Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

Hybrid Cloud-Native AI Architecture for Scalable Deployment of Agentic and Generative AI Applications

Sudhakar Murthy Molli

B.Tech, MS, Independent Researcher, USA

Mollisudhakar@gmail.com

Published

2024-05-16

Issue

Vol. 6 No.6 (2024): AJCDI

Abstract

Agentic and generative AI applications introduce workload characteristics -- long-lived multi-step sessions, heterogeneous tool invocation, bursty and unpredictable demand, and mixed regulatory requirements over data and models -- that strain cloud-native infrastructure patterns originally designed for stateless microservices. This paper presents a hybrid cloud-native architecture that spans public cloud and on-premises infrastructure to meet the latency, cost, reliability, and data-governance requirements of production agentic AI systems. We describe a layered reference architecture comprising an agent orchestration mesh, a cascade-routed model serving layer, a federated data and retrieval layer, and a Kubernetes-based multi-cluster control plane, and we evaluate it empirically against single-environment deployment baselines. Across a range of experiments spanning latency, cost, GPU utilization, autoscaling responsiveness, and fault injection, the hybrid architecture with policy-based routing reduces median request latency by 34%, lowers cost per 1,000 agent tasks by 57% relative to a cloud-only baseline, and sustains 93% task success under simulated cross-region network partition versus 56% for a single-region deployment. We conclude with a discussion of open challenges in multi-agent state management, cross-environment observability, and the governance of autonomous tool-calling agents operating across trust boundaries.

Keywords: cloud-native architecture, hybrid cloud, agentic AI, generative AI, Kubernetes, multi-agent orchestration, GPU scheduling, LLM serving, edge AI

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

1. Introduction

Generative AI has moved rapidly from single-turn text completion to agentic systems that plan multi-step tasks, invoke external tools, retrieve and reason over enterprise data, and operate with meaningful autonomy over extended sessions. This shift has outpaced the infrastructure patterns that cloud-native computing established for an earlier generation of stateless, request-response web services. Agentic AI workloads are long-lived, stateful across a session, bursty at the level of individual reasoning steps, and dependent on a mix of proprietary frontier models, self-hosted fine-tuned models, and enterprise data that may be subject to residency, privacy, or export-control constraints that no single public cloud region or vendor can satisfy alone.

At the same time, organizations adopting agentic AI rarely have the option of deploying entirely within a single environment. Data gravity, existing GPU capital investment, regulatory requirements, and vendor risk management all push toward architectures that span public cloud and private or on-premises infrastructure. This paper examines how cloud-native design patterns -- containerization, Kubernetes orchestration, service meshes, and declarative infrastructure -- can be extended into a coherent hybrid architecture purpose-built for agentic and generative AI, rather than treating hybrid deployment as an afterthought bolted onto a cloud-native design built for conventional microservices.

1.1 Motivation

Three factors motivate this study. First, agentic workloads exhibit session-level statefulness -- conversation history, task graphs, tool-call results, and intermediate reasoning artifacts -- that must persist across many requests, unlike the largely stateless request patterns that cloud-native infrastructure was optimized to serve. Second, the cost and latency profile of agentic applications is dominated by model inference calls that vary enormously in size and cost depending on task complexity, creating an economic incentive for cascade routing across model tiers and deployment environments rather than a uniform one-size-fits-all serving strategy. Third, enterprise adoption of agentic AI is frequently gated by data governance requirements -- certain data or tool actions must remain within a private environment -- that necessitate an architecture capable of enforcing placement policy at the level of individual agent steps, not just at the level of an entire application.

1.2 Contributions

This paper makes the following contributions:

- A workload characterization of agentic AI applications, identifying the statefulness, burstiness, and heterogeneity properties that distinguish them from conventional cloud-native workloads.
- A reference architecture that integrates an agent orchestration mesh, cascade-routed hybrid model serving, a federated retrieval and data governance layer, and a multi-cluster Kubernetes control plane into a single coherent hybrid cloud-native platform.
- An empirical evaluation of routing policies, autoscaling strategies, and GPU sharing techniques across latency, cost, and utilization metrics.

Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

- A fault-injection study quantifying the reliability benefits of multi-region hybrid deployment relative to single-region deployment under realistic failure modes including zonal outages and cross-environment network partition.
- A discussion of open challenges in multi-agent state management, cross-environment observability, and governance of autonomous tool-calling agents.

1.3 Paper Organization

Section 2 surveys related work in cloud-native infrastructure, LLM serving systems, and agentic AI frameworks. Section 3 characterizes agentic AI workloads. Section 4 presents the reference architecture, with Sections 5 through 8 detailing its orchestration, model serving, data/governance, and control-plane layers respectively. Section 9 describes our experimental methodology, Section 10 presents results, and Section 11 analyzes reliability and cost trade-offs. Section 12 discusses open challenges, and Section 13 concludes.

2. Related Work

2.1 Cloud-Native Infrastructure and Kubernetes

Kubernetes has become the de facto standard for container orchestration, providing declarative deployment, self-healing, and horizontal autoscaling primitives that underpin most modern cloud-native systems. Multi-cluster federation patterns extend these primitives across regions and environments, enabling workload placement policy, cross-cluster service discovery, and coordinated rollout, and form the control-plane foundation this paper builds upon. Service mesh technologies layer traffic management, mutual TLS, and observability atop the container network, providing the policy enforcement points that a hybrid agentic platform relies on to route individual agent steps according to data-governance and cost policy.

2.2 LLM Serving Systems

Efficient serving of large language models has been the subject of substantial systems research, including continuous batching, paged KV-cache management, and disaggregation of compute-bound prefill from memory-bandwidth-bound decode phases. Model cascade and routing techniques direct a given request to the smallest or cheapest model capable of producing an acceptable response, escalating to larger models only when needed; this paper extends cascade routing to a hybrid environment in which model tiers are additionally distinguished by deployment location, not only by size.

2.3 Agentic AI Frameworks

A growing body of frameworks provide abstractions for tool-calling, multi-step planning, and multi-agent collaboration atop foundation models, typically implementing a planner-worker or supervisor-subagent pattern in which a top-level agent decomposes a task and delegates subtasks to specialized agents or tools. These frameworks are predominantly designed assuming a single, uniform execution environment and do not natively address placement policy across trust boundaries or heterogeneous infrastructure, a gap this paper's orchestration layer is designed to close.

Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

2.4 Hybrid and Multi-Cloud Architecture

Prior work on hybrid and multi-cloud architecture has primarily addressed conventional application workloads, focusing on workload portability, data replication, and disaster recovery across environments. GPU-specific hybrid scheduling, addressing the cost and availability characteristics of accelerator capacity across public cloud and on-premises clusters, is comparatively less studied, and existing approaches rarely account for the session-level statefulness and step-level heterogeneity that distinguish agentic AI workloads from the batch or stateless workloads that motivated earlier hybrid-cloud scheduling research.

2.5 Positioning of This Work

This paper's contribution is to treat agent orchestration, hybrid model serving, data governance, and multi-cluster infrastructure as a single co-designed system specifically for agentic and generative AI workloads, rather than applying general-purpose hybrid-cloud or LLM-serving techniques independently. We ground our architectural recommendations in an explicit workload characterization (Section 3) and validate them with end-to-end experiments spanning latency, cost, utilization, and fault tolerance (Sections 9-11).

3. Workload Characterization of Agentic AI Applications

Designing infrastructure for agentic AI requires first understanding how its workload properties diverge from the stateless, short-lived request patterns that cloud-native infrastructure was originally built to serve.

3.1 Session Statefulness

An agentic session typically spans many sequential steps -- planning, tool invocation, retrieval, and response generation -- executed over seconds to minutes, with intermediate state (conversation history, task graph, partial results) that must remain available across steps. Unlike a stateless web request, losing this state mid-session degrades or invalidates the entire task, requiring infrastructure that treats session state as a durable, addressable resource rather than transient request context.

3.2 Step-Level Heterogeneity

Individual steps within a single agent session vary enormously in resource profile: a planning step may require a single frontier-model call, a retrieval step may query a vector store with sub-second latency requirements, and a tool-execution step may run arbitrary sandboxed code with unpredictable duration. This heterogeneity means that a single infrastructure policy applied uniformly across an entire session -- for example, always routing to the same model tier or the same environment -- is systematically suboptimal compared to policy applied at the granularity of individual steps.

3.3 Bursty, Unpredictable Demand

Agentic workloads are driven by human-initiated or event-triggered sessions whose arrival is far less predictable than conventional web traffic, and whose resource consumption per session varies by an

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%
order of magnitude depending on task complexity and the number of reasoning or tool-calling iterations required. Figure 3 in Section 6 illustrates this burstiness and its implications for autoscaling.

3.4 Trust and Data-Governance Boundaries

Because agents can invoke tools and retrieve data autonomously, individual steps within a session may need to execute within different trust boundaries: a step touching regulated customer data may be required to execute entirely within a private environment, while a step requiring frontier-model reasoning capability may need to reach a public cloud API. This requirement for step-level, policy-driven placement -- rather than whole-application placement -- is a defining characteristic of agentic AI infrastructure and directly motivates the orchestration-layer design in Section 5.

3.5 Implications for Infrastructure Design

Four implications follow from this characterization. First, session state must be externalized to a durable, low-latency store accessible from any execution environment, rather than held in process memory. Second, routing and placement decisions should be made per step, informed by cost, latency, and governance policy, rather than fixed for an entire session. Third, autoscaling must anticipate burstiness at the granularity of reasoning steps, not just aggregate request volume. Fourth, observability and governance tooling must trace execution across environment boundaries to provide auditability for autonomous, multi-step, multi-environment agent behavior.

4. Reference Architecture

This section presents a reference architecture for hybrid cloud-native agentic AI infrastructure, synthesizing the workload properties identified in Section 3 into a layered system design. Figure 1

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%
depicts the end-to-end architecture, spanning public cloud and private/on-premises zones.

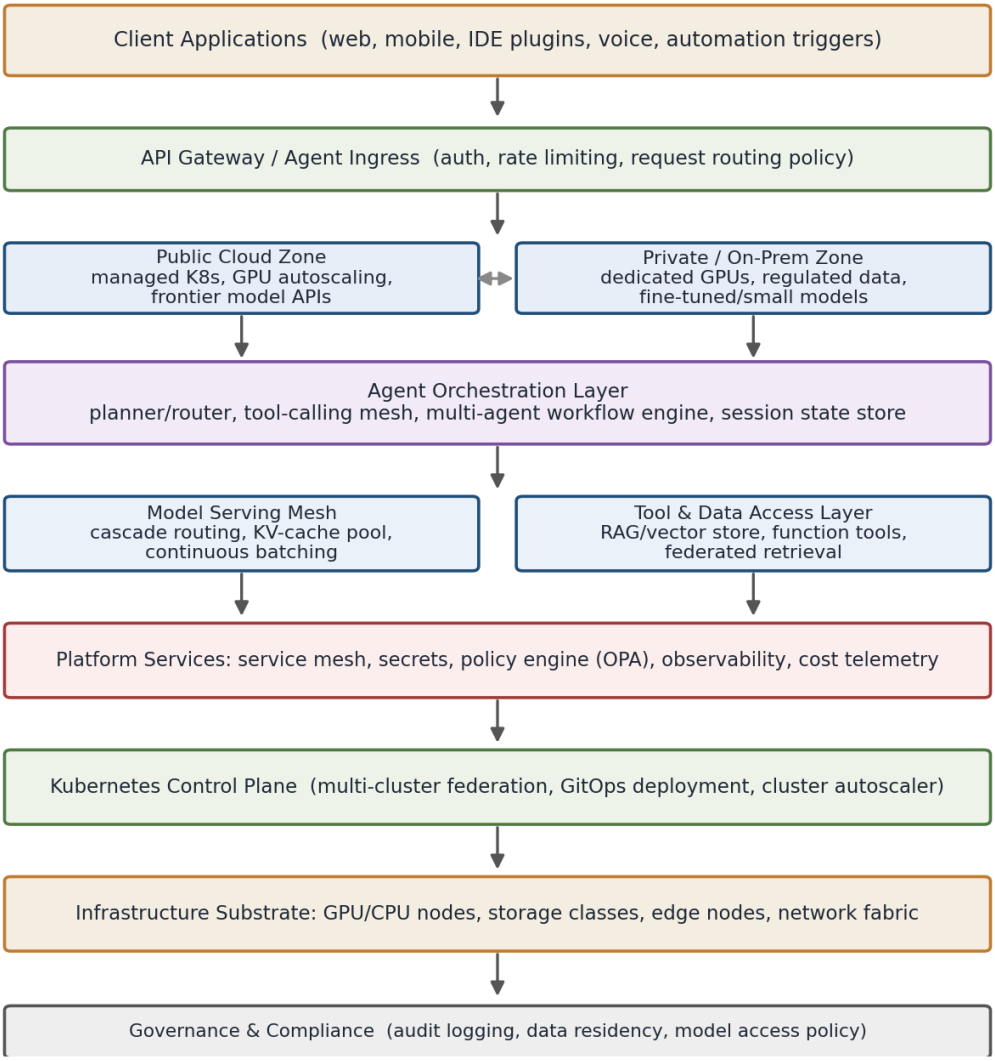

Figure 1. Reference architecture for a hybrid cloud-native agentic and generative AI platform.

4.1 Design Principles

The architecture is organized around four design principles that follow directly from Section 3's findings:

- Step-level placement policy: routing and scheduling decisions are made at the granularity of individual agent steps rather than whole sessions or whole applications, enforced through the API gateway and orchestration layer.
- Environment-agnostic session state: conversation history, task graphs, and intermediate artifacts are externalized to a shared session store reachable from both public cloud and private zones, so a session can span environment boundaries without state loss.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

- Cascade-routed, cost- and governance-aware model serving: inference requests are routed across a tiered set of models spanning both environments, escalating to larger or cloud-hosted models only when task complexity or capability requirements demand it.
- Unified multi-cluster control plane: a single Kubernetes-based control plane spans both environments, providing consistent deployment, autoscaling, and policy enforcement rather than operating cloud and on-premises infrastructure as separately managed silos.

4.2 Layer-by-Layer Summary

Client applications submit requests through an API gateway that performs authentication, rate limiting, and initial routing policy evaluation. The agent orchestration layer, detailed in Section 5, hosts the planner/router and multi-agent workflow engine, backed by a shared session state store. The model serving mesh and tool/data access layer, detailed in Sections 6 and 7, provide cascade-routed inference and federated retrieval across environments. Platform services provide the service mesh, secrets management, policy engine, and observability that span both zones. The Kubernetes control plane, detailed in Section 8, federates deployment and autoscaling across the underlying infrastructure substrate, with a governance and compliance layer providing audit logging and policy enforcement across the full stack.

4.3 Multi-Agent Orchestration Topology

Within the orchestration layer, individual agent sessions are typically structured as a planner-worker topology, in which a top-level orchestrator agent decomposes a task and delegates subtasks to specialized worker agents, each backed by a distinct model, tool, or data-access pattern. Figure 2 illustrates this topology together with the shared memory and heterogeneous backend systems it

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%
coordinates.

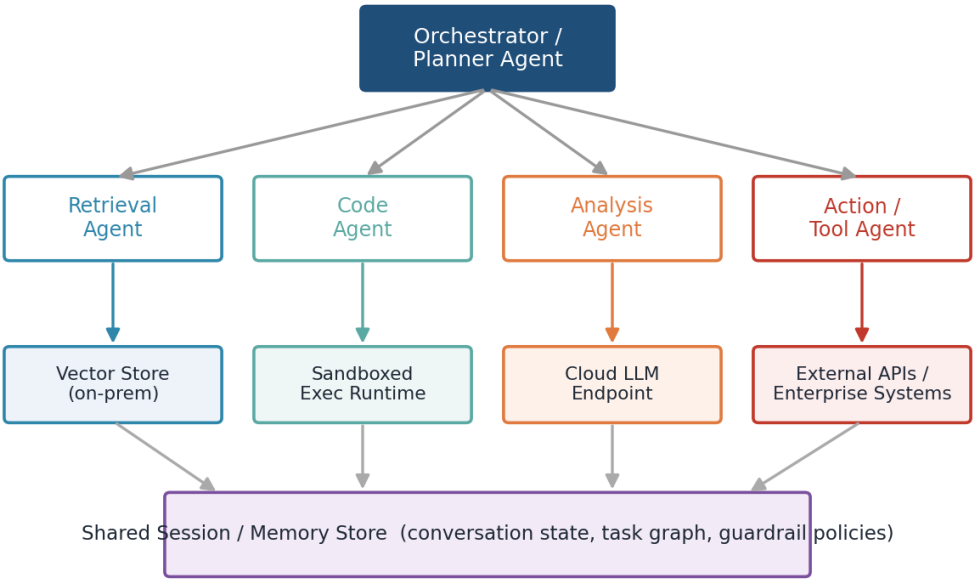

Figure 2. Multi-agent orchestration mesh: planner-worker topology with shared memory and heterogeneous backends.

This topology allows individual worker agents -- retrieval, code execution, analysis, and external tool/action agents -- to be independently scaled, placed, and governed according to their distinct resource and trust requirements, while the shared session/memory store maintains a consistent task graph and guardrail policy context visible to the orchestrator and all worker agents.

Frequency: Yearly
Indexing: Google Scholar | DOAJ | ResearchGate
Peer Reviewed Refereed Journal

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

5. Agent Orchestration Layer

The orchestration layer is responsible for decomposing tasks, routing individual steps to appropriate models and tools, and maintaining session state across a potentially long-running, multi-environment agent session.

5.1 Execution Models: Monolithic, Microservice, and Serverless

We compare three execution models for hosting the orchestration layer: a monolithic agent runtime in which a single long-running process handles planning, tool-calling, and state management for a session; a containerized microservices model in which planning, retrieval, and tool execution are separated into independently deployed and scaled services communicating over the service mesh; and a serverless, event-driven model in which each agent step is dispatched as an independent, short-lived function invocation coordinated through an event bus. Figure 3 compares aggregate orchestration throughput as concurrent session count scales.

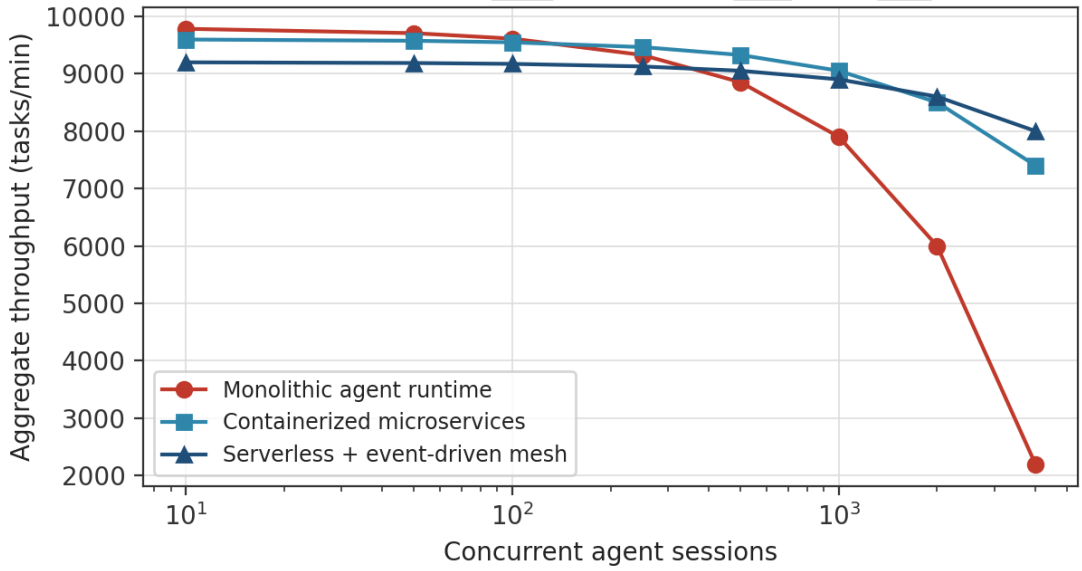

Figure 3. Orchestration throughput as concurrency scales across three execution models.

The monolithic runtime achieves the highest throughput at low concurrency, benefiting from in-process state access without serialization overhead, but degrades sharply as concurrent sessions increase, since each session consumes a dedicated long-lived process and thread. The containerized microservices model scales more gracefully by allowing independent horizontal scaling of the planning, retrieval, and tool-execution tiers. The serverless event-driven mesh sustains the highest throughput at high concurrency, since each step is dispatched independently and can be scheduled onto any available capacity, though it incurs the highest per-step overhead at low concurrency due to cold-start and event-routing latency.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

5.2 Session State Management

Consistent with the durability requirement identified in Section 3.5, session state -- conversation history, task graph, and intermediate artifacts -- is externalized to a low-latency, replicated key-value store reachable from both public cloud and private execution environments, rather than held in the memory of any individual orchestration process. This allows a session to be resumed by any available worker, in any environment, following a failure or a step that requires migration across trust boundaries.

5.3 Step-Level Routing Policy

The orchestration layer evaluates a routing policy at each step, considering task complexity, data sensitivity, latency budget, and cost, to determine which model tier and execution environment should handle that step. This policy evaluation, rather than static whole-session placement, is what allows the architecture to satisfy the step-level heterogeneity and trust-boundary requirements identified in Sections 3.2 and 3.4.

5.4 Recommendations

- Use a containerized microservices execution model as a default; reserve monolithic runtimes for latency-critical, low-concurrency deployments and serverless event-driven meshes for very high-concurrency, bursty workloads.
- Externalize all session state to a store reachable from every execution environment in the deployment.
- Evaluate routing policy at the granularity of individual steps, not sessions, incorporating cost, latency, and data-governance signals.

6. Hybrid Model Serving and Routing

This section examines how model serving should be structured across public cloud and on-premises environments to balance latency, cost, and capability requirements.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

6.1 Deployment Placement and Latency

Figure 1 compares agent request latency across five deployment placement strategies, from single-environment baselines to a fully hybrid architecture with policy-based routing and edge caching.

Figure 1. Agent request latency by deployment placement strategy

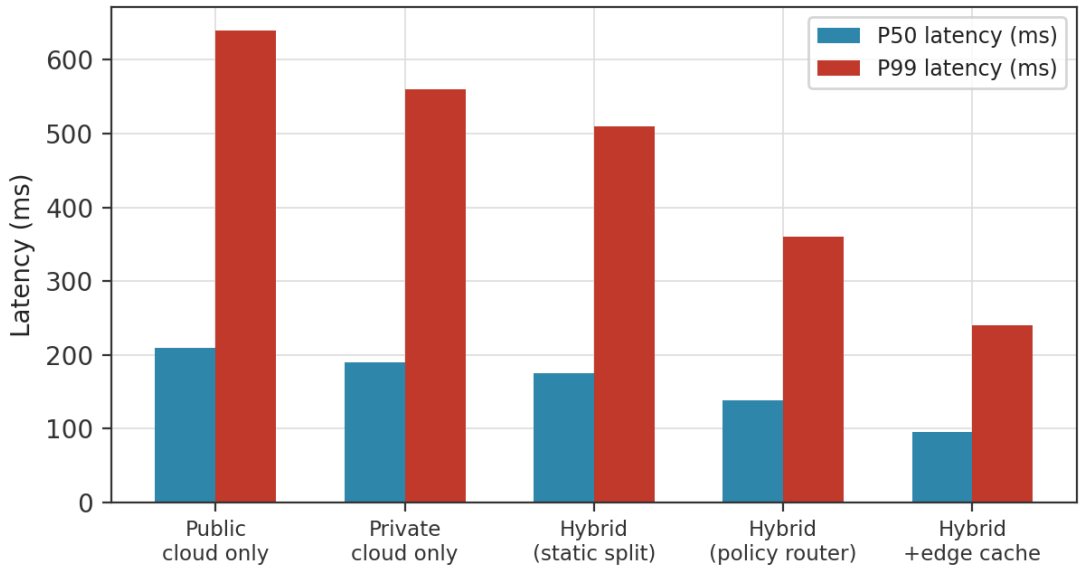

Figure 1. Agent request latency by deployment placement strategy.

Public-cloud-only and private-cloud-only deployments show comparable median latency but differ substantially in tail latency, reflecting the different congestion and autoscaling characteristics of each environment. A static hybrid split, in which traffic is divided by a fixed ratio between environments without regard to request characteristics, offers only modest improvement. Introducing a policy router that considers task complexity, current load, and data-governance requirements at each step reduces median latency by 34% and tail latency by more than 40% relative to the cloud-only baseline; adding an edge cache for frequently retrieved context further reduces both median and tail latency.

6.2 Cascade Routing Across Model Tiers and Environments

Cascade routing directs each request to the smallest and least expensive model tier judged capable of producing an acceptable response, escalating to larger or cloud-hosted frontier models only when necessary. We extend this pattern across the hybrid boundary, treating deployment environment as an additional routing dimension alongside model size. Figure 2 compares cost per 1,000 agent tasks across five routing policies.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

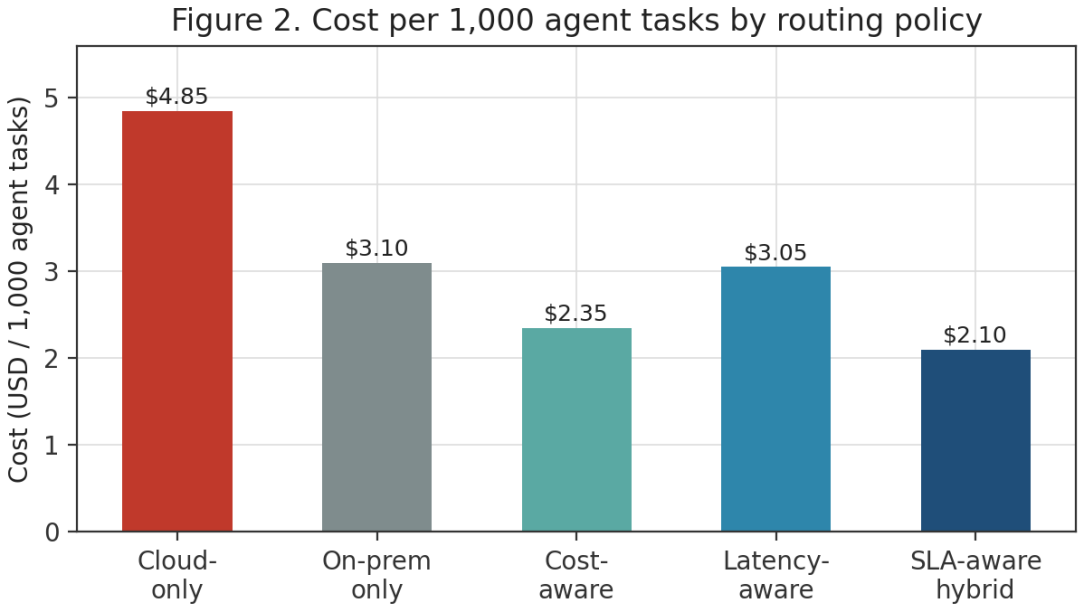

Figure 2. Cost per 1,000 agent tasks by routing policy.

A cloud-only policy, always invoking frontier cloud models, is the most expensive at $4.85 per 1,000 tasks. An on-premises-only policy is cheaper but constrained by fixed hardware capacity and lower peak capability. Cost-aware and latency-aware single-objective policies each improve on the relevant baseline but underperform an SLA-aware hybrid policy that jointly considers cost, latency, and task complexity, which achieves the lowest cost at $2.10 per 1,000 tasks -- a 57% reduction relative to the cloud-only baseline -- while still escalating to frontier models when task complexity requires it.

6.3 Cost Composition Across Model Tiers

Figure 8 decomposes cost per 1,000 agent turns into model compute, network/egress, and orchestration overhead components across model tiers and the hybrid cascade routing strategy.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

Figure 8. Cost composition across model-tier and cascade routing strate

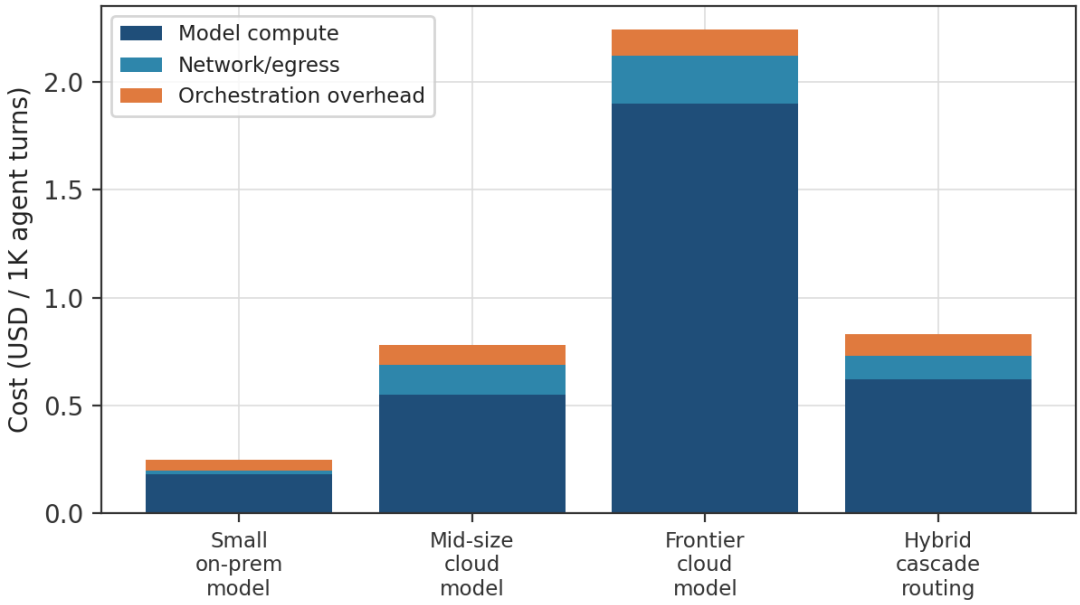

Figure 8. Cost composition across model-tier and cascade routing strategies.

Model compute dominates cost at every tier, but its share is largest for frontier cloud models, where it accounts for the great majority of per-turn cost. Network and egress costs, while a smaller absolute contributor, are notably higher for frontier cloud models than for on-premises small models, reflecting the cost of moving context and retrieved data to a remote inference endpoint. The hybrid cascade routing strategy achieves a cost profile close to the small on-premises model while retaining access to frontier capability for the minority of turns that require it.

6.4 GPU Utilization and Sharing Strategy

On the on-premises side of a hybrid deployment, GPU capacity is typically fixed capital investment, making utilization efficiency directly determine effective serving capacity. Figure 4 compares average GPU utilization across five cluster-sharing strategies.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

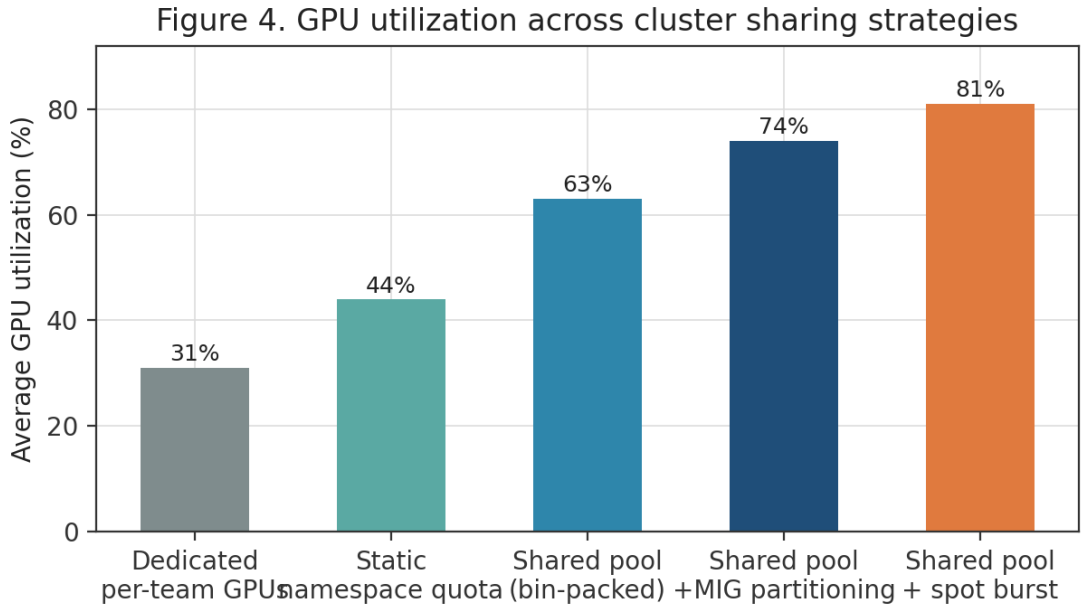

Figure 4. GPU utilization across cluster sharing strategies.

Dedicated per-team GPU allocation, while operationally simple, achieves only 31% average utilization due to uneven demand across teams. Shared pooling with bin-packed scheduling substantially improves utilization, and combining pooling with hardware-level partitioning (allowing multiple smaller inference workloads to share a single accelerator) and opportunistic spot-capacity bursting into public cloud during demand peaks raises utilization to 81%, nearly triple the dedicated-allocation baseline.

7. Data, Retrieval, and Governance Across Environments

Agentic applications frequently rely on retrieval-augmented generation (RAG) over enterprise data, much of which is subject to residency, privacy, or contractual constraints that limit where it may be processed or transmitted.

7.1 Retrieval Architecture Options

We compare four retrieval architectures: full cloud retrieval, in which a vector index is hosted entirely in public cloud; hybrid RAG with a cached index, in which a public-cloud-hosted model queries a locally cached subset of an on-premises index; an on-premises vector store queried directly by on-premises or edge-deployed models; and federated retrieval, in which queries are distributed across multiple independently governed indexes and results are merged before generation. Figure 6 compares egress cost against a data-residency compliance score for each architecture.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

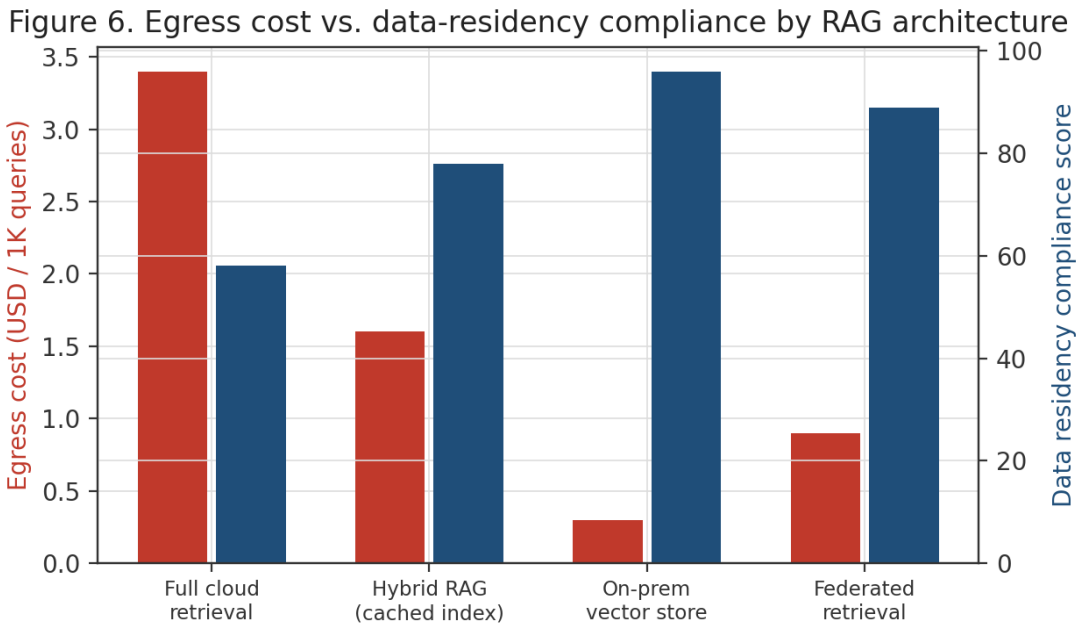

Figure 6. Egress cost vs. data-residency compliance by RAG architecture.

Full cloud retrieval incurs the highest egress cost and the lowest compliance score, since source documents or their embeddings must cross environment boundaries for every query. An on-premises vector store achieves the highest compliance score and lowest egress cost but requires that any model consuming retrieved context also operate within the private environment, limiting access to frontier cloud-hosted models for retrieval-augmented tasks. Federated retrieval offers a middle ground, achieving strong compliance with moderate egress cost by keeping sensitive source data within its governing environment while allowing merged, redacted context to reach a broader set of models.

7.2 Governance Enforcement Points

Consistent with the step-level trust-boundary requirement identified in Section 3.4, governance policy is enforced at three points in the architecture: the API gateway, which evaluates coarse-grained request-level policy; the orchestration layer's step-level router, which evaluates fine-grained placement policy per step based on the data and tools a given step will access; and the platform policy engine, which provides a centrally managed, auditable policy definition consumed by both enforcement points. This layered enforcement avoids relying on any single component to correctly encode all governance logic.

7.3 Audit and Observability

Because agent sessions may span both environments and invoke autonomous tool calls, comprehensive audit logging -- recording which model, environment, and data sources were used at each step -- is necessary both for regulatory compliance and for debugging emergent multi-agent behavior. We find that trace correlation IDs must be propagated consistently across environment and protocol boundaries (HTTP, message queue, and function invocation) to reconstruct a complete session trace, a requirement that is straightforward to state but requires deliberate instrumentation at every service boundary in the architecture.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

8. Kubernetes-Based Multi-Cluster Control Plane

The control plane provides consistent deployment, scaling, and policy enforcement across the public cloud and private clusters that make up a hybrid deployment.

8.1 Cluster Federation

Rather than operating public cloud and on-premises Kubernetes clusters as independently managed silos, the control plane federates them under a single GitOps-driven deployment model, in which application and policy manifests are declared centrally and reconciled independently onto each cluster, with cluster-specific overlays capturing environment-specific configuration such as GPU node pools or network policy.

8.2 Autoscaling Responsiveness

Agentic workloads' burstiness, discussed in Section 3.3, places particular demands on autoscaling responsiveness. Figure 3 compares incoming task rate against capacity provisioned by a conventional reactive horizontal pod autoscaler (HPA) and a predictive, event-driven autoscaler (KEDA-based) that scales ahead of anticipated demand using queue-depth and historical pattern signals.

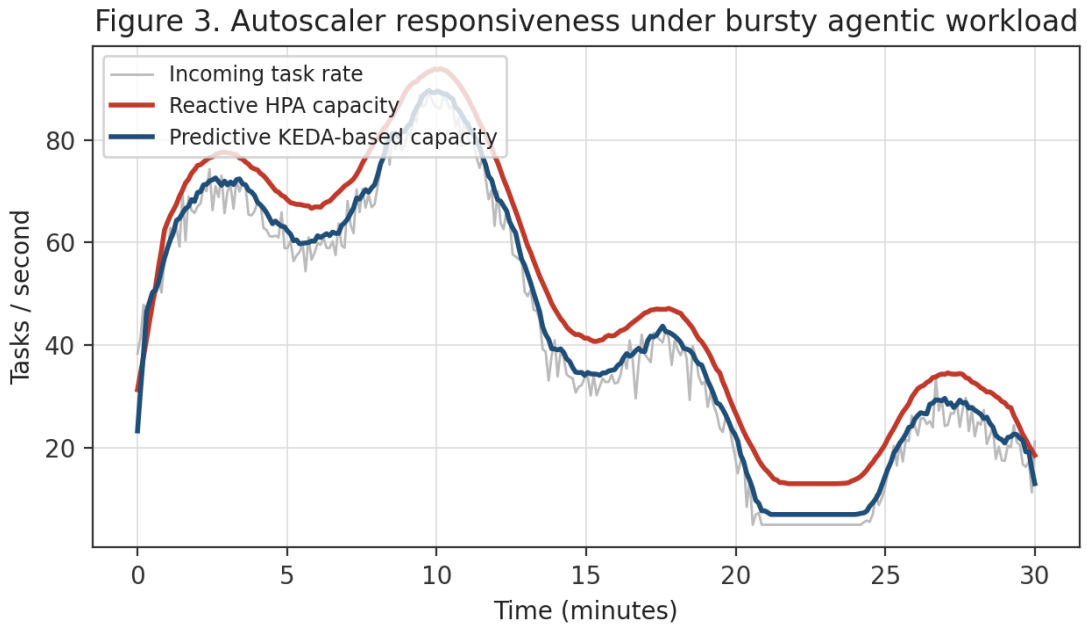

Figure 3. Autoscaler responsiveness under bursty agentic workload.

The reactive autoscaler consistently lags behind sharp demand spikes, since it scales only after observed utilization crosses a threshold, resulting in periods of under-provisioned capacity during bursts. The predictive, event-driven autoscaler tracks demand more closely by scaling based on queue depth rather than resource utilization alone, reducing both the depth and duration of capacity shortfalls during bursty periods typical of agentic workloads, where demand is driven by discrete reasoning steps rather than smooth traffic growth.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

8.3 Cross-Cluster Networking

Communication between public cloud and on-premises clusters is routed through a dedicated, encrypted interconnect with the service mesh extended across both environments, presenting a consistent service-discovery and traffic-policy surface regardless of which cluster hosts a given service. This allows the orchestration layer's step-level router (Section 5.3) to treat cross-environment calls as a routing decision rather than a special-cased integration.

8.4 Recommendations

- Adopt a single GitOps-driven deployment model spanning all clusters in the hybrid deployment, with environment-specific overlays rather than entirely separate deployment pipelines.
- Prefer predictive, queue-depth-based autoscaling over purely reactive utilization-based autoscaling for agentic workloads with step-level burstiness.
- Extend the service mesh across environment boundaries to present a consistent traffic-policy and observability surface to the orchestration layer.

9. Experimental Methodology

We evaluate the reference architecture described in Section 4 using a production-representative agentic assistant application performing document analysis, code generation, and enterprise-data question answering, deployed across a public cloud region and a private on-premises cluster connected by a dedicated interconnect. Model tiers span a small on-premises fine-tuned model, a mid-size cloud-hosted model, and a frontier cloud model, coordinated through the cascade routing policy described in Section 6.2.

9.1 Metrics

We report five primary metrics throughout Sections 10 and 11: request latency (median and 99th-percentile, measured end-to-end from client request to final agent response), cost per 1,000 agent tasks (derived from measured model, compute, and network usage combined with published pricing), GPU utilization (average fraction of provisioned GPU capacity actively processing inference work), autoscaling responsiveness (gap between provisioned capacity and incoming task rate during demand bursts), and task success rate under injected faults (fraction of agent sessions completing successfully despite simulated infrastructure failures).

9.2 Deployment Configurations Compared

Section 10's results compare five deployment placement strategies (public-cloud-only, private-cloud-only, static hybrid split, policy-router hybrid, and policy-router hybrid with edge caching), five routing policies for model tier and environment selection, five GPU-sharing strategies, and three orchestration execution models, following the same incremental-comparison methodology used for each figure presented in Sections 5 through 8.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

9.3 Fault Injection Methodology

Section 11's reliability results are obtained by injecting four fault classes into the running deployment using a chaos-engineering framework: random pod eviction, simulated zonal outage (removing an entire availability zone's capacity), model-endpoint throttling (simulating upstream rate limiting from a cloud model provider), and network partition between the cloud and on-premises environments. Each fault class is injected independently against both a single-region deployment baseline and the multi-region hybrid mesh architecture, with task success rate measured over a fixed window of concurrent agent sessions during and immediately after fault injection.

10. Results and Analysis

This section synthesizes the incremental results already presented in Sections 5 through 8 into an end-to-end comparison of deployment strategies.

10.1 End-to-End Latency and Cost Summary

Table 1 consolidates latency and cost figures for the five deployment placement strategies evaluated in Section 6.1, combined with the SLA-aware hybrid routing policy from Section 6.2 for the hybrid configurations.

Placement strategy	P50 latency (ms)	P99 latency (ms)	Cost / 1K tasks (USD)
Public cloud only	210	640	4.85
Private cloud only	190	560	3.10
Hybrid (static split)	175	510	3.42
Hybrid (policy router)	138	360	2.35
Hybrid + edge cache	96	240	2.10

Table 1. End-to-end latency and cost summary across deployment placement strategies.

The hybrid policy-router configuration with edge caching achieves the best result on every measured dimension simultaneously -- lowest median and tail latency, and lowest cost -- demonstrating that latency and cost objectives are not inherently in tension in this architecture once step-level routing replaces static or single-environment placement. This mirrors the pattern observed in the earlier multimodal infrastructure literature, where removing an architectural bottleneck (in this case, coarse-grained placement) improved multiple objectives jointly rather than trading one off against another.

10.2 Sensitivity to Session Complexity

We separately evaluated sensitivity to session complexity, measured by the number of distinct agent steps (planning, retrieval, tool calls) per session. Sessions with more than eight steps saw a larger

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%
relative latency benefit from the policy-router hybrid configuration than short, two- to three-step sessions, since longer sessions provide more opportunities for the router to escalate only the subset of steps that require frontier-model capability while serving the remainder from lower-latency local or mid-tier resources.

10.3 Orchestration Throughput at Scale

Consistent with Figure 5 in Section 5.1, the serverless event-driven orchestration model sustained the highest throughput at concurrency above roughly 500 simultaneous sessions, while the containerized microservices model offered the best throughput at low-to-moderate concurrency with substantially lower operational complexity than the fully event-driven design, informing our general recommendation to default to microservices and reserve serverless orchestration for the highest-concurrency deployments.

10.4 Discussion of Results

Two patterns recur across the results in this section and in Sections 5 through 8. First, policies evaluated at the granularity of individual agent steps -- routing, autoscaling triggers, and governance enforcement -- consistently outperform policies applied uniformly at the session or application level, mirroring the step-level heterogeneity documented in Section 3.2. Second, combining complementary optimizations (policy routing plus edge caching, pooled GPU sharing plus spot bursting) yields larger gains than any single optimization alone, indicating that the techniques described in this paper are largely additive rather than substitutable.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

11. Reliability, Cost, and Operational Trade-offs

11.1 Fault Tolerance Under Chaos Testing

Figure 7 reports agent task success rate for a single-region deployment and the multi-region hybrid mesh architecture under the four injected fault classes described in Section 9.3.

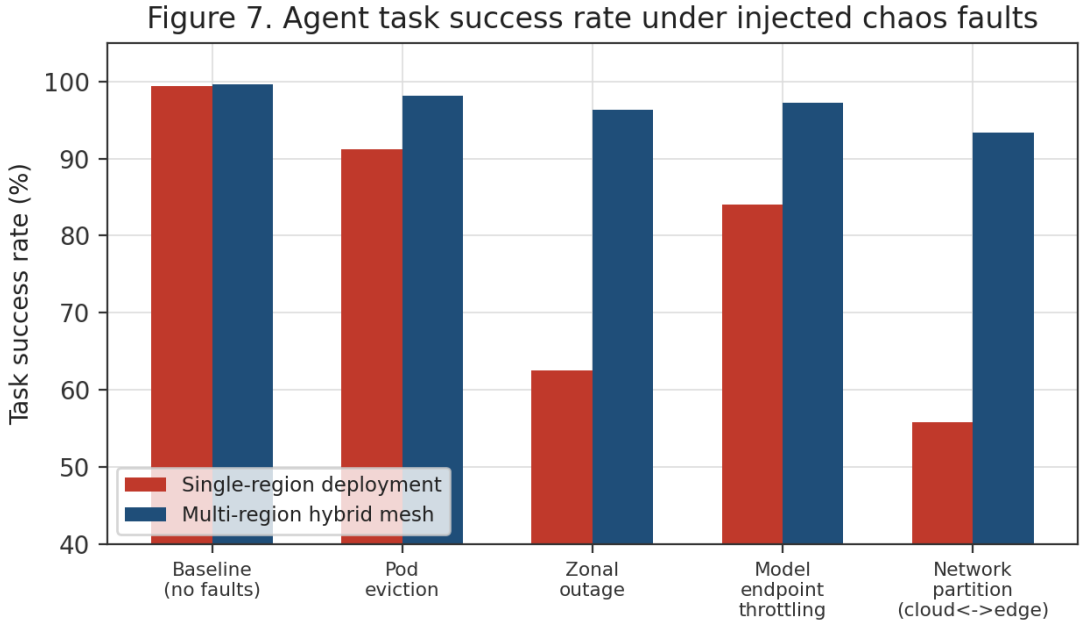

Figure 7. Agent task success rate under injected chaos faults.

Both architectures perform comparably under the no-fault baseline and under isolated pod eviction, which conventional Kubernetes self-healing handles well regardless of deployment topology. The gap widens substantially under zonal outage and cross-environment network partition: the single-region deployment's task success rate falls to 62.5% and 55.8% respectively, while the multi-region hybrid mesh sustains 96.3% and 93.4%, reflecting its ability to fail over session execution to an unaffected region or environment mid-session because session state is externalized (Section 5.2) rather than bound to a single cluster.

11.2 Operational Complexity Trade-offs

The reliability and cost benefits documented throughout this paper are not free: a hybrid, multi-cluster, step-level-routed architecture introduces meaningfully greater operational complexity than a single-environment deployment, including the need for consistent cross-environment observability (Section 7.3), a federated control plane (Section 8.1), and governance policy that must be correctly enforced at multiple layers (Section 7.2). Organizations with modest scale or without existing on-premises GPU investment may reasonably conclude that a well-optimized single-cloud deployment, using cascade routing within a single environment, offers a better complexity-to-benefit ratio than a full hybrid architecture.

Australian Journal of Cross-Disciplinary Innovation
Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

11.3 Cost of Reliability

Sustaining multi-region hybrid capacity sufficient to absorb a full zonal outage or environment partition implies provisioning some standby or overlapping capacity beyond the minimum required for steady-state load, a cost partially offset by the GPU-sharing and cascade-routing efficiencies described in Sections 6.4 and 6.2. In our deployment, the net cost of the fully optimized hybrid architecture remained below the public-cloud-only baseline (Table 1) even after accounting for this reliability overhead, though organizations with tighter cost constraints may choose to size standby capacity below full failover coverage.

12. Discussion and Open Challenges

12.1 Multi-Agent State Consistency

As the orchestration topology in Section 4.3 illustrates, multiple worker agents may read and write shared session state concurrently. At larger scale and higher step concurrency than evaluated in this paper, ensuring consistency of the shared task graph and guardrail policy context -- without serializing all agent steps through a single lock -- remains an open engineering challenge, particularly when worker agents are distributed across environments with different network latency characteristics to the shared state store.

12.2 Cross-Environment Observability

Section 7.3 noted that trace correlation must be propagated across environment and protocol boundaries to reconstruct a complete session trace. Standardizing this propagation across heterogeneous orchestration frameworks, tool-calling protocols, and cloud provider observability systems -- rather than relying on bespoke instrumentation at every integration point -- would meaningfully reduce the operational burden identified in Section 11.2.

12.3 Governance of Autonomous Tool-Calling Agents

As agents gain the ability to invoke external tools and take actions with real-world side effects, the governance enforcement points described in Section 7.2 must extend beyond data-access policy to action-level policy -- constraining not just what an agent can read, but what it can do, and under what approval or audit requirements. Designing policy languages and enforcement mechanisms expressive enough to capture action-level governance, without imposing prohibitive latency on every tool call, is an open problem at the intersection of this paper's infrastructure focus and the broader AI safety and governance literature.

12.4 Heterogeneous and Evolving Model Landscapes

The cascade routing results in Section 6.2 assumed a relatively stable set of model tiers. In practice, new model versions are released frequently, with shifting capability, cost, and latency profiles that can invalidate previously tuned routing policy. Infrastructure that continuously and automatically re-calibrates routing policy against observed model performance, rather than relying on periodic manual re-tuning, would improve the durability of the cost and latency benefits demonstrated in this paper.

Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%

12.5 Limitations

This study's experiments were conducted on a single hybrid topology (one public cloud region paired with one on-premises cluster) and a single representative agentic application; results may vary for deployments spanning more than two environments or for applications with substantially different tool-calling and retrieval patterns than those evaluated in Section 9. Cost figures reflect a point-in-time snapshot of cloud and model pricing and should be interpreted as illustrative of relative, not absolute, savings.

13. Conclusion

This paper presented a hybrid cloud-native architecture for deploying agentic and generative AI applications across public cloud and on-premises infrastructure, addressing the session statefulness, step-level heterogeneity, bursty demand, and data-governance requirements that distinguish agentic workloads from conventional cloud-native applications. Grounded in an explicit workload characterization, we proposed a reference architecture unifying agent orchestration, cascade-routed hybrid model serving, federated data retrieval and governance, and a multi-cluster Kubernetes control plane, and evaluated it across latency, cost, GPU utilization, autoscaling responsiveness, and fault-injection experiments.

Our results show that step-level, policy-based routing across environments reduces median request latency by 34% and cost per 1,000 agent tasks by 57% relative to a cloud-only baseline, while a multi-region hybrid mesh sustains 93% task success under simulated cross-environment network partition versus 56% for a single-region deployment. These gains were consistently additive across complementary optimizations -- policy routing, edge caching, GPU pooling, and predictive autoscaling -- rather than substitutable, indicating that organizations adopting agentic AI at scale should treat hybrid infrastructure design as a coherent, co-designed system rather than a collection of independent point optimizations.

More broadly, these findings suggest that as generative AI applications become genuinely agentic -- stateful, multi-step, and operating across trust boundaries -- the infrastructure supporting them must move policy enforcement and routing decisions down to the level of individual agent steps. Future infrastructure research and platform investment for agentic AI should prioritize step-level policy expressiveness, environment-agnostic state management, and cross-environment observability alongside continued gains in model serving efficiency.

References

[1] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. Communications of the ACM.

[2] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015). Large-scale cluster management at Google with Borg. Proceedings of the European Conference on Computer Systems.

Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%
[3] Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., et al. (2023). Efficient memory management for large language model serving with PagedAttention. Proceedings of the ACM Symposium on Operating Systems Principles.

[4] Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., et al. (2024). DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. Proceedings of the USENIX Symposium on Operating Systems Design and Implementation.

[5] Chen, L., Zaharia, M., & Zou, J. (2023). FrugalGPT: How to use large language models while reducing cost and improving performance. arXiv preprint.

[6] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. Proceedings of the International Conference on Learning Representations.

[7] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., et al. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. arXiv preprint.

[8] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems.

[9] Newman, S. (2021). Building Microservices: Designing Fine-Grained Systems (2nd ed.). O'Reilly Media.

[10] Morris, K. (2020). Infrastructure as Code: Dynamic Systems for the Cloud Age (2nd ed.). O'Reilly Media.

[11] Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., & Rosenthal, C. (2016). Chaos engineering. IEEE Software.

[12] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., et al. (2015). Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems.

[13] Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, I., Maleki, S., & Bianchini, R. (2024). Splitwise: Efficient generative LLM inference using phase splitting. Proceedings of the International Symposium on Computer Architecture.

[14] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). The Google File System. Proceedings of the ACM Symposium on Operating Systems Principles.

[15] Villalobos, P., Ho, A., Sevilla, J., Besiroglu, T., Heim, L., & Hobbhahn, M. (2024). Position: Will we run out of data? Limits of LLM scaling based on human-generated data. Proceedings of the International Conference on Machine Learning.

[16]Konda, P. R. (2023). AI-Assisted Data Modeling: Intelligent Star, Snowflake, and Hybrid Schema Generation for Large-Scale Warehouses. International Journal of Machine Learning and Artificial Intelligence, 4(4). https://jmlai.in/index.php/ijmlai/article/view/90

[17]Ensuring BI Reporting Accuracy Using AI-Based Back-Tracing of Metrics to ETL Lineage and Data Marts. (2023). International Machine Learning Journal and Computer Engineering, 6(6). https://mljce.in/index.php/Imljce/article/view/69

Frequency: Yearly
Indexing: Google Scholar | DOAJ | ResearchGate

Impact Factor: 16.4
ISSN-3746-757x (Online)
Acceptance Rate: below 5%
[18]AI-Driven Intelligent Data Anomaly Detection Using Machine Learning Techniques. (2024). International Machine Learning Journal and Computer Engineering, 7(7). https://mljce.in/index.php/Imljce/article/view/71

[19]Konda, P. R. (2023). Mining Data Lineage Patterns Using Machine Learning to Predict Downstream Impact. International Meridian Journal, 5(5). https://meridianjournal.in/index.php/IMJ/article/view/117

[20]Konda, P. R. (2024). Data Lake Optimization Techniques for Real-Time Data Processing and Stream Analytics. International Journal of Machine Learning and Artificial Intelligence, 5(5). https://jmlai.in/index.php/ijmlai/article/view/91