Command: winget install 7zip
Description: Installs software via Windows Package Manager.
System-modifying but legitimate and reversible; requires approval.
```

Illustrative OPA-based command classification examples used in the deployed system.

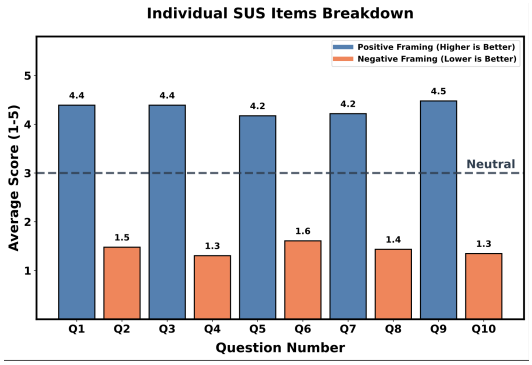

Figure 5: Item-level SUS score breakdown (1–5 scale). Positive-framed items (higher is better) and negative-framed items (lower is better) are shown separately. The dashed line indicates the neutral midpoint (3.0).

8.5 Expanded SUS Item-Level Analysis

To provide greater transparency into the System Usability Scale (SUS) results reported in the main text, we include an item-level breakdown of all ten SUS questions in Figure 5. Positive-framed items (Q1, Q3, Q5, Q7, Q9) are shown directly, where higher scores indicate stronger agreement and better usability perception. Negative-framed items (Q2, Q4, Q6, Q8, Q10) are shown in their original form (lower is better), allowing inspection of perceived complexity and friction. The neutral midpoint (3.0) is included for reference. This breakdown highlights consistency across usability dimensions and helps interpret the aggregate SUS score of 86.2. The questions are enumerated below for reference:

Q1. I think that I would like to use this system frequently.

Q2. I found the system unnecessarily complex.

Q3. I thought the system was easy to use.

Q4. I think that I would need the support of a technical person to be able to use this system.

Q5. I found the various functions in this system were well integrated.

Q6. I thought there was too much inconsistency in this system.

Q7. I would imagine that most people would learn to use this system very quickly.

Q8. I found the system very cumbersome to use.

Q9. I felt very confident using the system.

Q10. I needed to learn a lot of things before I could get going with this system.