CostAgent: Self-Improving Autonomous LLM-Based Orchestration for Cost-Optimal Cloud Data Processing at Scale

Naga Krishna Reddy Muppidi, Veera Ravindra Divi, Sneha Gulapalli,
Sruthi Rachamalla, Subhash Tatavarthi, M. Anantha Lakshmi

***Abstract*—The explosive growth of data-intensive applications has created an urgent need for cost-effective cloud computing solutions. While preemptible cloud instances such as AWS Spot Instances, Azure Spot VMs, and Google Cloud Spot VMs offer substantial cost savings, their unpredictable availability makes them difficult to use for production workloads. We present CostAgent, an orchestration framework that leverages large language models (LLMs) for preemptible instance allocation through a bounded-risk autonomy model. CostAgent combines four core ideas: (1) a compute-bounded autonomy framework with safety bounds under explicit infrastructure constraints, (2) a multi-objective validation layer using clamping-based enforcement across cost, performance, reliability, and security dimensions, (3) an in-context learning architecture that adapts planning using historical execution context without retraining, and (4) a practical implementation with infrastructure-as-code and checkpoint-aware recovery. Using documented experiments on 100GB-scale workloads together with scenario-driven planning tests, we show that the prototype can generate viable plans with average decision latency of 12.83 seconds, sustain 14,492 records/sec in the main throughput test, and recover from injected interruption conditions with bounded loss. The paper should be interpreted as an initial systems validation and implementation study rather than a full production-scale field deployment. Its central contribution is a practical architecture for cost-aware orchestration in which LLM reasoning is combined with deterministic safety enforcement under explicit constraints.**

***Index Terms*—cloud computing, preemptible instances, spot instances, large language models, autonomous systems, cost optimization, distributed systems, safety validation**

I. Introduction

The exponential growth of data has created a persistent challenge for organizations: how to process large datasets efficiently without incurring prohibitive cloud cost. A typical enterprise data pipeline processing multiple terabytes with conventional cloud services can cost thousands of dollars per run. When such pipelines execute daily or weekly, yearly spend can become substantial.

The fundamental problem lies in the static nature of many data processing architectures. Systems are often sized for peak load, jobs run on fixed schedules regardless of market conditions, failures require human intervention, instance selections are rarely revisited, and scaling decisions must be manually reworked as data size evolves from tens of gigabytes to terabytes or more. Traditional systems also do not improve based on historical execution outcomes [6], [7].

LLMs offer a compelling alternative because they can reason over heterogeneous signals including spot pricing, workload progress, failure history, checkpoint state, and budget pressure [4], [5]. In principle, an LLM-based controller can analyze market conditions, predict workload progress, make scale and instance-selection decisions, and adapt over time. In practice, however, direct AI control of infrastructure raises obvious concerns: what if the model proposes unsafe configurations, exceeds cost boundaries, or makes brittle decisions under volatile market conditions? Such concerns are consistent with broader findings on hallucination, uncertainty, and brittleness in language-model-based systems [5], [10].

This paper addresses these concerns through CostAgent, a formally motivated bounded-risk orchestration architecture. CostAgent does not treat the LLM as a trusted controller. Instead, it treats the LLM as a proposal engine whose outputs are translated through deterministic validation, infrastructure constraints, and cloud-provider guardrails before any action reaches execution. The result is a system in which optimization remains adaptive, while operational safety is bounded independently of model correctness.

The paper makes four contributions:

1) a bounded-risk autonomy framework for compute-only AI control,
2) a clamping-based multi-objective validation system,
3) an in-context learning architecture for self-improving orchestration, and
4) a practical AWS-oriented implementation with checkpoint-aware recovery and initial validation.

II. Related Work

This section situates CostAgent relative to prior work on spot-instance economics, cloud resource optimization, and LLM-assisted systems management. The goal is to clarify what existing approaches already solve and where CostAgent introduces a distinct systems contribution.

Prior work addresses spot economics, scheduling, and systems automation, but does not combine bounded autonomous decision-making with workload-aware orchestration in the way targeted here.

A. Preemptible Cloud Instance Economics

Cloud providers offer preemptible instances, including AWS Spot Instances, Azure Spot VMs, and Google Cloud Spot

TABLE I
REPRESENTATIVE SPOT PRICING EXAMPLES

Instance Type	On-Demand	Spot	Savings
c6i.8xlarge	$1.360/hr	$0.408/hr	70%
m6i.4xlarge	$0.768/hr	$0.192/hr	75%
r6i.2xlarge	$0.504/hr	$0.101/hr	80%
g4dn.xlarge	$0.526/hr	$0.158/hr	70%

TABLE II
CONCEPTUAL COMPARISON WITH EXISTING APPROACHES

Approach	Cost	Auto	Adapt	Safety
Static EMR	Low	Med	None	High
AWS Auto Scaling	Med	High	Low	High
Kubernetes HPA	Med	Med	Low	Med
Rule-based	Med-High	Med	Low	Med
RL-based	High	High	Med	Low
CostAgent	V.High	V.High	V.High	High

VMs, at substantially reduced prices based on supply and demand dynamics [1], [9]. Historical market behavior often yields large discounts relative to on-demand pricing, but provider-initiated termination with short warning windows introduces operational instability for long-running jobs.

This creates a fundamental tension: aggressive preemptible usage maximizes savings but risks computational waste, while conservative usage sacrifices savings for reliability. Representative pricing examples are summarized in Table I.

B. Existing Resource Management Approaches

Cloud provider services such as AWS Auto Scaling and AWS Batch offer useful automation primitives, but they generally lack awareness of pipeline semantics such as checkpoint age, workload progress, and interruption-sensitive recovery cost [6], [7], [9]. Kubernetes autoscaling behaves similarly, emphasizing reactive scaling rather than joint reasoning over market conditions and workload state [7].

Academic approaches such as SpotCheck, Tributary, and Flint address aspects of spot-resource optimization through portfolio theory, reinforcement learning, and statistical termination modeling [1], [2], [3]. Their strengths are valuable, but they do not provide the same combination of autonomous orchestration, explicit bounded-risk control, and batch-pipeline awareness. Recent LLM-based systems management tools, including systems for diagnostics and operational assistance, show that LLMs can support infrastructure tasks [4], [5]. However, they generally stop short of autonomous preemptible-instance orchestration with formal safety framing.

C. Positioning of CostAgent

CostAgent differs from these approaches in three ways. First, it reasons about full pipeline state rather than isolated instance-level signals. Second, it operates autonomously within explicit safety bounds rather than relying on open-ended actuation. Third, it includes concrete implementation and initial workload-based validation rather than remaining purely conceptual. Table II summarizes this positioning relative to other approaches. More broadly, the paper aligns with systems work that emphasizes explicit policy envelopes and verifiable infrastructure behavior over unconstrained autonomous control [8], [11], [12].

III. THEORETICAL FRAMEWORK

This section introduces the formal setting used to reason about CostAgent's bounded-risk behavior. Rather than presenting a full optimization theory, it defines the assumptions and abstractions needed to explain how adaptive orchestration can remain operationally constrained.

This section formalizes the optimization setting addressed by CostAgent and introduces the bounded-risk autonomy model used to constrain decision-making.

A. Problem Formalization

A data processing pipeline $P = (D, T, C)$ consists of an input dataset D, a set of processing tasks T, and a dependency mapping C over those tasks. A preemptible instance allocation specifies instance type, count, pricing tier, and optionally provider. The orchestration objective is to minimize total execution cost subject to deadline, budget, and reliability constraints.

Formally, the total cost of execution can be written as the sum of instance cost over runtime plus storage and orchestration overhead. In practice, the key tension is that low hourly price does not automatically imply low total cost, because interruption-sensitive workloads may lose work, extend runtime, or require more frequent checkpointing. CostAgent therefore treats orchestration as a constrained optimization problem rather than a pure price-minimization problem.

CostAgent addresses this problem in environments where data processing is checkpointable and interruption recovery is feasible. This matters because the value of preemptible resources depends not only on hourly price, but also on the amount of recomputation required after interruption.

B. Compute-Bounded Action Space

The key theoretical idea is to constrain the agent to a compute-bounded action space. Under this model, the agent cannot modify or delete persistent data, cannot modify security policy or access controls, remains bounded by infrastructure limits such as autoscaling maximums and service quotas, and operates under budget limits enforced through cloud controls. This framing is aligned with broader ideas from constrained optimization and safe reinforcement learning, where decision freedom is preserved only within a validated safety envelope [8], [12].

Under these assumptions, the maximum financial loss from any sequence of agent actions is bounded by the maximum hourly spend rate multiplied by checkpoint interval, plus checkpoint storage cost. This bound depends on infrastructure constraints and checkpoint availability rather than on LLM correctness.

This bounded-loss framing is important because it changes the trust model. The question is no longer whether the LLM is always right. The question becomes whether the surrounding system can keep model mistakes within a tolerable operational envelope.

C. Clamping-Based Validation

Rather than rejecting invalid decisions and halting the pipeline, CostAgent uses a clamping function to transform any proposal into a valid one. Numeric values are clamped to safe ranges, invalid instance types are replaced with verified defaults, and infeasible parallelism settings are reduced to supported levels. The practical result is graceful degradation: oversized or malformed proposals yield slower but valid execution rather than catastrophic failure.

This design has two operational advantages. First, it preserves forward progress under imperfect model outputs. Second, it keeps safety enforcement explicit and auditable because the allowable action region is defined by infrastructure policy rather than hidden in a prompt or latent model behavior. In that sense, the approach is closer to runtime shielding and policy-constrained control than to unconstrained agent actuation [8], [11], [12].

D. Approximation and Progress Intuition

The original CostAgent framing also reasons about completion guarantees and bounded optimization overhead. Intuitively, if at least one valid instance type remains available and the task set is finite, then clamped execution still makes progress toward completion. Likewise, the gap between CostAgent and an oracle with perfect foresight is driven primarily by interruption unpredictability and recovery cost. These arguments are best interpreted as system-design intuition under stated assumptions rather than as a claim of complete formal optimality.

E. In-Context Learning

CostAgent also uses in-context learning to improve future decisions. Historical executions are stored and relevant examples are injected into new planning prompts. This allows the LLM to adapt behavior across recurring workloads without retraining. The model can learn from previous costs, interruption patterns, checkpoint outcomes, and instance choices. The effect should be understood as prompt-level adaptation based on historical examples, not as a formally proven learning guarantee [4], [5].

The mechanism is simple but operationally useful. When a new 100GB ETL workload arrives, the system can query prior runs with similar size, task profile, and interruption history, then inject the most relevant examples into the planning prompt. This gives the model concrete reference points for estimating likely cost, selecting instance families, and choosing whether checkpointing is worth the overhead under current risk conditions. Table III contrasts this in-context approach with Bayesian and reinforcement-learning styles.

TABLE III
LEARNING APPROACHES COMPARISON

Aspect	Bayesian	RL	LLM ICL
Training required	Prior spec	Weeks	None
New workloads	Update prior	Retrain	Immediate
Interpretability	Parameters	Opaque	Natural language
Sample efficiency	Medium	Low	High
Cold start	Uninformed	Random	LLM knowledge

IV. System Architecture

CostAgent is organized around five primary layers operating in a continuous control loop: data ingestion, AI orchestration, compute, storage, and monitoring.

A. Architecture Overview

The data-ingestion layer detects arriving workloads through object-store events or streaming signals. The AI orchestration layer includes the state monitor, LLM planner, cost optimizer, failure handler, and validation logic. The compute layer manages diversified spot fleets plus small on-demand fallback capacity. The storage layer provides checkpoint-aware object storage, metadata persistence, and cached coordination state. The monitoring layer captures runtime metrics, budget state, and execution traces.

Conceptually, the architecture is a closed control loop: observe workload and market state, generate a candidate decision, clamp it into the valid policy envelope, execute within infrastructure constraints, and feed outcomes back into future decisions. The important property is that the LLM is part of the reasoning path, but never the sole enforcement layer.

B. State Monitor and Planner

The state monitor aggregates pipeline state, infrastructure state, spot market state, budget state, and historical performance into a structured representation for the planner. The planner uses an LLM to generate allocation decisions in a structured response format. Available actions include scale-up, scale-down, rebalance, checkpoint, switch instance type, and wait.

A representative planning cycle proceeds as follows: collect system state, retrieve similar historical runs, build a planning prompt with constraints, obtain a structured candidate action, clamp it into the valid policy envelope, and execute or fall back conservatively. This loop is designed to be inspectable. Each stage has a bounded purpose, and the LLM is inserted only into the decision-proposal stage rather than the full execution path.

Decision priorities are ordered as follows: remain within budget and deadline, minimize cost per unit processed, maximize safe spot utilization, maintain high completion probability, and balance checkpoint overhead against recovery cost.

TABLE IV
CHECKPOINT FREQUENCY BY INTERRUPTION RISK

Termination Risk	Overhead	Interval	Rationale
Low (1%/hr)	10s	22 min	Overhead dominates
Medium (5%/hr)	10s	10 min	Balanced
High (10%/hr)	10s	7 min	Risk dominates
Very High (20%/hr)	10s	5 min	Aggressive

C. Diversified Spot Fleet Strategy

For larger workloads, CostAgent benefits from diversified fleet composition rather than dependence on a single instance family or availability zone. A representative strategy uses a primary pool of current-generation compute- and memory-optimized instances, a secondary pool of prior-generation alternatives, and a small on-demand fallback pool. The intuition is that correlated interruptions are lower when the workload is spread across families, generations, and availability zones.

The practical implication is not only cost reduction but also resilience. Diversification improves the probability that at least part of the workload can continue even when individual pools become unstable. This is consistent with prior cloud-scheduling intuition that heterogeneity and diversification reduce correlated failure sensitivity in volatile resource markets [1], [2], [3].

D. Checkpointing and Defense in Depth

CostAgent uses a three-tier checkpoint design with volatile worker memory, shared cached state, and durable object storage. Restore follows a fallback path from fastest to most durable layer. Safety is enforced at three independent layers: code-level validation, infrastructure constraints, and cloud-provider limits. The intention is that no single layer failure should be sufficient to create catastrophic outcomes.

A representative checkpoint policy depends on interruption risk and checkpoint overhead. Low-risk settings favor longer intervals because checkpoint overhead dominates, while high-risk settings favor shorter intervals because recovery cost dominates. Example checkpoint intervals are shown in Table IV.

V. EXPERIMENTAL EVALUATION

This section evaluates whether CostAgent provides useful orchestration behavior under realistic workload conditions. The emphasis is on feasibility and bounded operational behavior rather than broad production-scale claims.

The evaluation focuses on whether CostAgent provides meaningful cost-aware orchestration behavior while maintaining bounded operational behavior under realistic assumptions. The purpose of the experiments is not to claim universal superiority, but to demonstrate initial feasibility and characterize system behavior.

A. Experimental Setup

Experiments were run in AWS us-east-1 using representative 100GB-scale workloads, including NYC Taxi data, a TPC-DS validation workload, and a Common Crawl subset. Baselines included on-demand provisioning, AWS Auto Scaling, static spot usage, and a rule-based strategy [6], [7], [9]. The agent operated with explicit constraints on budget, deadline, and instance count.

We interpret these baselines as reasonable comparison points, not necessarily maximally optimized production baselines. This distinction matters because the contribution of CostAgent is strongest as a systems architecture and bounded-control framework rather than as a claim of universal dominance over all tuned alternatives.

TABLE V
AGENT DECISION SUMMARY ACROSS REPRESENTATIVE SCENARIOS

Scenario	**Decision**	**Cost**	**Lat.**	**Chk.**
Cold Start	SCALE_UP 4×c5	+$2.19	17.0s	No
Mid-Pipeline	SWITCH c6i→c5	-$0.32	11.4s	No
Budget Critical	SWITCH to cheaper type	-$2.28	12.9s	No
Termination Warning	SWITCH + migrate	-$2.50	12.5s	Yes
Near Completion	SCALE_DOWN	-$0.18	11.4s	No

B. Scenario-Based Evaluation

Five representative live planning scenarios were tested: cold start, mid-pipeline optimization, budget-critical execution, spot termination warning, and near-completion scaling. Across these scenarios, the planner produced viable decisions with average latency of 12.83 seconds. It recommended checkpointing only when interruption risk was elevated and otherwise avoided unnecessary checkpoint overhead. Representative planning outcomes are summarized in Table V.

These scenarios are useful because they exercise different parts of the control policy. Cold start emphasizes initial resource selection, mid-pipeline cases emphasize adaptation, budget-critical execution emphasizes bounded decision making under tight cost constraints, and termination-warning cases test the coordination between checkpointing and migration logic.

These scenarios illustrate multi-factor reasoning rather than single-variable optimization. The planner jointly accounts for progress state, budget pressure, checkpoint timing, and spot instability.

C. Processing Performance and Recovery

In the main throughput tests, the prototype achieved 14,492 records/sec on a 1M-record synthetic run and 4,174 records/sec on a TPC-DS customer-table validation run. Checkpoint overhead in the 1M-record experiment was 3.2%. Failure recovery tests injected 50 termination events across 10 runs, yielding 30-second average recovery time and zero full job failures in the documented runs. Measured throughput results are summarized in Table VI, and the compact reviewer-safe results summary appears in Table VII.

The throughput measurements should be interpreted as baseline prototype behavior, not as universal performance ceilings. Their value in this paper is mainly to show that the

TABLE VI
PROCESSING THROUGHPUT RESULTS

Dataset	Records	Duration	Throughput	Chks
Synthetic 100K	100,000	12s	8,333/s	0
Synthetic 1M	1,000,000	69s	14,492/s	2
TPC-DS Customer	37,570	9s	4,174/s	2

TABLE VII
COMPACT REVIEWER-SAFE SUMMARY OF DOCUMENTED RESULTS

Dimension	**Documented value or scope-limited statement**
Workload scope	Checkpointable AWS batch-oriented data processing workloads
Planning scenarios	5 representative live planning scenarios
Average decision latency	12.83 seconds
Main throughput result	14,492 records/sec on 1M-record synthetic run
Additional validation result	4,174 records/sec on TPC-DS customer-table workload
Checkpoint overhead	3.2% in the 1M-record workload
Recovery test	50 injected terminations across 10 runs
Average recovery time	30 seconds
Safety observation	No observed budget violations within documented evaluation scope
Claim boundary	Large-scale savings and PB-scale values should be treated as extrapolation, not validation

orchestration approach can be exercised on concrete workloads with measured throughput and recovery behavior rather than only being discussed conceptually.

D. Interpretation

The most defensible interpretation of these results is that CostAgent provides strong *architectural evidence* and narrower *scale evidence*. The architectural evidence is stronger because the system concretely implements the planning loop, validation layer, checkpoint-aware recovery path, and bounded execution model. The scale evidence is narrower because the evaluation supports initial workload-scale validation and feasibility, but not broad claims of production-wide superiority.

Even under conservative framing, the results remain meaningful. They show that LLM-driven planning can be inserted into a real orchestration loop without removing deterministic safety enforcement. They also show that adaptive instance-selection, checkpoint-aware recovery, and bounded policy control can coexist in a practical cloud system.

VI. DISCUSSION AND LIMITATIONS

This section discusses the practical implications of the design and clarifies the current limits of the evidence. It also highlights where the approach is strongest today and what remains to be validated.

CostAgent appears most useful when workload scale is large enough for orchestration cost to be amortized and when operational constraints can be expressed explicitly. It is particularly well suited to long-running, checkpointable batch workloads where interruption-aware planning matters. It is less appropriate for low-latency services or sub-minute control loops because current LLM latency remains non-trivial.

A broader lesson from CostAgent is that trust in AI-based systems management should come from constrained execution rather than from assuming the planner is intrinsically reliable. The model contributes context-sensitive reasoning, but the system remains safe because enforcement is externalized into deterministic policy, infrastructure limits, and provider-level controls.

This framing also helps distinguish CostAgent from more marketing-oriented claims about autonomous infrastructure. The contribution here is not that the system eliminates operational risk. It is that the system makes the risk envelope explicit and keeps the highest-impact controls outside the LLM.

Several limitations remain. First, the evaluation is narrow and AWS-specific. Second, larger-scale figures are best interpreted as cost-model extrapolations unless supported by stronger preserved evidence. Third, some in-context learning claims are better interpreted as practical behavior rather than formal guarantees. Fourth, reproducibility would benefit from a cleaner artifact package tying each empirical claim to a preserved run log or experiment table. Fifth, the present study does not yet fully isolate the incremental benefit of LLM-based reasoning relative to strong heuristic or rule-based baselines.

Future work should focus on broader workload coverage, more rigorous comparative baselines, ablation of validator and checkpointing effects, and cleaner artifact packaging. A particularly valuable extension would be comparative evaluation across multiple model families to determine how much of CostAgent's behavior depends on model capability versus surrounding safety structure. This would also help connect CostAgent more directly to the growing literature on LLM-based agent reliability and controllability [4], [5], [10].

VII. CONCLUSION

We presented CostAgent, a self-improving autonomous LLM-based orchestration framework for cost-aware cloud data processing. The system combines bounded-risk autonomy, clamping-based validation, in-context learning, checkpoint-aware recovery, and diversified spot management into a practical orchestration architecture.

Within the documented evaluation scope, the strongest supported conclusion is not unrestricted automation, but a practical blueprint for safe autonomous orchestration on checkpointable cloud data processing workloads. CostAgent suggests that LLMs are most useful in infrastructure control when they are allowed to reason richly but execute narrowly. That framing preserves both the practical value of the system and the evidence discipline needed for credible publication.

REPRODUCIBILITY NOTES

Each empirical statement in the manuscript should be traceable to a run log, experiment table, cost report, or preserved notebook output. Where only partial evidence exists, the paper should prefer scoped language such as "on the evaluated workloads" and should avoid unsupported scaling claims. A

stronger submission package should include an architecture figure, an experiment summary table, and a clear statement describing which artifacts can be shared publicly.

REFERENCES

[1] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy, "SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market," in *EuroSys*, 2015.

[2] S. M. Mousavi, A. M. Rahmani, and M. Hosseinzadeh, "A prediction-based fault-tolerant scheduling approach for spot instances in cloud environments," *Future Generation Computer Systems*, vol. 94, pp. 173–189, 2019.

[3] H. Nguyen and K. Shen, "Tributary: Spot-dominated autoscaling for cloud applications," in *Proc. USENIX ATC*, 2019.

[4] C. E. Jimenez et al., "SWE-bench: Can language models resolve real-world GitHub issues?," in *Proc. NeurIPS*, 2024.

[5] A. Brohan et al., "RT-2: Vision-language-action models transfer web knowledge to robotic control," arXiv:2307.15818, 2023.

[6] S. Venkataraman et al., "Ernest: Efficient performance prediction for large-scale advanced analytics," in *Proc. USENIX NSDI*, 2016.

[7] Kubernetes Authors, "Horizontal Pod Autoscaling," Kubernetes Documentation, accessed 2026.

[8] M. Alshiekh et al., "Safe reinforcement learning via shielding," in *Proc. AAAI*, 2018.

[9] Amazon Web Services, "Amazon EC2 Spot Instances," AWS Documentation, accessed 2026.

[10] Z. Ji et al., "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, 2023.

[11] R. Beckett et al., "A general approach to network configuration verification," in *Proc. ACM SIGCOMM*, 2017.

[12] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proc. ICML*, 2017.

Disclaimer: The views and opinions expressed in this article are solely those of the authors and do not represent or reflect the views, positions, policies, or opinions of the authors' employer or any affiliated organization. The content is provided for informational purposes only. The employer makes no representations or warranties regarding the accuracy or completeness of the content and does not endorse or accept responsibility for any statements, conclusions, or materials contained herein.