International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V6I4P136
Eureka Vision Publication | Volume 6, Issue 4, 253-262, 2025

*Original Article*

# Counterfactual Deployment Event Graphs for Explainable Cost Attribution and Resource Efficiency Optimization in Kubernetes Workloads

Nilesh Mutyam
Senior Software Development Engineer, PayPal Inc, Dallas, TX, USA.

***Received On: 24/11/2025*** ***Revised On: 19/12/2025*** ***Accepted On: 24/12/2025*** ***Published On: 30/12/2025***

*Abstract - Kubernetes has become the dominant substrate for cloud-native workload orchestration, yet its cost behavior remains difficult to explain because infrastructure expenditure is produced by a dynamic interaction among deployment events, scheduler decisions, autoscaling loops, resource requests, workload dependencies, and shared-node allocation policies. Existing cost observability systems often report expenditure retrospectively at namespace, service, or cluster levels, but they rarely explain why a cost increase occurred, which deployment event produced it, or what alternative configuration would have reduced waste without violating service-level objectives. This paper proposes Counterfactual Deployment Event Graphs (CDEGs), a conceptual and methodological framework for explainable cost attribution and resource efficiency optimization in Kubernetes workloads. CDEGs model Kubernetes operational history as a temporally indexed, causally annotated graph connecting deployments, pods, nodes, autoscalers, telemetry streams, billing records, configuration changes, and service dependencies. The framework integrates causal counterfactual reasoning, graph-based provenance, workload telemetry, and FinOps-oriented cost allocation to estimate how observed costs would have changed under alternative deployment decisions. Unlike purely correlational dashboards, CDEGs support path-specific cost explanations, actionable recourse recommendations, and guarded optimization policies for rightsizing, autoscaling, scheduling, and consolidation. The paper defines the problem, presents the graph model and counterfactual attribution procedure, describes a reference architecture, and proposes evaluation criteria for attribution fidelity, counterfactual validity, optimization effectiveness, operational overhead, and explanation usability. Analytical discussion demonstrates how CDEGs can distinguish legitimate elasticity from avoidable overprovisioning, identify deployment-induced waste, and bridge engineering and financial accountability. The study contributes a research agenda for explainable Kubernetes cost intelligence that is auditable, causally grounded, and practically aligned with production reliability constraints.*

*Keywords - Kubernetes, Counterfactual Explanation, Cloud Finops, Cost Attribution, Resource Efficiency, Autoscaling, Causal Inference, Deployment Event Graph, Observability, Microservices.*

## 1. Introduction

Kubernetes has transformed cloud infrastructure from a collection of individually managed servers into a declarative control plane for scheduling, scaling, and repairing containerized applications. The economic consequence of this transformation is substantial: cost is no longer attributable only to provisioned virtual machines, reserved capacity, or storage volumes, but to a high-frequency stream of deployment events and resource claims distributed across shared nodes. A single change to a resource request, horizontal autoscaler threshold, affinity constraint, container image, sidecar, rollout strategy, or node pool can modify the bin-packing state of the cluster and produce downstream cost changes that are difficult to assign to a responsible engineering decision. This opacity is intensified in multi-tenant clusters, where services share nodes, teams share namespaces, and workloads generate uneven mixtures of CPU, memory, network, and storage demand. The lineage from “deployment change” to “billing impact” is therefore not linear; it is a graph of operational dependencies, temporal effects, and scheduler-mediated interactions. The design lineage of Kubernetes itself reflects lessons from large-scale cluster managers such as Borg and Omega, where utilization, scheduling, and workload heterogeneity were central operating concerns [1].

Cost attribution in Kubernetes is also a socio-technical problem. Finance teams require chargeback or showback reports that are stable enough for budgeting, while platform teams require operational explanations that are precise enough to guide rightsizing and autoscaling actions. A cost spike may be caused by legitimate traffic growth, a failed rollout that repeatedly restarts pods, a memory leak that triggers horizontal scale-out, a pod anti-affinity rule that prevents consolidation, or a resource request set far above observed usage. Conventional dashboards usually show that cost increased, but they seldom answer the counterfactual question that matters to practitioners: “What would the cost have been if the deployment had used a different configuration, while maintaining reliability constraints?” Cloud financial management literature has emphasized

collaborative accountability, continuous optimization, and shared vocabulary between finance and engineering, but Kubernetes introduces a layer of indirection that makes these practices technically harder to implement [4].

This paper argues that explainable cost attribution for Kubernetes must move beyond static allocation rules toward causally structured operational histories. A cost report that allocates node cost proportionally to CPU requests may be useful for accounting, but it can be misleading for optimization because it does not distinguish between necessary reserved capacity, scheduler fragmentation, idle safety margin, workload-induced scale-out, and control-plane policy decisions. Similarly, a machine-learning model that predicts future cost may improve forecasting but may not provide a trustworthy explanation of which deployment event should be changed. In production systems, the quality of predictive or diagnostic models depends strongly on model selection, feature representation, and performance evaluation, as shown in comparative work on software defect prediction [3]. Kubernetes cost intelligence requires the same discipline, but with an additional requirement: explanations must be actionable under operational constraints.

The proposed framework, Counterfactual Deployment Event Graphs (CDEGs), treats Kubernetes cost as the outcome of an event graph rather than as an isolated metric. Each graph node represents an operational entity or event, such as a Deployment revision, ReplicaSet, Pod, node, container, autoscaler decision, service dependency, metric window, or billing record. Each edge represents a typed relation, such as "created," "scheduled-on," "scaled-by," "communicates-with," "requests," "uses," "constrained-by," or "allocated-to." The graph is temporal, because the same service can pass through multiple revisions and scaling states; it is causal, because some edges are interpreted as candidate mechanisms for cost change; and it is counterfactual, because alternative interventions can be simulated over the graph to estimate avoided or induced cost. The framework is intended to complement, not replace, existing Kubernetes metrics, observability pipelines, and FinOps practices.

The research motivation is grounded in three gaps. First, Kubernetes cost allocation often remains descriptive, while optimization requires causal reasoning over deployment events. Second, autoscaling and rightsizing techniques frequently optimize resource utilization without producing explanations that are understandable to service owners and financial stakeholders. Third, explainable AI methods have matured in prediction domains, but their adaptation to cloud-native cost attribution remains underdeveloped. This paper therefore contributes a formal problem statement, a graph-based methodology, a conceptual architecture, and a set of evaluation criteria for explainable Kubernetes cost optimization.

The remainder of the paper is organized as follows. Section 2 reviews background and related work. Section 3 formulates the problem. Section 4 presents the proposed CDEG methodology. Section 5 describes the conceptual architecture. Section 6 defines evaluation metrics. Section 7 provides analytical discussion. Section 8 explains practical implications. Sections 9 and 10 discuss limitations and future research directions. Section 11 concludes the paper.

## 2. Background and Related Work

The technical foundation of Kubernetes cost behavior lies in cluster scheduling and resource isolation. Borg demonstrated that high utilization in large clusters requires coordinated admission control, task packing, resource estimation, and workload isolation across heterogeneous jobs. These ideas remain visible in Kubernetes through pod scheduling, resource requests, controller reconciliation, and node-level resource accounting. However, Kubernetes exposes many of these controls to application teams, which means that cost behavior is partly governed by declarative workload configuration rather than by a single central operator. The Borg experience shows that utilization is not merely a hardware-efficiency metric but an emergent property of workload declarations, scheduler behavior, and operational policy [5].

Kubernetes also inherits architectural principles from systems that separated scheduling concerns while preserving shared cluster state. Omega proposed parallel scheduler architecture using optimistic concurrency over shared state, illustrating that scheduling decisions are not isolated; they interact through contention for resources and through the timing of updates. In Kubernetes, multiple controllers, autoscalers, operators, and admission policies may act over the same workload graph. This creates an attribution challenge because a pod's cost may be influenced not only by its request but also by when it arrived, which node pool had spare capacity, and what other controllers were doing at the time. The scheduler's decision is therefore a mediating variable between deployment intent and realized cost [21].

Kubernetes practice literature emphasizes that resource requests and limits are central abstractions for workload management, but the economic interpretation of these abstractions is nuanced. Requests influence scheduling and capacity reservation, while limits influence runtime enforcement. A pod can be cheap in observed CPU usage but expensive in reserved capacity if its request prevents consolidation. Conversely, low requests can reduce apparent cost while increasing throttling risk or node pressure. Introductory Kubernetes texts provide the operational mechanics of controllers, scheduling, services, and declarative configuration, but production cost attribution requires a higher-level model that connects those mechanics to financial outcomes [17].

Autoscaling has been widely studied as a mechanism for balancing performance and efficiency. Non-disruptive vertical scaling and resource estimation approaches attempt to resize pods based on observed demand, reducing the gap between declared and actual resource needs. Such work is important because overestimated requests are a major source of avoidable cost in Kubernetes clusters, while

underestimated requests create instability and degraded service quality. For cost attribution, vertical scaling research supplies a way to interpret excess request as a measurable counterfactual: if a pod had requested an amount closer to its observed safe requirement, the scheduler might have packed the cluster differently and reduced node-hour expenditure [7].

Large-scale production systems have also demonstrated the value of automatic resource configuration. Google's Autopilot adjusts workload resources by combining historical observations, heuristics, and safety margins, illustrating that rightsizing must balance slack reduction against the risk of out-of-memory termination or CPU degradation. This insight is central to CDEGs: the framework does not define efficiency as minimizing cost alone, but as minimizing avoidable cost subject to reliability, performance, and safety constraints. A counterfactual that reduces cost by violating a latency or availability objective is not an acceptable explanation for optimization [16].

Container orchestration research has further shown that heterogeneous resources and cost-aware placement can materially affect infrastructure expenditure. Cost-efficient container orchestration strategies evaluate how workloads can be allocated across heterogeneous instances to improve utilization and reduce spending. Yet many such approaches focus primarily on optimization rather than explainability. CDEGs extend this line of work by asking not only where a workload should run, but why the observed placement produced a particular cost and which event or configuration change would have altered it [10].

Microservice-specific resource management adds another layer of complexity because end-to-end performance depends on dependency chains rather than isolated container metrics. SHOWAR, for example, addresses rightsizing and scheduling of microservices by jointly considering replicas and CPU allocation. This is relevant because a Kubernetes service owner may overprovision one tier to mask bottlenecks in another tier, causing cost to appear at the wrong location. CDEGs explicitly represent service dependencies so that cost attribution can distinguish between local resource waste and upstream or downstream pressure [22].

Machine-learning-based resource managers such as Sinan demonstrate that tracing data and QoS-aware models can guide resource allocation for microservice systems. They show that resource decisions should account for inter-tier dependencies and tail-latency objectives rather than relying only on CPU utilization. In CDEGs, this insight is incorporated through dependency-aware counterfactuals: changing the resources of one deployment may affect another service through queueing delay, retry storms, or backpressure. Therefore, counterfactual cost estimation must consider not only the direct cost of a pod but also the induced cost of dependent workloads [24].

Edge and distributed Kubernetes environments intensify the problem because clusters may have heterogeneous capacity, variable latency, and different node price models. Machine-learning-based scaling for Kubernetes edge clusters shows that workload placement and scaling policies must be adaptive to local resource availability. For CDEGs, this means that the same deployment event may have different counterfactual cost effects depending on the cluster region, node pool, hardware type, or edge location. The graph model must therefore include contextual attributes rather than treating all resource units as interchangeable [19].

Explainability research provides concepts for making model outputs understandable to users. LIME introduced local surrogate explanations that approximate a black-box model around a specific prediction. While Kubernetes cost attribution is not necessarily a classification problem, the local explanation principle is useful: operators often need an explanation for a specific cost anomaly, deployment revision, or namespace rather than a global theory of the whole cluster. CDEGs adapt this idea by producing local counterfactual explanations around a selected event window [6].

SHAP formalized additive feature attribution using Shapley values, offering a principled way to distribute prediction outcomes among features. In Kubernetes cost attribution, simple feature attributions can be misleading because features such as CPU request, replica count, and node type are not independent. However, the additive explanation idea remains valuable if adapted to graph paths and causal interventions. CDEGs use attribution over event paths rather than over independent tabular features, thereby preserving dependency structure while still producing decomposable explanations [11]. Counterfactual explanations are particularly relevant because they answer "what would need to change" rather than merely "which feature was important." In operational settings, this distinction is essential. A statement that CPU request contributed 40% of a cost increase is less actionable than a statement that reducing a request from an inflated value to a safe percentile would have avoided one node scale-out event without violating memory constraints. Counterfactual explanation theory therefore provides a conceptual bridge between interpretability and operational recourse [14].

Causal inference supplies the formal language needed to distinguish correlation from intervention. Pearl's structural causal model framework defines counterfactuals using interventions, causal graphs, and structural equations. This is important for Kubernetes because many cost variables are correlated with traffic, deployment frequency, and time of day, but not all are causes of cost. For example, a high error rate may correlate with cost during an incident, but the causal driver may be retry amplification after a bad rollout. CDEGs treat causal assumptions as explicit model components rather than hidden dashboard logic [2]. Modern causal inference also emphasizes invariance, mechanisms, and assumptions about how interventions modify a system. This matters because Kubernetes is controlled by

reconciliation loops: changing a deployment specification changes the desired state, after which controllers and schedulers act through stable mechanisms. Such mechanisms make counterfactual reasoning plausible, but only when assumptions are carefully stated. CDEGs therefore encode structural equations for resource demand, scheduling feasibility, autoscaler response, and cost allocation, while allowing uncertainty when mechanisms are only partially observed [23].

Graph-based provenance is another relevant foundation. The W3C PROV model standardizes concepts such as entities, activities, agents, generation, usage, and attribution. Kubernetes operational histories naturally resemble provenance graphs: a deployment revision generates pods, pods use nodes, controllers act as agents, and billing records are derived from infrastructure usage. CDEGs extend provenance from retrospective traceability to counterfactual analysis by adding typed causal mechanisms and intervention semantics [9]. Low-overhead observability is necessary because Kubernetes cost attribution cannot rely only on billing data, which is delayed and coarse-grained. eBPF and BPF-based tooling offer kernel-level visibility into process, network, and system behavior with relatively low overhead. Such observability is useful for connecting container-level activity to resource consumption and service communication, especially when application instrumentation is incomplete. A CDEG implementation can use eBPF-derived signals to enrich graph edges for network dependency, syscall intensity, and runtime contention [15].

The origins of packet filtering and kernel-level observability are also relevant because they show how low-level measurement can be made efficient enough for production environments. The BSD Packet Filter introduced a high-performance architecture for user-level packet capture, and its design lineage informs later BPF and eBPF systems. For CDEGs, the lesson is methodological: explainability depends on measurement, but measurement must not perturb the system so heavily that counterfactual cost estimates become invalid [20]. Finally, distributed systems observability literature emphasizes that logs, metrics, and traces answer different diagnostic questions. Metrics indicate magnitude, logs preserve discrete events, and traces reveal causal or temporal paths across services. Kubernetes cost attribution requires all three: metrics for resource usage, logs for deployment and controller events, and traces for service dependencies. CDEGs integrate these signals into a unified graph so that cost explanations are grounded in operational evidence rather than isolated billing aggregates [25].

## 3. Problem Statement

Let a Kubernetes platform operate over a time horizon (T), during which workloads generate deployment events, autoscaler actions, scheduling decisions, telemetry observations, and billing records. The observed total cost (C_T) is not directly generated by a single workload but by the interaction of workload declarations, node provisioning, resource usage, and allocation rules. The central problem is to attribute portions of (C_T) to deployment events and configuration decisions in a way that is explainable, causally meaningful, and actionable for optimization. This requires distinguishing between cost that is necessary to satisfy service objectives and cost that is avoidable under feasible alternative configurations.

The first challenge is shared infrastructure. A node may host pods from several namespaces, and its cost may be billed at the node, instance, cluster, or cloud account level. A naive proportional allocation can support financial reporting, but it cannot fully explain whether a workload caused the node to exist, prevented scale-down, fragmented available memory, induced network egress, or consumed only residual capacity. The attribution problem is therefore not merely one of dividing a bill; it is one of estimating marginal and path-specific effects of workload events on infrastructure state.

The second challenge is temporal mediation. Deployment changes often have delayed effects. A rollout can create transient replica overlap, trigger readiness failures, induce retries, and cause autoscalers to react several minutes later. A rightsizing recommendation may reduce requests immediately but reduce cost only after node consolidation occurs. Consequently, attribution must model event sequences and controller delays rather than treating cost as a contemporaneous function of resource usage.

The third challenge is counterfactual feasibility. Not all lower-cost alternatives are valid. A counterfactual deployment that reduces replicas below safe capacity, removes required affinity, or violates memory safety is not operationally meaningful. Feasible counterfactuals must respect constraints such as service-level objectives, pod disruption budgets, quality-of-service classes, topology spread, regulatory isolation, and team ownership. Reliability engineering practice has long emphasized that availability and operational safety must be treated as first-class design objectives rather than afterthoughts [12].

The fourth challenge is explanation quality. Engineering teams need explanations that map cost to controllable levers: resource requests, limits, replicas, autoscaling policies, image versions, sidecars, scheduling constraints, and dependency behavior. Financial stakeholders need explanations that map cost to services, products, and teams. Platform teams need explanations that reveal cluster-level inefficiencies such as fragmentation and node pool mismatch. A useful framework must therefore support multiple explanation granularities without producing inconsistent narratives.

This paper formulates the research problem as follows: given a temporal record of Kubernetes deployment events, workload telemetry, dependency traces, scheduling outcomes, and cost records, construct a causally annotated event graph that estimates the cost contribution of each deployment event and generates feasible counterfactual alternatives that improve resource efficiency while preserving operational constraints.

## 4. Proposed Framework / Methodology

A Counterfactual Deployment Event Graph is defined as (G_T=(V,E,,A,X,Y)), where (V) is a set of nodes, (E) is a set of typed directed edges, () assigns timestamps or intervals, (A) denotes admissible interventions, (X) denotes observed attributes, and (Y) denotes outcomes such as cost, utilization, latency, error rate, and scale events. Nodes include Kubernetes objects, infrastructure resources, telemetry windows, billing records, and human or automated agents. Edges encode operational relations such as ownership, scheduling, scaling, deployment derivation, communication, constraint satisfaction, and cost allocation.

The graph is constructed from five event classes. Deployment events include changes to Deployments, StatefulSets, DaemonSets, Jobs, Helm releases, container images, ConfigMaps, and resource specifications. Control-plane events include scheduler decisions, Horizontal Pod Autoscaler actions, Vertical Pod Autoscaler recommendations, Cluster Autoscaler node changes, evictions, and admission-controller mutations. Runtime events include CPU, memory, disk, network, restart, and latency metrics. Dependency events include traces, service calls, retries, and ingress patterns. Financial events include node-hour cost, storage cost, network egress, managed service charges, and allocation labels. Each event is normalized to a graph schema that preserves time, owner, namespace, service, revision, and confidence.

The causal layer interprets selected graph edges as mechanisms. For example, a Deployment revision creates a ReplicaSet; a ReplicaSet creates Pods; Pods request resources; the scheduler places Pods on Nodes; unschedulable Pods may trigger node scale-out; running Pods consume resources; resource consumption affects latency and error rates; node uptime contributes to cost. These mechanisms are encoded as structural equations with uncertainty terms. The purpose is not to claim perfect causal identification, but to make assumptions inspectable and testable. Causal reasoning is valuable only when its assumptions are visible enough to be challenged by operators. CDEG attribution begins by computing the observed cost allocation (C^{obs}_{i,t}) for workload (i) over interval (t). This allocation may combine request-weighted, usage-weighted, and marginal-node methods. Request-weighted allocation assigns shared node cost according to declared CPU and memory requests. Usage-weighted allocation assigns cost according to measured utilization. Marginal-node allocation estimates whether a workload's presence or request caused a node to be provisioned or prevented from being removed. The framework does not require a single allocation method; it records the method as part of the explanation and compares attribution stability across methods.

Counterfactual attribution then evaluates interventions of the form (do(a a')), where (a) is a deployment action or configuration value and (a') is a feasible alternative. Examples include reducing CPU request to a safe percentile, changing a memory limit, modifying an HPA target, selecting a different node pool, disabling an unused sidecar, changing topology spread, or staggering rollout surge. The counterfactual graph (G_T^{a'}) is generated by replacing the intervened attribute and propagating effects through scheduling, scaling, and cost equations. The attributable cost effect is (C = C^{obs} - C^{cf}), where (C^{cf}) is the estimated cost under the alternative. To avoid unsafe recommendations, CDEGs define feasibility constraints. A counterfactual is feasible only if it satisfies resource safety, scheduling feasibility, dependency stability, service-level objectives, and governance rules. Resource safety can be estimated from historical percentiles and burst margins. Scheduling feasibility can be checked through scheduler simulation or bin-packing approximation. Dependency stability can be evaluated by tracing whether upstream or downstream services would experience increased queueing or retries. Governance rules can exclude regulated workloads, stateful systems, or critical production services from automatic action.

The explanation layer converts counterfactual estimates into human-readable narratives. A path-specific explanation might state that a new Deployment revision increased memory request, which reduced node packing density, which caused the Cluster Autoscaler to add a node, which produced additional node-hour cost. Another explanation might show that a traffic increase caused legitimate HPA scale-out and therefore should not be treated as waste. This distinction is central: CDEGs should not stigmatize necessary elasticity, but should expose avoidable inefficiency. The framework also supports attribution confidence. Confidence is reduced when telemetry is missing, when traces are sampled too sparsely, when scheduler state is incomplete, when workloads have non-stationary demand, or when counterfactual interventions extrapolate beyond observed regimes. Confidence scores prevent overinterpretation and allow explanations to be ranked by evidential strength. This is especially important when applying deep learning or sequence models to operational data, since model sophistication alone does not guarantee interpretability or causal validity [8].

Optimization is performed as a constrained search over feasible interventions. The objective function can be written as minimizing expected cost plus penalties for SLO risk, operational disruption, explanation uncertainty, and policy violation. Candidate interventions are ranked by net benefit, confidence, reversibility, and blast radius. Low-risk interventions can be recommended automatically, while high-impact interventions require human approval. The output is not merely a cost-saving number but an auditable optimization proposal with graph evidence.

The methodology therefore contains six stages: event ingestion, graph normalization, causal mechanism annotation, observed cost allocation, counterfactual simulation, and explanation-driven optimization. Each stage is modular so that organizations can adopt CDEGs incrementally. A platform team may begin with request-based cost attribution and deployment event lineage, then

add dependency traces, scheduler simulation, and causal confidence as instrumentation matures.

## 5. System Architecture or Conceptual Model

A reference CDEG architecture consists of seven layers. The first layer is the event acquisition layer, which collects Kubernetes API events, audit logs, controller events, resource specifications, node states, autoscaler actions, and CI/CD deployment metadata. This layer must preserve object identity across revisions; otherwise, cost attribution cannot distinguish between a long-running service and a new rollout. It also records admission mutations and policy decisions, since these can change the effective configuration that reaches the scheduler.

The second layer is the telemetry layer. It ingests metrics from cluster monitoring systems, traces from service instrumentation, logs from workloads and controllers, and kernel-level signals where available. CPU and memory usage are not sufficient by themselves because many cost effects are mediated by network egress, storage I/O, retries, and latency. The telemetry layer therefore maps resource consumption to service behavior, not only to container identity.

The third layer is the billing and pricing layer. It imports node price, instance family, region, storage class, network egress rate, reserved or committed-use discounts, spot or preemptible pricing, and managed service charges. The architecture separates raw cloud bills from allocation rules because different stakeholders may require different allocation semantics. For example, finance may prefer stable request-based allocation, while optimization may require marginal-node analysis.

The fourth layer is the graph construction layer. It materializes a temporal property graph in which Kubernetes objects, events, metrics, and cost records become typed vertices and edges. The graph store supports interval queries, path traversal, lineage reconstruction, and neighborhood extraction around anomalies. The use of provenance-style modeling ensures that explanations can be audited from raw events to derived cost claims.

The fifth layer is the causal and counterfactual engine. This component maintains structural equations for scheduling, scaling, utilization, and cost allocation. It can call a scheduler simulator, a bin-packing estimator, an autoscaler model, or a workload forecast model. It also tracks uncertainty and validates whether proposed interventions remain inside observed operating envelopes. The engine's role is not to produce a single unquestionable truth, but to generate transparent estimates with assumptions that can be inspected.

The sixth layer is the explanation and recommendation layer. It translates graph paths and counterfactual deltas into stakeholder-specific explanations. Engineers may see deployment-level root causes, platform operators may see cluster fragmentation and node-pool inefficiency, and finance teams may see service-level cost drivers. Explanations include the observed path, the counterfactual alternative, estimated cost difference, SLO risk, confidence, and reversibility.

The seventh layer is the governance and actuation layer. It integrates with policy engines, GitOps workflows, approval systems, and change-management processes. CDEGs should not directly mutate production resources without safeguards. Instead, they generate pull requests, policy recommendations, or staged experiments. Automatic actuation may be appropriate only for low-risk changes such as non-production rightsizing or scheduled downscaling. This design aligns with the principle that production reliability depends on disciplined operational control as well as automation [12].

The conceptual model supports both retrospective and prospective use. Retrospectively, it explains why a cost anomaly occurred. Prospectively, it evaluates proposed deployment changes before they are merged. For example, a pull request that increases memory requests can be evaluated against historical workload demand and scheduler capacity to estimate whether it would trigger node scale-out. This enables cost-aware review without reducing Kubernetes to a static budgeting tool.

Security and privacy must also be considered. Deployment graphs can reveal service topology, traffic patterns, team ownership, and potentially sensitive workload metadata. The architecture should enforce role-based access, namespace scoping, data minimization, and retention limits. Explanations should disclose enough evidence to support action while avoiding unnecessary exposure of unrelated workloads in shared clusters.

## 6. Evaluation Criteria / Performance Metrics

The first evaluation criterion is attribution fidelity. Fidelity measures whether the framework assigns cost to the events and workloads that actually influenced infrastructure expenditure. Since ground truth is difficult to obtain in production, fidelity can be evaluated using controlled experiments, replayed traces, synthetic deployments, and known interventions. A useful test is to inject a known resource-request change, observe the resulting scheduling and cost behavior, and verify whether the CDEG identifies the correct event path.

The second criterion is counterfactual validity. A counterfactual estimate is valid when it represents a plausible alternative system trajectory under the specified intervention. Validity can be assessed by comparing predicted effects with canary experiments, shadow scheduling, or historical natural experiments. If the framework predicts that reducing CPU requests would avoid node scale-out, a controlled non-production replay should show similar consolidation behavior. Counterfactual validity is more important than visual explanation quality because an attractive but invalid explanation can lead to unsafe optimization.

The third criterion is optimization effectiveness. Metrics include cost reduction, request-to-usage slack reduction, node utilization improvement, avoided node-hours, reduced fragmentation, and decreased cost per request. These metrics should be interpreted with SLO-aware penalties. A system that reduces cost while increasing tail latency, restarts, or error rates is not efficient in the operational sense. Evaluation must therefore report cost metrics alongside p95 or p99 latency, error budget burn, restart rate, eviction rate, and throttling.

The fourth criterion is explanation usefulness. Usefulness can be measured through operator studies, action acceptance rate, time to diagnose cost anomalies, recommendation adoption, and disagreement between engineering and finance stakeholders. Explanation quality includes completeness, conciseness, actionability, and confidence calibration. A useful explanation identifies a controllable lever and provides enough evidence for a service owner to act.

The fifth criterion is computational and operational overhead. CDEGs must not impose excessive API-server load, telemetry cost, storage overhead, or runtime latency. Overhead metrics include event ingestion latency, graph update throughput, query latency, storage growth per workload, and instrumentation overhead. Low-overhead measurement is especially important when using kernel-level or trace-level telemetry in production environments.

The sixth criterion is robustness. Kubernetes systems are noisy: metrics can be missing, traces can be sampled, pod names are ephemeral, and workloads can be non-stationary. Robustness should be evaluated under incomplete telemetry, delayed billing data, workload bursts, controller failures, and label inconsistency. Robust CDEG implementations should degrade gracefully by lowering confidence rather than producing unsupported certainty.

The seventh criterion is fairness and accountability. Cost attribution methods can shift expenses between teams depending on whether allocation is request-based, usage-based, or marginal. Evaluation should therefore report sensitivity to allocation policy and identify cases where shared infrastructure costs cannot be uniquely assigned. This is not a weakness; it is a necessary feature for trustworthy financial governance.

## 7. Results and Discussion / Analytical Discussion

Because this paper proposes a conceptual framework rather than reporting a completed empirical deployment, the results are analytical. The expected contribution of CDEGs is not a universal percentage reduction in cost, but a structured improvement in how cost is explained, disputed, and optimized. The framework's value is clearest in cases where conventional metrics are ambiguous.

Consider a workload whose CPU request is set far above observed usage. A dashboard may show low utilization and high allocated cost, but it may not reveal whether the inflated request actually caused additional infrastructure expenditure. If the cluster had sufficient unused capacity, the request may have affected internal allocation but not the cloud bill. If the request prevented other pods from fitting on existing nodes, it may have caused node scale-out. CDEGs distinguish these cases by tracing the path from request to scheduling feasibility to node provisioning to cost. The explanation is therefore marginal rather than merely proportional.

A second case involves autoscaling. Suppose a deployment increases from five to twenty replicas during a traffic surge. A simplistic cost report may identify the service as expensive, while an optimization tool may recommend lowering replicas. CDEGs can classify the cost as legitimate elasticity if traces and ingress metrics show increased demand and latency protection. The counterfactual of not scaling would then violate SLO constraints, so the framework would avoid labeling the scale-out as waste. This distinction supports reliability-aware FinOps rather than indiscriminate cost cutting.

A third case involves rollout surge. Kubernetes rolling updates can temporarily run old and new replicas at the same time. If a deployment uses an aggressive surge setting during peak traffic, it may briefly trigger node scale-out and leave a node underutilized until scale-down delay expires. CDEGs can attribute the cost increase to the rollout configuration and timing rather than to steady-state service demand. A feasible counterfactual might recommend reducing surge, changing rollout time, or pre-scaling down non-critical workloads.

A fourth case involves service dependency. A downstream service may appear costly because it scaled out, but the root cause may be an upstream deployment that increased retry volume after a latency regression. In a tabular allocation model, the downstream service absorbs the cost. In a CDEG, trace edges and temporal event paths can show that the downstream scale-out was induced by upstream behavior. This supports more accurate engineering accountability and prevents optimization from targeting the symptom rather than the cause.

A fifth case involves node-pool mismatch. A workload may request a small amount of CPU but require a specialized node pool because of affinity, GPU toleration, compliance isolation, or architecture constraints. Its proportional resource usage may look small, yet its marginal cost may be large if it keeps a specialized node alive. CDEGs can represent node-pool constraints explicitly and estimate whether relaxing a constraint, moving a workload, or consolidating similar workloads would reduce cost. The analytical advantage of CDEGs lies in their ability to separate allocation, causation, and recourse. Allocation answers "who is charged?" Causation answers "what produced the cost?" Recourse answers "what could be changed?" These questions are often conflated in existing cost dashboards. By separating them, the framework allows a finance team to preserve a stable chargeback rule while allowing platform teams to use marginal counterfactuals for

optimization. The framework also improves explainability by grounding explanations in event paths. A statement such as "namespace A wasted 30% CPU" is not an explanation; it is an aggregate metric. A CDEG explanation identifies the deployment revision, the resource request change, the scheduler effect, the autoscaler reaction, the node-hour cost, and the feasible alternative. This path-based explanation is more aligned with how engineers debug production systems. The cost of this richer explanation is model complexity. CDEGs require clean labels, event retention, telemetry integration, and causal assumptions. Organizations with weak ownership metadata or inconsistent deployment practices may initially obtain incomplete graphs. However, this limitation can itself be useful: missing graph edges expose governance gaps that make cost accountability difficult. In this sense, CDEGs serve both as an optimization tool and as a maturity diagnostic for cloud operations. The framework also provides a bridge between predictive modeling and operational reasoning. Machine-learning models can predict cost anomalies or resource demand, but their outputs require interpretation. CDEGs can use predictions as inputs while constraining recommendations through graph-based causal structure. This reduces the risk of opaque optimization policies and aligns with broader explainable AI principles that emphasize transparency, stakeholder relevance, and trust [18].

## 8. Practical Implications

For engineering teams, CDEGs transform cost optimization from a periodic cleanup exercise into a deployment-aware feedback loop. Developers can see how configuration changes affect cost paths, not only whether a namespace became more expensive. This supports earlier intervention during code review, release planning, and capacity design. For platform teams, the framework provides a way to identify systemic inefficiencies such as fragmentation, incompatible node pools, excessive rollout surge, and autoscaler misconfiguration. Rather than issuing generic rightsizing recommendations, platform teams can prioritize interventions by marginal cost effect, confidence, and operational risk. This helps avoid optimization fatigue, where teams ignore dashboards because recommendations are too broad or unsafe. For finance and FinOps teams, CDEGs provide a defensible explanation layer beneath showback and chargeback reports. A service owner can dispute an allocation by examining the graph evidence and counterfactual assumptions. This can reduce unproductive debates over billing methodology and shift discussion toward actionable changes. It also aligns cloud financial management with the collaborative practices emphasized in FinOps literature [4]. For reliability teams, CDEGs preserve the distinction between waste and resilience. Redundancy, headroom, and autoscaling are not inherently inefficient; they are often necessary for reliability. By embedding SLO constraints into counterfactual feasibility checks, the framework prevents optimization from undermining error budgets. This is essential because cost reduction that increases incident risk is economically misleading. For governance, CDEGs support policy-as-code integration. Organizations can define which workloads are eligible for automatic rightsizing, which require approval, and which must be excluded because of compliance or criticality. Explanations can be attached to pull requests or change tickets, creating an audit trail from cost anomaly to recommendation to action.

## 9. Limitations

CDEGs depend on the quality and completeness of operational telemetry. Missing labels, incomplete traces, delayed billing data, and short retention windows can reduce attribution confidence. In many organizations, Kubernetes objects are not consistently mapped to products, teams, or business services, making financial attribution difficult even when technical telemetry is available. Causal identification remains challenging. Kubernetes systems are dynamic, and many events occur simultaneously. A cost spike may coincide with a deployment, traffic surge, incident, and node pool change. CDEGs can make assumptions explicit and estimate counterfactuals, but they cannot eliminate uncertainty when interventions are not observed or when workload behavior changes abruptly.

Counterfactual simulation may also be inaccurate when scheduler behavior, autoscaler policies, or cloud-provider pricing details are not fully modeled. For example, a predicted node consolidation may not occur because of pod disruption budgets, daemonset overhead, topology spread constraints, or provider-specific scale-down delays. Implementations must therefore validate recommendations through staged experiments. The framework may introduce storage and computation overhead. Temporal graphs can grow quickly in large clusters, especially when traces and metrics are attached at fine granularity. Practical implementations will need summarization, sampling, retention policies, and hierarchical graph representations to remain scalable. Finally, explanation does not guarantee action. Teams may lack time, ownership clarity, or incentives to adopt recommendations. CDEGs can improve the technical quality of cost explanations, but organizational alignment remains necessary for sustained efficiency.

## 10. Future Research Directions

Future research should develop benchmark datasets for Kubernetes cost attribution. Such datasets should include deployment histories, scheduler events, workload metrics, traces, billing records, and known interventions. Without shared benchmarks, it will be difficult to compare attribution fidelity or counterfactual validity across methods.

A second direction is causal discovery for deployment graphs. While this paper assumes that many Kubernetes mechanisms are known, real systems include undocumented dependencies, hidden retries, and external managed services. Combining domain knowledge with causal discovery could improve graph completeness while preserving interpretability. Future work should examine how invariance-based causal methods can operate under Kubernetes workload drift [23].

A third direction is integration with reinforcement learning and safe control. CDEGs could provide explanation

constraints for automated optimizers, ensuring that policies recommend actions with interpretable causal paths and bounded risk. This would be especially valuable for edge clusters and heterogeneous infrastructure, where manual tuning is difficult and workload conditions change rapidly [19].

A fourth direction is explanation personalization. Engineers, finance analysts, executives, and platform operators need different views of the same cost event. Future work should study how to generate explanations at different abstraction levels while preserving consistency between technical and financial narratives.

A fifth direction is model comparison for operational prediction. Kubernetes cost optimization may use regression models, sequence models, graph neural networks, or hybrid causal models. Prior work comparing classical machine-learning approaches for fault prediction suggests that model selection should be empirical rather than assumed [13]. Similar comparative studies are needed for resource forecasting, anomaly explanation, and counterfactual cost estimation.

A sixth direction is multi-objective sustainability. Cost is often correlated with energy use and carbon intensity, but not perfectly. Future CDEGs could include carbon-aware scheduling, region-specific emissions, and sustainability constraints, enabling explanations such as "this deployment increased cost and carbon because it forced scale-out in a high-intensity region."

## 11. Conclusion

This paper proposed Counterfactual Deployment Event Graphs as a framework for explainable cost attribution and resource efficiency optimization in Kubernetes workloads. The central argument is that Kubernetes cost cannot be explained adequately through static allocation alone because expenditure emerges from temporal interactions among deployment events, scheduler decisions, autoscaling loops, workload dependencies, and shared infrastructure. CDEGs address this challenge by modeling operational history as a causally annotated temporal graph and estimating feasible counterfactual alternatives.

The framework contributes a structured methodology for event ingestion, graph construction, causal mechanism annotation, observed cost allocation, counterfactual simulation, and explanation-driven optimization. It supports path-specific explanations, distinguishes necessary elasticity from avoidable waste, and aligns engineering action with FinOps accountability. Although empirical validation remains future work, the analytical discussion shows that CDEGs provide a rigorous foundation for moving Kubernetes cost intelligence from descriptive reporting toward causal, auditable, and actionable optimization. As Kubernetes platforms continue to scale across cloud, edge, and multi-tenant environments, explainable counterfactual cost attribution will become increasingly important for balancing reliability, efficiency, and financial governance.

## References


[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70–93, Jan. 2016, doi: 10.1145/2898442.2898444.

[2] J. Pearl, *Causality: Models, Reasoning, and Inference*, 2nd ed. Cambridge, U.K.: Cambridge University Press, 2009, doi: 10.1017/CBO9780511803161.

[3] S. K. Gunda, "Analyzing Machine Learning Techniques for Software Defect Prediction: A Comprehensive Performance Comparison," *2024 Asian Conference on Intelligent Technologies (ACOIT)*, KOLAR, India, 2024, pp. 1–5, https://doi.org/10.1109/ACOIT62457.2024.10939610.

[4] A. F. Baarzi and G. Kesidis, "SHOWAR: Right-Sizing and Efficient Scheduling of Microservices," in *Proc. ACM Symposium on Cloud Computing (SoCC '21)*, Seattle, WA, USA, 2021, pp. 427–441, doi: 10.1145/3472883.3486999.

[5] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th European Conference on Computer Systems (EuroSys '15)*, Bordeaux, France, 2015, pp. 1–17, doi: 10.1145/2741948.2741964.

[6] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?': Explaining the Predictions of Any Classifier," in *Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, San Francisco, CA, USA, 2016, pp. 1135–1144, doi: 10.1145/2939672.2939778.

[7] G. Rattihalli, M. Govindaraju, H. Lu, and D. Tiwari, "Exploring Potential for Non-Disruptive Vertical Auto Scaling and Resource Estimation in Kubernetes," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, Milan, Italy, 2019, pp. 33–40, doi: 10.1109/CLOUD.2019.00018.

[8] S. K. Gunda, "A Deep Dive into Software Fault Prediction: Evaluating CNN and RNN Models," *2024 International Conference on Electronic Systems and Intelligent Computing (ICESIC)*, Chennai, India, 2024, pp. 224–228, https://doi.org/10.1109/ICESIC61777.2024.10846549.

[9] L. Moreau, P. Missier, K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes, "PROV-DM: The PROV Data Model," W3C Recommendation, Apr. 30, 2013. [Online]. Available: https://www.w3.org/TR/prov-dm/

[10] Z. Zhong and R. Buyya, "A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources," *ACM Transactions on Internet Technology*, vol. 20, no. 2, Article 15, pp. 1–24, Apr. 2020, doi: 10.1145/3378447.

[11] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, Long Beach, CA, USA, 2017, pp. 4765–4774.

[12] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, "Sinan: ML-Based and QoS-Aware Resource Management for Cloud Microservices," in *Proc. 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*, Virtual, USA, 2021, pp. 167–181, doi: 10.1145/3445814.3446693.

[13] S. K. Gunda, "Fault Prediction Unveiled: Analyzing the Effectiveness of Random Forest, Logistic Regression, and KNeighbors," *2024 2nd International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)*, Erode, India, 2024, pp. 107–113, https://doi.org/10.1109/ICSSAS64001.2024.10760620.

[14] S. Wachter, B. Mittelstadt, and C. Russell, "Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR," *Harvard Journal of Law & Technology*, vol. 31, no. 2, pp. 841–887, Spring 2018, doi: 10.2139/ssrn.3063289.

[15] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, Scalable Schedulers for Large Compute Clusters," in *Proc. 8th ACM European Conference on Computer Systems (EuroSys '13)*, Prague, Czech Republic, 2013, pp. 351–364, doi: 10.1145/2465351.2465386.

[16] K. Rzadca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand, and J. Wilkes, "Autopilot: Workload Autoscaling at Google," in *Proc. Fifteenth European Conference on Computer Systems (EuroSys '20)*, Heraklion, Greece, 2020, pp. 1–16, doi: 10.1145/3342195.3387524.

[17] B. Burns, J. Beda, K. Hightower, and L. Evenson, *Kubernetes: Up and Running: Dive into the Future of Infrastructure*, 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2022.

[18] A. Adadi and M. Berrada, "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52138–52160, 2018, doi: 10.1109/ACCESS.2018.2870052.

[19] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine Learning-Based Scaling Management for Kubernetes Edge Clusters," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 958–972, Mar. 2021, doi: 10.1109/TNSM.2021.3052837.

[20] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," in *Proc. USENIX Winter 1993 Conference*, San Diego, CA, USA, Jan. 1993, USENIX Association. [Online]. Available: https://www.usenix.org/conference/usenix-winter-1993-conference/bsd-packet-filter-new-architecture-user-level-packet.