Agentic AI

Cost Optimization & Performance Control

Boris Zibitsker

BEZNext

Published by BEZNext
www.beznext.com

Acknowledgments

I would like to express my deepest gratitude to all the people who contributed to the development of this book, supported my research, and inspired the ideas presented here.

My sincere thanks go to Dr. Alex Lupersolsky, my long-time collaborator and co-author on multiple research papers. His profound expertise in modeling and optimization has shaped many of the concepts in this work.

Special thanks to the engineers at BEZNext for developing software tools and continuously pushing the boundaries of automated observability and modeling, helping customers optimize costs and performance. Their creativity and commitment made the practical examples in this book possible.

I am grateful to Dan Graham, Dr. Alex Podelko, Dr. Alex Gigur, Dan Kaberon, Mark Friedman, Pekka Kostamaa, Dr. Dominique Heger, Richard Winter, Norbert Kremer and Dr. Yuri Balasanov for reviewing the draft of this book and friends in the ICPE, SPEC Research Group, and CMG communities, whose discussions, papers, and workshops provided valuable insights into the challenges organizations face when implementing and optimizing modern AI and cloud systems.

I am also grateful to the many professionals from Fortune 1,000 companies who trusted us with their IT performance and cost challenges. Their real-world use cases inspired much of the applied guidance and examples included here.

Finally, I want to acknowledge the unwavering support of my wife, Ella; my daughter, Julia; and my grandchildren, Curtis and Jackson.

To everyone who contributed - directly or indirectly - thank you!

Preface

Agentic AI represents a profound shift in how organizations design, manage, and optimize business processes. While promising unprecedented automation and intelligence, these systems introduce new challenges in performance management, workload management, resource allocation, and cost control - areas that traditional FinOps and monitoring practices are not fully prepared to address.

This book was written to close that gap.

Over the past several years, I have had the privilege of working with global enterprises, helping optimize strategic and tactical decisions and focus on continuously meeting business performance goals at the lowest cost.

These experiences underscored the need for a more rigorous framework and a data-driven approach to understand agentic workloads, combining observability, workload

characterization, queueing network modeling, and gradient optimization to enable continuous cost and performance control.

Objective of the book

In this book, we present a framework and review examples of iterative queueing network modeling and gradient optimization. A primary goal in managing agentic AI systems is to achieve business process performance targets at the lowest possible cost. We review a framework and examples of closed-loop cost optimization and performance control.

What you will find in these chapters is both practical and technical:

- A unified framework for analyzing, optimizing, and managing agentic AI systems
- A modeling and optimization approach that automates observability and improves agentic AI cost optimization and control decisions
- Examples drawn from Snowflake, Databricks, Teradata Vantage, BigQuery, Synapse, Oracle, and other platforms supporting agentic AI systems
- Overview of the modeling and optimization technology used to evaluate options and develop cost optimization and performance control recommendations
- Recommendations for building closed-loop cost and performance control applications for agentic AI.

We discuss how BEZNext technology enhances the FinOps, PlatformOps, DevOps, and DataOps frameworks by automating observability, applying modeling and optimization, and enabling closed-loop cost optimization and performance control.

It includes collecting performance and cost metrics for agents across the agentic AI infrastructure layers that host the orchestration, RAG, LLM, MCP and Vector Store platforms and use of closed queueing network models and gradient optimization to evaluate options, optimize costs and performance decisions, and set realistic expectations. We review several examples of platform and configuration selection, organizing dynamic capacity management, and sizing new applications.

We also review the results verification, comparing actual costs and performance with expected/predicted values, and organizing a closed-loop performance and cost control.

Intended Audience

This book is intended for IT, business, finance, and operations executives, architects, and other professionals concerned with the cost, performance, and governance of agentic AI systems, including:

- **Executive and business leaders** seeking predictable cost, performance, and risk control for agentic AI investments.
- **Enterprise architects and IT leaders** responsible for platform selection, sizing, and scaling agentic AI architectures.

- **Engineering managers and developers** building autonomous agents who need early feedback on cost, performance, and scalability.
- **FinOps, PlatformOps, DevOps, and DataOps teams** adapting operational practices to non-deterministic, agent-driven workloads.
- **Operations and site reliability teams** responsible for day-to-day performance, availability, and incident response in agentic AI environments.
- **Anybody interested in how models and optimization work under the hood and curious about modeling and optimization technology.**

The approach presented in this book improves collaboration across business, financial, and technical teams. It helps organizations **better understand how complex agentic AI systems actually behave**, align decisions across roles, and **reduce the risk of performance and financial surprises** as agentic AI systems scale into production.

Boris Zibitsker

bzibitsker@beznext.com
Chicago, 2025

Contents

1 Introduction

Over the past couple of years, ChatGPT, Gemini, DeepSeek, and other AI tools have become integral to our lives, demonstrating the value of AI. This has motivated vendors, venture capitalists, and organizations to invest heavily in agentic AI to transform business processes to levels never seen before. Most organizations have initiated agentic AI projects. Some of them deployed several agents and have high expectations from agentic AI projects.

1.1 Life cycle of an agentic AI project

The life cycle of an agentic AI project is fundamentally different from that of traditional software, analytics, or even machine-learning systems. Agentic AI systems are autonomous, goal-driven, and adaptive, which means they must be designed, deployed, and operated as closed-loop systems rather than as one-time implementations.

At a high level, an enterprise-grade agentic AI system arises from the synergy of three pillars:

- **Intelligence** (LLMs, SLMs, reasoning and planning),
- **Knowledge** (retrieval-augmented generation, memory, vector stores),
- **Infrastructure and Control** (orchestration, observability, security, and operations).

Because agents continuously plan, act, and adapt, their life cycle is **iterative** and **ongoing**, not linear.

The challenge is to reduce uncertainty and risk of cost and performance surprises during all phases of the agentic AI project's life cycle.

1.1.1 Strategic planning and problem framing

The project's life cycle begins by defining the objectives and constraints for each business process. This phase establishes:

- Agents that will support each business process
- Agents' success criteria (business KPIs, service-level goals, cost boundaries).

This step defines the "business logic" that agents will later implement and enforce.

1.1.2 System and agentic architecture design

The agentic system is designed as a distributed system rather than a single AI model. Key decisions include:

- Agent roles and responsibilities (planner, executor, verifier, optimizer)
- Single-agent vs. multi-agent topology
- Orchestration and coordination mechanisms
- Choice of LLMs and smaller, task-specific models

- Knowledge access patterns (RAG, memory, tools, APIs). It covers : **What the agent knows** (RAG/Memory) and **What the agent can do** (Tools/APIs).

At this stage, the architecture explicitly defines **how the agents reason, interact, and act**, and how their behavior can be controlled and observed.

1.1.3 Knowledge, data, and environment preparation

Agents are only as reliable as the environment in which they operate. This phase prepares:

- Enterprise documents, APIs, and systems the agents will use
- Knowledge pipelines for retrieval and grounding
- Rules governing what agents are allowed to access or execute
- Sandbox environments for safe experimentation.

This step ensures agents operate on a trusted, explainable "ground truth" rather than on hallucinated or uncontrolled inputs.

1.1.4 Agent development and behavior engineering

Agents are implemented by defining behavior, not just by prompts. This includes:

- Role-specific reasoning strategies
- Planning and reflection loops
- Tool-use policies
- Retry, fallback, and escalation logic
- Explicit limits on recursion, concurrency, and cost.

This phase treats agent design as behavior engineering, analogous to designing control logic for complex systems.

1.1.5 Testing and evaluation in an agentic sandbox

Traditional unit tests are insufficient for autonomous agents. Instead, testing focuses on the following:

- Trajectory validation: verifying not just outcomes, but the reasoning and action paths agents take
- Failure and adversarial testing to ensure agents respect business rules and security boundaries
- Automated evaluation using reference tasks, simulations, and judgment criteria.

The goal is to validate correctness, safety, and predictability, not just accuracy.

1.1.6 Deployment and operational readiness

Deployment introduces agents gradually into production environments with:

- Controlled autonomy
- Human-in-the-loop supervision
- Strict limits on actions and spending

- Continuous logging and tracing.

This phase brings together multiple operational disciplines: infrastructure, observability, and cost management to ensure the system remains stable and governable.

1.1.7 Continuous optimization and closed-loop control

Once deployed, agentic AI systems must be continuously monitored and optimized. Actual behavior is compared against expected performance and cost to:

- Detect anomalies and inefficiencies
- Tune agent strategies and model selection
- Adjust capacity, concurrency, and routing decisions
- Prevent runaway costs or unstable behavior.

This closed-loop observe, model, optimize, and control is the core of sustainable agentic AI.

1.1.8 Evolution and scaling into an agentic ecosystem

Agentic AI projects are never "finished." Over time, organizations:

- Add specialized agents for new tasks or domains
- Expand multi-agent collaboration across departments
- Improve local or task-specific models using observed agent behavior
- Refine governance and autonomy boundaries.

At this stage, the project evolves into an enterprise agentic ecosystem that supports multiple business processes with shared infrastructure and control.

1.2 Agentic AI transforms business process management

Agentic automation is revolutionizing business process management by moving beyond simple automation to autonomous, goal-driven actions that adapt to changing business conditions. These systems can break down complex objectives into smaller tasks, execute them, and learn from the results. For example:

- For Customer Facing & Service Operations agentic AI enhances customer experience by providing more intelligent, end-to-end support
- For Supply Chain and Logistics agents, it brings resilience and efficiency to complex, dynamic supply chains
- For Financial and Administrative Functions, agentic AI drives efficiency and improves accuracy in core back-office processes, including Finance and Accounting, Compliance Monitoring, and Human Resources (HR)
- For IT and Software Development, agents accelerate development cycles and improve system efficiency.

Each business process has specific performance requirements for response time and throughput. These requirements can be translated into specific, measurable requirements, including:

- **Latency goals** (e.g., response < 2s for customer queries, <200ms for retrieval calls)
- **Throughput goals** (e.g., 100 requests/sec sustained)
- **Cost efficiency goals** (e.g., <$0.01 per query or <$X per business process per month)
- **Reliability goals** (availability %, error/failure rate).

We will call the business process requirements Service Level Goals (SLGs) in contrast to Service Level Objective (SLO), Service Level Indicator (SLI), and SLA (agreement, contractual obligation), terms often used in industry.

A 2025 Google Cloud study found that 52% of enterprises have successfully deployed AI agents in production environments across various industries. Most of these implementations use a small number of agents.

Currently, the main focus in deploying agentic AI is on the trustworthiness of the results. Because agents are autonomous and non-deterministic (they make their own choices), the evaluation criteria for agentic AI systems must include reliability, reasoning quality, ethical governance, cost per task (CPT), latency, accuracy and error rate, security, and stability.

But this is just the beginning of the agentic automation revolution. According to Gartner and other industry analysts, the expansion of intelligent process automation and AI agents will continue to accelerate throughout this decade. This implies rapid growth in the number of business processes managed by agentic automation and in the number of autonomous agents deployed across enterprises.

1.3 Key takeaways

- **From Static to Autonomous:** agentic AI moves beyond traditional automation to autonomous, goal-driven systems that reason, act, and adapt to changing business conditions
- **Iterative Lifecycle:** unlike linear software projects, agentic AI lifecycles are continuous and iterative, requiring ongoing planning, behavior engineering, and scaling
- **The Operational Challenge:** the primary hurdle is not simply deploying agents, but operating them economically and reliably on a scale
- **Defining SLGs:** success requires translating business needs into measurable SLGs, including latency, throughput, and cost efficiency
- **Trajectory-Based Testing:** Testing must focus on validating reasoning paths and action trajectories rather than just simple outcomes

2 Agentic AI Architecture Overview

Agentic automation uses information about business processes to optimize and execute timely business process management decisions [2,3,4].

An agentic AI system consists of several functional layers and components that work together to achieve business objectives autonomously:

- Action/Execution layer (integrating business APIs, databases, and tools)
- Orchestration layer – responsible for task decomposition, planning, coordination among agents, and managing workflows
- LLM layer – provides reasoning, language understanding, and decision generation capabilities through large and small language models (LLM/SLM)
- MCP (Model Context Protocol) – manages communication, synchronization, and policy enforcement among agents, ensuring consistency, safety, and resource optimization
- Vector store layer – maintains long-term memory and supports retrieval-augmented generation (RAG) by storing and supplying relevant embeddings and contextual information
- Observation & Optimization layer (monitoring SLGs, cost, and performance).

Together, these layers enable goal-oriented reasoning, adaptive planning, and continuous optimization across business processes. The orchestration layer processes user and application requests, triggering one or more agents to handle them. Agents process requests concurrently and often revisit the same layer multiple times.

Agentic AI typically operates within a multi-layer architecture (Figure 2.1). During processing, a single request may pass through RAG (Retrieval-Augmented Generation), LLM / SLM reasoning layers, vector stores and other data or tools via Model Context Protocol (MCP) servers multiple times, depending on the task's complexity.

Each business process is typically supported by multiple agents working together. Therefore, the overall end-to-end response time across all agents managing a workflow must meet the performance SLG for that business process.

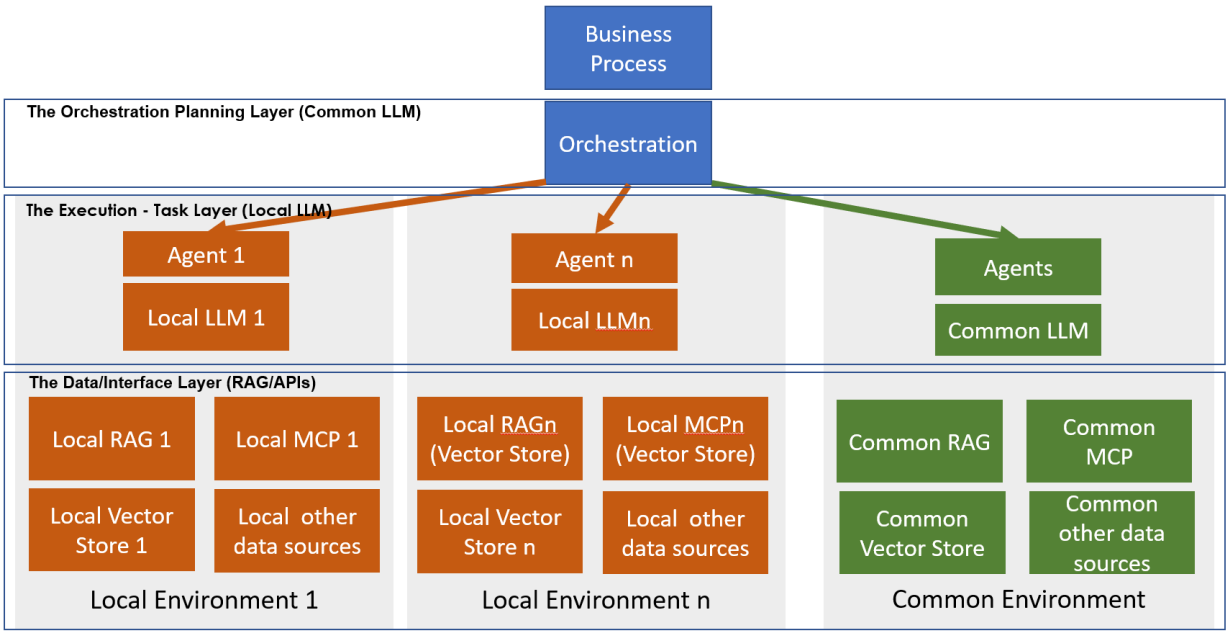


Figure 2.1 Components of the agentic AI enabling agentic automation

Users' and applications' requests are handled by orchestration scheduling agents to access common or local resources.

Local LLM, RAG (Vector Store), are dedicated to a single agent or a small group of agents with the same function.

Common LLM and RAG resources are shared across the entire fleet of agents and managed by a central orchestrator or a "Supervisor Agent." These resources typically fall into four main categories:

- Shared Knowledge Bases (**Common RAG/Memory**), including Centralized RAG (Vector Database and Long-Term Memory / Blackboards
- Computational Resources, including **Common LLM and Orchestrator Agent** for complex planning, reasoning, and task decomposition.
- External Tools and APIs, Enterprise APIs through standardized interfaces like the **MCP server** allow agents to interact with external business systems (e.g., Salesforce, ERP, or a payment processor)
- **Authentication/Authorization Service** manages the credentials and permissions for all agents, ensuring they only access the data and tools they are authorized for
- **Environment and Coordination Mechanisms**: Virtual space and rules govern agents' interactions; Communication Channels (Message Queues) enable asynchronous communication, allowing agents to work in parallel without waiting on each other; Task Allocator: a central agent or service that manages the queue of incoming user tasks and allocates them to the most suitable specialized agent based on current load and capability.

2.1 Key factors affecting agentic ai performance and cost

This section describes the primary factors affecting the response, throughput and costs of running AI agents supporting business processes.

Agentic AI architecture dramatically expands the variability of performance and cost. Agentic AI performance is often unpredictable. Unlike traditional software that follows a fixed scenario, agents make their own decisions on fly, such as when to retry a step or search for more data. Consequently, system performance depends heavily on how well the infrastructure handles the agents' autonomous behavior.

Many factors affect the cost and performance of agentic AI systems. Effective governance requires automated observability and the ability to manage platforms, configuration, scalability, and workload management to enable continuous workload optimization and resource allocation.

2.1.1 Agentic ai workload management decisions

Agents' workload management decisions affect concurrency and priorities, taking into consideration:

- **Compound Interaction Volume**: Unlike traditional linear applications, agentic AI operates continuously and autonomously. A single user's "transaction" can trigger a cascade of actions: multiple LLM calls, vector database queries, external API requests, and recursive validations resulting in dozens of backend interactions for one user goal
- **Resource Contention**: Poorly managed agents can spawn uncontrolled processes, leading to high concurrency. When too many agents compete for shared resources (compute, memory, I/O), performance degrades sharply
- **Recursion and Loops**: Agents often utilize retry mechanisms and recursive logic to solve problems. Without strict controls, this can lead to "runaway" resource consumption and infinite loops
- **Task Variability**: Performance profiles vary significantly by task type; RAG-based retrieval, code generation, and complex reasoning chains each impose different loads on the system
- **Orchestration Overhead**: Recurring costs from recursive agent calls, autonomous retries, and tool usage
- **Tooling**: Licensing and compute costs for orchestration frameworks (e.g., Temporal, LangGraph, Airflow) used to manage state and coordination.

2.1.2 Architecture and layer interactions decisions

- **Multi-Layer Dependencies**: As illustrated in Figure 2.1, agentic systems span multiple layers: orchestration, reasoning, RAG, vector stores, and execution. Bottlenecks in one layer (e.g., retrieval latency) cascade through the entire workflow
- **Component Distribution**: Agents may utilize local SLMs for fast answer while relying on shared LLMs for complex reasoning

- **Observability**: Performance control requires integrated observability across the Vector Store (e.g., FAISS, Pinecone, Weaviate), the orchestration layer, and external tools.

2.1.3 LLM/SLM deployment tradeoffs

- **Token Latency**: Inference latency increases linearly with prompt length and generation size.
- **Model Size**: Larger parameter models provide better reasoning but require significantly more compute resources and introduce higher latency
- **Cold Start**: Serverless or idle model deployments often suffer from initialization delays before processing the first token
- **LLM / SLM Training Costs**: Includes compute (GPU/TPU hours), data preprocessing, storage, and distributed orchestration overhead
- **Inference Costs**: Driven by token-based pricing (e.g., OpenAI API) or provisioned throughput. Real-time SLAs often require overprovisioning resources to handle spikes, increasing costs
- Use of **Common LLM vs Local SLM**: cost of accessing LLM through API vs cost of training SLM using proprietary business data.

2.1.4 MCP servers implementation decisions

Performance stability varies between self-hosted open-source MCP servers and managed enterprise solutions, particularly regarding throughput and uptime guarantees.

2.1.5 Knowledge base and data platform implementation decisions

- **Semantic Layer Efficiency**: Performance is heavily influenced by how effectively the agent utilizes logical data models and semantic layers sitting atop enterprise Data Warehouses and Data Lakes
- **Storage Formats**: There is often a trade-off between the interoperability/convenience of Open Table Formats (OTFs) and the raw performance of native, optimized storage formats
- **Query Performance**: The speed of SQL-based access (e.g., Snowflake, Databricks) directly impacts agent wait times. Undersized clusters or queries spilling data to remote storage can cause severe queuing.

2.1.6 RAG and Vector Store implementation decisions

- **Embedding Latency**: Search speed depends on index size, the distance metric used (e.g., cosine similarity), and the number of results retrieved
- **Indexing Strategy**: The choice between batch vs. real-time indexing affects data freshness and system responsiveness
- **Retrieval Quality**: Poor retrieval relevance forces the LLM to process more noise, leading to longer completion times or failed reasoning steps
- **Infrastructure**: Costs associated with vector database operations (search + write throughput), embedding generation, and continuous index refresh cycles

- **Context Management**: Large context windows increase token consumption and I/O latency, directly impacting cost per query.

2.1.7 Cloud Infrastructure, Network, Scheduling, and Design Decisions

- **Compute and Storage**: Expenses for on-demand vs. reserved compute (DBUs, Snowflake credits, GPUs) and storage for model weights, logs, and embeddings
- **Data Transfer**: Inter-layer communication costs, particularly when data moves across availability zones or clouds (e.g., LLM - RAG - Vector DB)
- **Network Latency**: In multi-cloud or hybrid setups, physical separation between the LLM, Vector DB, and Data Lake increases round-trip time
- **Prompt Engineering**: Inefficient prompt structures result in excessive token usage and slower inference
- Planning Depth: Agents that chain multiple sequential steps (long-horizon planning) inherently increase total latency.

2.1.8 Observability, security, and governance decisions

- **Monitoring Tools**: Costs for telemetry pipelines (OpenTelemetry, LangSmith, Prometheus) required to track non-deterministic agent behavior
- **Compliance**: Overhead for logging, auditing, access control, and ensuring compliance with frameworks like HIPAA, GDPR, and SOC 2.

2.1.9 LLM/SLM training decisions

Understanding the total cost of training requires analyzing the tradeoff between control and expense. Decisions should be guided by the following hierarchy of cost-performance profiles:

- **Train from Scratch**: Highest control and specialization, but highest cost (compute, data, talent) and complexity
- **Fine-Tune Pretrained**: Balanced control and cost; essential for domain-specific compliance or style adaptation
- **Prompt Engineering**: Low infrastructure cost, but performance is limited by the base model's constraints
- **API Access**: Easiest integration with zero maintenance, but high recurring operational costs at scale.

2.2 Agentic AI manageability decision risks

What if you, as an executive, manager, or architect, are selecting the wrong agentic AI platform, using inefficient scalability and workload management rules, or incorrectly determining the minimum configuration needed to meet business SLGs?

If the platform type or configuration is misjudged, the consequences are immediate and costly. C-level executives, IT leaders, and business management are forced to confront

escalating costs, unstable performance, and missed Service Level Goals that directly affect business operations.

An undersized platform causes latency spikes, agent failures, and SLG violations in critical business processes. An oversized platform, often chosen "to be safe," results in chronic overprovisioning, especially when peak load scenarios driven by many concurrently running agents serve as the sizing baseline. If unused or idle environments are not shut down, organizations waste millions on underutilized compute, storage, and networking resources.

FinOps and PlatformOps initiatives are expected to guide CFOs and CIOs in these decisions. However, without workload-aware observability and modeling, platform selection becomes guesswork rather than an informed, defensible decision.

2.2.1 What if you size a new Agent incorrectly during DevOps?

Incorrect sizing during the DevOps phase is one of the fastest ways to erode trust in an Agentic AI initiative. Mistakes by application developers, operations teams, or business managers often surface later as budget overruns, performance surprises, and missed delivery commitments.

During development and testing, costs can escalate rapidly. Each failed execution, prompt refinement, or logic adjustment typically requires a full, production-priced inference run. Unlike traditional applications, agentic AI systems incur real costs for every experiment.

Because autonomous agents rely on iterative reasoning, retries, and multi-step workflows, hidden costs can accumulate quickly. Without early workload characterization and cost modeling, development environments can burn through large portions of the annual AI budget before the system even reaches production.

2.2.2 What if your dynamic capacity management is ineffective?

Dynamic capacity management is essential but dangerous when poorly designed. In Agentic AI systems, autonomous agents can independently trigger retries, recursive calls, and parallel execution. A small logic flaw or unexpected condition can cause an agent to loop inefficiently for hours, consuming massive amounts of tokens, GPU time, memory, and I/O.

In multi-agent systems, risk multiplies. A slow, overloaded, or failing agent can become a systemic bottleneck, blocking downstream agents and halting the entire workflow. The result is cascading latency, missed deadlines, SLG violations, and a loss of confidence in automated decision-making.

Without continuous observability, policy enforcement, and closed-loop control, dynamic capacity management becomes a runaway cost and performance risk rather than an optimization mechanism.

2.2.3 What if your DataOps is not effective?

Ineffective DataOps silently undermines both performance and cost control in agentic AI environments. DBAs, architects, and operations teams rely on DataOps to shorten the long, complex cycles of moving data from source → pipeline → analytics → models.

When DataOps lacks visibility or coordination, latency issues and cost spikes go undetected across platforms such as Snowflake, Databricks, Teradata, BigQuery, Synapse, and S3-based pipelines. Data duplication, inefficient scans, and uncontrolled data movement inflate costs and degrade agent response times.

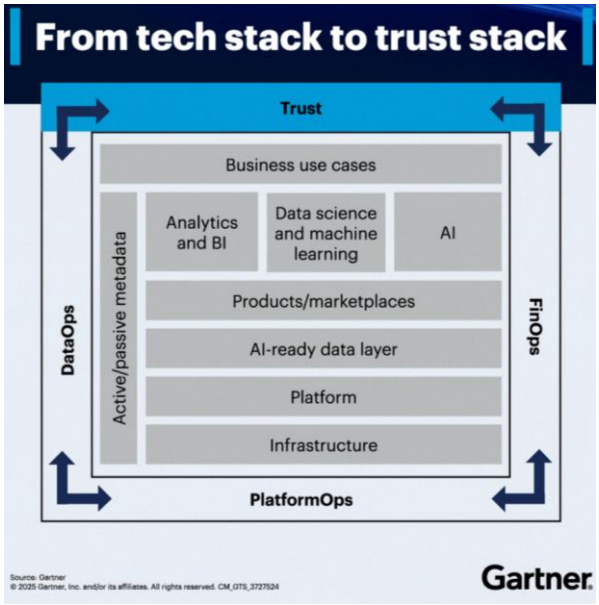
Effective DataOps provides centralized orchestration, metadata management, monitoring, and version control across hybrid multi-cloud environments. Without it, Agentic AI systems operate on fragmented data pipelines, making reliable performance, accurate modeling, and cost predictability nearly impossible.

2.2.4 The bottom line

Agentic AI does not fail quietly. Wrong platform choices, poor DevOps sizing, ineffective dynamic capacity management, or weak DataOps practices do not merely increase costs. They amplify risk, erode trust, and threaten the viability of AI-driven business processes.

To combat rising costs, leading organizations are adopting FinOps, PlatformOps, DevOps and DataOps practices [1,2]. Agentic AI manageability requires integrated FinOps, PlatformOps, DevOps, and DataOps, supported by automated observability, modeling, optimization, and continuous verification not by intuition or static rules.

According to a recent Gartner report, integrated operational frameworks such as FinOps, DevOps, and DataOps are foundational for delivering cost-efficient, reliable, and scalable cloud-based AI systems, because they align business goals with Figure 2.2 Optimization of



FinOps, PlatformOps, DevOps and DataOps decisions builds trust in agentic AI systems, according to Gartner Group.

performance and financial accountability across infrastructure and organizational layers [24].

However, while the vision of integrated Ops is compelling, the technology to support this convergence in practice remains immature or fragmented. Most organizations today struggle with siloed tools, disconnected telemetry, manual handoffs between teams, and a lack of actionable models that unify observability, forecasting, optimization, and closed-loop control.

Achieving performance goals at the lowest cost for each business process in a multi-layered, agentic AI system is highly complex. Therefore, automating observability becomes essential to continually monitor the cost and performance of each AI agent across layers and components. Gathering performance, resource utilization, and data usage metrics for workload characterization and resource allocation in an agentic AI environment is challenging.

2.3 Key Takeaways:

- **Multi-Layer Interdependency:** Architectures span orchestration, LLM, RAG, and execution layers; bottlenecks in any single layer cascade non-linearly across the entire system
- **Unpredictable Behavior:** System performance and cost are highly variable due to agent-driven concurrency, recursive reasoning loops, and autonomous retries
- **High Mistake Costs:** Platform selection and sizing errors have immediate financial consequences, leading to either chronic overprovisioning or cascading systemic failures
- **Quantitative Governance:** Defensive governance requires automated observability and modeling to replace intuition-based decision-making
- **Integrated Ops:** Mission-critical reliability demands the tight integration of FinOps, DevOps, DataOps, and PlatformOps.

3 Agentic AI Observability Automation

3.1 Observability approach overview

If you are an IT manager, architect, or FinOps engineer responsible for organizing observability processes in an agentic AI environment, you face a challenge in scaling observability as the number of agents managing business processes and the volume of data processed in the agentic AI system grow. Let's review a methodology and examples for automating observability, including data collection, workload characterization, problem identification, and performance tuning, to reduce costs in agentic AI worlds.

Data collected on each layer and different components of the agentic AI environment are transformed into a standard format and used for workload characterization and problem determination. Machine learning algorithms identify performance and cost anomalies and their root causes, narrowing down performance tuning efforts.

The results of observability are used as input to queueing network models and gradient optimization to optimize cost and performance and set cost and performance expectations. It enhances the operations function by comparing actual measurements with expected values, enabling the organization of closed-loop, continuous cost optimization and performance control.

Observability is an integral part of cost optimization, operating performance, and cost control.

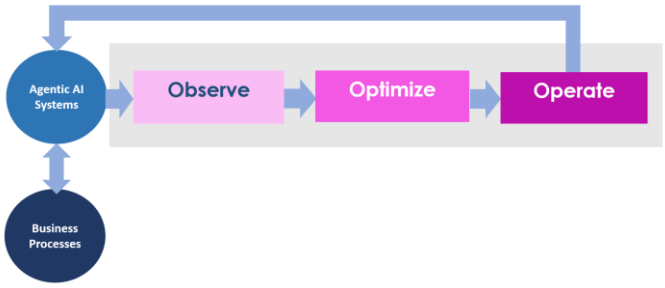


Figure 3.1 Integration of observe, optimize, and operate functions in an agentic AI environment.

3.2 Automating observability for Agentic AI

Observability agents and third-party tools collect performance, resource utilization, data usage, and cost data across platforms and layers of the agentic AI system.

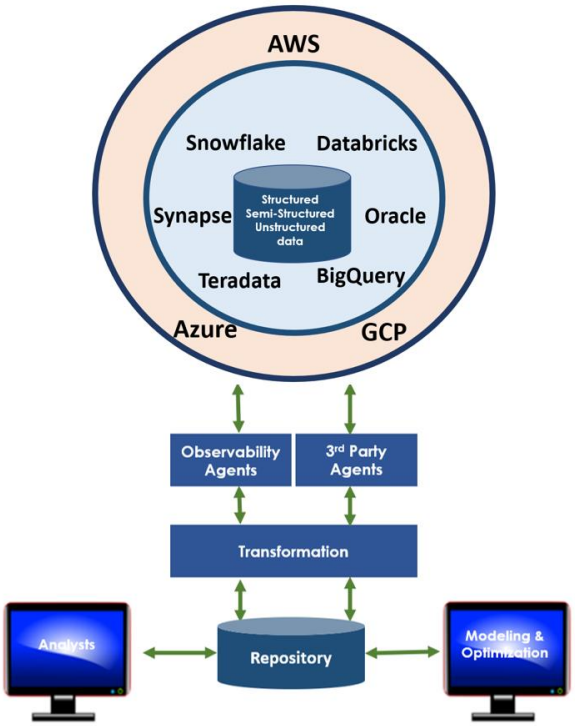


Figure 3.2 Observability Agents extract measurement data characterizing agentic AI Agents' performance, resource utilization, data use, and cost. Measurement data used by observability, optimization, and closed-loop cost and performance control

3.2.1 Core observability metrics used by modeling and optimization

Performance metrics include response times and request counts. Resource utilization includes CPU and GPU time, memory, storage, and network usage. Cost metrics include token counts, credits used, and API usage costs [8–10].

Cloud-native tools such as AWS CloudWatch, Azure Monitor, and GCP Cloud Monitoring (Ops Suite) monitor CPU, memory, I/O, and network usage. For GPU-accelerated workloads, NVIDIA DCGM monitors GPU utilization and queue depth.

To ensure SLG compliance, organizations typically use Prometheus for performance monitoring and Grafana for visualization and dashboarding. OpenTelemetry, Datadog, and LangSmith are essential for end-to-end tracing and latency analysis, often used for detecting and verifying SLG compliance.

The complexity of an agent's tasks significantly affects both latency and cost. In most agentic workflows, the number of LLM calls per request is the primary driver of overall cost and latency.

Layer	Function	Metrics
Agent & Workflow	Goal decomposition, planning, collaboration	Goal success rate, reasoning steps, retries, delegation to sub-agents, planning vs. execution time
LLM Usage	Reasoning, generation, decision-making	Tokens usage (prompt/completion), model latency, retries/error rate, cost per run
Memory & Retrieval	Context/memory supply for agents and LLMs	Query volume, hit/miss ratio, latency per query, storage growth
External Tools & APIs	Execute actions, fetch real-world data	Calls per tool/API, latency per call, failure/timeout rate, cost per call
Infrastructure	Runtime, scaling, compute/storage	CPU, memory, GPU, I/O, network
SLG Compliance	Overall system health	Latency targets, cost per session, success thresholds

Table 3.1 Metrics characterizing each layer of the agentic AI environment

The following information is based on internal benchmarking conducted on a very small configuration in our lab, and on industry observability results. It incorporates measurement data from the RAG/LLM agent and the Vector Store cloud data platforms, including Snowflake, Databricks, and Teradata, as well as publicly available data from hybrid multi-cloud environments. Response-time ranges are derived from industry references and from the usage patterns of vector stores such as FAISS, Pinecone, and Weaviate.

3.2.2 LLM performance monitoring

Use Case	LLM Calls per Request
Simple Q&A	1
Document Summarization	1–2
Tool-Augmented Agent (RAG, search)	2–4
Planning Agent with Sub-Agents	5–15
Complex Multi-Step Decision Process	10–50+

Table 3.2 Variability of the number of LLM calls per request affects cost and performance.

LLM response time, measured in a small environment, ranges from 50 seconds to 2.5 minutes. Automated scaling is needed to meet SLG requirements and reduce costs.

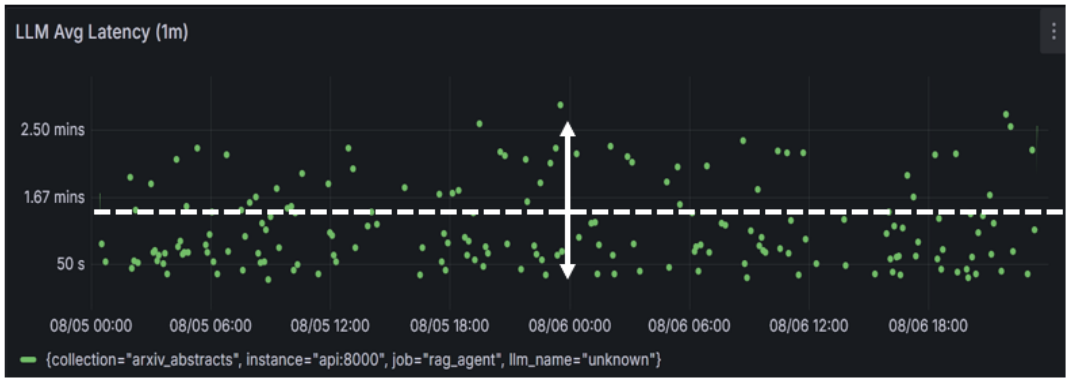


Figure 3.3 The average LLM Response Time is 1.34 minutes, with a range between 50 seconds and 2.5 minutes during the benchmark test. Automated scaling up is needed to satisfy SLG, and scaling down to reduce cost.

Cost is measured by the number of tokens sent to and from the LLM. Measured LLM response time and the number of tokens are just one of the components of the total response time and total cost.

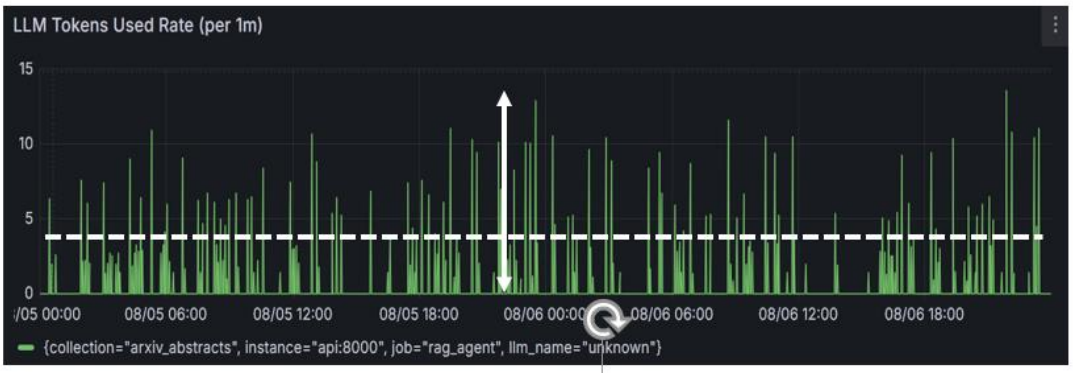


Figure 3.4 The average LLM token rate is three tokens per minute, with a maximum of 13 tokens per minute.

3.2.3 RAG performance monitoring

Various methods of RAG evaluations are described in [15-18]. An example of the metrics collected in our test environment is shown in Figure 3.5 and Figure 3.6 below.

The RAG latency varies widely, and automated scaling is needed to continuously meet SLGs at the lowest cost.

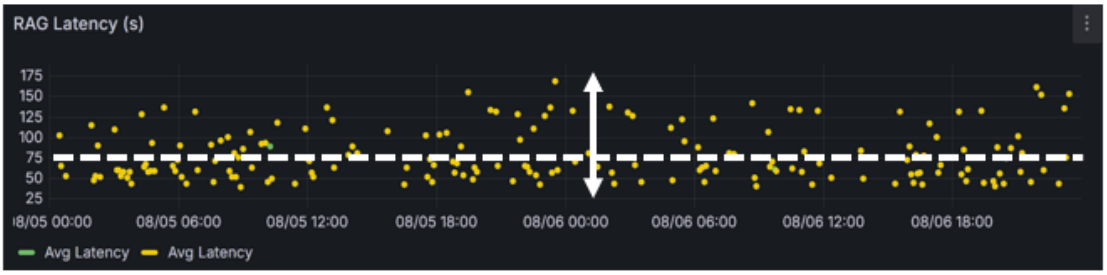


Figure 3.5 The average RAG latency during the benchmark test is between 25 and 175 seconds. Automated scaling is needed to satisfy the business process SLG

The CPU time consumed by RAG is varied as well.

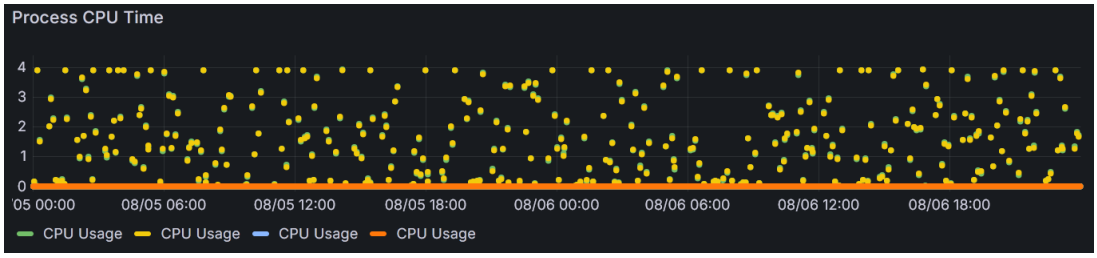


Figure 3.6 The total RAG CPU time (in cores used) deviates between 0 and 4 during the benchmark test. It demonstrates the variability of resource usage supporting service demand change

3.2.4 Vector Store performance monitoring

Vector stores provide context and memory to agents, especially in RAG systems. The number of accesses depends on the number of agents' retrievals, validations, and information stores during a task.

Scenario	Vector Store Access Frequency
One-Shot Q&A (basic RAG)	1 retrieval per task
Multi-Step Planning Agent	2–5 times for fact retrieval, validation, and storage
Research or Summarization Agents	5–20+ retrievals for iterative refinement
Autonomous Agents with Memory	10+ retrievals and updates per session
Real-Time Monitoring Agents	Continuous, based on event frequency

Table 3.3 An example of the vector store access frequency for different types of requests.

Figure 3.7 High variability in Vector DB (qdrant) search and scroll latency between 0 and 2 seconds

3.2.5 Response time ranges for each layer

Each agentic AI layer uses specific components, and their response times vary widely. In general, an agent's request can visit the orchestration layer, LLM, and Vector Store several times. The total response time should meet the business process SLG.

Layer	Function	Components	Typical Response Time Range
1. Agent & Workflow	Goal decomposition, planning, and collaboration	Agent shells, orchestrators	100 ms – 3 sec per decision step
2. LLM Execution	Reasoning, generation, decision-making	Local SLMs, common LLMs (e.g., GPT-4, Claude, Mistral)	200 ms – 10 sec (varies with prompt size and model)
3. Memory & Retrieval	Context and memory supply for agents and LLMs	Vector stores (e.g., FAISS, Weaviate, Pinecone), RAG modules	10 ms – 500 ms
4. External Tool & API	Execute actions, fetch real-world data	APIs, databases, function tools (e.g., search, code, math)	100 ms – 5+ sec
5. Infrastructure	Runtime, scaling, compute, and storage	CPU, GPU, memory, and autoscaling systems	Adds baseline latency and can impose constraints

Table 3.4 An example of the response time ranges for different agentic AI layers.

3.2.6 Performance and resource usage by the agentic AI system

Performance tuning recommendations determination is based on the analysis of the response time components and corresponding resource utilization on each layer

Layer	Response Time (ms)	CPU Usage (%)	IO Operations (Count)	Memory Usage (MB)

Planning (LLM Call)	200	15	50	200
RAG Retrieval fetches relevant context	400	10	150	150
LLM Inference	2500	70	300	600
Tool/API Calls	1000	20	200	250
Orchestration	200	10	80	100
Memory Access	100	5	120	80
Verification	300	10	60	120
Telemetry Logging	50	2	40	60

Table 3.5 An example of the average response time and resource usage by moderately complex agentic requests

3.3 Cost and performance monitoring by cloud data platform vendors

If you are an executive, manager, architect, or FinOps analyst involved in selecting an agentic AI platform, you have to consider the opportunities and limitations of each platform's observability.

Each cloud data platform uses different methods to log performance and resource utilization data, employs different metrics, and offers different options for organizing traces, cost tracking, and AI-specific telemetry. Although cloud data platform architectures differ, the monitoring objectives remain consistent: assess performance, allocate costs, trace agent behavior, and enable closed-loop control of cost and performance. When selecting an appropriate cloud data platform, both observability and monitoring, as well as performance and resource usage, are essential factors for comparison.

3.3.1 Snowflake observability overview

Snowflake provides isolated, independently scalable Virtual Warehouses, enabling predictable performance and rapid start/stop with per-second billing. This makes it well suited for mixed agentic AI workloads, separating RAG, embeddings, and operational analytics without noisy-neighbor effects. Its optimizer uses automatic micro-partitioning, predicate pruning, caching, and centralized metadata services to deliver stable, predictable scan and retrieval performance.

Snowflake Cortex adds native AI observability, capturing telemetry, traces, cost metrics, and evaluations in internal Snowflake tables for SQL-driven analysis. MCP support is strong and natively aligned with Cortex, enabling efficient tool calling and governance. Billing is credit-based, with auto-suspend/auto-resume and Resource Monitors supporting strong cost-control discipline.

Best fit: predictable SLG compliance, isolated workloads, centralized governance, enterprise RAG orchestration.

3.3.2 Google BigQuery observability overview

BigQuery is a fully serverless analytics and RAG environment with automatic autoscaling, powered by the Dremel execution engine and massive parallelism across distributed columnar storage. This architecture is ideal for bursty, unpredictable agentic workloads that require elastic retrieval and vector access, without cluster management overhead.

Observability is deeply integrated via BigQuery logs, Cloud Monitoring, Looker, Vertex AI telemetry, and streaming analytics, making it a strong hub for analyzing agent traces and costs at scale. MCP support is available through the MCP Toolbox, with medium maturity. Billing supports both on-demand TB scanned and slot-based capacity Editions, balancing flexibility and predictability.

Best fit: large-scale RAG, elastic retrieval, no-ops environments, rapid agent instrumentation.

3.3.3 Databricks observability overview

Databricks provides a unified Lakehouse with strong governance through Unity Catalog and isolation via workspaces, supporting agentic workloads across retrieval, vector search, orchestration, and LLM pipelines. Performance is driven by Photon (vectorized execution), Spark Adaptive Query Execution, and Delta optimizations, making it highly effective for high-throughput pipelines and complex hybrid workloads.

Observability includes System Tables, MLflow Tracing, Vector telemetry, and Lakehouse Monitoring. MCP support is emerging with first-party integration and strong ecosystem alignment. Costs are billed in DBUs and depend on SKU and cluster types, requiring workload mapping discipline.

Best fit: engineering-driven platforms, hybrid vector/SQL workloads, high-throughput AI pipelines.

3.3.4 Teradata VantageCloud observability overview

VantageCloud delivers elastic, independent compute clusters with industry-leading concurrency, workload management, and a powerful cost-based optimizer. It excels at mission-critical, structured enterprise decision workloads, delivering predictable performance under high agent concurrency.

Observability is exceptionally strong through MCP telemetry, DBQL analytics, and resource usage statistics, enabling full traceability from agent action → SQL workload → resource → cost. MCP support is broad and mature. Billing typically follows elastic reserved compute models suited for stable enterprise workloads.

Best fit: regulated environments, mission-critical agent workflows, and high-concurrency structured data operations.

3.3.5 Azure Synapse observability overview

Azure Synapse supports both Dedicated SQL Pools (DWU capacity) and Serverless per-TB-scanned models. Performance depends heavily on effective table distribution strategies; misconfiguration can introduce latency variability for analytical and RAG preprocessing workloads.

Observability is deeply integrated with Azure Monitor, Log Analytics, and the AI Foundry analytics ecosystem. MCP support is limited and requires custom implementation, adding development cost and latency risk. Billing is dual-model (capacity + consumption), requiring governance discipline.

Best fit: Azure-centric enterprises, integrated analytics + AI ecosystems, and organizations comfortable with DevOps tuning.

3.3.6 Oracle AI Database 26ai / HeatWave observability overview

Oracle AI Database 26ai embeds AI directly into the database engine, significantly reducing latency by eliminating external AI services. HeatWave provides massively parallel in-memory acceleration for vector search, hybrid SQL + vector retrieval, and analytics, making it highly efficient for large-scale RAG and enterprise AI workloads. Observability and governance are deeply embedded in Oracle enterprise frameworks, with strong compliance and auditability. MCP support is limited and DIY. Billing follows capacity-oriented enterprise models aligned with sustained high-performance AI environments.

Best fit: highly governed industries, large structured AI memory workloads, and enterprises prioritizing security and predictability.

3.3.7 Comparative table across platforms

Platform	Architecture & Scaling	Optimizer / Execution Engine	Observability & Monitoring Depth	MCP Support & Maturity	Billing Model & Cost Predictability	Best Fit
Snowflake	Independent Virtual Warehouses, predictable isolation, fast start/stop	Micro-partitioning, pruning, caching, and centralized services	Cortex observability + full AI telemetry + SQL analytics repository	Strong native + open source MCP, highest maturity	Credit-based, per-second billing, auto-suspend, strong governance	Predictable SLG workloads, enterprise RAG orchestration, and cost governance
BigQuery	Fully serverless, elastic autoscaling, no cluster management	Dremel + massive parallel columnar execution	Deep Google Cloud Monitoring + SQL + ML-based analytics	Supported via MCP Toolbox, medium maturity	On-demand TB scan or Editions slot-based capacity	Bursty RAG workloads, elastic retrieval, “no-ops” environments
Databricks	Lakehouse with strong isolation via Workspaces & Unity Catalog	Photon + AQE + Delta optimization	System Tables + MLflow + Vector Telemetry	Emerging first-party MCP integration	DBU-based by SKU & workload; requires discipline	Hybrid vector + SQL, high-throughput agent pipelines

Platform	Architecture & Scaling	Optimizer / Execution Engine	Observability & Monitoring Depth	MCP Support & Maturity	Billing Model & Cost Predictability	Best Fit
Teradata VantageCloud	Elastic compute clusters, industry-leading concurrency. Support for a Vector Store, and in the engine, vector operations.	Mature cost-based optimizer	Exceptional end-to-end observability (MCP + DBQL + ResUsage)	Broad vendor + community, high maturity	Reserved/elastic enterprise capacity	Mission-critical, regulated, high-concurrency environments
Azure Synapse	Dedicated SQL Pools + Serverless TB scan	Performance depends heavily on distribution strategy	Azure Monitor + Log Analytics + AI Foundry	Limited DIY MCP	Mixed DWU capacity + per TB scan	Azure ecosystem orgs, willing to tune
Oracle AI Database 26ai / HeatWave	AI embedded in DB engine + in-memory HeatWave vector acceleration	In-memory execution, strong enterprise governance	Embedded enterprise telemetry, governance-first	Limited DIY MCP	Enterprise capacity aligned to sustained workloads	Highly governed industries, secure enterprise AI

Table 3.6 Comparing cloud data platforms as a base for agentic AI

3.3.8 Cross-platform cost monitoring comparison

Platform	Compute Cost	Storage Cost	LLM/RAG Cost	Agent Telemetry
Snowflake	Credits	Per-TB	Cortex API	Full
Teradata	Vantage Units	Low-cost object store	External	Full (MCP + DBQL)
Databricks	DBUs	Delta storage	Mosaic Serving	Full (MLflow + System Tables)
Synapse	DWUs / Serverless TB Scan	ADLS	Azure OpenAI	Full via Azure Monitor
BigQuery	TB scanned	Table storage	Vertex AI	Full (SQL + ML)
Oracle	OCPU	Block/Object Store	OCI GenAI	Full via Audit/Logging

Table 3.7 Cost Observability & Metering

3.3.9 Data metrics used by modeling and optimization technology

Performance and cost repository stores measurement data collected by observability agents and 3rd-party monitoring tools.

Measurement data are aggregated using workload aggregation rules. The repository contains the results of workload characterization, including hourly performance, resource utilization, and data usage profiles for each workload.

It also stores configuration information, modeling scenarios, and optimization results. Quantitative analysis generates performance-tuning recommendations, and modeling

and optimization technologies evaluate alternatives, generating performance-prediction results and recommendations, enabling closed-loop cost and performance control.

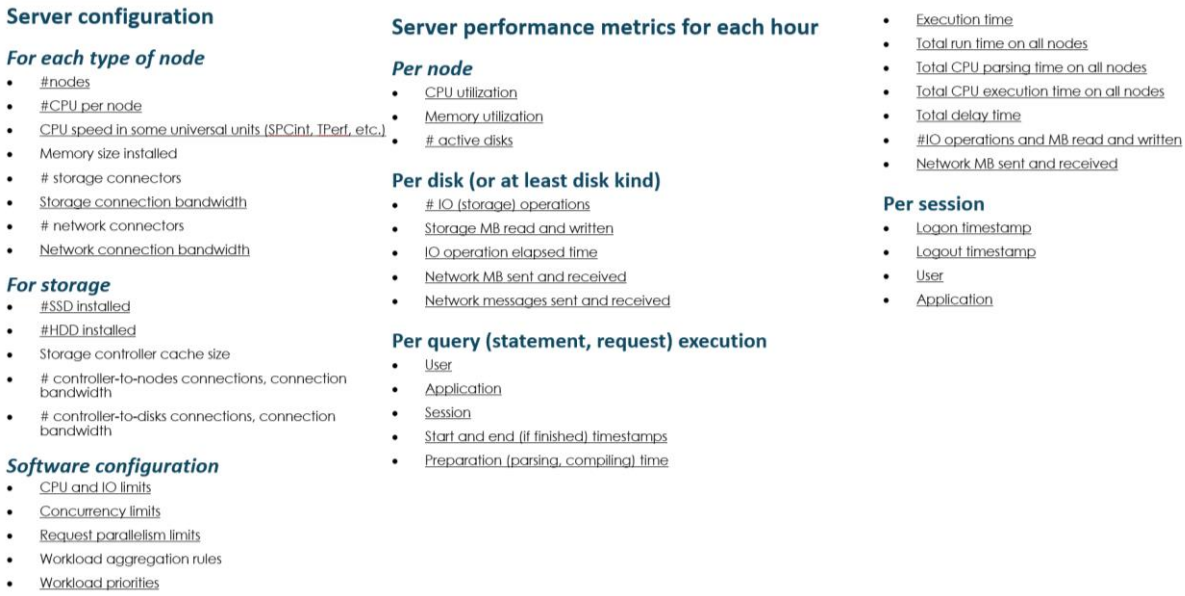


Figure 3.8 Data needed for our modeling and optimization

3.3.10 How to reduce monitoring overhead

The data collection overhead increases as more business processes are supported by agentic AI, especially when the number of agents rises, the vector store size expands, or business process goals change.

We recommend grouping agents by similar functions and only monitoring the representative agents. If the same type of performance, resource usage, or cost problem persists within a specific group, temporarily enable more detailed monitoring and tracing.

3.4 Key Takeaways: Observability Automation

- **Granular Telemetry:** Observability must extend beyond basic metrics to track granular data per agent, per task, per API call, and per token
- **Foundation for Control:** Observability is not merely a reporting tool; it is the essential foundation for predictive modeling and closed-loop control
- **Traceability for Compliance:** Integrated tracing across every layer from the LLM to the Vector Store is vital for ensuring SLG compliance
- **Granularity of Monitoring:** High-resolution telemetry identifies hidden cost drivers like "prompt bloat" and inefficient retry cycles
- **Intelligent Overhead Management:** Organizations can reduce monitoring costs by clustering similar agents and monitoring representative functional groups.

4 Agentic AI System Performance Tuning

4.1 Agentic AI workload characterization

Results of the workload characterization help explain how the complex agentic AI environment works. People supporting FinOps and engineers can develop regular daily, weekly, and monthly performance and cost-saving reports and recommendations for DBAs and developers. Workload characterization aggregates measurement data characterizing performance, resource utilization, data usage, and the cost of individual agents into business workloads and builds hourly performance, resource utilization, data usage, and cost profiles for each business process (workload) on each agentic AI layer.

4.1.1 Workload aggregation Rules

Workload aggregation rules define agentic AI components supporting each business process and group them into workloads.

4.1.2 Automated workload characterization

Workload characterization automatically generates hourly **Performance** (response time, throughput), **Resource utilization** (CPU, memory, GPU, I/O usage, and tokens consumption per agent), **Data usage** (frequency of accessing data objects by different agents; Vector store behavior: query data volume, vector size, read/write ratios, IO operations), and **Cost** (credits used and token/API-related cost distribution, input/output tokens count, context expansion from RAG) **workload profiles on each layer**.

Workload profiles help determine performance and cost patterns, complexity, seasonality and trends for each workload. They enable anomaly detection and tuning recommendations.

Modeling and optimization technology uses workload characterization results for each group of agents supporting each business process as input. It applies these results to evaluate options and develop recommendations for cost optimization and performance control.

4.1.3 Examples of the workload characterization of agentic AI

Below is a dashboard showing daily performance (response time, throughput), CPU and storage utilization by each business workload for one of the layers of the agentic AI system.

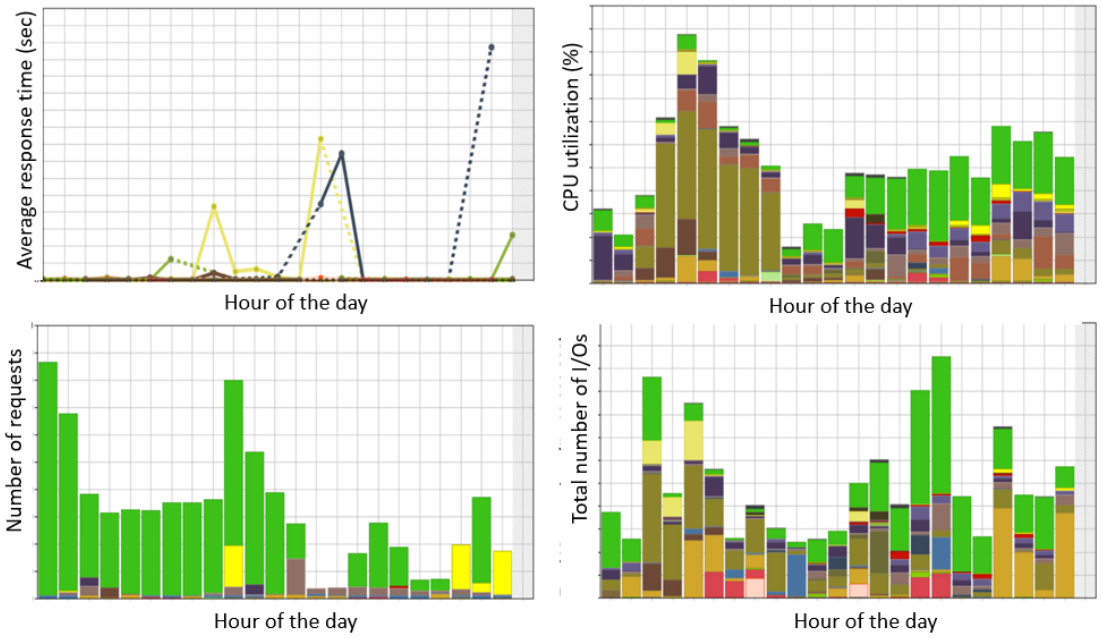


Figure 4.1 An example of the dashboard tiles visualizing performance, resource utilization, and data usage across different layers and platforms for various workloads, shown in different colors over time.

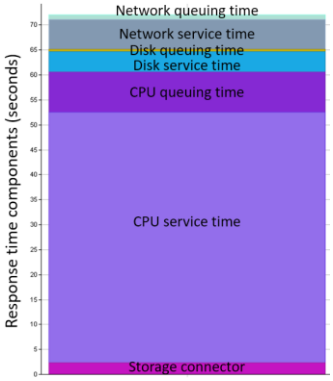


Figure 4.2 An example of the workload response time components. The CPU service time is the largest one. In this case, either application tuning or a faster CPU is needed to improve the response time.

To simplify analysis, we create clusters of workloads, with similar resource demand. Figure 4.3 shows the short, medium, and complex groups by data volume (KB) and CPU service time per request.

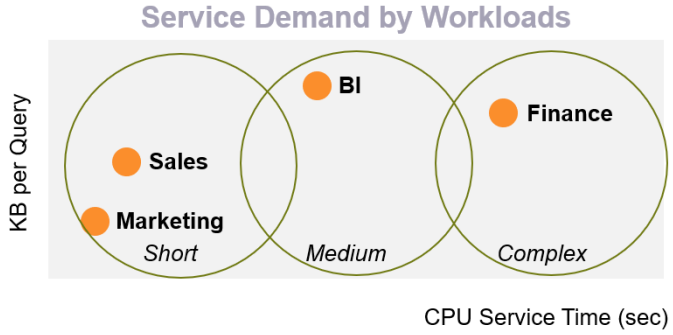


Figure 4.3 The agents/workloads supporting different business processes have different demands for resources and different SLGs. It affects the workload management and resource allocation rules.

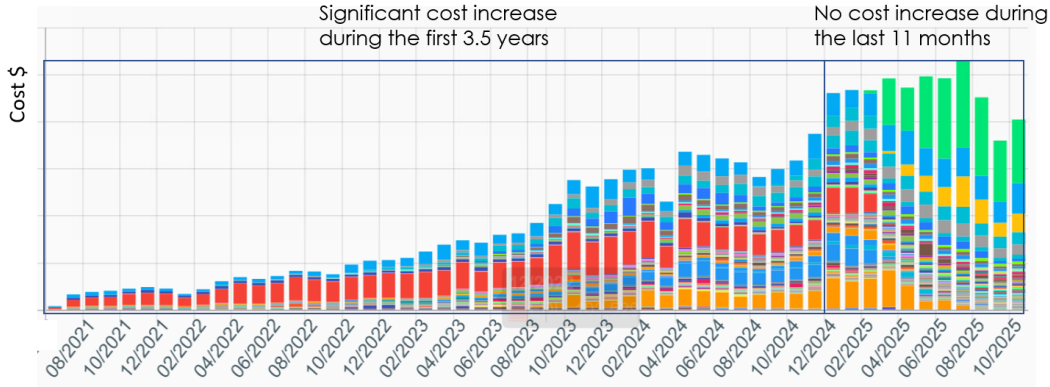


Figure 4.4 This example illustrates that without proper control costs can increase almost exponentially. During the 3.5 years, the monthly cost increased almost 6 times. As a result of tuning and consolidation, the total cost has not grown during the last 11 months

Year	Month	GB processed				
Application		Sales	Finance	CRM	HR	APP
2024	10	13,920,327		10,790,609		9,864,205
2024	11	33,432,732		25,007,250		8,579,563
2024	12	22,567,226		22,430,883		9,693,571
2025	1	23,195,727		29,860,745		13,107,165
2025	2	29,893,191	0	29,958,984	1,144,036	14,610,133
2025	3	24,948,949	165,279	26,838,803	26,789,188	17,087,876
2025	4	27,332,750	6,879,958	29,609,392	32,105,192	16,087,286
2025	5	18,787,959	26,790,185	26,558,194	54,990,734	12,030,433
2025	6	19,137,979	39,419,815	27,013,872	64,227,894	9,551,932
2025	7	17,194,630	91,315,873	34,580,777	104,275,663	9,545,493
2025	8	14,262,797	127,053,570	27,660,369	84,115,790	9,468,548
2025	9	19,176,441	78,130,239	29,863,353	80,827,485	9,851,141

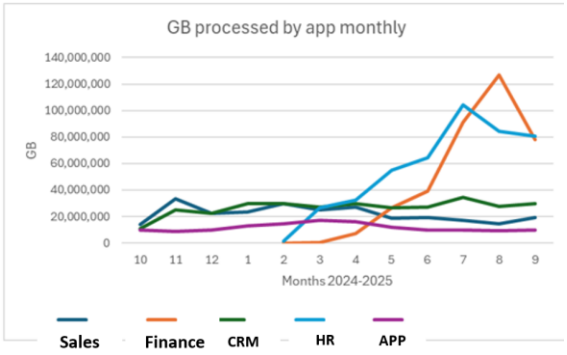


Figure 4.5 Automatic observability detects trends in the volume of data processed by each workload to develop proactive cost optimization and performance control recommendations

For the two new workloads, deployed in February (2nd month), the volume of processed data was growing rapidly, but decreased recently due to queries and table tuning.

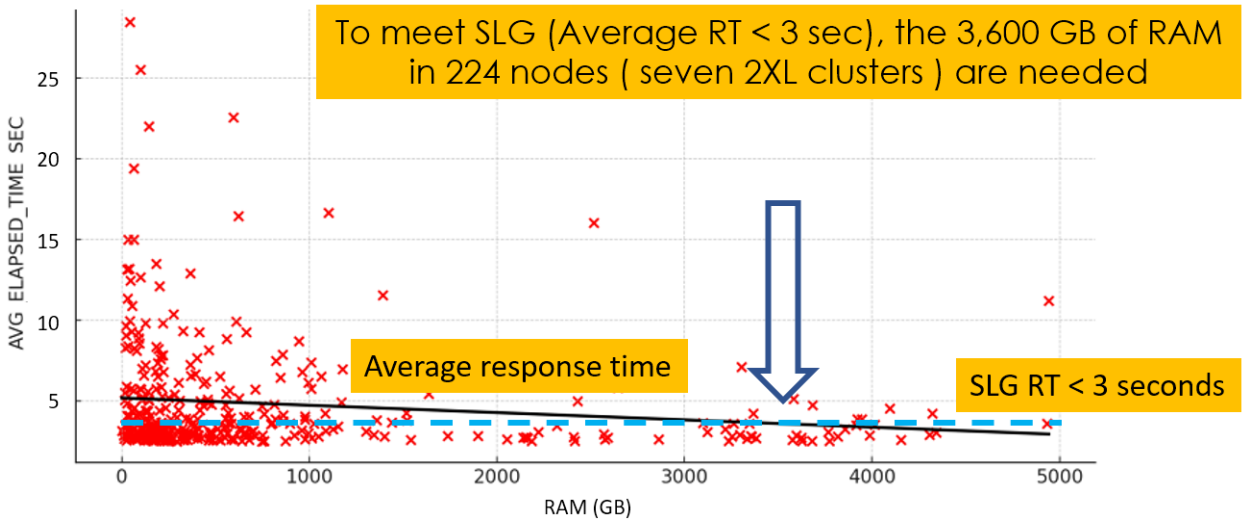


Figure 4.6 This example illustrates the relationship between performance and memory size. To achieve a 3-second response time SLG, about 3,600 GB of RAM on 224 nodes are needed

4.2 Examples of performance tuning based on observability results

The goal of performance management for an agentic AI environment is to use the results of observability and workload characterization to identify performance and cost problems, on each layer and component, and determine their root causes to continuously adjust the configuration and workload management rules, and tune applications to meet SLGs. Insights from automated observability, workload analysis, and anomaly and seasonality detection help identify issues and focus performance management efforts.

ML models and Statistical Process Control use observability results to identify cost and performance-related anomalies and to develop tuning recommendations.

4.2.1 Performance and cost problems detection

Observability data gathered through workload characterization acts as input for machine learning and AI algorithms. These algorithms find usage patterns, detect anomalies, and identify root causes by comparing real-time data with expected behavior. This process is crucial for managing performance and costs, especially in the fast-changing world of agentic AI.

Determine and tune critical applications and queries having the largest number of cost and performance SLG violations to reduce application costs. Reduce the overhead caused by failed queries. Tune queries that spill a large volume of data to remote storage.

Monitoring is a continuous process of ensuring SLG compliance (Figure 4.7). This involves:

- **Setting Thresholds:** Establishing clear limits for key metrics like latency (e.g., maximum of 5 seconds) and cost (e.g., less than $0.20 per session)
- **Triggering Alerts:** Automatically generating alerts when these thresholds are breached
- **Tracking Success Rates:** Monitoring the percentage of requests that successfully meet their defined goals
- **Logging Granular Metrics:** Capturing tokens usage per reasoning step to identify inefficiencies like "prompt bloat."

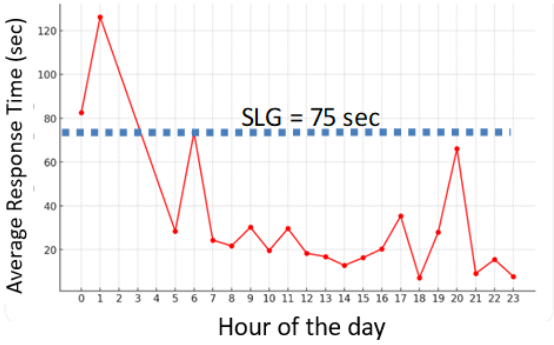


Figure 4.7 SLG-oriented monitoring detects SLG violations. Analysis of the frequency and severity of these types of violations is needed to justify the tuning efforts

Cost, performance, and resource usage anomaly detection focuses on identifying outliers that deviate from expected behavior and determining their root causes:

- **High-Latency Outliers**: Often caused by tool timeouts, excessive retries, or slow external API calls
- **High-Cost Outliers**: Can result from long prompts, inefficient RAG retrievals that require multiple attempts, or agents getting stuck in a loop
- **High-resource outliers**: Usually related to prompts involving large documents retrieved or extensive, computationally intensive reasoning chains.

By systematically identifying and analyzing these patterns, organizations can better understand their agentic AI systems and take focused actions to improve performance and manage costs.

4.2.2 Seasonality pattern determination

The performance, resource utilization, and cost profiles created for each workload reveal recurring patterns and seasonal trends in performance and resource use. Recognizing these trends is key to effective proactive capacity planning.

Recognizing these patterns - whether they are hourly, daily, or weekly - allows for a more proactive approach to managing resources. Instead of responding reactively to performance issues, organizations can optimize resource allocation in advance. For example, if a particular workload consistently spikes every Friday afternoon, resources can be automatically scaled up to meet the demand and scaled back down to cut costs during off-peak times. This predictive method ensures SLGs are achieved without over-allocating resources.

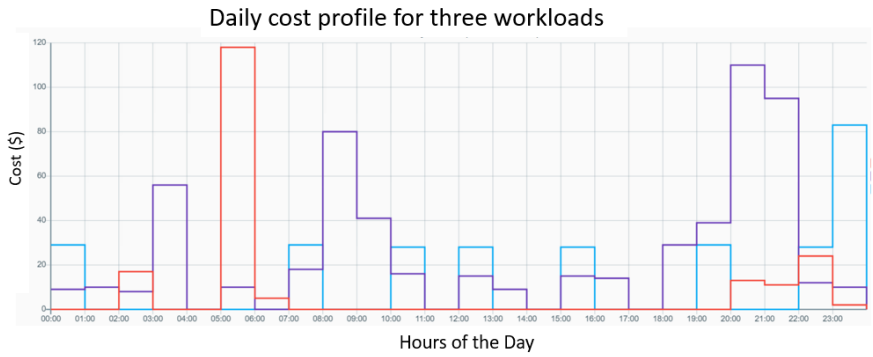


Figure 4.8 An example of detecting seasonality for three applications to optimize resource allocation rules.

4.2.3 Performance and cost anomaly and root cause detection

Applications with inconsistent service demands cause unexpected performance and cost surprises. Figure 4.9 shows the frequency and severity of the performance and cost anomalies, where severity is the function of the duration and amplitude of the anomaly.

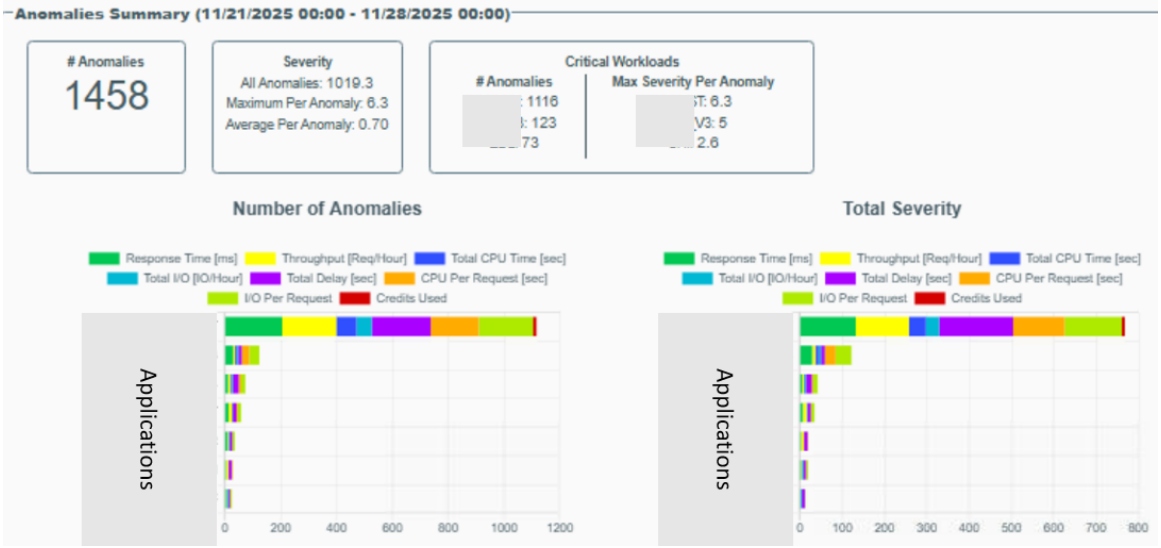


Figure 4.9 Applications' performance and cost anomalies sorted by frequency and severity of anomalies during one week.

Determining the applications and virtual warehouses with the highest anomaly frequency and severity of anomalies (Table 3.1), finding their root causes (Table 4.2) and identifying the requests causing the anomalies (Table 4.3) narrows the scope of tuning efforts.

Application Name	Date	Time	Virtual Warehouse Name	Parameter Name	Duration	Amplitude	Severity	# of Root Causes
Applications	11/24/2025	12:00	Warehouses	Response Time [ms]	12	274,189	6.30	19
	11/25/2025	05:00		Response Time [ms]	10	233,194	5.70	6
	11/22/2025	12:00		CPU Per Request [sec]	12	143	5.00	0
	11/25/2025	05:00		I/O Per Request	7	2,324,703	4.70	0
	11/22/2025	12:00		I/O Per Request	8	4,433,857	4.60	0
	11/25/2025	11:00		Throughput [Req/Hour]	7	142	4.40	0
	11/24/2025	14:00		CPU Per Request [sec]	10	77	4.20	0
	11/23/2025	16:00		I/O Per Request	7	3,468,319	4.00	0
	11/26/2025	18:00		Response Time [ms]	6	400,614	3.80	9
	11/22/2025	18:00		Response Time [ms]	6	281,062	3.80	11

Table 4.1 Top 10 individual anomalies by the severity – with application and virtual warehouse name, date and hour, and the abnormal parameter

Anomalies' root causes are other events that happened at the same time and could affect the application performance and cost. In the table below they are grouped by parameter (like CPU time per request), application, user and database and sorted by the total severity of anomalies they caused. Root causes of the most severe anomalies should be investigated and fixed.

App Name	Virtual Warehouse Cause	Parameter Cause	User Cause	Database Cause	# Anomalies Caused	Total Severity	# Workloads affected
Applications and Warehouses		I/O Per Request	Users and databases		23	12.39	1
		I/O Per Request			22	11.20	1
		CPU Per Request [sec]			19	9.86	1
		CPU Per Request [sec]			17	8.30	1
		I/O Per Request			14	5.87	1
		I/O Per Request			7	5.45	1
		CPU Per Request [sec]			16	5.37	1
		I/O Per Request			13	5.19	1
		CPU Per Request [sec]			13	5.03	1
		Throughput [Req/Hour]			8	3.98	1

Table 4.2 Determining the root causes with the highest severity narrows down the tuning efforts

App Name	Database	Query Type	Query Id	User	Virtual Warehouse Name	Start Time	End Time	Elapsed Time	Execution Time	MB Processed
Applications and databases		CREATE_TABLE_AS_SELECT	01c08a2a-0210-0351-4aa2-0709e9fabe4f		User and warehouse name	2025-11-21 12:58:01.488	2025-11-21 13:26:36.438	1714.950	1681.805	3,467,817
		SELECT	01c09dac-0210-0a47-4aa2-0709fddfe0bf			2025-11-25 00:12:06.694	2025-11-25 01:23:19.518	4272.824	4260.918	3,219,996
		SELECT	01c09dd5-0210-05c4-4aa2-0709fe0d9163			2025-11-25 00:53:33.888	2025-11-25 01:35:01.974	2488.086	2472.976	3,212,674
		SELECT	01c0a04c-0210-0905-4aa2-070a00579e73			2025-11-25 11:24:22.962	2025-11-25 11:47:18.743	1375.781	1373.520	3,035,286
		SELECT	01c09d83-0210-1315-4aa2-0709fdbab95f			2025-11-24 23:31:57.592	2025-11-25 00:12:04.355	2406.763	2392.544	2,996,882

Table 4.3 Determining queries causing anomalies with the highest service demand provides candidates for tuning

This approach helps IT managers and engineers to provide actionable information to DBAs and developers to focus tuning efforts on improving performance and reducing the cost of the most critical applications and agents.

4.2.4 Failed queries

Credits used for the failed queries are lost as they do not produce the expected results. Table 4.4 shows the history of failures by applications, and Figure 4.10 shows the credit lost by requests that should be tuned. IT managers and engineers can provide this information to application developers to recommend tuning with an estimate of how tuning will reduce cost. After tuning, they can compare the actual cost reduction with the expected.

		Percent failed executions					Percent failed credits				
YEAR	MONTH	SALES	FINANCE	CRM	HR	APP	SALES	FINANCE	CRM	HR	APP
2024	11	0.11		0.01		0.00	13.47		2.34		2.45
2024	12	0.15		0.01		0.01	10.88		2.63		0.84
2025	1	0.17		0.00		0.01	3.05		1.76		0.28
2025	2	0.26	0.00	0.00	0.01	0.00	11.89	0.00	0.52	17.22	1.45
2025	3	0.19	0.01	0.00	0.00	0.00	2.49	0.01	0.30	5.19	4.28
2025	4	0.14	0.01	0.00	0.00	0.01	3.56	2.51	1.44	1.34	1.06
2025	5	0.10	0.01	0.00	0.00	0.06	4.59	2.63	0.35	5.13	0.76
2025	6	0.08	0.01	0.00	0.02	0.01	2.39	2.52	1.23	1.29	0.61
2025	7	0.12	0.01	0.00	0.01	0.01	2.38	3.56	0.47	1.70	1.32
2025	8	0.09	0.01	0.01	0.10	0.00	1.98	2.46	1.83	0.86	0.00
2025	9	0.09	0.01	0.00	0.01		1.98	3.16	1.47	1.94	
2025	10	0.11	0.04	0.00	0.00		1.43	1.90	0.61	1.50	

Table 4.4 Determining the applications and queries with the highest failure rates helps focus tuning on reducing wasted cost

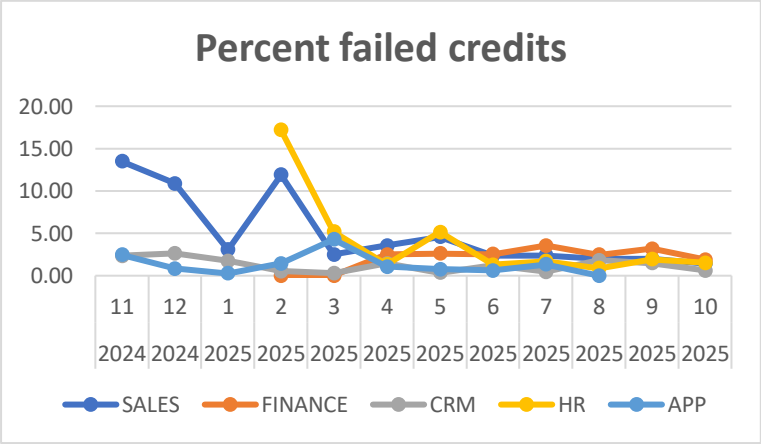


Figure 4.10 Percent of credits lost for failed queries trend for five applications. This information helps to narrow down tuning efforts to the most critical applications and their queries to reduce wasted cost

4.2.5 Credits used by top queries

Top credit-consuming queries below in Table 4.5 (defined by the tag and max query IDs) by application, database, error code, with their execution time, GB processed, and GB spilled to remote storage during one month, are the candidates for tuning.

DATABASE	Processed query tag	Error code	Error message	Execution count	Execution time (hours)	GB processed	GB spilled local	GB spilled remote	GB network	Query credits
	"ciirpt_intg-w			1	4.00	7.97	16,455.07	8,398.15	127,251.76	128.00
	"ciirpt_intg-w			1	3.47	7.96	14,367.16	6,333.76	110,418.18	110.84
	72b514967be			50	4.00	77.54	9,652.43	0.00	104,740.01	61.12
	f85420b4ceb			3	2.61	2.59	0.00	0.00	0.99	36.63

Table 4.5 Determining the data objects requests IDs, error codes, and error messages, number of requests, summary execution time, volume of data processed, and volume ofdata spilled to local and remote storage, network usage, and cost paid narrows down the scope of tuning efforts.

IT managers and engineers can provide application developers with information about the queries that incur the most credits use. Tuning these queries will provide a significant reduction in cost.

4.2.6 Reducing credit loss caused by virtual warehouse idle time

Determining the cost lost when virtual warehouses are idle provides another tuning opportunity.

For example, Snowflake bills per-second of compute time, and warehouses continue to consume credits even when they are idle and no queries are running. This means idle warehouses accumulate costs without providing value.

Our analysis shows that cost losses by application when warehouses are idle range from 15% to 40%, with the average credit loss consistently between 20% and 25%.

To reduce credit loss, organizations reduce Auto-Suspend timeout and set Auto-Resume to True. This reduces warehouse time sitting idle between jobs and automatically restart VW when a query arrives. Consolidating workloads by combining low-volume or non-overlapping workloads into shared VWs helps as well.

To reduce cost losses, modeling and optimization help determine the minimum configuration required to meet SLG.

4.2.7 Reducing the spilling to remote storage

The amount of spilled volume of data depends on the local storage size and the volume of data processed by a request. The latter can be reduced by proper table partitioning and/or indexing.

Tuning virtual warehouses and requests to reduce spilling to remote storage and to decrease the execution time and credits used is another performance and cost improvement opportunity.

The information shown in Table 4.6 and Table 4.7 identifies warehouses, databases, and queries with the biggest volume of spilled data as candidates for tuning.

Warehouse	GB processed	GB spilled remote
P01_LOAD_01_WH_XL	74,896,735	2,239,024
P01_LOAD_01_WH_2XL	83,523,249	1,745,654
P01_CQDM_LOAD_WH_2XL	8,868,409	1,243,602
P01_CDA_TV_LOAD_WH_4XL	14,428,309	994,418

Table 4.6 Volume of data processed vs. spilled to remote storage by warehouse

CLASSIFICATION	DATABASE	Processed query tag	Error code	Error message	Execution count	Execution time (hours)	GB processed	GB spilled local	GB spilled remote	GB network	Query credits	Max query ID
l		9d3383f82845ebfc2f637f569bd11	0		31	35.19	68,206.59	353,950.05	54,866.20	287,329.13	544.38	01c015ab-020f-9d10-4aa2-0709693d4f8f
l		2574fdcbb4c5405980fcdf9b43d5	0		31	28.86	65,127.74	323,978.73	43,632.37	273,055.13	532.79	01c0152b-020f-9864-4aa2-0709689c6c67
l		{job_id: 141513942 etl_stp_sqnc_	630	Statement reac	1	10.00	9,023.82	44,932.65	36,490.53	61,542.60	110.62	01bfc421-020f-37e0-4aa2-070911c0572b

Table 4.7 Queries sorted by the volume of data (GB) spilled to remote storage are candidates for tuning

Applying modeling and optimization, which we discuss in the next chapter, can determine the potential benefits of improved performance and reduced costs from reduced spillage.

4.2.8 Seasonal peaks

Seasonal peaks are recurring patterns in application behavior that occur at specific times. Performance management (providing the right amount of resources on time) should be synchronized with response time peaks to meet SLGs.

Seasonal peak statistics by application and warehouse name are shown in Table 4.8 and Figure 4.8.

App Name	WH	Parameter Name	Peak Type	Peak Start	Duration(Hours)	Avg Amplitude	Standard Deviation	Min Value	Max Value	Peak Length STD
		Total I/O [IO/Hour]	Daily	16	1	222,121,448	120,314,735	115,073,819	653,584,639	0.00
		Response Time [ms]	Daily	0	1	33,074	15,706	5,190	73,197	0.00
		Total Delay [sec]	Daily	3	1	4,848	3,973	4	20,219	0.00
		Total Delay [sec]	Daily	16	1	5,863	5,118	3,043	32,423	0.00
		Throughput [Req/Hour]	Daily	16	1	10,931	3,608	5,208	16,032	0.00
		I/O Per Request	Daily	4	1	417,042	244,870	13,905	927,651	0.00

Table 4.8 Detected seasonal peaks are used to optimize resource allocation by workload

4.3 Summary of FinOps observability for agentic AI

We reviewed an automated observability framework tailored for agentic AI systems. The approach provides a business-process-centric view that spans every layer of the agentic AI stack, ensuring that all optimization efforts remain directly aligned with business objectives.

An automated performance and cost analysis detects anomalies and identifies their root causes. Intelligent monitoring techniques further reduce system overhead by clustering similar agents and adjusting data-collection frequency based on workload stability.

It narrows the scope of performance management to address the most critical problems affecting cost and performance. It starts by identifying the business processes, agents, applications, and critical queries with the highest frequency and severity of anomalies. Then it determines the root causes, the most resource-intensive and expensive queries,

the queries with the highest frequency of failures, and the queries with the highest volume of spilled data to identify hundreds of critical queries from millions of queries being processed.

Determining the most significant response time components among orchestration, RAG, LLM, vector store, and CPU, memory, I/O, and network bottlenecks helps predict the potential outcome of the tuning efforts.

4.4 Key Takeaways: Performance tuning for agentic AI systems

- **Workload Characterization:** Automated characterization transforms raw measurement data into actionable performance and cost profiles for each business process.
- **Proactive vs. Reactive:** Seasonal patterns and trends identified in workload profiles enable proactive capacity management instead of reactive "firefighting".
- **Targeted Tuning:** Significant cost and performance gains typically come from addressing a small subset of critical anomalies and resource-intensive queries.
- **Root-Cause Discovery:** Effective tuning requires identifying the specific concurrent events that cause severe performance outliers.
- **Wasted Cost Elimination:** Tracking failed queries and virtual warehouse idle time identifies immediate opportunities to reduce wasted credits without impacting performance.

References to Chapters 1, 2, 3 and 4

FinOps & Cost Management for Agentic AI

[1] **Storment, J.R.; Fuller, M.** *Cloud FinOps* (2nd ed., O'Reilly, 2023). Foundational FinOps playbook used by most enterprises. Amazonfinops.org

[2] **B. Zibitsker, A. Lupersolsky**, "Cost Optimization and Performance Control in The Hybrid Multi-Cloud Environment", ICPE 2025

[3] **FinOps Foundation WG.** "Cost Estimation of AI Workloads" (May 2024). Practical templates and methods across dev→prod. finops.org

[4] **FinOps Foundation WG.** "How to Forecast AI Services Costs in Cloud" (July 7, 2025). AI-specific forecasting guidance. finops.org

[5] **Nawrocki, P. et al.** "FinOps-driven optimization of cloud resource usage for ML" (Journal article, 2024). ML + FinOps modeling. ScienceDirect

[6] **ESG (Dell Technologies).** "Understanding the Total Cost of Inferencing Large Language Models" (2025, analyst white paper). On-prem vs cloud RAG inference economics. Dell

[7] **InferSave (research).** "Cost-Efficient LLM Serving in the Cloud" (Apr 2025). VM selection + KV-cache economics for inference. arXiv

Observability (General → GenAI/Agents)

[8] **Majors, C.; Fong-Jones, L.; Miranda, G. Observability Engineering** (O'Reilly, 2022). Modern observability foundations. Amazoninfo.honeycomb.ioshroffpublishers.com

[9] **OpenTelemetry (Spec)**. "Semantic conventions for Generative AI systems" (living spec, 2024–2025). Traces/metrics for LLMs, Vector DBs, Agents. OpenTelemetry+3OpenTelemetry+3OpenTelemetry+3

[10] **OpenTelemetry Blog**. "AI Agent Observability – Evolving Standards and Best Practices" (Mar 6, 2025). Status of GenAI SIG work and scope. OpenTelemetry

Agentic AI (Concepts & Surveys)

[11] **Park, J.S. et al**. "Generative Agents: Interactive Simulacra of Human Behavior" (CHI 2023). Seminal agent architecture (memory, planning, reflection). ACM Digital LibraryarXiv

[12] **Durante, Z. et al**. "Agent AI: Surveying the Horizons of Multimodal Interaction" (2024). Broad agentic-AI survey with multimodal focus. arXiv

[13] **Pati, A.K. et al**. "Agentic AI: A Comprehensive Survey…" (2025, survey). Overview of autonomy/memory/goal-directed behavior. ResearchGate

[14] "Generative to Agentic AI: Survey, Conceptualization, and Roadmap" (Apr 26, 2025). Deep survey of the paradigm shift to agents. arXiv

RAG & Evaluation/Observability for RAG Systems

[15] **Shahul Es et al**. “RAGAS: Automated Evaluation of Retrieval-Augmented Generation” (EACL 2024 + arXiv 2023). Reference-free eval metrics (faithfulness, context relevance/recall). ACL Anthology+1arXiv

[16] **Zhang et al**. “Evaluation of Retrieval-Augmented Generation: A Survey” (July 2024). Taxonomy of RAG metrics/datasets. arXiv

[17] **Meta-survey**. “Retrieval-Augmented Generation Evaluation in the LLM Era: A Survey” (Apr 2025). Updated landscape and meta-analysis. arXiv

[18] **Snowflake Engineering**. “Eval-Guided Optimization of LLM Judges for the RAG Triad” (Feb 4, 2025). Practical eval loop for RAG/agents using LLM-as-judge. Snowflake

Practical Tracing/Evals Tooling (useful secondary references)

[19] **LangSmith** Docs. Evaluation quick start & tracing concepts (ongoing). LangSmith

[20] **TruLens** Docs. Feedback-function-based evaluations for agents/RAG. trulens.org

[21] **LangSmith** ecosystem overviews (industry reports/guides). Useful for “related work” context. LangChain BlogAnalytics Vidhya

Other

[22] Herzig et al., 2021 “Unlocking Datasets: Overcoming Limitations of Text Datasets for Language Model Training.”

[23] UiPath_2026_AI_and_Agentic_Automation_Trend_Report

[24] Dennis Smith, Marco Meinardi, Ken Rothenberger, Ang Troy. Gartner Magic Quadrant for Cloud Financial Management Tools, 15 Sep 2025.

5 Optimizing Cost And Performance Control Decisions For An Agentic AI

In this chapter, we will review the opportunities and limitations of benchmarking and modeling for optimizing strategic and tactical decisions in an agentic AI environment.

We will discuss what is happening under the hood of Queueing Network Models and Gradient Optimization and review several typical examples that are valuable for executives, managers, and architects. The main point is that modeling and optimization generates cost and performance expectations, enabling result verification and organizing a closed-loop performance and cost control.

Decisions regarding the selection of appropriate platforms in multi-layered agentic AI architecture, as well as designing new agents, optimizing orchestration, and managing workloads, significantly impact the cost and performance of agentic AI environments. There are significant uncertainty and a high risk that poor decisions could be very costly.

The goal of modeling and optimization is to evaluate options and develop recommendations to optimize cost and performance, including:

- **New applications sizing before deployment**: optimize design decisions, size new applications during the DevOps process before deployment and determine the appropriate platform and minimum configuration needed to meet SLGs on each Layer
- **Performance management**: optimize performance management decisions and support optimization decisions for agent structure, tool usage, and scheduling
- **Dynamic capacity management and cost optimization**: find configuration and workload management changes needed to support expected workload growth and continuously meet SLGs, evaluate the credits reservation vs. pay-as-you-go on-demand strategies, and predict the realistic budget needed to meet SLGs of business processes with the lowest cost

Observability results are used by modeling and optimization to determine the minimal configurations, resource allocation and workload management rules, and budgets needed to meet SLGs for all Agents using the RAG, LLM, MCP, and Vector Store components of the agentic AI system.

5.1 Agentic AI systems platforms overview

How to select appropriate cloud platforms for RAG, LLM, Vector Store, and Orchestration for agentic AI agents to meet SLGs of business processes?

There are many platform options for building agentic AI systems [5,13]. Let's focus on the cloud data platforms most frequently used as a base for building enterprise-agentic AI systems. We will examine the similarities and differences among the major platforms that influence the performance and cost of agentic AI applications.

The architecture of each cloud data platform directly influences **performance, scalability, and cost-efficiency** for agentic AI workloads. These architectural choices determine how quickly Agents can access data, how effectively they can scale under variable load, and how predictable costs remain during fluctuating demand.

Let's compare the differences of the cloud data platforms' features supporting agentic AI:

- **Architectural Influence:** The architecture of the underlying cloud data platform such as its execution engine and isolation model directly dictates cost-efficiency and scalability
- **Explicit Billing Models:** Different billing units (Snowflake credits, BigQuery slots, Databricks DBUs) must be mapped to specific agent behaviors and SLGs to predict budgets accurately
- **No Universal "Best" Platform:** Platform selection depends on specific workload shapes, such as whether a workload is highly bursty or requires deep structured data concurrency
- **Quantified Trade-offs:** Modeling turns platform selection from a qualitative guess into a quantitative trade-off analysis.

5.2 Role of benchmarking in the evaluation of agentic AI

Benchmark / Suite	Task Types	Metrics Measured	Supported Domains / Notes	Best Use Case
GAIA (General AI Assistant)	Multi-step reasoning, tool use, planning, web tasks	Success rate, steps to completion, tool use correctness, latency	General agent tasks across open environments	Broad agent capability evaluation
WebArena / OSWorld / Mind2Web	UI navigation, web browsing, file & environment interaction	Interaction success, time-on-task, error recovery, robustness	Web and desktop simulations	Agents requiring UI automation & integration
SWE-Bench / Terminal-Bench	Coding, shell command execution, system tasks	Functional correctness, execution cost, safety	Code and environment manipulation	Tool-use & execution quality evaluation
AstaBench	Diverse set: math, logic, code, puzzles, reasoning	Task success, quality scores, cost per task, latency	Hybrid open tasks + benchmark task battery	Holistic agent ranking + cost evaluation
Salesforce CRM Benchmarks	Enterprise task workflows (CRM tasks, data lookup)	Accuracy, latency, cost, business utility, safety	Commercial workflows	Enterprise automation evaluation
AI Agent Benchmark Compendium	Catalog of many benchmarks	Depends on sub-benchmark (tool use, logic, navigation)	Community-curated, multi-domain	Exploratory selection & profiling
AILuminate Agentic Maturity	Safety, reliability over production scenarios	FP / FN safety metrics, uptime, error rate	Product maturity eval	Go-to-market readiness & risk
UAVBench / Domain-specific	Navigation, planning, mission execution	Mission success, safety, LLM precision, energy		

Table 5.1 Most frequently used agentic AI Benchmarks

5.3 Key metric categories (across all benchmarks)

Category	Representative Metrics	Why It Matters
Task Success	Completed goal, % success	Measures reliability of agent logic
Efficiency	Steps taken, time to finish	Indicates planning effectiveness
Quality	Correctness, mistakes, hallucination	Ensures results are meaningful
Tool Use Effectiveness	API accuracy, integration correctness	Critical for real-world agents
Cost	Compute cost (GPU/CPU), token cost	Central for FinOps in production
Latency	Average / worst-case latency	Critical for SLG/SLA compliance
Robustness / Safety	Error recovery, safe outputs	Important for trust & compliance
Resource Usage	GPU/CPU hours, memory, bandwidth	Helps sizing & cost projection
Stability	Consistency over time	Needed for production reliability

Table 5.2 Benchmark Key Metric Categories

5.4 Core problems with relying on benchmarks in agentic ai

Below are the key reasons why benchmarks alone are insufficient and often harmful for agentic AI cost and performance optimization.

5.4.1 Benchmarks measure static not dynamic behavior

Benchmarks assume fixed workload, predictable execution flow and consistent resource use.

Agentic AI reality: agents spawn other agents, retries, recursion, backtracking, varying reasoning paths per run, **execution cost changes based on *decisions the agent makes*.** So, a benchmark may show "Cost = $0.05 per task", but real system may become: "Cost suddenly $0.50 because agents entered a reasoning loop". In other words, the benchmarks dramatically underestimate ***true*** cost and performance risk.

5.4.2 Benchmarks assume deterministic performance

Benchmarks cannot reflect **variance risk**, yet variance is what kills budgets and SLGs. Benchmarks give averages while real businesses suffer from tail performance.

5.4.3 Benchmarks test capability - enterprises need reliability

Benchmarks ask: "Can the agent solve this?" Businesses need to ask, "Can the agent solve this 1,000,000 times reliably this month without blowing the budget?"

Benchmarks rarely measure consistency, drift over time, reliability, regression after model updates, or degradation under load. As a result, a platform that benchmarks well may fail in long-running real-world use.

5.4.4 Benchmarks rarely measure the total cost of execution

They often miss GPU idle cost, orchestration overhead, memory footprint of vector stores, storage and retrieval fees, retries and backoffs, failure cleanup cost, cost of hallucinations and corrections, token expansion due to longer contexts, concurrency scaling penalties and multi-cloud routing differences.

Most benchmark results simplify cost to: "token cost + compute time", but agentic AI FinOps reality is **much broader and more expensive**. As a result, benchmarks create **false financial confidence**.

5.4.5 Benchmarks optimize for accuracy, but enterprises need cost-for-value

In the enterprise environments, the key question is "What is the ***cheapest predictable configuration*** that meets business SLGs?"

Benchmarks do not answer that. As a result, the benchmarks produce "best performing" systems instead of "best business systems". Using benchmarks, you optimize for a lab toy instead of your real business.

5.4.6 Vendors can (and do) tune to benchmarks

Vendors optimize pipelines around benchmark workloads, special-case certain prompts/tasks, tune cache/retrieval/execution flow, and build shortcuts for benchmark suites. So, benchmark results represent "best-case engineering showcase", not "expected reality in your messy environment".

Benchmarks increasingly tell you how well the vendor performs in benchmarking, not in real performance.

5.4.7 Benchmarks do not scale to business forecasts

They do not predict future workload growth, forecast GPU demand curves, predict performance under increased concurrency, estimate next-year budget, or calculate elasticity under burst load.

Enterprises care deeply about future spending, scale resilience, and predictable operations. Benchmarks are snapshots; businesses need **forecasts**.

5.5 Key Takeaways: Role of benchmarks

Benchmarks are useful to compare basic capability, sanity-check technology and measure specific subsystems, but they **cannot** optimize cost, guarantee performance, ensure reliability and support enterprise decision-making [17].

For agentic AI, benchmarks must be combined with **real-world workload observability, modeling, optimization, and continuous, closed-loop control.**

6 Modeling And Optimization Technology

For people interested in how modeling and optimization technology works in general and what is modified to model and optimize decisions in complex agentic AI environments, we will review several examples that explain this.

Modeling and optimization use data from observability, workload forecasting, and scenario planning as input:

- Observability provides measurement data characterizing agents' performance, resource utilization, data usage, cost, and current configuration
- Workload forecasting describes the expected increase in volume of data, in the number of agent types, and in the number of agents running simultaneously and SLGs
- Scenario planning describes the configuration options and trade-offs that need evaluation.

We evaluated various modeling technologies [8-12]. While simulation models offer high accuracy, they are time-consuming to develop and use. Machine Learning and Generative AI models, although promising, require extensive training on diverse measurement data that may not cover all scenarios. Therefore, we selected queueing network models combined with gradient optimization to achieve the desired accuracy and efficiency for ongoing SLG compliance across applications.

We adapted universal closed queueing network models (QNM) to support multi-layer agentic AI environments. Gradient optimization techniques are used to determine the smallest configurations and workload management adjustments needed to meet business SLGs across all applications and platforms.

Our modeling results establish clear performance and cost expectations, enabling the implementation of a continuous, closed-loop feedback control system.

6.1 Introduction to queueing network models (QNM)

Let’s review an example of predicting workload performance for expected growth over the next 12 months using a single server. For illustration, let’s use a simple open QNM with a single server, supporting requests arriving at an average rate of 2.5 per second. The server's average service time is 0.3 seconds (Figure 6.1).

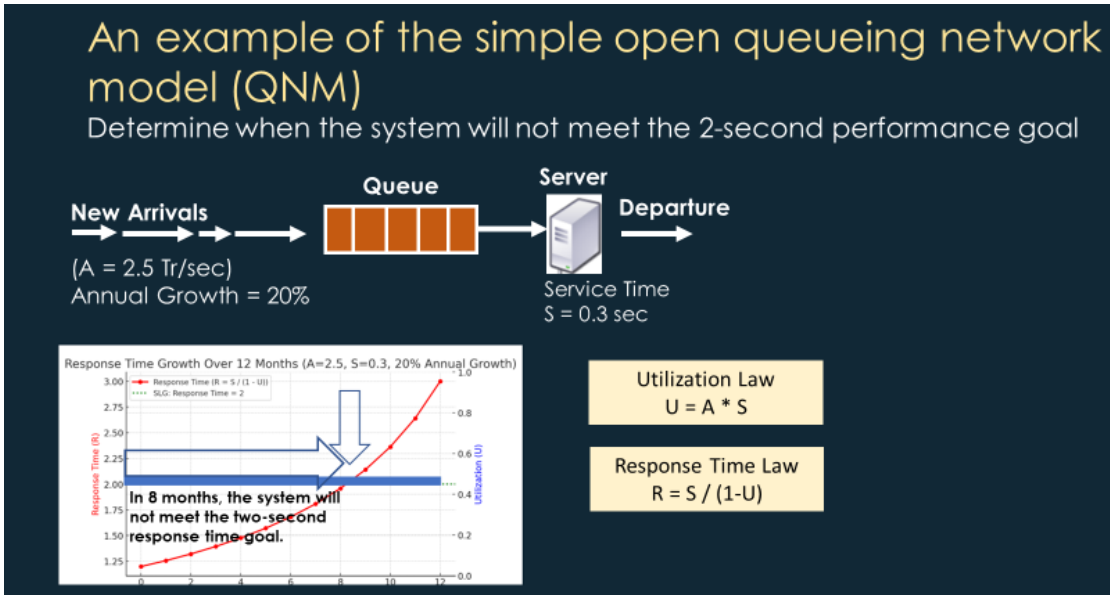


Figure 6.1 Predicted response time for the next 12 months using Utilization and Response Time Laws.

Let's use the Utilization Law to calculate server utilization and the Response Time Law to predict the response time.

As we can see on Figure 6.1 the current configuration will not be able to support a 2-second SLG in 8 months, and a configuration change will be needed.

Among the available server types, we identified the one that reduced service time from 0.3 seconds to 0.25 seconds per request. Optimization here means finding the server capacity that reduces service time to the value necessary to meet the SLG (Figure 6.2).

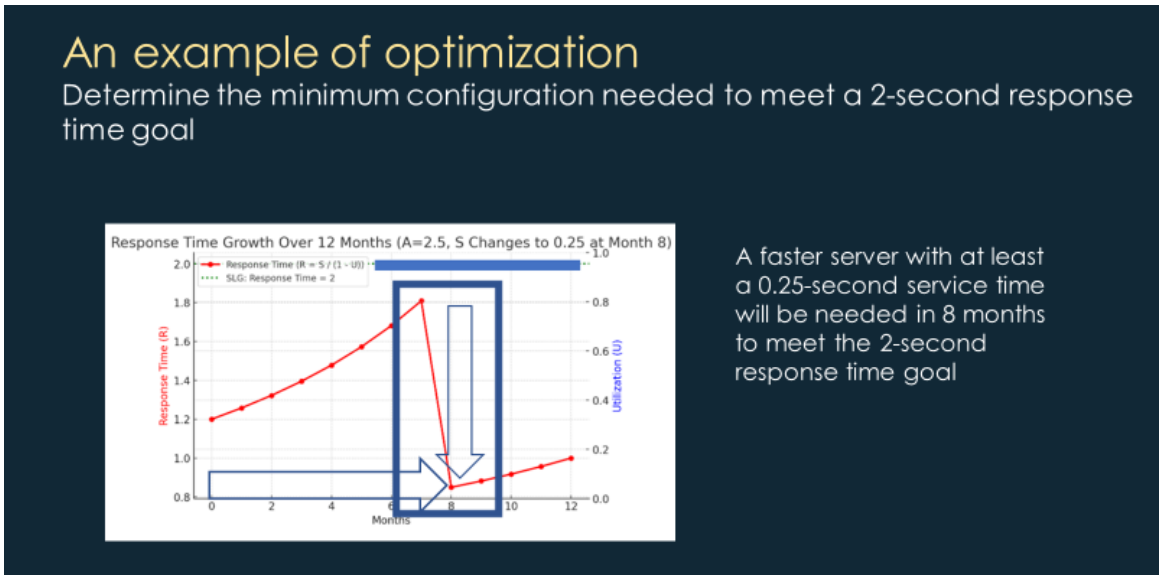


Figure 6.2 The minimum server configuration available will reduce the service time from 0.3 seconds to 0.25 seconds. It will support response time SLG till the end of the year.

The described open QNM has a limitation. The model assumes that the arrival rate of requests is independent of server utilization. To model more general cases, organizations use closed QNMs. The closed QNM assumes that each user waits for the server's response before preparing ("think time") and sending the next request. That means the throughput in closed QNMs depends on the server response time. It's more realistic in most cases.

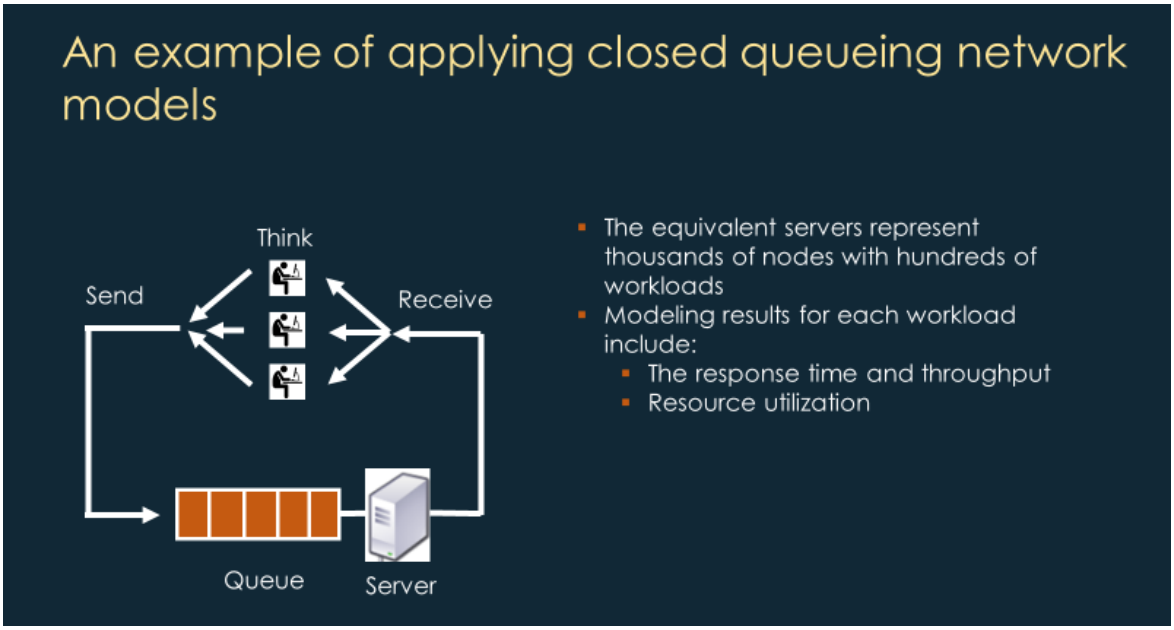


Figure 6.3 The closed queueing network model. The request round-trip includes think time when the user prepares the request, queueing time when requests wait for the server, and the service time of the request.

6.2 Queueing network modeling and gradient optimization of agentic ai systems

Agentic AI systems employ intelligent agents that independently plan, reason, and interact across multiple layers of software and infrastructure supporting RAG retrieval engines, LLMs, vector databases, orchestration frameworks, and cloud data platforms. This layered, interconnected configuration supports dynamic workloads.

We use closed QNMs [7] in each step of gradient optimization. This enables us to evaluate the performance and cost of the system configuration at each step and, finally, determine the optimal configuration, workload management, and costs needed to meet SLGs for each workload.

Each server's QNM includes multiple nodes that correspond to hardware resources, such as compute nodes, storage arrays, and interconnecting channels. Workload requests enter queues for these nodes, wait for service, and are serviced according to the hardware and software configuration of each node.

We adapted the Mean Value Analysis (MVA) algorithm [8] to model workload management parameters, including priorities, concurrency levels, and resource utilization limits.

The iterative, approximate MVA algorithm, which allows queue sizes to converge to steady-state values quickly, is a more efficient approach for complex queuing network models where workloads share resources and requests are processed concurrently. This method supports various service-time distributions for processor-sharing nodes and exponential distributions for FIFO queues, providing a practical approximation of real-world computing processes.

To model a multi-layer environment, we created a multi-tier, multi-server network of QNMs, where each server is represented by its own QNM. Figure 6.4 shows an example of such a complex model. The output from the upper server is an input (request) to the lower (next) server on different layers. The response time of the next server is included in the upper (calling) server's response time. A single upper server request can result in multiple calls to different lower servers.

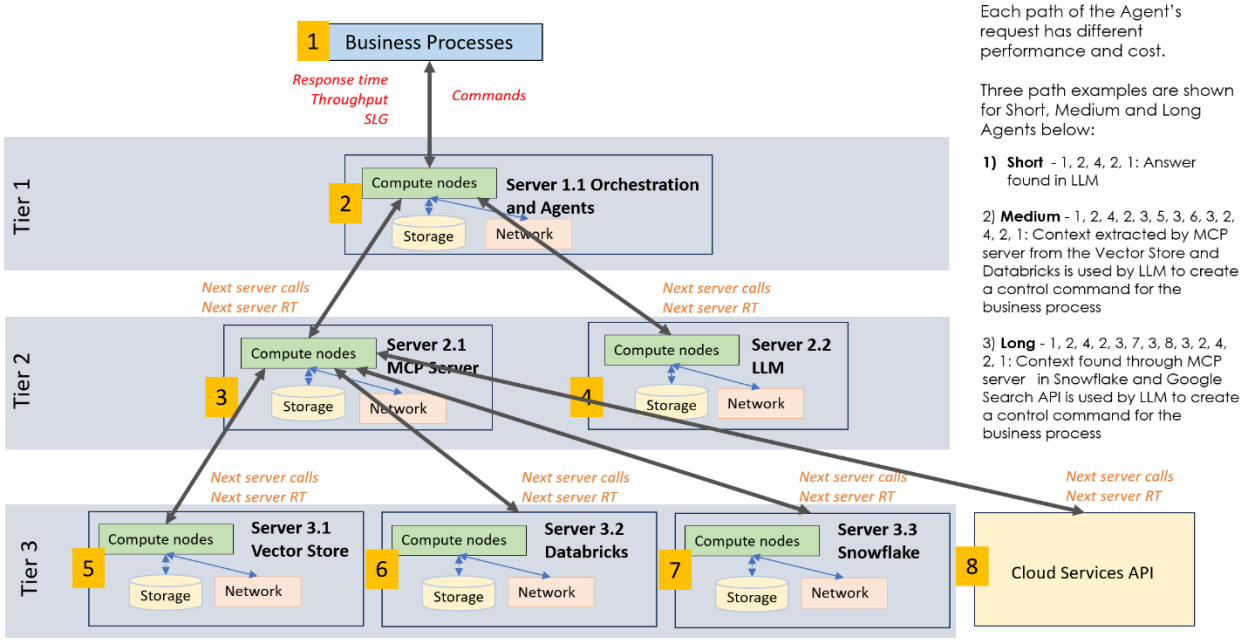


Figure 6.4 Example of Closed QNM of Multi-Layer Agentic AI system

Based on observability data collected at each layer and each component/server of the agentic AI system, we use workload characterization to generate a pair (one per called server): the next server ID and the number of next-server calls per workload. These pairs are then recalculated into routing probabilities for the workload's requests as they move through the network of QNMs.

During model calibration, the measured next-server response time is used to calibrate the calling server's QNM. Calibration means finding values of the model parameters that

we cannot measure. It is an iterative process that stops when the model response time and throughput become close enough to their measured values.

During the evaluation of proposed configuration changes, each server's QNM is calculated from the bottom to the top of the network to provide the new next server response time for the calling server's QNM. On the other hand, both changes to the upper server's configuration and changes to the next server's response time affect the calling server's throughput, which is ultimately reflected in the equivalent "think time" for the calling servers. Thus, the evaluation process becomes iterative as well, and it stops when it converges to some stable values of each server's response time and throughput.

A special kind of next server calls are API calls to some services that are "black boxes", and their configuration and resource consumption are unknown. For these services, the measured response time remains unchanged during the scenario evaluation.

An example of three agents concurrently supporting a business process is shown in Figure 6.5.

A different color represents each agent's execution time. Some agents interact with the same components of the agentic AI system multiple times.

The light-green area shows that only the short agent (yellow) meets its SLG, while the medium (brown) and long (blue) exceed the business process SLGs.

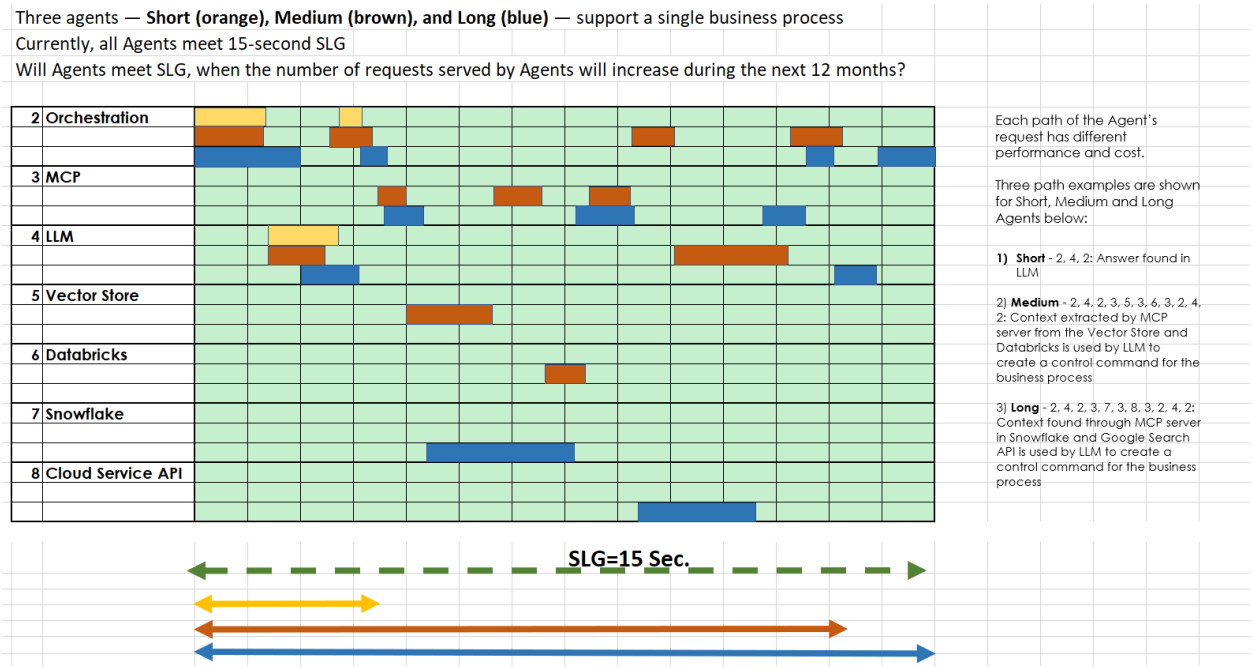


Figure 6.5 An example of the time diagram for three agents supporting a business process

6.3 Example of applying the gradient optimization for agentic AI systems

On each optimization step for each server

- The optimization algorithm changes the hardware configuration and workload management parameters
- The optimization algorithm sets the SLG for the called servers
- The model determines new performance
- The optimization process compares the predicted response time with SLGs
- This process is repeated until the SLGs are met for all workloads.

As SLGs are defined by business owners for the entire environment, our optimization algorithm [9] begins by evaluating the current or proposed configuration of each server in the execution chain. The QNM assesses server performance, with emphasis on the **top-level (user- or business-facing) server**, calculating the average response time for each workload. These response times are compared against their corresponding SLGs to determine the configuration adjustments required for this server, as well as the **target response times** that must be met by any downstream servers it invokes. These targets effectively become the SLGs for those underlying servers.

Optimization of the top-level server continues iteratively until all business SLGs are satisfied at the lowest possible cost. The same process is then applied to each server called by the top-level server, followed by the servers they depend on, and so forth - propagating the requirements downstream until all underlying servers meet the SLGs established by the servers that depend on them.

There are two optimization phases:

- **Redistribution of existing resources**. Resources are reallocated across workloads by adjusting priorities based on each workload's ratio of average response time to its SLG. This approach maximizes the effectiveness of the current configuration without increasing capacity.
- **Resource allocation adjustment**. If redistribution alone cannot satisfy SLGs, the ratio of predicted response time to the SLG determines the required magnitude of configuration changes, while the weights of each response time component guide how this adjustment should be distributed across individual resources.
  Conversely, if workloads exceed their SLGs (i.e., perform faster than required), the algorithm identifies which resources can be safely reduced - lowering cost while maintaining compliance with SLGs.

The optimization process is illustrated in Figure 6.6.

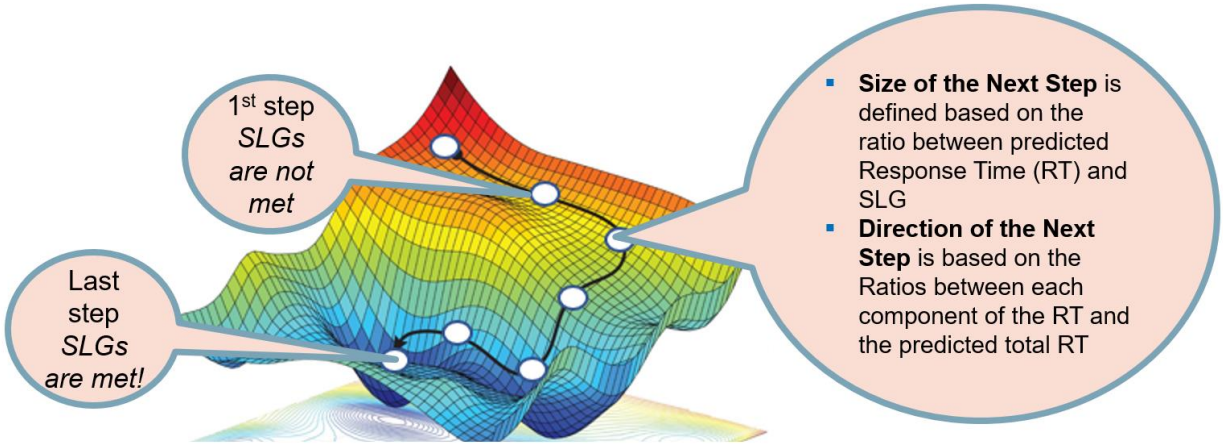


Figure 6.6 Queueing network modeling and gradient optimization process

6.4 Automating observability, modeling, and optimization process

A typical process includes the following steps:

- **Observability setup:** Install the software and establish automated observability across all platforms. The objective is to detect performance and cost anomalies, identify their root causes, and generate proactive performance-tuning recommendations.
- **Workload profiling:** Create detailed profiles for each business workload, capturing key metrics such as cost, performance, resource utilization, and data usage for every measurement interval.
- **Selection of representative intervals:** Choose time periods that accurately reflect typical workload behavior - including peak, average, and off-peak conditions - so the model captures true variability over time.
- **Modeling and optimization:** Apply QNM and gradient-based optimization to determine the minimal configuration and workload-management adjustments required to meet SLGs for each workload at different times of day and throughout the year.
- **Budget estimation:** Estimate the cost of the optimized configuration using the pricing models of the relevant cloud providers, ensuring that the proposed budget aligns with performance requirements.
- **Performance and cost expectations:** Deliver realistic expectations for performance, resource utilization, and cost based on the optimized configuration, enabling proactive, ongoing cost and performance control.

The number of agents in an agentic AI system is growing rapidly. Based on current adoption patterns, we expect Fortune-2000 organizations to experience exponential-like growth in the number of Agents between 2025 and 2027. This expansion is driven by increasing automation, multi-agent orchestration, and platform-level tools that programmatically generate new agents. As a result, enterprises that deploy dozens of agents in 2025 may operate hundreds or even thousands by 2027.

To reduce the overhead of observability and simplify the QNM modeling and optimization process, we group agents with similar performance and resource usage patterns. Then, we identify the minimum configuration required to meet SLGs for each group.

6.5 Key Takeaways: role of modeling and optimization

- Queueing Network Models (QNMs) provide a **system-level view** of agentic workloads
- Gradient optimization enables **multi-objective trade-offs** between cost, latency, and throughput.
- Modeling converts observability data into **predictive insight**, not just historical reporting.
- **The Minimum Configuration:** Optimization algorithms identify the smallest possible hardware configuration and workload rules that satisfy business SLGs.
- **Closed-Loop Enablement:** Accurate modeling establishes the performance and cost expectations necessary to run a continuous feedback control loop.

7 Examples of Applying Modeling And Optimization

We will review several typical examples of optimizing strategic and tactical decisions, which are valuable for everyone interested in applying modeling and optimization in a multi-layer system managing business processes with different SLGs [13-23].

During the journey toward the agentic AI world, organizations typically make many strategic and tactical decisions, including selecting a platform, determining the minimum configuration and cost needed to meet business process SLGs, sizing new Agents before deployment to production, implementing dynamic capacity management, and maintaining continuous closed-loop performance and cost control.

Let's review several examples of applying modeling and optimization to address these key challenges. We will review examples illustrating:

- Selecting the appropriate agentic AI platform and determining the minimum configuration required to meet SLGs for RAG, LLM, Vector Store, and Orchestration components.
- Sizing new agents before deployment, ensuring they meet performance objectives without unnecessary overprovisioning.
- Organizing dynamic capacity management to adapt to changing workloads throughout the day and across seasons.
- Developing a realistic and cost-effective budget based on actual workload behavior and optimized configurations.

In these examples, the modeling and optimization methodology recognizes that workload fluctuations, workload management, and configuration changes to a single component of the multi-tier agentic architecture can directly affect the performance of other components. By accounting for these interdependencies, it identifies the **minimum set of configuration and workload-management adjustments** necessary to meet business SLGs **at the lowest cost**.

7.1 Select an Appropriate Agentic AI Platform

Enterprise Agentic AI platform decisions affect response time, concurrency stability, governance, and cost predictability. Let's review differences in architecture and the impact of several platforms on performance and cost.

Snowflake leads in predictable isolation, strong native MCP alignment, and credit-governed cost control.

BigQuery delivers the most elastic serverless model with automatic scaling and minimal DevOps effort.

Databricks excels in engineering-driven Lakehouse AI environments with strong pipelines and hybrid workloads.

Teradata VantageCloud remains the strongest choice for mission-critical, highly regulated, high-concurrency environments, with unmatched observability depth. Teradata has recently added a Vector Store and long-term support for vector operations in the engine. Agentic AI work is well suited to the traditional Teradata workload management controls. Teradata also supports Bring Your Own Model (BYOM), which can be useful for many AI applications, including RAGs. Open-source catalogs, such as Hugging Face, offer a wide range of models useful for many scenarios when building your AI-based applications.

Azure Synapse benefits organizations fully committed to Azure, though performance depends heavily on data distribution and requires a disciplined tuning approach.

Oracle AI Database 26ai offers a uniquely integrated "AI inside the database" model with HeatWave in-memory acceleration and strong governance for industries prioritizing security, compliance, and predictable performance.

Sometimes organizations use benchmark test results during platform selection, but in practice, the "best platform" depends not only on benchmarks but also on workload shape, governance priorities, performance-predictability requirements, dynamic scaling needs, and cost-model tolerance.

7.1.1 Decision recommendation matrix

The table below shows the recommended platform, based on the most important decision factors

Decision Priority	Recommended Platforms	Why
Lowest Risk, Predictable SLG Compliance	Snowflake, Teradata	Strong isolation, predictable execution, mature governance
Most Elastic / Bursty Agent Workloads	BigQuery, Databricks	Autoscaling / high throughput, resilient pipelines
Deepest Observability & Traceability Needs	Teradata, Snowflake	Full cost + telemetry correlation
Strongest Native MCP Ecosystem Alignment	Snowflake, Teradata, Databricks	Built-in or mature MCP integration
Minimal Operations Effort	BigQuery, Snowflake	Serverless + auto-suspend, simplified management
Highly Regulated / Governance-Intensive Environments	Teradata, Oracle	Built-in governance, audit, predictable controls
Azure-Centric Enterprise Strategy	Azure Synapse	Ecosystem integration advantage
Engineering-Heavy AI Innovation Platform	Databricks	Powerful execution engines + pipeline control
Security First + AI Embedded in DB Strategy	Oracle HeatWave	In-memory AI with enterprise controls

Table 7.1 Recommended Platform by Primary Decision Priority

7.1.2 Fast guidance heuristic

- If your priority is **predictability and governance → Snowflake / Teradata**
- If your priority is **elastic scaling and low DevOps → BigQuery**
- If your priority is **AI engineering power and pipelines → Databricks**
- If your priority is **Azure alignment → Synapse**
- If your priority is **compliance + embedded AI → Oracle**

If you want to compare the costs required to meet SLGs for your agentic workloads, you can use modeling and optimization to derive realistic performance and cost expectations.

7.1.3 Determining the minimum configuration and cost

After selecting the platform type, you need to determine the **minimum configuration and budget** required for Orchestration, Common and Local LLM, MCP Server and Vector Store to meet business SLGs on different platforms.

This includes identifying the optimal **node types**, **number of nodes**, and **storage configuration** across different hours of the day, days of the month, and months of the year.

Models use the results of observability and workload characterization, aggregating agents' requests supporting each business process into short, medium and complex workloads.

The multi-tier model accounts for the fact that some agents request visits to each component of the agentic AI system multiple times. The total response time for an agent is the sum of the times requests spent on each tier. Different platforms use different instance types, with different optimizers, workload management mechanisms, scalability, and resource allocation rules.

All these factors affect the minimum configuration of each component of agentic AI needed to meet the business process SLG.

These universal models can be used to address a variety of what-if questions across platforms.

7.1.4 Example of the observability results used by modeling and optimization

Workload parameters	Workload name		
Parameter	AgentsShort	AgentsMedium	AgentsLong
Concurrent connections	10	10	10
Total requests per hour	1,800	1,800	1,800
Average response time (s)	3.200	12.250	14.000
Average execution time (s)	3.200	12.250	14.000
Average parallel tasks per request	1.000	1.000	1.000
CPU utilization (%)	4.549	9.749	9.622
IO rate (operations/s)	1.000	2.000	3.000
MB per IO operation	0.125	0.125	0.125
Total network messages per hour	9,000	18,000	18,000
Next server ID		83980	83980
Total next server calls		1,800	1,800
Next server ID	83990	83990	83990
Total next server calls	1,800	3,600	3,600

Table 7.2 Orchestration performance observability results

Configuration	
Tier ID	10
Server ID	83970
Number of nodes	2
Instance type	c6gd.2xlarge
CPUs per node	8
Storage connectors per node	2
Local storage	
Disk type	SSD
Search time (ms)	0.01
Transfer rate (MBps)	2500
Number of disks	2
Local storage hit ratio (%)	62.76
Remote storage	
Disk type	SSD
Search time (ms)	0.5
Transfer rate (MBps)	2500
Number of disks	2
Weighted host channel bandwidth (MBps)	2883
Weighted disk channel bandwidth (MBps)	3128
Average network message size (KB)	100
Network bandwidth (MBps)	1000
Max parallel tasks per request	16

Table 7.3 Orchestration server configuration

Workload parameters	Workload name		
Parameter	AgentsShort	AgentsMedium	AgentsLong
Concurrent connections		10	10
Total requests per hour		1,800	1,800
Average response time (s)		4.800	8.100
Average execution time (s)		4.800	8.100
Average parallel tasks per request		1.000	1.000
CPU utilization (%)		5.920	8.362
IO rate (operations/s)		10.000	15.000
MB per IO operation		0.125	0.125
Total network messages per hour		10,800	16,200
Next server ID		84000	84000
Total next server calls		3,600	3,600

Table 7.4 MCP Server performance observability results

Configuration	
Tier ID	20
Server ID	83980
Number of nodes	2
Instance type	c6gd.2xlarge
CPUs per node	8
Storage connectors per node	2
Local storage	
Disk type	SSD
Search time (ms)	0.01
Transfer rate (MBps)	2500
Number of disks	2
Local storage hit ratio (%)	62.76
Remote storage	
Disk type	SSD
Search time (ms)	0.5
Transfer rate (MBps)	2500
Number of disks	2
Weighted host channel bandwidth (MBps)	2883
Weighted disk channel bandwidth (MBps)	3128
Average network message size (KB)	100
Network bandwidth (MBps)	1000
Max parallel tasks per request	16

Table 7.5 MCP Server configuration

Workload parameters	Workload name		
Parameter	AgentsShort	AgentsMedium	AgentsLong
Concurrent connections	10.000	10.000	10.000
Total requests per hour	1,800	3,600	3,600
Average response time (s)	1.299	1.699	0.949
Average execution time (s)	1.299	1.699	0.949
Average parallel tasks per request	43.517	24.880	60.745
CPU utilization (%)	9.938	14.934	19.884
IO rate (operations/s)	200.000	300.006	6,009.554
MB per IO operation	0.125	0.125	0.125
Total network messages per hour	45,000	60,000	80,000

Table 7.6 LLM performance observability results

Configuration	
Tier ID	20
Server ID	83990
Number of nodes	20
Instance type	c6gd.2xlarge
CPUs per node	8
Storage connectors per node	2
Local storage	
Disk type	SSD
Search time (ms)	0.01
Transfer rate (MBps)	2500
Number of disks	20
Local storage hit ratio (%)	62.76
Remote storage	
Disk type	SSD
Search time (ms)	0.5
Transfer rate (MBps)	2500
Number of disks	20
Weighted host channel bandwidth (MBps)	2883
Weighted disk channel bandwidth (MBps)	3128
Average network message size (KB)	100
Network bandwidth (MBps)	1000
Max parallel tasks per request	160

Table 7.7 LLM server configuration

Workload parameters	Workload name		
Parameter	**AgentsShort**	**AgentsMedium**	**AgentsLong**
Concurrent connections		10.000	10.000
Total requests per hour		3,600	3,600
Average response time (s)		1.299	2.499
Average execution time (s)		1.299	2.499
Average parallel tasks per request		9.041	9.630
CPU utilization (%)		12.912	26.817
IO rate (operations/s)		150.001	300.000
MB per IO operation		0.125	0.125
Total network messages per hour		21,600	21,600

Table 7.8 Vector Store performance observability results

Configuration	
Tier ID	30
Server ID	84000
Number of nodes	7
Instance type	c6gd.2xlarge
CPUs per node	8
Storage connectors per node	2
Local storage	
Disk type	SSD
Search time (ms)	0.01
Transfer rate (MBps)	2500
Number of disks	7
Local storage hit ratio (%)	62.76
Remote storage	
Disk type	SSD
Search time (ms)	0.5
Transfer rate (MBps)	2500
Number of disks	7
Weighted host channel bandwidth (MBps)	2883
Weighted disk channel bandwidth (MBps)	3128
Average network message size (KB)	100
Network bandwidth (MBps)	1000
Max parallel tasks per request	112

Table 7.9 Vector Store server configuration

Cost is measured for each Agent on each layer in the Snowflake Cortex environment

Workload Name	Avg. LLM Calls per Request	Monthly Token Volume (In/Out)	Monthly Compute Credits (VWs)	Vector Store / Storage Cost	Total Predicted Monthly Cost	Cost per Business Task (CPT)
AgentsShort	1	5.4M Tokens	145 Credits	$50.00	$1,500.00	$0.08
AgentsMedium	2	18.2M Tokens	480 Credits	$320.00	$5,200.00	$0.29
AgentsLong	2	125.5M Tokens	1,850 Credits	$1,250.00	$21,500.00	$1.19

Table 7.10 Measured performance and cost of running three Agents in Snowflake agentic AI environment.

7.1.5 Cost per task (CPT) review

- **Complexity Penalty**: As agents move from simple Q&A (AgentsShort) to complex multi-step reasoning (AgentsLong), CPT increases exponentially rather than linearly. This is driven by the cascade of actions: recursive validations, retries, and high-volume backend interactions required for a single user goal.
- **Tokens vs. Infrastructure Balance**: For AgentsShort, the CPT is dominated by the initial LLM call. For AgentsLong, CPT includes significant infrastructure overhead from 2XL+ clusters needed to handle massive parallel tasks and high I/O rates.
- **Impact of Wasted Credits**: These CPT figures assume optimized **Auto-Suspend** settings. Without proper idle-time management, the CPT across all workloads would likely be **20% to 25% higher** due to credit loss from inactive virtual warehouses.

7.1.6 Modeling and optimization results – minimum configuration and cost needed to meet business SLG

In this example, we assume that the agents' workload will grow at a rate of 5% per month.

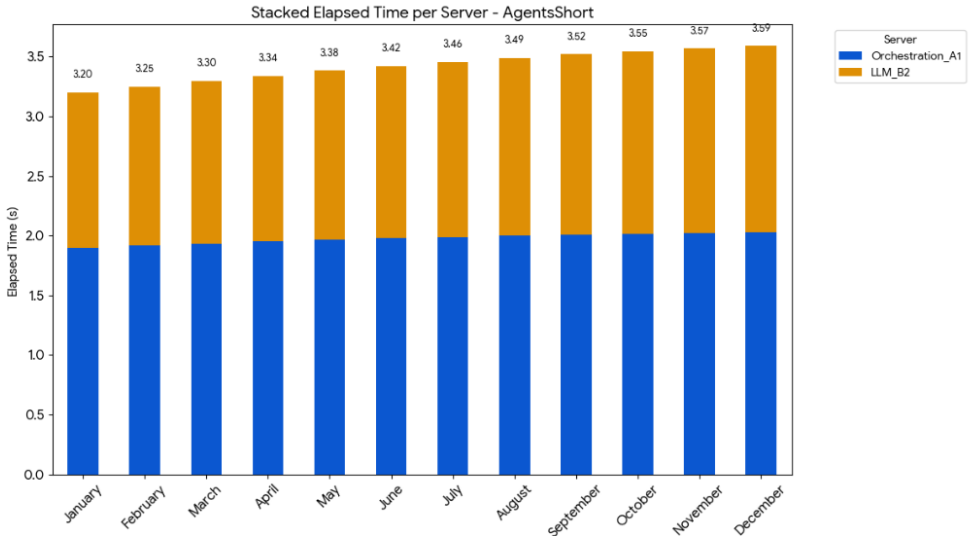


Figure 7.1 Response time for Agents Short accessing Orchestration and LLM Layers will reach 3.6 sec in 12 months which is below 15 seconds SLG

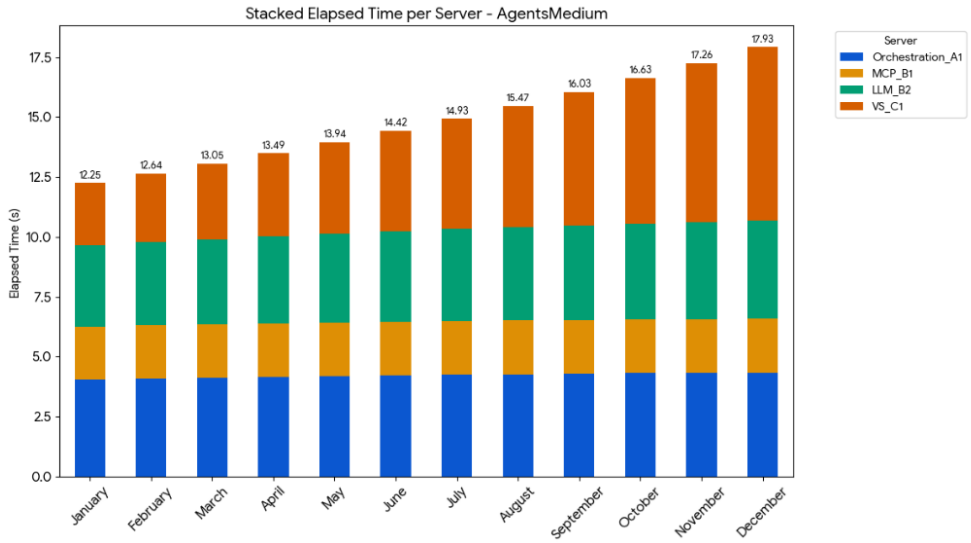
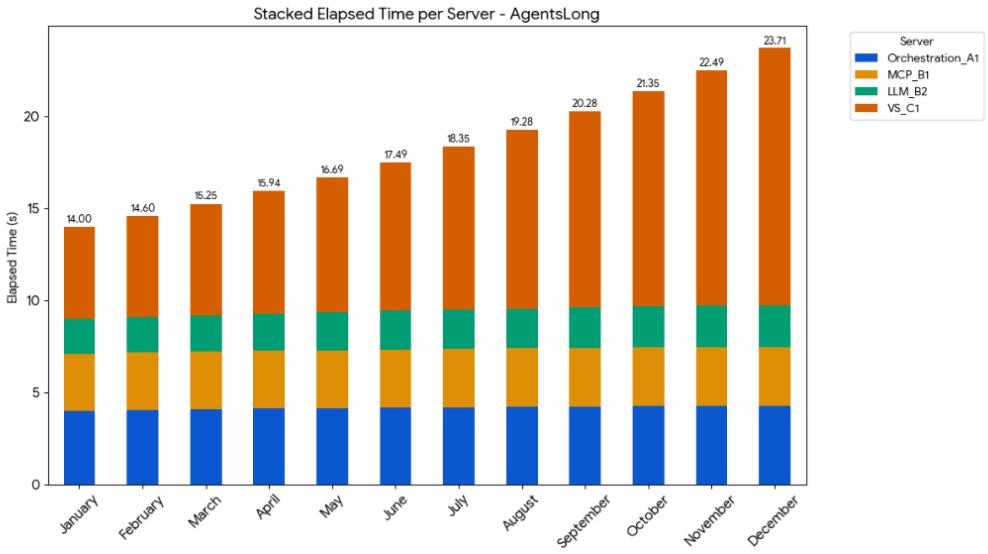


Figure 7.2 The predicted elapsed time for AgentMedium will exceed the 15 sec SLG starting August.

Figure 7.3 The predicted Response time of AgentsLong will exceed the SLG starting March



7.1.7 What should be done to satisfy the business process SLG?

Workload management optimization, tuning Agents, and tuning Vector Store can improve performance, but for illustration purposes, we will review how configuration changes can affect performance. Modeling and optimization help to determine the minimum configuration needed to meet the business process SLG of 15 seconds.

The following stacked bar charts display the predicted Elapsed Time per Request for the `AgentsShort`, `AgentsMedium`, and `AgentsLong` workloads during the next 12 months with the increased Vector Store server configuration. Modeling and optimization determined the minimum Vector Store configuration that will be needed starting March to meet the 15 seconds SLG of the business process. Colors represent response time components on each layer of the agentic AI system.

AgentsMedium workload spans four tiers. After Vector Store configuration increases starting in March, the total response time is significantly lower in the latter part of the year and will be lower than 15 seconds SLG.

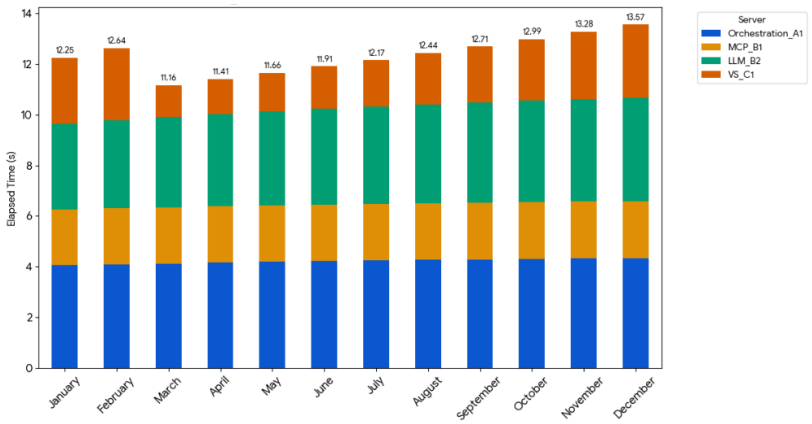


Figure 7.4 Proposed configuration change will allow meeting SLG for AgentsMedium workload during the next 12 months

Similar to the **AgentsMedium** workload, the **AgentsLong** chart reflects the impact of the recommended configuration change in March. As a result of the recommended configuration change, the total response time for AgentsLong Agent will meet SLG during the next 12 months.

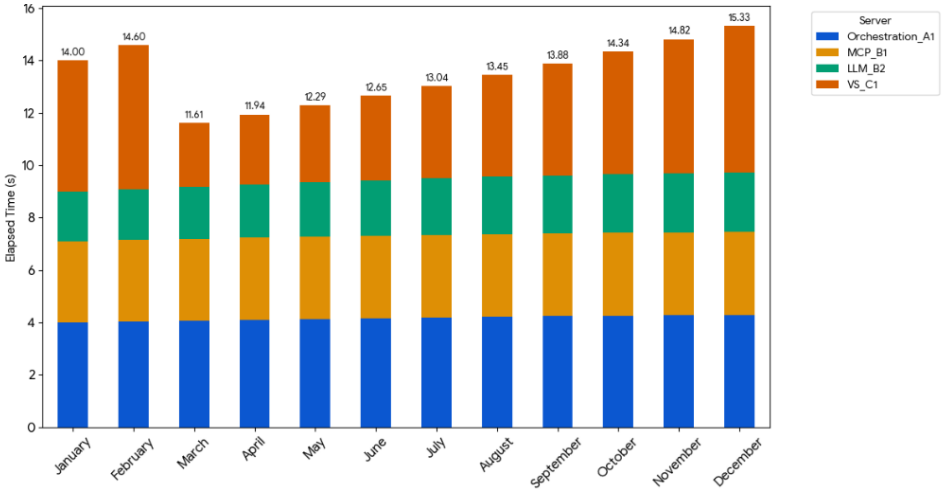


Figure 7.5 Proposed configuration change will allow meeting SLG for AgentsLong workload during the next 12 months.

Month	AgentsShort	AgentsMedium	AgentsLong
January	$1,500.00	$5,200.00	$21,500.00
February	$1,530.00	$5,356.00	$22,420.00
March	$1,560.00	$8,420.00	$34,500.00
April	$1,590.00	$8,588.00	$35,190.00
May	$1,620.00	$8,760.00	$35,894.00
June	$1,650.00	$8,935.00	$36,612.00
July	$1,680.00	$9,114.00	$37,344.00
August	$1,710.00	$9,296.00	$38,091.00

September	$1,740.00	$9,482.00	$38,853.00
October	$1,770.00	$9,672.00	$39,630.00
November	$1,800.00	$9,865.00	$40,423.00
December	$1,830.00	$10,062.00	$41,231.00
Total Year	$19,980.00	$103,550.00	$421,688.00

Table 7.11 Predicted cost, assuming using more powerful Snowflake Virtual Warehouses starting in March.

In this example, we predicted the minimum configuration and cost needed to meet business process SLG for three types of agents in a growing environment.

7.2 An example of comparing three platforms

If you are an executive interested in comparing the cost of several platforms needed to meet your agentic AI workloads, this example is for you [10].

Tables 7.12 and 7.13 present the predicted minimum configuration and associated cost required to meet SLGs across three platforms over three operational shifts over the next 12 months. In this example, workloads have different resource demands during the day, night, and evening. The number of users and the volume of data processed are expected to grow over the next 12 months.

Modeling determines the minimum configuration required for each platform, for each shift, and for each of the 12 months.

Platform	Instance Type	Shift	# Instances (Clusters) / Month											
			JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
Platform 1	Instance Type	1st	10	10	10	10	10	10	10	10	10	10	11	11
		2nd	32	34	34	34	34	34	36	36	36	36	38	38
		3rd	13	13	13	13	13	13	13	13	14	14	14	14
Platform 2	Instance Type	1st	52	52	52	54	54	54	56	56	58	58	58	60
		2nd	130	130	130	140	140	140	140	150	150	150	150	150
		3rd	72	74	74	76	76	78	78	80	80	82	82	82
Platform 3	Instance Type	1st	5	5	6	6	6	6	6	6	6	6	6	6
		2nd	3	3	3	3	3	3	3	3	3	3	3	3
		3rd	5	5	5	5	5	5	5	5	5	5	5	5

Table 7.12. Predicted minimum configuration of three platforms needed to meet SLGs during three shifts for the next 12 months

Publicly available pricing data is used to predict the cost of the configuration required for each platform to meet SLGs across shifts and the 12 months.

		Jan	Feb	. . .	Dec	Annual Cost	Relative Cost
Platform 1	Cost per month	$234,778	$241,453		$261,479	$2,964,189	1
	Cost per **query**	$0.0040	$0.0041		$0.0040	$0.0040	
Platform 2	Cost per month	$807,206	$813,466		$926,131	$10,468,877	3.53
	Cost per **query**	$0.0139	$0.0138		$0.0143	$0.0141	
Platform 3	Cost per month	$1,255,660	$1,255,660		$1,301,740	$15,528,720	5.24
	Cost per **query**	$0.0210	$0.0208		$0.0196	$0.0205	

Table 7.13. Predicted cost of the minimum configurations needed to meet SLG for each platform for one year

Each platform can meet performance service goals, but the cost to achieve that for the 1st platform is 5 times lower than for the 3rd one.

7.3 An example of sizing new application during DevOps

If you are a developer, executive, or are involved in operations, and would like to avoid the risk of performance and financial surprises after deploying new applications, this example is for you.

During the DevOps process, the measurement data is automatically collected during testing of the new application. This data is used as an input to the modeling and optimization process. This allows us to determine - before actual deployment - the minimum configuration (Figure 7.8) and budget (Figure 7.9) required for the application to meet its production SLG [13].

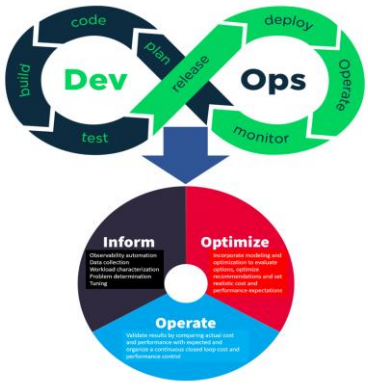


Figure 7.6 Integration of FinOps and DevOps to size new applications before deployment.

Modeling and optimization results determine the minimum configuration and budget needed to meet business SLG after deployment.

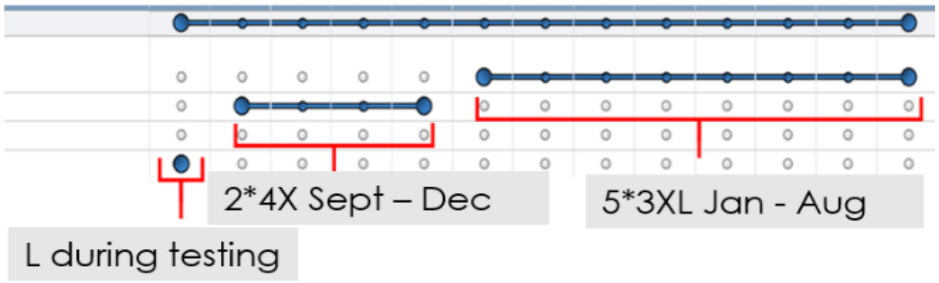


Figure 7.7 Predicted minimum Snowflake configuration needed to meet SLG after deployment of the new application.

Modeling and optimization rely on assumptions about expected workload growth and increased data volume to determine the minimum configuration needed at different hours of the day over the next 12 months following the deployment of the new application.

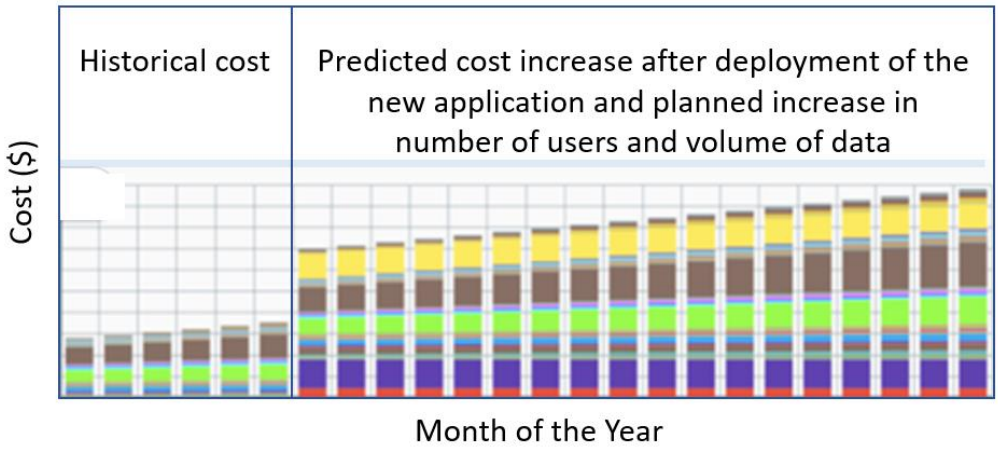


Figure 7.8 Predicted cost needed to meet SLGs will double after deployment of the new application. The cost will increase to support the expected growth.

Modeling and optimization results provide cost and performance expectations, enabling results verification and control.

7.4 An example of determining the configuration and cost needed to finish data load in time

If you are a DBA, IT manager, or responsible for operations, and are concerned with loading data on time, this example is for you.

The current elapsed time of loading data is longer than 7 days, and this should be reduced to 3 days. The load process comprises 4 phases with varying resource demands, as shown in Figure 7.10.

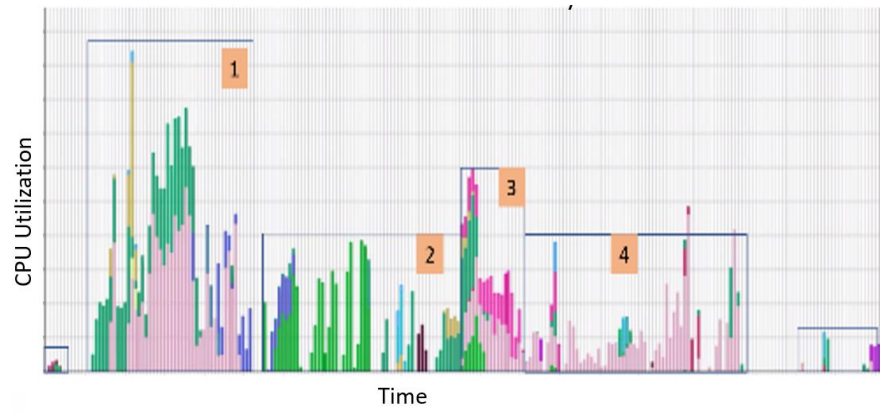


Figure 7.9 The current elapsed time of loading data is longer than 7 days

Predicted results (Table 7.14) determine the size and cost of the Snowflake configuration needed to load data in three days after migration. This is modeling expectations.

Interval On-Prem	On-prem Time (hours)	Snowflake Time (Hours)	Snowflake Clusters	Credits	Monthly Cost
4/02 8am - 4/04 4am	44	14.5	2*4XL	3,705	
4/04 8am - 4/06 5am	45	14.9	1*4XL	1,901	
4/06 8am - 4/07 8am	24	8.3	1*4XL	1,059	
4/07 8am - 4/09 9pm	61	15.8	1*4XL	2,028	
Total	174	53.5		8,693	$20,863

Table 7.12 Recommended configurations and predicted cost after migration.

Predicted cost and performance expectations are used to verify results and organize performance and cost control.

7.5 Organizing automatic dynamic capacity management

Automation observability, performance tuning, and proactively addressing workload management and resource allocation rules are critically important for a dynamic agentic AI world. If you are an IT executive, DBA, developer, or involved with operations, this example is for you.

Organizing a continuous dynamic capacity management process includes several steps. Some of them we discussed in previous chapters.

Dynamic capacity management relies on observability automation, workload characterization, and anomaly and root-cause determination to develop proactive performance management and cost-saving recommendations. What should be done to

fix current problems and avoid problems in the future? Each recommendation has performance and cost expectations. We discussed these topics in the previous chapters.

Dynamic capacity management relies on modeling and optimization to address strategic capacity management decisions by evaluating what-if scenarios.:It recommends the minimum configuration of orchestration platform, LLM, MCP, vector store, and DBMS components, workload management, and software parameter changes needed to continuously meet business SLGs at the lowest cost.

7.6 Modeling and optimization help to build a realistic budget

Traditional budget-planning methods rely primarily on historical cost data and projected resource needs for departments and lines of business. However, this approach does not account for the SLGs of business processes. As a result, organizations face a significant risk of over- or under-configuring, leading to performance and cost financial surprises. If you are an IT or finance executive, an IT manager and concerned with building a realistic budget for existing and new workloads, this example is for you.

Modeling and optimization technology helps to determine the **minimum configuration and corresponding budget** required to meet SLGs for existing Agent workloads, as well as for new Agent workloads and new applications planned for deployment. Figure 2.12 illustrates an example budget derived from identifying the optimal configuration and the necessary workload-management and tuning actions needed to meet SLGs for all existing and new Agents and applications.

This example shows how modeling and optimization can be applied to determine **next year's budget** based on SLG-driven performance and resource requirements.

The resulting performance and cost expectations enable **continuous closed-loop cost and performance control**, supporting proactive tuning of the agentic AI environment and ongoing refinement of workload-management and resource-allocation rules to meet SLGs at the **lowest possible cost**.

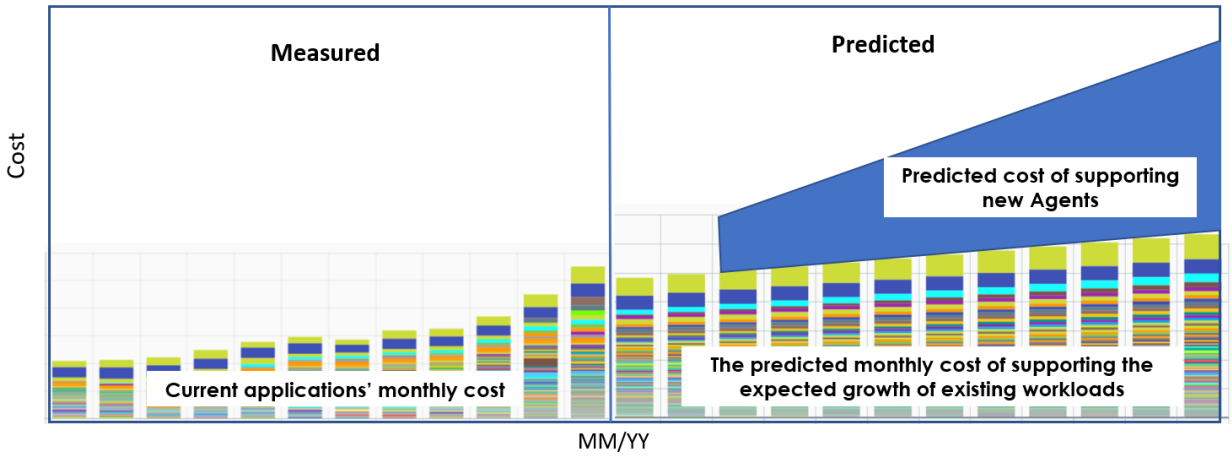


Figure 7.10 An example of applying modeling and optimization to determine next year's budget needed to meet SLGs

7.7 Modeling and optimization recommendation

Applying closed queueing network algorithms and gradient optimization enables informed, cost-effective decisions that improve performance management, strategic planning, and SLG compliance across hybrid multi-cloud architectures.

One challenge in using these techniques is the limited availability of detailed metrics for application- and query-level resource usage. We tackle this issue through automated model calibration.

We use this approach to evaluate alternative configurations and improve both strategic and tactical decisions, including sizing new Agents before deployment, managing performance, organizing dynamic capacity management, and producing financially sound budgets for agentic AI environments.

The case studies demonstrate the practical benefits of this methodology across multiple projects, including selecting agentic platforms, applying dynamic capacity management, and sizing new Agents before deployment.

7.8 Key Takeaways: applying modeling and optimization

- **Modeling and optimization transform planning and operations from guesswork to an engineering discipline**, enabling evidence-based decisions about platform choice, configuration, workload management, and investment.
- **Minimum-cost / SLG-compliant configurations can be determined in advance**, helping organizations avoid overprovisioning, excessive spending, and late-stage performance failures.
- **Interdependencies across Orchestration, LLM, MCP, Vector Store, and data platforms are explicitly captured**, ensuring platform and configuration decisions reflect real system behavior rather than isolated metrics.
- **New agents and applications can be sized before deployment**, preventing performance surprises, accelerating DevOps, and reducing financial and operational risk.
- **Dynamic capacity management becomes proactive rather than reactive**, supported by continuous observability, workload characterization, prediction, and automated optimization.
- **Strategic budgeting becomes accurate and defensible**, because it is based on workload behavior, performance requirements, and optimized configurations rather than historical spending trends.
- **Closed-loop performance and financial control is achievable**, enabling organizations to continuously meet SLGs at the lowest possible cost across hybrid multi-cloud environments.

References to Chapters 5, 6, and 7:

FinOps Optimize

[1] FinOps Foundation. What is FinOps? https://www.finops.org

[2] Slingshot: https://www.capitalone.com/tech/cloud/introducing-slingshot/

[3] "Cost Estimation of AI Workloads" by FinOps.org

[4] Top 20 FinOps tools to consider in 2024: https://www.finout.io/blog/finops-tools-guide

[5] “Cloud Cost Optimization: A Comprehensive Review of Strategies and Case Studies” (2023). https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4519171

[6] "FinOps for Cloud Cost Management": https://www.ibm.com/think/topics/finops-cloud-cost management

[7] J.R. Storment & Mike Fuller, Cloud FinOps, O’Reilly Media, 2020

Modeling and Optimization Technology

[8] L. Kleinrock. 1976. Computer Applications (1st ed.). Queueing Systems, Vol 2. Wiley-Interscience, New York.

[9] M. Reiser, S. Lavenberg. 1980. Mean value analysis of closed multichain queueing networks. J.ACM 27, 2 (April 1980), 313-322. https://doi.org/10.1145/322186.322195

[10] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[11] B.Zibitsker and A. Lupersolsky. A new approach to cloud cost optimization and performance control: https://bit.ly/4hwKU0b

[12] Zhang, L., Xu, M., & Buyya, R. (2021). “Evolutionary Multi-Objective Optimization for Multi-Cloud Resource Scheduling.” Journal of Parallel and Distributed Computing

Application of Modeling to Cost and Performance Optimization

[13] B. Zibitsker. "Which Platform is Best for your Cloud Data Warehouse? https://www.beznext.com/wp-content/uploads/2022/02/BEZNext-White-Paper-Which-Platform-is-Best-for-your-Cloud-Data-Warehouse-2-17-2021.pdf

[14] B.Zibitsker and A. Lupersolsky. 2022. The Journey to the Cloud-Hybrid multi-cloud. https://www.beznext.com/wp-content/uploads/2022/02/220225-BEZNext-White-Paper.pdf

[15] B. Zibitsker. 2022. Cloud Performance and Financial Governance Optimization (FinOps). Conference session. In CMG Impact 2022.

[16] B. Zibitsker and Alex Podelko. ICPE 2020. Performance Testing and Modeling for New Analytic Applications. Video. (21 May 2021). Retrieved November 16, 2023, from https://youtu.be/dnTrMiIYR98?si=pzugRScwJEeODMpZ

[17] B.Zibitsker “Cloud benchmarks aren’t enough. The use of modeling to address the limitations of benchmark tests” bit.ly/49cqq8I

[18] Herbst, N. R., Kounev, S., Reussner, R. (2013). “Elasticity in Cloud Computing: What It Is, and What It Is Not.”: 10th International Conference on Autonomic Computing (ICAC ’13), pp. 23-27, https://www.usenix.org/system/files/conference/icac13/icac13_herbst.pdf

[19] Wu, S., Li, J., & Kumar, S. (2021). “Machine Learning-based Multi-cloud Resource Allocation for Real-time Big Data Analytics.” IEEE Transactions on Parallel and Distributed Systems

[20] Singh, R. K., Gupta, P., & Alshamrani, S. (2023). “Workload-aware Hybrid Cloud Orchestration: Survey” Journal of Cloud Computing.

[21] Fu, A., Li, R., & Chen, H. (2021). “Adaptive DB Tuning with Deep Reinforcement Learning in Cloud-based Data Lakes” Proceedings of the VLDB Endowment

[22] Sun, X., Jermaine, C., & Mozafari, B. (2022). “Autonomous Query Optimization in the Cloud via Learned Cost Models.” ACM SIGMOD Conference

[23] Predicting Cloud Data Platforms Carbon Footprint for Large Data Warehouses: bit.ly/49cqq8I; bit.ly/3ScrAu4

8. Implementing Closed-Loop Performance and Cost Control for Agentic AI

The integration of observability, analytic modeling, and optimization provides the foundation for realistic cost and performance expectations. When combined with automation, these elements enable continuous closed-loop control, ensuring that agentic AI systems remain within their operational and financial guardrails.

In previous chapters, we examined the multi-layered factors—agent design, orchestration, infrastructure selection, and workload management—that drive cost and performance. We established how strategic modeling allows organizations to meet Service Level Goals (SLGs) at the lowest possible cost.

This chapter demonstrates how comparing actual performance and costs with model-driven expectations creates a closed-loop system. This feedback loop significantly reduces the risk of "bill shock" or operational failure. To organize this effectively, one must automate data collection and workload characterization to detect anomalies and inefficiencies early, narrowing the scope of tuning efforts as detailed in Chapters 3 and 4.

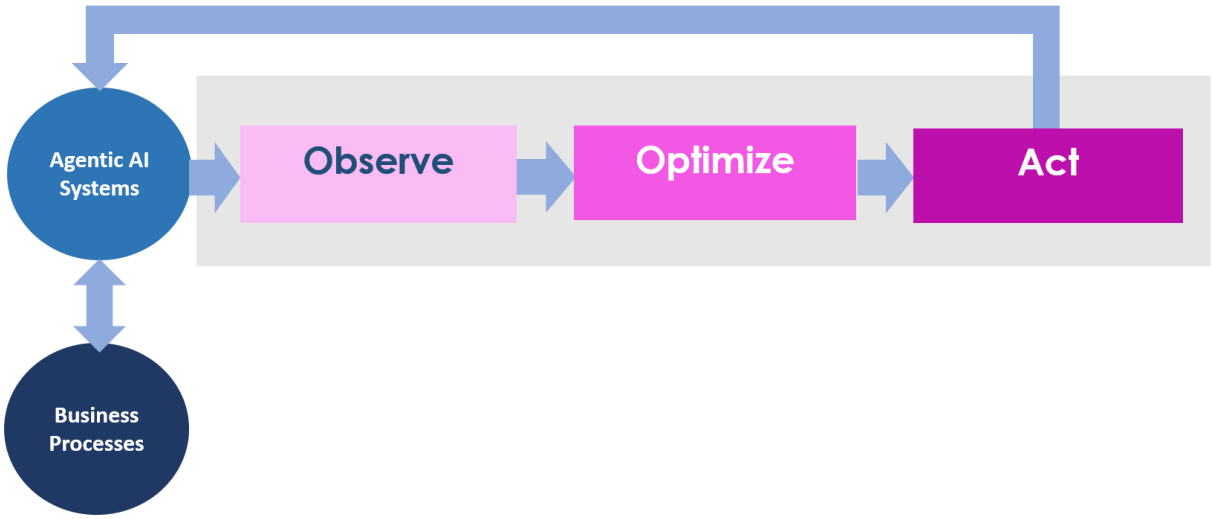


Figure 8.1: Integrating Observe, Optimize, and Act functions in organizing Closed-Loop Cost Optimization and Performance Control

The results from the observability tools serve as the primary input for the optimization phase. Optimization efforts target both cost reduction and performance stability through the following high-impact strategies:

7.8.1 High-impact strategies

- **LLM Selection & Complexity Reduction:** Deploying "small language models" (SLMs) for routine tasks and fine-tuning specific models to improve accuracy without the overhead of oversized general-purpose models.
- **Input/Output Optimization:** Prompt engineering to ensure conciseness, directly reducing token consumption and lowering latency.
- **Key-Value (KV) Caching:** Caching repetitive system instructions to avoid redundant processing, which can reduce latency by up to 80% and costs by up to 90%.
- **Efficient API Patterns:** Using batch APIs for non-time-sensitive tasks to secure significant discounts and implementing streaming to improve the user's perceived speed.

7.8.2 Architectural strategies

- **Quantization & Pruning:** Utilizing Post-Training Quantization (PTQ) to reduce model size for faster inference. Pruning removes unnecessary weights for permanent structural savings.
- **Intelligent Orchestration:** Dynamically matching tasks to the appropriate infrastructure (e.g., high-tier GPUs for reasoning, lower-tier instances for simple routing).
- **AI Gateways:** Implementing an abstraction layer to manage LLM calls centrally, enforce global policies, and track usage without altering the core application code.

7.8.3 Specialized performance techniques

- **Speculative Decoding:** Utilizing a small "draft" model to predict tokens that a larger model verifies in parallel, accelerating generation without retraining.
- **Hardware & Network Optimization:** Selecting specific hardware (GPUs/TPUs) for parallel workloads and co-locating servers near AI providers to minimize network hop latency.

7.9 Cost Verification

Verification is the "closing" of the loop. Table 7.6 (from the previous chapter) provided the optimization results, including the predicted minimum configuration and cost required to meet SLGs over an eight-month period.

Figure 8.2 compares the actual cost (credits used) against these expected results. In this case study, the variance remained within 10% of the model's predictions. The sole exception occurred in Month 3, where a redundant load execution caused a temporary spike.

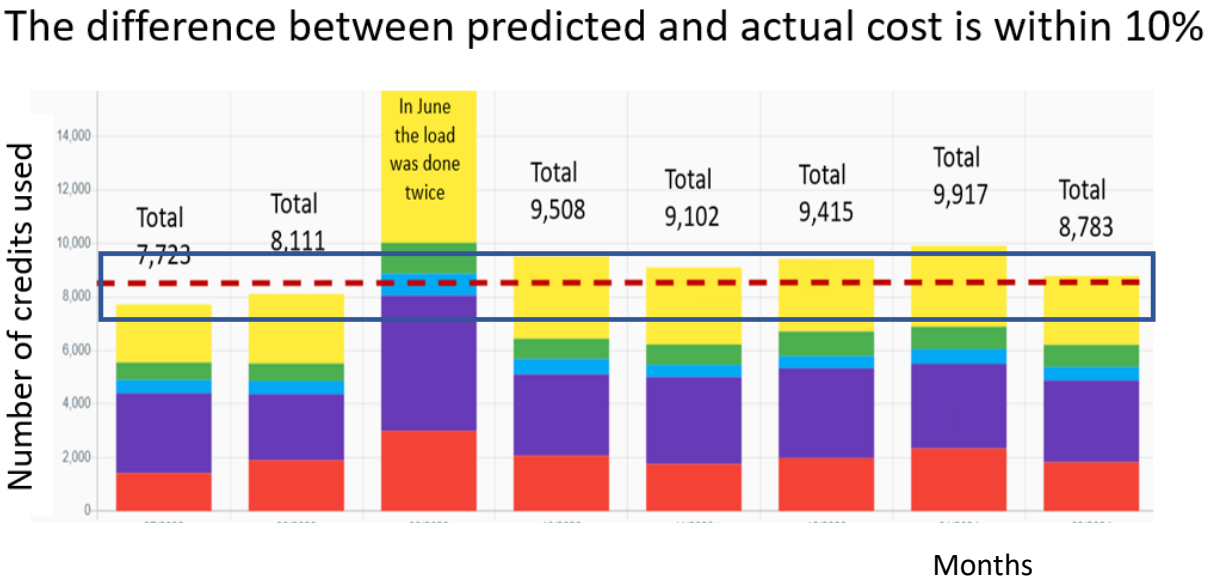


Figure 8.2: Actual measured costs remain within the 10% tolerance threshold of predicted values.

7.10 Performance verification

In this project, the defined SLG required data loading to be completed within three days. However, as shown in Figure 8.3, the actual duration frequently reached four days.

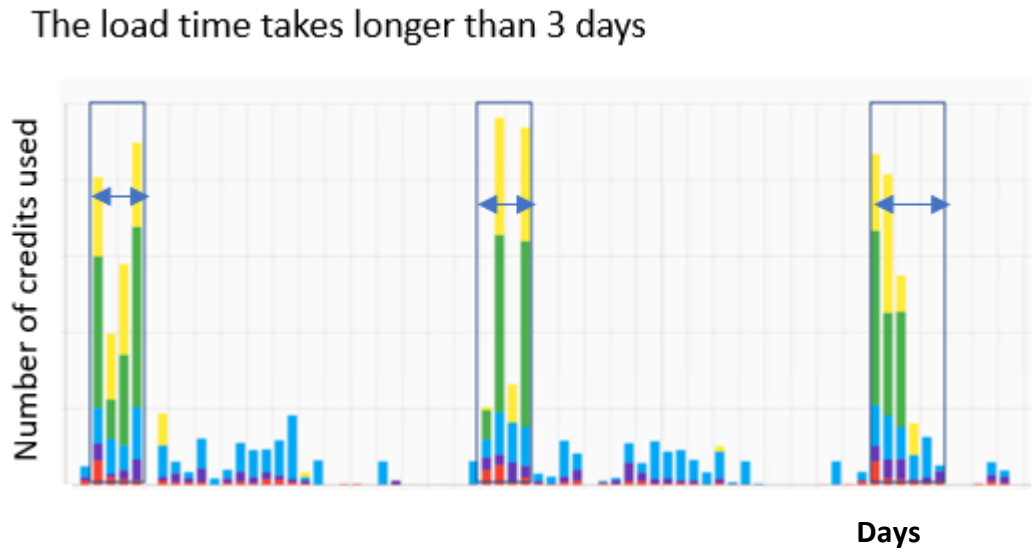


Figure 8.3: Throughput SLG (3-day load window) is not met; tuning is required.

To close the loop and return the system to SLG compliance, we generated specific tuning recommendations:

1. **High-Credit Query Optimization:** Identifying and refactoring the specific queries consuming the most credits.
2. **Anomaly Mitigation:** Addressing queries with the highest frequency and severity of performance failures.
3. **Technology Migration:** Transitioning virtual warehouses to more modern, efficient hardware instances.

7.10.1 Key Takeaways: Closed-loop control

- **Trust through Verification:** Verification is essential for maintaining confidence in autonomous AI agents.
- **Continuous Correction:** Comparing measured results with predicted outcomes allows the system to learn and self-correct over time.
- **Risk Mitigation:** Closed-loop control prevents cost drift, SLG violations, and "runaway" agent behavior.
- **Scalability:** Automation is mandatory; manual oversight cannot keep pace with growing agent populations.
- **Sustainability:** Long-term Agentic AI operations require the integration of observability, modeling, and autonomous enforcement

7.10.2 References to Chapters 7 and 8

Foundations: closed-loop control & autonomic systems

[1] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, Feedback Control of Computing Systems. Wiley, 2004.

[2] IBM, "An Architectural Blueprint for Autonomic Computing," 4th ed., 2006.

[3] ETSI, "GR ENI 017 v2.2.1: The IBM MAPE-K control loop (ENI reference model)," 2024.

[4] Google SRE, Implementing SLOs (workbook chapter), 2023–2025.

Cloud-native autoscaling: balancing SLOs, performance, and cost

[5] Y. Garí et al., "Reinforcement Learning-based Application Autoscaling in the Cloud," Engineering Applications of AI, 2021.

[6] S. Alharthi et al., "Auto-Scaling Techniques in Cloud Computing: Issues and Challenges," Applied Sciences, 2024 (survey).

[7] A. Marchese et al., "SLO- and Cost-Driven Container Autoscaling on Kubernetes," Proc. ICSOFT, 2025.

[8] A. Ullah et al., "A Control-Theoretical View of Cloud Elasticity: Taxonomy & Survey," Cluster Computing, 2018.
[9] C. Wu, V. Sreekanti, and J. M. Hellerstein, "Autoscaling Tiered Cloud Storage in Anna," PVLDB, 2019.

Hybrid / multi-cloud scheduling & placement (cost-aware)

[10] M. Luo et al., "Starburst: A Cost-aware Scheduler for Hybrid Cloud," USENIX ATC, 2024.
[11] Z. Qiu et al., "Optimizing Placement of Data and Compute in Hybrid Clouds," SOSP, 2025 (Moirai). [12] P. Nawrocki et al., "A Survey of Cloud Resource Consumption Optimization," J. Grid Computing, 2025.

Autonomic data/AI systems (closed-loop tuning for performance/cost)

[13] A. Pavlo et al., "Self-Driving Database Management Systems," CIDR, 2017 (Peloton/NoisePage vision).

[14] CMU-DB, "NoisePage: Self-Driving DBMS (project overview)," 2020–2025.

Agentic/GenAI observability & evals to feed the loop

[15] OpenTelemetry, "Semantic Conventions for Generative AI," 2024–2025.
[16] LangChain, "LangSmith: Online Evaluations & Production Tracing for LLM/Agent Apps," 2024–2025
[17] A. Afzal et al., "Towards Optimizing a Retrieval-Augmented Generation Pipeline," ACM AIES Wkshp, 2024.

[18] E. Martell, "Closed-Loop Retrieval-Augmented Generation for Recommendations," MSc Thesis, 2025.

Cloud budget guardrails (automated actions = cost closed loop)

[19] AWS, “Configuring Budget Actions (automated stop/scale/notify),” Docs, 2025. [20] Google Cloud, “Programmatic Budget/Anomaly Notifications via Pub/Sub (automate responses),” Docs, 2025.
[21] Microsoft Azure, “Budget Scenarios with Automation (Action Groups + Runbooks),” Docs, 2025. [22] FinOps Foundation, “Workload Management & Automation (match supply to demand),” Framework, 2024–2025

[22] B. Zibitsker and A. Lupersolsky, “Cost Optimization and Performance Control in the Hybrid Multi-Cloud Environment,” in Proc. ICPE 2025, 2025.

8 Summary and Key Takeaways:

8.1 Agentic ai cost and performance optimization

Agentic AI introduces a fundamental shift from deterministic software to **autonomous, goal-driven systems** that continuously reason, act, and adapt. While this enables unprecedented business automation, it also introduces **non-linear performance behavior, volatile costs, and elevated operational risk**. Traditional FinOps, monitoring, and capacity-planning practices are insufficient for managing these systems at scale.

This book presents a **closed-loop framework** for delivering **trustworthy, cost-efficient, and performance-predictable agentic AI systems**.

8.2 Agentic AI must be operated, not just deployed

- Agentic AI systems are **living systems**, not one-time implementations.
- Success depends on lifecycle discipline: planning, deployment, optimization, verification, and evolution.
- Without continuous control, autonomous agents become **unpredictable, expensive, and risky**.

8.3 Cost and performance are system properties

- Agentic AI performance and cost emerge from **interactions across multiple layers**: orchestration, LLMs/SLMs, RAG, vector stores, data platforms, and infrastructure.
- A single user request can trigger **dozens of backend interactions**, amplifying latency and cost.
- Bottlenecks and inefficiencies cascade across layers, making siloed optimization ineffective.

8.4 Observability is the foundation of trust

- Effective governance requires **agent-level, task-level, and business-process-level observability**.
- Granular telemetry (latency, tokens, retries, resource usage, and cost) exposes hidden cost drivers.
- Automated observability narrows optimization efforts to the **small subset of agents and workflows that matter most**.

8.5 Modeling replaces guesswork with predictability

- Queueing Network Models and optimization transform raw telemetry into **predictive insight**.
- Modeling enables organizations to:
    - Size new agentic AI applications **before deployment**

    - Select platforms and configurations with **quantified risk**
    - Forecast realistic budgets tied to **Service Level Goals (SLGs)**
- Without modeling, platform selection and capacity planning remain speculative.

8.6 Optimization enables minimum-cost SLG compliance

- The objective is not maximum performance, but **meeting SLGs at the lowest possible cost**.
- Optimization identifies:
    - Minimal viable configurations
    - Effective workload-management rules
    - Reserved vs. on-demand capacity tradeoffs
- This approach delivers **measurable ROI** while preserving performance guarantees.

8.7 Verification Closes the Loop

- Trustworthy agentic AI requires continuous **verification of actual vs. expected results**.
- Closed-loop control detects drift, validates assumptions, and corrects decisions automatically.
- This prevents runaway costs, performance regressions, and governance failures.

8.8 Integrated Ops is a strategic imperative

- Sustainable agentic AI operations require tight integration of:
- **FinOps** (cost governance)
- **DevOps** (deployment and testing)
- **DataOps** (data reliability and efficiency)
- **PlatformOps** (infrastructure and workload control)
- Automation across these disciplines is essential to scale safely.

8.9 Bottom Line

Agentic AI delivers transformative business value **only when operated with discipline**. Organizations that combine **automated observability, rigorous modeling, optimization, and closed-loop control** can scale agentic AI with confidence achieving predictable performance, controlled costs, and durable trust.

This book provides the framework, methodology, and real-world examples to make that possible.

8.10 Related work

FinOps methodology and technology are widely used to optimize cloud costs through observability, modeling, and control [1]. Storment and Fuller provide a comprehensive overview of FinOps principles, emphasizing the importance of continuous observability and feedback loops for effective cost and performance management [7].

However, applying FinOps in **agentic AI hybrid multi-cloud environments** presents significant challenges. These include the complexity of deploying and managing FinOps at scale, the difficulty of achieving unified monitoring across disparate cloud platforms, and inconsistent metrics from cloud providers that hinder true end-to-end observability.

Current FinOps observability tools lack a **systemic, automated approach**. Their recommendations typically do not include cost–performance expectations and do not allow customers to validate whether implemented actions actually delivered the intended results [2].

Articles [3, 4, 5, 6] outline the Inform–Optimize–Operate framework for cloud cost management and highlight the need for accountability and real-time insights. Yet these works do not specify the minimum resource requirements, workload-management rules, or budgets needed to meet SLGs for diverse applications in hybrid multi-cloud environments. Although these tools identify anomalies and their root causes, they often fail to pinpoint the requests and queries that are most expensive, resource-intensive, failure-prone, and likely to produce significant data spills to storage. Some tools offer tuning suggestions but generally do not estimate the expected cost and performance impact of those changes.

Modeling approaches in [18] define metrics for elasticity and propose conceptual models for evaluating elastic resource provisioning. Work in [19] explores predictive modeling that combines time-series forecasting and reinforcement learning to optimize resource allocation across multiple clouds, demonstrating how machine learning can improve the latency and cost efficiency of real-time analytics pipelines. Survey [22] reviews workload orchestration techniques in hybrid clouds, focusing on trade-offs among real-time and batch processing, cost and performance objectives, and vendor lock-in.

Although these models help evaluate individual decisions, they have **limited ability to assess the combined effects** of configuration and workload-management changes on cost and performance in environments hosting mixed workloads with diverse profiles and SLGs.

Other research, such as [21], applies performance modeling and reinforcement learning to query and database tuning, adjusting caching, partitioning, and indexing parameters in distributed SQL engines. Reference [22] introduces a learned cost model integrated

into a cloud query optimizer, demonstrating how data-driven models can enhance query planning. However, these techniques focus on DBMS-internal optimization rather than workload and application tuning across customer-specific cloud environments.

A growing body of work highlights the role of machine learning, particularly reinforcement learning and multi-objective optimization, in cloud database tuning, resource allocation, and workload management. As hybrid and multi-cloud adoption accelerates, new models are emerging to address complex resource distributions, diverse pricing schemes, and dynamic workloads. Multi-objective optimization explicitly recognizes trade-offs among cost, response time, energy consumption, carbon footprint, and fault tolerance [12], often using evolutionary or Pareto-based methods.

Although this paper focuses primarily on cost optimization and performance control, our framework also supports estimating **power usage and carbon footprint** for individual applications across cloud data platforms [23].

Numerous research and industry publications further acknowledge the limitations of existing observability automation and modeling approaches in addressing cost-optimization and performance-control challenges in complex, agentic AI–driven hybrid multi-cloud environments.

8.11 Glossary of Terms

"Agentic AI Cost Optimization and Performance Control"

A

Agent – An autonomous software component capable of reasoning, planning, taking actions, and interacting with other agents or external systems to achieve defined goals.

Agentic AI – A system of intelligent agents that autonomously plan, reason, retrieve knowledge, call tools, and make decisions to optimize business processes.

Agent Telemetry – Logs, traces, and metrics that describe an agent's actions, reasoning steps, tool calls, latency, token usage, and costs.

AISQL (Snowflake Cortex) – SQL-based functions enabling LLM/RAG operations directly within Snowflake's governed environment.

Anomaly Detection – Identification of events or behaviors that deviate from expected performance or cost baselines.

Auto-Scaling – Automatic adjustment of resources (CPU, GPU, memory, nodes) in response to real-time workload demand.

B

Batch Workload – Non-interactive workloads with relaxed SLGs, often executed periodically (e.g., nightly loads or analytics).

BEZNext – Observability automation, Modeling, optimization, and workload management platform used for cost optimization and performance control in the hybrid multi-cloud and agentic AI environments.

BigQuery Slots – Units of query processing capacity used for serverless SQL and AI workloads in Google BigQuery.

C

Capacity Planning – Determining the minimum compute, memory, GPU, and storage capacity required to meet SLGs.

Closed-Loop Control – A control system that continuously compares measured results with expectations and adjusts configuration or workload management.

Closed Queueing Network Model – A performance model where throughput depends on system response time and "think time" between requests.

Cloud FinOps – A framework for financial management of cloud resources, focused on visibility (Inform), cost optimization (Optimize), and operational control (Operate).

Cloud-Native Observability – Monitoring tools provided by cloud platforms (CloudWatch, Azure Monitor, GCP Monitoring).

Cortex – Snowflake's integrated AI engine for LLMs, RAG, vector search, and evaluation.

Cost Attribution – Assigning cloud costs to agents, workloads, teams, or business processes.

Credits (Snowflake) – Snowflake's unit of compute consumption for Virtual Warehouses.

D

Databricks Unit (DBU) – Normalized measure of compute consumption in Databricks environments.

Data Spill – The process where queries exceed local memory and spill data to remote storage, increasing latency and cost.

Dynamic Capacity Management – Adjusting resources, workload rules, or configurations in real time to maintain SLGs at minimal cost.

E

Embedding – A numeric vector representation of text or objects used for similarity search in RAG and vector stores.

Elasticity – The ability of a system to scale resources up or down in response to workload changes.

Evaluation Metrics (LLM/RAG) – Measures such as relevance, groundedness, hallucination risk, and correctness used to assess agent and model quality.

F

FinOps (Inform / Optimize / Operate) – Cloud financial operations lifecycle:

Inform: observability, allocation, cost visibility

Optimize: cost-performance optimization, modeling

Operate: continuous control against SLGs

Funnel Analysis (LLM/RAG) – Breaking down LLM or RAG workflow steps to identify where latency or cost accumulates.

G

Gradient Optimization – Iterative optimization method used to determine minimal configurations to meet SLGs while minimizing cost.

GPU Utilization – Percentage of GPU processing used by LLM inference, embedding generation, or vector search.

H

HeatWave (Oracle) – In-memory, scale-out vector and analytical engine integrated with MySQL for LLM/RAG workloads.

Hybrid Multi-Cloud – An architecture spanning multiple cloud providers and on-premise environments.

I

Inference Cost – Cost associated with LLM tokens (input + output), API calls, or model-serving compute time.

Isolation (Workload Isolation) – Separation of workloads or agents to prevent interference and noisy-neighbor effects.

K

Key Performance Indicator (KPI) – Metrics such as latency, throughput, cost/session, or agent success rate that define SLG adherence.

Knowledge Base (KB) – Structured and unstructured enterprise information used for RAG retrieval.

L

Lakehouse – Databricks architecture combining data lake and data warehouse functionality.

Latency (p50, p90, p95) – Percentile-based response time distribution metrics.

LLM (Large Language Model) – AI model used for natural language reasoning, planning, and generation.

Local LLM – A model deployed per agent or service for deterministic performance or data isolation.

M

MCP (Model Context Protocol) – Open standard enabling agents to securely interact with databases, tools, and external services via structured requests.

Mean Value Analysis (MVA) – Queueing model algorithm used to analyze multi-chain closed queueing networks.

Memory Store (Agent Memory) – Long-term storage for embeddings and agent knowledge, typically implemented via a vector database.

Metrics Pipeline – System that collects and aggregates logs, traces, and performance indicators.

N

Noisy Neighbor – Another workload or agent consuming shared resources, causing performance degradation.

O

Observability – The ability to collect, aggregate, analyze, and interpret system telemetry (logs, metrics, traces).

On-Demand Compute (BigQuery) – Pricing model where customers pay per TB scanned rather than for reserved capacity.

Optimization Scenario – A modeling configuration used to compare different platform or resource decisions.

P

Partitioning (Data) – Organizing data into micro-partitions or segments to improve pruning and reduce scan volume.

Performance Verification – Comparing measured performance to predicted values after tuning or scaling.

Prompt Bloat – Excessive or unnecessary prompt content causing increased cost and latency.

Pub/Sub – Google Cloud's real-time messaging system for streaming telemetry to BigQuery.

Q

Queueing Network Model (QNM) – Analytical technique used to model response times, throughput, concurrency, bottlenecks, and resource saturation.

Query Pruning – Skipping irrelevant micro-partitions to reduce scan time and cost.

R

RAG (Retrieval-Augmented Generation) – Technique in which LLMs retrieve relevant context (documents, embeddings) to improve accuracy and reduce hallucination.

Resource Monitor (Snowflake) – Tool that caps credit usage to prevent runaway cost.

Root-Cause Analysis – Identifying underlying reasons for performance or cost anomalies.

S

Scalability – The ability of a system to handle increased load without violating SLGs.

Service Level Goal (SLG) – Business-driven performance targets (latency, throughput, cost per workflow, availability).

Serverless Compute – Execution model with automatic provisioning and scaling of compute resources (e.g., BigQuery, Databricks Serverless).

Slots (BigQuery) – Compute units for BigQuery edition-based workloads.

Spill (Local/Remote) – Overflow of processing to disk/storage when memory is insufficient.

Synapse DWU – Azure Synapse unit for measuring compute power in dedicated SQL pools.

T

Telemetry – Data collected from systems describing behavior, usage, and performance; includes logs, metrics, and traces.

Teradata TASM – Teradata Active System Management: workload throttles, filters, and concurrency controls.

Think Time – Interval in a closed queueing network when a client is preparing a new request.

U

Utilization (CPU, GPU, I/O) – Degree to which system resources are consumed.

V

Vector Database – Specialized storage system for high-dimensional embeddings (FAISS, Pinecone, Weaviate, Qdrant, Databricks Vector Search).

Vector Search – Similarity-based retrieval mechanism used in RAG workflows.

Virtual Warehouse (Snowflake) – Compute cluster dedicated to a workload with independent scaling and isolation.

W

Workload Characterization – Aggregating measurement data into hourly performance, resource, and cost profiles.

Workload Management – Rules for concurrency, prioritization, workload isolation, and scheduling.

Z

Z-Ordering (Databricks) – Data clustering technique to improve retrieval and reduce scan volumes in Delta Lake.

8.12 Version History

Version	Date	Description
0.9 (Draft)	September 2025	Initial draft completed.
1.0 (First Edition)	January 2026	Full editing pass,
1.1		
1.2		

Table Version information

8.13 About the Author

Dr. Boris Zibitsker is an expert in applying modeling and optimization to performance engineering, capacity management, and cost control within Agentic AI and cloud ecosystems. With more than four decades of experience optimizing the management of complex computing environments, he has enabled global enterprises to achieve transformative gains in performance, stability, cost, and efficiency.

As the Founder and CEO of BEZNext, Dr. Zibitsker leads the development of automated observability, performance modeling, and continuous cost optimization for agentic AI and hybrid multi-cloud environments. His work integrates advanced queuing network models, gradient optimization, and FinOps best practices to manage the high-variability workloads common in modern AI and data-intensive systems.

A respected voice in the field, Dr. Zibitsker has co-authored numerous academic and industry papers and is a frequent contributor to ICPE, the SPEC Research Group, and CMG. His innovations have supported Fortune 2000 leaders across the financial, telecommunications, and healthcare sectors, helping them manage mission-critical workloads on platforms including Snowflake, Teradata Vantage, Databricks, Google BigQuery, and Azure Synapse.

Dr. Zibitsker holds a Ph.D. in Computer Science and has dedicated his career to applying modeling and optimization to achieve closed-loop performance and cost control across enterprise systems. His current research focuses on a unified, continuous-optimization framework that integrates autonomous control and modeling to balance the performance and cost of agentic AI.

BEZNext provides software and strategic services, specializing in selecting optimal platform configurations, sizing new AI agents before deployment, and implementing dynamic capacity management.

For organizations seeking a deeper understanding of cost optimization and performance control in agentic AI cloud environments, BEZNext offers hands-on workshops using customer data. Learn more at: **www.beznextworkshop.com**.