

AI-Driven Production Support: Applying Large Language Models and Model Context Protocol to Distributed Systems Diagnostics

Makarand Gujarathi

Independent Researcher, USA

Received: 06.02.2026

Accepted: 08.02.2026

Abstract

Modern distributed microservices architectures make it difficult to perform post-mortem analysis on production outages occurring in independently managed services. LLMs using the MCP can automate this by outputting optimized database queries, identifying and aggregating correlated telemetry across service boundaries, and automatically generating root cause analysis reports. Thus, the work addresses the cognitive overheads of diagnostic tasks on distributed systems by encasing expert knowledge in an AI agent context, enabling automated multi-step diagnosis workflows that need to mimic engineer behavior. The system systematically documents schema, queries, architectural knowledge, and operational patterns in Knowledge Bases (KBs) for AI. In production, context engineering approaches reduce the incident investigation time, improve the MTTR, and convert expert diagnostic tasks into tasks executable by less knowledgeable engineers. The AI-generated queries were also exposed to engineers, providing the former with a continuous learning opportunity and allowing the engineers to upskill through passive observation. As a result, AI-assisted diagnostic work has shifted enterprise incident response from predominantly manual tasks to human-AI work, lowering the risk of bottlenecks arising from the concentration of knowledge.

Keywords: Large Language Models, Model Context Protocol, Production Support Automation, Distributed Systems Diagnostics, Observability

1. Distributed Systems Observability Challenges

Modern microservices architectures have fundamentally changed how software systems are built and deployed: rather than monolithic applications, systems are now often decomposed into a set of independent services. While these newer architectures provide several advantages, they also introduce new challenges to understanding system behavior when failures occur. The microservices pattern is based on the principle that services must have autonomy over their own data storage, deployment cycle, and operational characteristics. While this architecture increases organizational scalability and flexibility, it also creates the important challenge of observability [1].

Maintaining distributed state and tracking diagnostics pose challenges when an application is divided into multiple deployable components. Each microservice emits telemetry from its perspective, so it is necessary to correlate information between services to complete scenarios that span multiple services in an end-to-end transaction flow. Operators investigating problem scenarios during live service need to understand different schemas, navigate dependencies between services, and reconstruct the complex causal relationships between separate sources of information. This mental burden can far exceed the effort of debugging a non-distributed application, where the relevant application state is located within a single process boundary [2].

10.48047/jocaaa.2026.35.02.09

For distributed systems in production, that problem is even harder due to the complex topology of services, communication patterns, and failure propagation. It also requires abstract reasoning about how requests go through a chain of services, how faults in one service affect the responsiveness of peer services, and how system properties emerge from the services. Building mental models of distributed systems is particularly challenging in practice because each organization has different practices for service decomposition, inter-service communication, and data management. As a result, developing diagnostic expertise in these complex distributed systems can be a substantial investment for organizations [1].

The concentration of operational knowledge also creates potential fragility in production support workflows. Senior engineers, with many years of experience on specific system architectures, are often the ones with the deepest knowledge of failure modes, diagnostics, and investigation. This method provides faster incident response due to experienced personnel, but it also creates key-person dependencies and reduces organizational agility. Systems may struggle to scale with adequate operational support if the on-call team lacks individuals with critical knowledge because of a shortage of experienced engineers.

A single incident's telemetry data may be split across multiple data stores. Each may have different collection, querying, and retention behaviors. Engineers need to know which data sources to use for what types of incidents, the details for querying a given data source for evidence of an incident or problem, and how to combine observations from multiple data sources. This mechanical investigation work can take a disproportionate amount of time because it causes delays in identifying the root cause and the duration of the service outage. Since a large number of investigation work items are repetitive, this presents an opportunity for automation, provided that it can replicate the cognitive effort that engineers typically put in.

2. Model Context Protocol Architecture

The Model Context Protocol (MCP) specifies a set of standard interfaces for Large Language Model agents, enabling them to programmatically interact with different kinds of data and computation tools. The MCP provides the standardization needed to solve the basic problems of integrating AI systems with enterprise data infrastructure. In contrast, existing solutions require a custom code for each data source and incur an active maintenance fee. The protocol abstracts the implementation details for each data source and exposes only the capabilities that AI agents need. This means a single agent implementation works with relational databases, document stores, time-series databases, and other data sources [3].

Data-intensive applications are defined by many aspects, including the volume of data to be handled, the efficiency of their query processing, and consistency in the management of distributed application state. Key challenges in data management systems involve schema evolution, query scalability, and consistency in distributed storage systems. These architectural choices directly impact the way that agents interface with enterprise data infrastructure, as strategies for producing a given data query are intimately tied to underlying properties of the data system itself (such as partitioning schemes, indexes, and database optimizations). AI agents can generate queries that accurately operate against production databases of any size by utilizing these data system properties [3].

Large Language Models have achieved impressive results in few-shot learning and can be prompted to complete new tasks given examples in prompts. Thus, large language models are suitable candidates for database query generation, where agents can learn organization-specific query patterns given a library of examples. These models suggest that the agents possess pattern-matching capabilities, enabling them to

10.48047/jocaaa.2026.35.02.09

identify structural similarities between new diagnostic situations and previously encountered ones, which facilitates the generalization of examples to other situations and queries involving previously unseen incidents. Consequently, few-shot learning eliminates the need for extensive model tuning for organization-specific data, thereby simplifying implementation and speeding up adaptation to new system designs [4]. Combining a set of protocols with the capabilities of large language models provides the basis for powerful abstractions for production support automation. When equipped with protocol-based data access, AI agents will be able to efficiently probe for relevant data in large, complex information spaces in terms of relevant sources and types of queries. This standardization of the protocol allows the agent implementation to be the same regardless of what data source it is accessing, while still allowing for the particulars of the data sources to be separate. This enables a broader coverage of data sources without necessitating modifications to the agent [4].

Autonomous query generation goes beyond existing SQL code completion tools, which generate SQL snippets based on partial input, because AI agents autonomously generate entire investigative strategies as sequences of interdependent queries that can answer diagnostic questions. Using reasoning, they plan for the information needed, the sequencing of the queries, and how individual queries can be adjusted based on previous responses, in a way similar to how skilled engineers use knowledge incrementally via forming and revising hypotheses. Shifting from hand-coded queries to AI-assisted investigation changes the nature of production support in the era of LLMs.

Integration Component	Traditional Approach	Model Context Protocol
Interface Design	Custom code per source	Standardized primitives
Data Source Coverage	Single implementation	Multiple sources
Schema Discovery	Manual documentation	Automated introspection
Query Execution	Source-specific syntax	Universal interface
Maintenance Overhead	High per source	Centralized
Agent Adaptability	Limited flexibility	Seamless extension

Table 1: Protocol Architecture Components and Capabilities [3, 4]

3. Context Engineering Methodologies

Additionally, prompt engineering involves designing context, structuring documents, and systematically gathering examples so that organizations can develop knowledge bases that detail system architecture, principles of operation, and troubleshooting procedures in formats that are digestible for use by an AI. Prompt engineering refers to formulating prompts to convey information clearly, succinctly, and in a structured format that provides agents with sufficient context to perform their tasks without overwhelming them with irrelevant information. Balancing between providing enough context and being concise is a key challenge in context engineering [5].

Context documentation for production support agents includes the knowledge types, or dimensions, needed to carry out diagnostic reasoning tasks. Schema documentation consists of the database schema, field semantics, and table relationships needed to make valid queries. The knowledge that is used for this purpose includes architectural knowledge, such as relationships between services and communication

10.48047/jocaaa.2026.35.02.09

characteristics, and operational knowledge, such as known failures and symptoms and efficient diagnostic strategies. Such knowledge will allow agents to reason using pattern libraries when experiencing similar

situations. Knowledge dimensions are organized to provide a full context for autonomous investigation capabilities [5].

Prompt pattern catalogs are systematic guides for designing effective prompts for different applications, a collection of common sense, heuristics, and design patterns that have been shown to elicit specific behaviors in language models like structured reasoning, iterative refinement, and constraint satisfaction. Production use cases include few-shot learning patterns that provide queries with examples of correct formatting and strategies, chain-of-thought patterns that guide reasoning about diagnostics, and output structuring patterns that mandate consistent formatting of model outputs. Organizations can leverage existing prompt patterns to quickly construct context using tested templates tailored to an organization's requirements [6].

People must consider tradeoffs between coverage, diversity, and the quality of example queries when building a library of example queries. On the one hand, it needs coverage of common types of diagnostic tasks and diverse querying methods. On the other hand, it should be efficient, both in terms of practical query optimizations, and well-documented. Such examples serve a dual purpose. They both provide training patterns for agents and serve as documentation of best practices for human programmers. Organizations maintain example libraries by adding new patterns through operational experience when new kinds of incidents occur and by improving older examples as query optimization techniques improve [6].

Domain-specific context engineering balances the generalist nature of AI technology against the context of specific domains. While large language models have a wide context, specific domains require the modeling of contexts in domain-specific operational environments, such as systems, practices, and the knowledge of organizations. Context engineering identifies the gaps between knowledge of the model and knowledge of the new domain and fills in those gaps with examples. The agent can now perform tasks in the new domain while retaining the flexibility to reason like a general-purpose model.

Knowledge Dimension	Content Type	Purpose	Example Applications
Schema Documentation	Database structures, field semantics	Enable valid query formulation	Table relationships, data types
Architectural Knowledge	Service relationships, communication patterns	Understand system topology	Dependency mapping, data flows
Operational Patterns	Common failure modes, symptom signatures	Pattern-based diagnosis	Incident classification, hypothesis generation
Query Libraries	Example queries, optimization strategies	Demonstrate best practices	Join patterns, partition pruning
Domain Context	Organization-specific conventions	Bridge knowledge gaps	Internal terminology, system naming

Table 2: Context Engineering Knowledge Dimensions [5, 6]

4. Autonomous Investigation Workflows

Planning agents that act coherently are an important enhancement to the capabilities of artificially clever systems. By linking reasoning to actions, agents can decide what to do, do it, and revise future plans based on the consequences of their actions. The reasoning-acting coupling enables agents working in domains of diagnosis to revise hypotheses more freely by deciding what evidence to collect next, rather than following strict investigative scripts. There is a good deal of flexibility in how an agent proceeds, given the current evidence status and its diagnostic objectives. This autonomous decision-making allows for adaptively choosing an investigation strategy based on the characteristics of the incident [7].

To solve a diagnosis problem, an agent makes a selection. It does this by analyzing the incident description for interesting system parts, making initial hypotheses about the cause, and deciding which questions to ask. This structured problem decomposition directly reflects the cognitive processes expert diagnosticians use when analyzing failures outside their experience base. An explanation of the reasoning traces generated by agents during an investigation reveals their diagnostic decision-making and allows the human engineers to validate agent conclusions. Making agent reasoning processes interpretable increases trust and human-AI collaboration during the incident response [7].

Language models can be trained and prompted to use external tools rather than needing to be explicitly taught what to do with them; such tool usage can be learned effectively through examples and experimentation. There are production support applications, such as database query engines, log analysis systems, and metric visualization systems. An agent needs to learn to choose the right tool, tune parameters, and interpret the tool's output. Self-teaching allows agents to quickly adapt to new tools as the organization continues to refine its observability infrastructure without wide-ranging manual retraining [8].

Multi-system correlation in diagnostic systems allows for the correlation of evidence with multiple, disparate sources across service boundaries. Agents should be capable of identifying correlation keys between related events. They should also be resilient to clock skew and network delay and be able to recognize cause-and-effect relationships through timing relationships. In the correlation step, queries are sent to multiple back-end data sources. Timestamps are normalized for the synchronization offsets. Explanations of the root cause of the incidents are provided. To make the process work, agents must have a good understanding of the architecture, how various services interact with failures, and how the outputs of different services impact the overall system behavior [8].

The iterative nature of independent investigation workflows permits diagnostics to be progressively sharpened. Initial questions can be used to collect high-level information that provides situational awareness. Hypotheses are formed, and analysis of initial findings leads to asking further questions to challenge other plausible explanations. When sufficient evidence is observed, agent confidence is directed towards root causes. Explanations must be confirmed through supporting evidence from several sources. The refinement process allows analysis with little knowledge of the underlying cause, based on what is known from the original description of the incident. Unlike structured methods in which the procedures written into the program fail if the results are unexpected, AI-based methods adjust the investigation method automatically based on earlier results.

Workflow Stage	Agent Activity	Output Generated	Evidence Sources
Problem Decomposition	Analyze the incident description	System components identified	Incident metadata
Hypothesis Formation	Generate potential causes	Competing explanations	Architectural context

10.48047/jocaaa.2026.35.02.09

Initial Query Generation	Retrieve baseline telemetry	High-level system state	Multiple databases
Evidence Analysis	Evaluate query results	Refined hypotheses	Telemetry patterns
Correlation Execution	Join multi-system data	Transaction flows reconstructed	Cross-service logs
Validation Queries	Test hypothesis validity	Supporting evidence	Targeted data extracts
Root Cause Synthesis	Integrate findings	Comprehensive diagnosis	All evidence sources

Table 3: Autonomous Investigation Workflow Stages [7, 8]

5. Knowledge Democratization Impact

Automation and technological changes continue to impact the work of the entire professional workforce. Artificial intelligence, in particular, has the potential to disrupt work skills, organizational structure, and value creation. To understand human-AI collaboration, it is critical to investigate which tasks can and should be automated, how expertise changes in collaboration with AI systems, and how work organizations can balance human and AI capabilities. The financial services sector is exemplary for studying the potential of technological transformations, such as in algorithmic trading or automated analytics supporting human decisions in high-velocity environments [9].

In this way, democratization of expert capabilities through AI assistance could permit more workforce participants to engage in specialized tasks that currently require many years of training to develop mental models of the system architectures, diagnostic strategies, and questioning techniques appropriate to these production support tasks. AI agents that represent expert knowledge allow less expert engineers to diagnose problems in ways that were previously only possible for more experienced engineers by monitoring their actions and guiding them through the sequence of events or even performing the mechanical investigations automatically. Senior engineers concentrate on developing original solutions, while AI systems follow standard protocols for performing computations with data [10].

AI agent capabilities are expressed transparently in the generated output. This continual learning process can enhance human capabilities and accelerate their development. For instance, agents that generate database queries for diagnostic workflows provide transparent examples that engineers can use in their own work. This observational learning provides context and concrete examples of how to apply techniques. Engineers benefit from learning best practices for query optimization, join patterns, and diagnostics by exposing them to positive cases with these techniques repeatedly. This learning happens in the normal flow of work and is not done in isolation from operational work [9].

Change management challenges for companies embracing AI-assisted workflows include scaling skills, integrating roles, and building trust. Engineers initially reluctant to work on AI projects will only feel comfortable integrating AI agents with human collaborators once these agents show dependability. The explicit and interpretable nature of agent reasoning builds trust as humans are able to trace the reasoning and understand the agent's logic. Over time, patterns of collaboration will emerge as engineers learn how to instruct the AI agents, interpret their output, and contextualize the outcome within the overall investigation. These patterns of collaboration create new functionalities and capabilities, which are not achievable by humans or AI alone [10].

10.48047/jocaaa.2026.35.02.09

The long-term effect of democratizing AI is not just efficiency but a more resilient and adaptable organization. Common diagnosis avoids key person dependencies and the fragility that comes with them. Engineers may expand the in-band on-call capacity as they grow more comfortable using AI supplements to manage production issues. Well-documented patterns in the context of agents provide an organizational memory that survives attrition, thus helping to reduce the loss of institutional knowledge. These structural improvements enable the organization to better manage system complexity as architectures change and operational contexts grow.

Impact Category	Traditional Model	AI-Assisted Model	Improvement Area
Learning Curve	Extended timeline	Accelerated development	Skill acquisition speed
On-Call Capacity	Limited rotation	Expanded participation	Operational coverage
Knowledge Distribution	Senior engineer concentration	Broader team capability	Risk reduction
Concurrent Investigations	Sequential handling	Parallel execution	Incident response capacity
Skill Transfer	Documentation-based	Observational learning	Learning effectiveness
Organizational Memory	Individual-dependent	System-encoded	Knowledge retention

Table 4: Knowledge Democratization Impact Metrics [9, 10]

6. Practical Advancement in Enterprise Systems Engineering

High-performing technology organizations have unique capabilities that enable them to deploy and release software quickly and reliably while maintaining a high level of stability in operations. These unique capabilities include technical capabilities (such as continuous integration and automated deployment), architectural capabilities (such as modularity and testability), organizational capabilities (such as cross-functional teams), and cultural capabilities (such as experimentation and learning from failure). The scientific study of these practices has identified a variety of evidence-based capabilities, particularly within the field of software delivery performance [11].

Integrating AI capabilities into production support workflows is an application of the general DevOps principles of automation, measurement, and continuous improvement. Automation reduces toil, allowing engineers to spend more time on higher-value work like proactively improving reliability rather than reactively searching for the cause of incidents. Data-driven impact can be assessed through metrics on diagnostic precision, time to solve, and the distribution of capabilities, while cycles of iterative development on agent context and query libraries provide feedback loops for long-term capability improvement. These practices support the building of high-performance technology organizations that are incorporating AI as a collaborative partner in workflows [11].

An important part of continuous delivery is the ability to reliably and repeatedly deploy code changes to production. For continuous delivery, this means that the build, test, and deployment process is fully automated to create deployment pipelines. The same principles apply to AI-based diagnostic agents, where

10.48047/jocaaa.2026.35.02.09

consistent context engineering, prompt design, and agent validation can lead to strong incident investigation capabilities. Organizations that embrace AI-assisted workflows can use the tenets of continuous delivery when upgrading agent versions (context updates, prompt design) and treat them as software updates that require testing and validation before deployment [12].

Reliability requirements for production support automation are often higher than for other AI use cases. As well, faults and inaccuracies in AI-generated diagnostics may lead investigators astray, meaning longer outages and worse user experience. Any organization using agents for production tasks will need processes to ensure the accuracy and reliability of said agents, provide fallback mechanisms if the agent can no longer complete its task, and monitor agent performance to recognize when it has started to degrade. The engineering of these processes follows similar patterns to other production systems [12]. The transition to agent-augmented engineering workflows is underway, but organizations still explore new integration patterns as agent capabilities and collaborative workflows improve thanks to amassed experience and continuous improvements to the underlying technology. Use cases with relatively simple diagnostics where success or accuracy can be defined, and where there are low costs associated with incorrect results, are often typical starting points, building confidence for more complex cases. In this way, organizations can begin to realize value using AI support with low risk, then progress towards more advanced ways of human-AI collaboration.

Conclusion

AI-assisted EASEled (Enterprise Systems Engineering for incident analysis) systems are transforming incident diagnosis workflows from wide-ranging human expertise to augmenting human capability: The combination of structured knowledge from large language models and the Model Context Protocol, along with real-time telemetry data, enables autonomous investigation workflows that perform at the same level as expert engineers. Context engineering approaches formalize the organizational knowledge of experts into machine-consumable contexts. This enables production-ready database queries, cross-service evidence correlation, and full root cause analysis. This democratization removes knowledge silos and increases the call capacity of on-call engineers while also accelerating the learning of junior engineers through observational learning. Deployments into production with these capabilities achieve competitive improvements in MTTR while being supported by a structured approach to validation and monitoring. These architectures and integrations provide foundational concepts for deploying AI in support workflows. As companies improve patterns of human-AI collaboration and agent capabilities, they will have a sustained means of scaling operational support to cope with the growing complexity of their systems and demands across their distributed platform environments.

References

- [1] Sam Newman, "Building Microservices, 2nd Edition," O'Reilly Media, 2021. [Online]. Available: <https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/>
- [2] Niall Richard Murphy et al., "Site Reliability Engineering: How Google Runs Production Systems," O'Reilly Media, 2016. [Online]. Available: <https://www.oreilly.com/library/view/site-reliabilityengineering/9781491929117/>
- [3] Martin Kleppmann, "Designing Data-Intensive Applications," O'Reilly Media, 2017. [Online]. Available: <https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/>
- [4] Tom B. Brown et al., "Language models are few-shot learners," arXiv:2005.14165, 2020. [Online].

10.48047/jocaaa.2026.35.02.09

Available:

<https://arxiv.org/abs/2005.14165>

- [5] Claude Documentation, "Prompt engineering overview." [Online]. Available: <https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview>
- [6] Jules White et al., "A prompt pattern catalog to enhance prompt engineering with ChatGPT," arXiv:2302.11382, 2023. [Online]. Available: <https://arxiv.org/abs/2302.11382>
- [7] Shunyu Yao et al., "ReAct: Synergizing reasoning and acting in language models," arXiv:2210.03629, 2023. [Online]. Available: <https://arxiv.org/abs/2210.03629>
- [8] Timo Schick et al., "Toolformer: Language models can teach themselves to use tools," arXiv:2302.04761, 2023. [Online]. Available: <https://arxiv.org/abs/2302.04761>
- [9] Xiaojing Dong and Shelby H. McIntyre, "The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies," 2014. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/14697688.2014.946440?utm_medium=article&utm_source=researchgate.net
- [10] Thomas H. Davenport, "Only Humans Need Apply: Winners and Losers in the Age of Smart Machines," Leaders Excellence Presentation, 2016. [Online]. Available: https://leadersexcellence.com/wp-content/uploads/dlm_uploads/2016/08/Davenport-Leaders-Excellencepresentation.pdf
- [11] Nicole Forsgren, Jez Humble, and Gene Kim, "Accelerate: The Science of Lean Software and DevOps," IT Revolution Press. [Online]. Available: <https://ebooks.karbust.me/Technology/Accelerate%20The%20Science%20of%20Lean%20Software%20and%20DevOps%20Building%20and%20Scaling%20High%20Performing%20Technology%20Organizations%20by%20Nicole%20Forsgren%20Jez%20Humble%20Gene%20Kim.pdf>
- [12] David Farley and Jez Humble, "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation," O'Reilly, 2010. [Online]. Available: <https://www.oreilly.com/library/view/continuous-delivery-reliable/9780321670250/>