FunPilot: Runtime Performance Diagnosis and Remediation for Serverless Applications with LLMs

Changyuan Lin
University of British Columbia

Yawen Wang
Google

Mohammad Shahrad
University of British Columbia

Abstract

In serverless computing, optimizing performance and cost is challenging due to the large configuration space in the presence of time-varying workloads and rapidly scaling infrastructure. Human-driven optimization is fundamentally limited in this setting: it is too slow for the short control timescales of serverless platforms, prohibitively expensive at scale, and unable to continuously explore high-dimensional configuration spaces in real time. Existing diagnostic agents are designed for long-running services and do not adapt well to the unique characteristics of serverless, such as short timescales, cost efficiency, and ephemeral runtimes.

We present `FunPilot`, a system that enables rapid LLM-assisted diagnosis and remediation for serverless applications. `FunPilot` uses an event-driven control loop to diagnose active symptoms, derive control knob updates, and validate remediation decisions while coordinating with the underlying autoscaler. `FunPilot` completes diagnosis in 6.04 s at a cost of $0.0082 per diagnosis on average, fitting within a single metrics collection interval while incurring modest cost relative to the applications it manages. Starting from the initial application configuration, `FunPilot` reduces average alert active time by 70.9% and resource consumption by 38.3%. Across real-world workloads and synthetic workloads with injected anomalies, firing alerts resolve in 80.57 s on average after the `FunPilot` diagnosis engine is triggered.

CCS Concepts

• **Computer systems organization** → **Cloud computing**; • **Software and its engineering** → **Software performance**; • **Computing methodologies** → **Intelligent agents**.

Keywords

Cloud Computing, Serverless Computing, Performance Engineering, Large Language Models

APSys '26, Bangkok, Thailand

ACM ISBN 979-8-4007-2912-6/26/09
https://doi.org/10.1145/3838177.3841720

ACM Reference Format:
Changyuan Lin, Yawen Wang, and Mohammad Shahrad. 2026. FunPilot: Runtime Performance Diagnosis and Remediation for Serverless Applications with LLMs. In *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '26), September 17–18, 2026, Bangkok, Thailand.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3838177.3841720

1 Introduction

Serverless computing has become a mainstream cloud paradigm, with major providers offering platforms like AWS Lambda [55] and Google Cloud Run [16], and open-source frameworks like Knative [38] being widely adopted. Serverless abstracts infrastructure management and provides pay-per-use billing and autoscaling from zero to thousands of instances. However, it does not eliminate the complexity of performance engineering. Instead, serverless turns such optimization into a runtime control problem over a large space of interdependent knobs under dynamic traffic, shifting resource demands, and applications with varying cold start latency and concurrency limits [3, 25, 28, 31, 64, 65].

Platforms expose the control surface through knobs for resource allocation, traffic management, and scaling policy: vCPUs, minimum replicas, concurrency, scaling targets, etc. [17, 34, 50]. The effects of these knobs and their interplay are often non-obvious. For example, lowering scaling targets may improve burst responsiveness but induce scaling oscillations that increase tail latency [41, 44]. As traffic and resource demands shift over time, serverless tuning is not a one-time task but a continuous operational challenge [3, 10].

Prior work addresses parts of this problem through offline profiling and optimization of platform-exposed knobs [3, 24, 45, 49, 56]. However, these typically rely on offline analysis of historical metrics and produce static recommendations for a limited set of knobs, such as CPU and memory allocation. They are not designed to diagnose live performance symptoms, identify application-specific issues, or revise control knobs as traffic and resource demands change at runtime.

More recently, large language model (LLM) agents have been applied to application and infrastructure operations, targeting root cause analysis (RCA) for databases, microservices, and cloud infrastructure [6, 12, 13, 20, 27, 33, 43, 48, 59, 66, 68]. These systems demonstrate strong diagnostic capabilities

for operational failures, such as unreachable ports, out-of-memory errors, and failure propagation, and some can diagnose and apply remediation at runtime. However, they typically target large, persistent, stateful workloads that can tolerate agent execution lasting tens to hundreds of seconds and costing tens of cents to several dollars per decision [5, 29, 59]. Such designs do not transfer well to serverless due to its unique characteristics. Specifically, serverless autoscalers act on short timescales (e.g., Knative Pod Autoscaler (KPA) evaluates scaling decisions every 2 s [36]). Agents with long execution time risk acting on stale observations and state. Also, cost efficiency is a primary driver of serverless adoption, which imposes strict cost constraints on agents. For example, a per-diagnosis agent cost of $0.498 (i.e., AWS DevOps Agent one-minute diagnosis cost [5]) is equivalent to running a 1 vCPU function for 4.82 hours under AWS Lambda pricing [54], an overhead that may outweigh the value of agent remediation outcome and hinders frequent use. Moreover, since serverless applications typically run in stateless, ephemeral sandboxes that scale to zero on multi-tenant infrastructure [1, 62], diagnostic methods that rely on continuous per-replica monitoring become inapplicable [21, 51].

To fill this gap, we present `FunPilot`, a system that runs alongside the serverless platform (i.e., Knative on Kubernetes), operates on the platform-exposed control knobs, and enables runtime performance diagnosis and remediation for serverless applications through LLM-assisted reasoning and validation. `FunPilot` is designed around four objectives derived from the serverless context: **rapid response within a single metrics collection interval** (e.g., 15–30 s [37, 42, 60]), **cost proportional to the workload** it manages, **complementing the platform autoscaler rather than replacing it**, and **alignment with the serverless execution model** (e.g., event-driven execution, scale-to-zero, revision lifecycles, and multi-tenant isolation).

To achieve low latency and cost while delivering high-quality changes, `FunPilot` pre-fetches and processes the diagnostic context (e.g., metrics and recent knob updates) before LLM invocation and adapts each diagnosis process along two dimensions: dynamic diagnostic depth and dynamic inference backend selection. When the preprocessed diagnostic context already supports a bounded, low-risk remediation (e.g., increasing capacity via tuning a single control knob for resource allocation or scaling target), `FunPilot` takes a *fast path* to apply it directly after guardrail checks. Otherwise, `FunPilot` falls back to the *slow path* that gathers additional evidence and performs additional reasoning to propose and compare alternative remediation plans before applying any update. `FunPilot` also uses tiered LLM backend selection at runtime. Early LLM calls use high-throughput backends to reduce inference latency, while later calls use lower-cost routes to limit the incremental cost of deeper diagnosis.

We implement and deploy `FunPilot` on Kubernetes with Knative and evaluate it with both real-world workloads from production traces and synthetic workloads with injected anomalies. The evaluation (§3) shows that `FunPilot` completes in an average of 6.04 s at an average cost of $0.0082 per diagnosis, fitting within the observability stack's metrics collection interval (15 s). `FunPilot` can effectively identify root causes and derive effective control knob settings to remediate performance issues of serverless applications.

2 System Design

The unique characteristics of serverless impose strict constraints on system design in this domain. Beyond general principles for agentic systems (e.g., guardrails, grounded reasoning, and auditability [20, 22, 23, 67]), we identify four serverless-specific design objectives (§2.1) and present the `FunPilot` system design that realizes them (§2.2 and §2.3).

2.1 Serverless-Specific Design Objectives

Rapid response. Serverless operates on substantially shorter timescales than other deployment models, in terms of scaling decisions, execution durations, and replica lifetimes. For example, KPA evaluates scaling decisions every 2 s over a 6–60 s window, whereas autoscaling in VMs, container services, and databases typically operates over tens of seconds to minutes [7, 8, 15, 36]. Under such rapid scaling, metrics, replicas, and quality of service (QoS) conditions shift second-to-second. Replicas may be created and destroyed within seconds, making per-replica diagnosis difficult. Traffic with time-varying volume and resource demands further accelerates state changes [61]. This leaves only a narrow time window for effective intervention, ideally within the observability stack's metrics collection interval [42].

Cost efficiency. Cost efficiency is a primary driver of serverless adoption, and serverless workloads are often short-lived and billed at millisecond granularity [31, 44, 50, 54, 58]. An expensive remediation mechanism can easily incur disproportionate costs (§1). The system should aim for per-diagnosis cost to be small relative to the application cost it manages or the potential savings it targets.

Coordination with the platform autoscaler. The system should complement, not replace, the platform autoscaler. Serverless autoscalers already provide a degree of self-healing. With proper settings, rapid horizontal scaling absorbs traffic fluctuations and clears transient alerts without external intervention [39, 63]. Existing platform autoscalers (e.g., KPA and KEDA [4, 36]) are inherently faster and cheaper than LLM-based solutions. Rather than directly managing scaling (i.e., replica count), the system should leverage rich observability context, semantic knowledge, and domain-aware reasoning to diagnose root causes beyond the autoscaler's scope and complement it by deriving effective control knob settings.

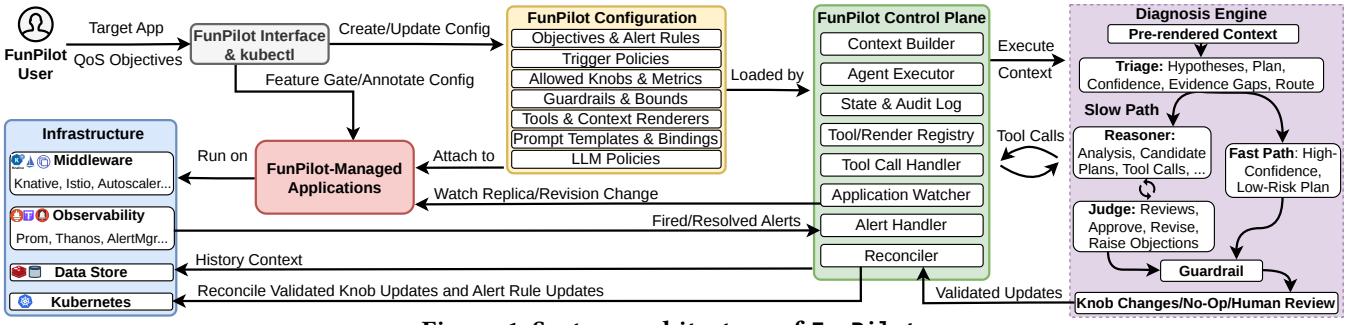

Figure 1: System architecture of FunPilot.

Alignment with the serverless model. The system should match the event-driven execution model and lifecycle of serverless applications: it should activate strictly on demand, impose zero overhead when an application is scaled to zero replicas, and maintain full revision awareness as deployments evolve [18, 35]. In multi-tenant environments, it must isolate metrics, logs, control settings, and agent state across tenants for security and privacy reasons. Lastly, through guardrails and allowlists, access to out-of-scope data/logs and dangerous knob settings should be restricted.

2.2 System Architecture

Figure 1 presents the FunPilot system architecture. FunPilot runs alongside the serverless platform and operates on its exposed control knobs. Runtime diagnosis and remediation are triggered by application-scoped alert rules [53] (e.g., average latency is above a certain threshold) that FunPilot registers in the observability stack (e.g., Prometheus and Alertmanager). Users interact with the system through the FunPilot interface and attach a FunPilot configuration to the target application. The configuration is a structured, JSON-based specification that states QoS targets, alert rules, trigger semantics, allowed knobs, metric definitions, prompt templates, reasoning step limits, and tool call handlers. This allows users to customize the diagnosis and remediation logic without changing the system.

The application watcher tracks FunPilot-enabled applications in an event-driven manner. It registers and deregisters alert rules as applications scale from or to zero, eliminating monitoring overhead for idle applications. On new deployments, it snapshots current settings and lifecycle metadata into revision history, forming a temporal context to help distinguish new anomalies from pre-existing conditions. The alert handler receives fired/resolved alerts via webhook and enforces per-application trigger semantics, such as cooldown periods after control updates and blackout windows after revision creation to avoid cascading diagnosis runs before prior changes have propagated or new revisions have stabilized. It also maintains per-revision alert histories for temporal context. The diagnosis engine is event-driven and triggered only on demand by valid alerts. The agent executor orchestrates agent execution, routes tool calls to the corresponding handlers, forwards validated remediation actions to the reconciler, and logs LLM requests, responses, and costs. The reconciler applies control knob and alert rule updates, manages application revisions, and enforces deterministic guardrails that reject non-allowlisted, out-of-range changes (e.g., changes arising from LLM hallucination).

Prior work on RCA for microservices shows that domain-specific tools, especially tools that summarize or analyze telemetry, improve diagnostic accuracy and reduce token cost [20]. FunPilot adopts the same principle for serverless remediation by exposing domain-specific tools, such as metric queries, time series processing (e.g., trend and anomaly analysis and statistics calculation [20]), alert/revision/knob update history queries and summarization, and capacity analysis (e.g., Little's law [46]). Before each diagnosis run, FunPilot invokes these tools, fetches, processes, and summarizes the metrics (as defined in the configuration attached to the application) and history, and materializes them into a structured diagnostic context. Thus, diagnosis starts from grounded runtime evidence rather than from issuing tool calls for routine observability framework queries and metrics analyses on the critical path. This is important for runtime diagnosis in the serverless context with tight latency and cost constraints. When the pre-materialized context is insufficient, the diagnosis engine can request additional evidence through tool calls, such as metrics over a specific revision and a longer time window.

FunPilot uses prompt templates with named placeholders (e.g., `{{.AlertHistory}}`) [26, 47] to support flexible context construction. The context builder resolves each placeholder by dispatching it to the corresponding renderer for prompt construction, which fetches and transforms the underlying data into a structured context through a uniform interface. FunPilot manages renderers and LLM-callable tools through registries. Users register renderers and tools through the FunPilot interface, and use the FunPilot configuration to bind placeholders to renderers and specify which tools are exposed to each agent workflow. Renderers and tools that involve application-specific data access are scoped to the

target application or revision, preventing a diagnosis engine instance from accessing metrics, logs, or configuration state of co-located workloads in a multi-tenant cluster.

2.3 Runtime Diagnosis and Remediation

As discussed in §2.2, validated alerts trigger `FunPilot` to launch a diagnosis engine instance to perform LLM-based diagnosis and remediation on the target serverless application at runtime. The engine runs a multi-agent workflow with three role-specialized LLM agents: a *triage*, a *reasoner*, and a *judge*. The *triage* is intentionally scope-limited and prompted to reason only over the pre-fetched and processed diagnostic context, and determines whether the evidence is sufficient for a bounded, low-risk remediation with a single LLM call, such as increasing capacity via a single knob for resource allocation or scaling target. If so, the engine takes a fast path and applies the update after guardrail checks without invoking more reasoning steps on the slow path.

The *reasoner* and *judge* run on the slow path, which is typically entered for cases such as ambiguous initial evidence, interacting effects across multiple control knob changes, or evidence gaps that lead to low confidence in specific control updates. The *reasoner* is instructed to analyze the triage assessment, identify evidence gaps, call relevant tools for broader context and decisive evidence when needed, and reason about candidate remediation plans. Each plan specifies plausible causes, proposed control knob updates (or a no-change/human-review outcome), supporting evidence, expected benefit and risk, and confidence. The *judge* reviews candidate remediation plans, and may approve the selected plan, revise it to a smaller or safer change, or reject it with feedback for another round of reasoning bounded by the number of reasoning steps and time limits. The reconciler applies the approved or revised changes after guardrail checks.

Since runtime diagnosis and remediation in serverless are constrained by tight latency and cost budgets, `FunPilot` uses tiered LLM backend selection to balance inference latency and cost. `FunPilot` is agnostic to where the LLM is hosted and can use either external LLM APIs or locally hosted model endpoints. Multiple inference providers often serve the same model at different prices, with varying latency (e.g., time-to-first-token) and throughput. Accelerator-backed providers can offer substantially higher throughput and lower latency, but typically at a higher per-token price. `FunPilot` therefore treats backend selection as part of its runtime control policy. Early LLM calls (e.g., first three calls) are routed to a high-throughput backend to cover latency-critical decisions, including fast-path remediations. Later invocations are routed through a lower-cost backend, reducing the incremental cost of additional evidence collection and re-reasoning.

3 Evaluation

We implement `FunPilot` on Kubernetes with Knative [38], a production-grade, open-source serverless platform used by public offerings such as Google Cloud Run [16] and IBM Code Engine [19]. `FunPilot` coordinates with the platform autoscaler: KPA makes replica-level scaling decisions, while `FunPilot` updates Knative-exposed control knobs. Enabling `FunPilot` requires only two Kubernetes annotations [40] (feature gate and configuration ID). This allows `FunPilot`-managed and regular applications to coexist in the same cluster without interference. We evaluate `FunPilot` end-to-end across ten real-world and synthetic serverless workloads.

Testbed. We deploy `FunPilot` on a seven-node Kubernetes cluster (v1.33.4) with one controller and six workers, totaling 92 vCPUs (Intel Xeon Skylake) and 205 GB of memory. The cluster runs Knative v1.19.6 and the required middleware and observability stack (e.g., Istio, Prometheus, and Thanos). A separate machine generates traffic with FaaSProfiler [57].

Workloads. We evaluate `FunPilot` using ten serverless workloads: three trace-driven, four synthetic with controlled traffic dynamics/injected anomalies, and three that emulate common serverless performance scenarios. We use two production traces for orthogonal purposes: Huawei traces [30, 31] provide per-request demand for workload replay, while IBM Code Engine traces [50] provide production Knative knob settings for constructing baselines. Specifically, we select three applications from Huawei traces (i.e., `C3-F405`, `C4-F1014`, and `C3-F1014`; `C` and `F` denote cluster and function IDs), whose average CPU time exceeds 20 ms and whose invocation counts in the first 1800 s are closest to 90k, 180k, and 360k (labeled as HW). We replay each request's CPU demand using a workload-emulating function, following prior trace-replay methodology [9, 32]. The four synthetic traffic workloads use Poisson arrivals with three 10-minute phases (numbers show arrival rates in RPS): 50–100–200, 200–100–50, 200–0–100, and 20–200–20, representing increasing load, decreasing load, mid-run idleness, and a 10× burst, in which each request uses 52 ms CPU time with a 40% CPU utilization (i.e., a wall clock duration of 52/40% = 130 ms), matching the average observed in Huawei traces (51.8 ms and 42.9%) [31, 44]. The three performance scenarios are a low concurrency limit (10 in-flight requests per container), a remote-API bottleneck (with 200 ms of injected delay), and a long cold start latency (15 s). The first two use HW360k, while the long cold start scenario alternates Poisson traffic every minute between 20 and 100 RPS.

Experimental setup. Each workload defines a latency SLO and a utilization target. For trace-based workloads, we set the p99 latency SLO to the original trace p99 rounded up to the nearest 50 ms (250 ms for HW90k and 200 ms for HW180k and HW360k). For workloads with Poisson traffic,

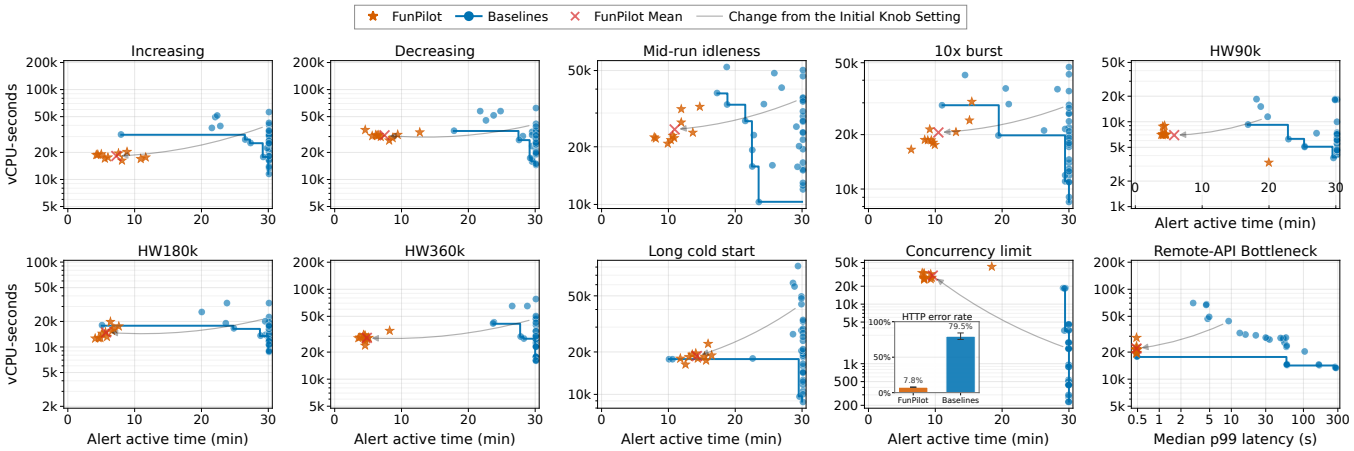

Figure 2: Resource cost versus alert active time. Arrows represent the change from the initial configuration to the mean results of `FunPilot`. For the remote-API bottleneck, we report median p99 latency since the latency target is unattainable.

we set the average latency SLO to 150 ms (rounded up to the nearest 50 ms). Across all workloads, the utilization target is average CPU utilization above 20%. We encode these objectives as alert rules (i.e., avg/p99 latency > workload-specific threshold and CPU utilization < 20%) and use three-minute cooldown/blackout windows. The `FunPilot` configuration contains eleven application- and revision-scoped metric definitions, including request latency, resource usage, pod count, traffic characteristics (e.g., request arrival and completion rates, error rates, and status codes), and platform autoscaler-specific signals (e.g., concurrency and panic mode). All LLM calls are routed through OpenRouter [52] and use the open-weight model `gpt-oss-120b` [2] with tiered routing between an accelerator-backed, high-throughput Cerebras endpoint [11] for the first three calls of a run and a lower-cost Google Cloud endpoint [14] for subsequent calls.

Baselines. We derive baseline settings from IBM Code Engine traces [50], a Knative-based production serverless platform. We vary three knobs: *concurrency scaling target* $\in \{100, 50, 200\}$, *CPU limit* $\in \{1, 0.125, 0.25\}$ vCPUs, and *min-scale* $\in \{0, 1, 2, 10\}$. The first three values for each knob are the most common observed settings, covering 94.8%, 85.0%, and 94.7% of the corresponding knob settings, respectively. *min-scale* of 10 adds an aggressive baseline for reserved capacity. This yields 36 baseline configurations for each workload. For `FunPilot`, we use an initial knob setting with the most common observed values in the trace: *scaling target* = 100, *CPU limit* = 1 vCPU, and *min-scale* = 1. `FunPilot` may then revise these knobs at runtime in response to validated alerts. We repeat each `FunPilot` experiment 10 times and each baseline experiment 3 times. Each experiment runs for 30 minutes.

Remediation Effectiveness. Figure 2 shows the tradeoff between application resource consumption and objective adherence across ten workloads. Alert active time (AAT) is the cumulative time of objective violations during a 30-minute run. As the injected 200 ms delay makes the p99 < 200 ms latency target unattainable, we report the median p99 latency for the remote-API bottleneck. The envelope curve is the lower-cost frontier over baseline runs: for each AAT threshold t, it reports the minimum vCPU-seconds achieved by any baseline run with AAT $\leq t$.

Starting from the initial knob setting, `FunPilot` reduces AAT by 70.9% and vCPU-seconds by 38.3% (13,270.4) on average, excluding the concurrency-limit scenario, where latency/utilization does not capture the main failure mode (failed requests), and the remote-API bottleneck scenario with always-active alerts. Across the remaining eight workloads, the mean `FunPilot` point achieves a Pareto improvement in six workloads and remains close to the envelope in the other two. Baseline configurations that achieve comparable or better resource efficiency/SLO adherence (e.g., HW180k) indicate the self-healing ability of KPA when knobs are properly configured, which is consistent with the design objective of coordinating with the platform autoscaler (§2.1).

For the three performance scenarios, `FunPilot` identifies the dominant mechanism and takes proper actions. It increases warm capacity for long cold starts by raising *min-scale* and *scale-down-delay*; sets a hard concurrency limit and a smaller scaling target for the scenario with a user-container concurrency limit, reducing the HTTP error rate from 79.5% to 7.8% on average; and identifies and escalates the application-level bottleneck for human review after an average of 3.1 diagnosis runs.

Diagnosis and Remediation Behaviors. Across all 284 diagnosis engine runs in the evaluation, 75.7% are triggered by latency alerts, 16.5% by utilization alerts, and 7.7% by both. Of these runs, 73.2% take the fast path, and 269 lead

to control knob updates. On the slow path, the *judge* approves 61.2% of remediation plans and revises 14.9% of them (typically by reducing the proposed capacity increase or deferring scaling down to zero). Across the fast path and all reasoning rounds on the slow path, 8.6% of candidate plans are rejected by the *judge* or guardrails, typically because they contradict observed evidence/metrics (e.g., reducing capacity despite recent resource pressure), over-provision aggressively, or contain malformed LLM responses due to instruction-following failures.

Excluding the remote-API bottleneck scenario with the unattainable latency SLO, 11.3% of the 238 runs that produce control knob updates leave the application in a worse alert rule violation state. We compare the three-minute window starting one minute after an update with the three minutes preceding it, and consider the state worse if the share of time in which an alert rule is violated is at least 10 percentage points higher. Most of these cases (10.1% of runs) result from traffic changes (e.g., traffic volume increases after the knob update), making the applied adjustment insufficient for the new load. The remaining 1.2% over-provision to resolve a latency violation and subsequently trigger a low-utilization alert. Because each diagnosis includes history context (i.e., recent alert and control knob update history), these insufficient or counterproductive changes are typically corrected by the next diagnosis run rather than persisting.

Latency and Cost. `FunPilot` can act fast with minimal overhead. For the 269 runs that lead to control knob updates in the evaluation, `FunPilot` completes the path from alert trigger to the creation of a new revision with updated knobs in 6.04 s on average (median 2.81 s and p95 24.11 s), and 86.6% of runs complete within 15 s. This latency falls within the observability stack's metrics collection interval (15–30 s [37, 42, 60]) and is comparable to the timescale of serverless autoscaler decisions, allowing remediation to act on the latest system state rather than stale observations. Excluding the remote-API bottleneck scenario, `FunPilot` resolves the firing alert in 80.57 s on average, measured from the fired alert that triggers diagnosis to the corresponding resolved event reported by Prometheus. Overall, a diagnosis run costs $0.0082 on average, using 21.3k input and 3.2k output tokens. Fast-path runs cost $0.0052 on average (11.4k/1.6k input/output tokens), while slow-path runs cost $0.0175 (51.1k/8.0k tokens). Even accounting for all diagnosis runs in an experiment (average 2.84 runs), the total cost remains much lower than the average cost savings from reduced vCPU-seconds ($0.382 under AWS Lambda pricing).

4 Discussion

`FunPilot`'s fast diagnosis partly relies on high-throughput, low-latency LLM endpoints. Our analysis of OpenRouter data [52] shows that, as of May 2026, only a few accelerator-backed providers (e.g., Cerebras and Groq) offer a narrow set of general-purpose models with throughput > 500 tokens/s at prices up to 10× those of lower-throughput endpoints for the same model. This throughput–price gap motivates tiered backend selection. We also examined smaller accelerator-backed models, including `Llama-3.1-8B` and `gpt-oss-20b`, but found them unreliable for runtime diagnosis. They often exhibited instruction-following failures, text degeneration, schema violations, or repeated tool misuse. We therefore use a larger reasoning model, `gpt-oss-120b`. The main remaining failure mode was overconfidence (§3): proposing actionable remediation plans despite insufficient evidence, often rejected by the *judge* or blocked by guardrails. As LLM inference becomes faster and cheaper, we expect more models to satisfy both latency and capability requirements within modest per-diagnosis budgets, making systems like `FunPilot` viable across a broader range of deployments.

Our evaluation uses third-party inference APIs and does not explore self-hosted, in-cluster LLMs. Self-hosted LLMs can improve data locality and amortize inference costs at sufficiently high alert/traffic volumes within the cluster, but may incur idle costs and require inference capacity planning. Also, fine-tuned, domain-specific models for performance engineering may further improve this tradeoff among diagnosis latency, cost, and capability. These choices also depend on the billing model: diagnosis and remediation as a free platform feature, a premium service, or a pay-per-use tool. This suggests LLM backend selection is also a control-plane decision for runtime cloud operations and raises open systems questions: which model to use for LLM-assisted performance engineering, whether fine-tuned models can replace general reasoning models, when to pay for accelerator-backed inference, and at what traffic volume an in-cluster model endpoint becomes cheaper than third-party APIs.

5 Conclusion

This work presents `FunPilot`, a runtime diagnosis and remediation system for serverless applications. Our results show that `FunPilot` navigates the resource efficiency/SLO adherence tradeoff effectively, while keeping latency and inference overhead low enough for practical online use.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), through research grants RGPIN-2021-03714 and DGECR-2021-00462, and the Canada Graduate Scholarship (CGS D) program. Cloud resources from the Digital Research Alliance of Canada (RAS and RAC allocations) facilitated this research.

References

[1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 419–434.

[2] Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925* (2025).

[3] Nabeel Akhtar, Ali Raza, Vatche Ishakian, and Ibrahim Matta. 2020. COSE: Configuring serverless functions using statistical learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 129–138.

[4] KEDA Authors. 2026. KEDA | Kubernetes Event-Driven Autoscaling. https://keda.sh/ [Online; accessed May-9-2026].

[5] AWS. 2026. Forecasting AWS DevOps Agent costs for your organization. https://repost.aws/articles/AR0juLZf1VT8yQD_kj4QEK3Q/forecasting-aws-devops-agent-costs-for-your-organization [Online; accessed May-9-2026].

[6] AWS. 2026. Frontier agent – AWS DevOps Agent – AWS. https://aws.amazon.com/devops-agent/ [Online; accessed Mar-2-2026].

[7] AWS. 2026. Performance and scaling for Amazon Aurora PostgreSQL. https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraPostgreSQL.Managing.html [Online; accessed May-8-2026].

[8] AWS. 2026. Target tracking scaling policies for Amazon EC2 Auto Scaling. https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-target-tracking.html [Online; accessed May-8-2026].

[9] Mohammadamin Baqershahi, Changyuan Lin, Visal Saosuo, Paul Chen, and Mohammad Shahrad. 2026. Hierarchical Integration of WebAssembly in Serverless for Efficiency and Interoperability. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. 2481–2498.

[10] Muhammad Bilal, Marco Canini, Rodrigo Fonseca, and Rodrigo Rodrigues. 2023. With great freedom comes great opportunity: Rethinking resource allocation for serverless functions. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 381–397.

[11] Cerebras. 2026. OpenAI GPT OSS - Cerebras Inference. https://inference-docs.cerebras.ai/models/openai-oss [Online; accessed May-11-2026].

[12] Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. 2025. AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds. *Proceedings of Machine Learning and Systems* 7 (2025).

[13] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2024. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 674–688.

[14] Google Cloud. 2026. Agent Platform Pricing. https://cloud.google.com/gemini-enterprise-agent-platform/generative-ai/pricing [Online; accessed May-11-2026].

[15] Google Cloud. 2026. Autoscaling groups of instances. https://docs.cloud.google.com/compute/docs/autoscaler [Online; accessed May-8-2026].

[16] Google Cloud. 2026. Cloud Run functions | Google Cloud. https://cloud.google.com/functions [Online; accessed May-2-2026].

[17] Google Cloud. 2026. Configure Cloud Run services. https://docs.cloud.google.com/run/docs/configuring [Online; accessed May-4-2026].

[18] Google Cloud. 2026. Manage revisions | Cloud Run | Google Cloud Documentation. https://docs.cloud.google.com/run/docs/managing/revisions [Online; accessed May-14-2026].

[19] IBM Cloud. 2026. IBM Cloud Code Engine. https://www.ibm.com/products/code-engine [Online; accessed May-2-2026].

[20] Alessandro Cornacchia, Iliyas Alabdulaal, Ibraheem Saghier, Albaraa Mirdad, Omar Fayoumi, and Marco Canini. 2025. Between Promise and Pain: The Reality of Automating Failure Analysis in Microservices with LLMs. In *Proceedings of the 16th ACM SIGOPS Asia-Pacific Workshop on Systems*. 155–167.

[21] Pubali Datta, Isaac Polinsky, Muhammad Adil Inam, Adam Bates, and William Enck. 2022. ALASTOR: Reconstructing the provenance of serverless intrusions. In *31st USENIX Security Symposium (USENIX Security 22)*. 2443–2460.

[22] Liming Dong, Qinghua Lu, and Liming Zhu. 2024. AgentOps: Enabling observability of LLM agents. *arXiv preprint arXiv:2411.05285* (2024).

[23] Yi Dong, Ronghui Mu, Gaojie Jin, Yi Qi, Jinwei Hu, Xingyu Zhao, Jie Meng, Wenjie Ruan, and Xiaowei Huang. 2024. Building guardrails for large language models. *arXiv preprint arXiv:2402.01822* (2024).

[24] Simon Eismann, Long Bui, Johannes Grohmann, Cristina Abad, Nikolas Herbst, and Samuel Kounev. 2021. Sizeless: Predicting the optimal size of serverless functions. In *Proceedings of the 22nd International Middleware Conference*. 248–259.

[25] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 135–153.

[26] Go. 2026. template package - text/template - Go Packages. https://pkg.go.dev/text/template [Online; accessed May-1-2026].

[27] Google. 2026. Gemini for Google Cloud documentation | Google Cloud Documentation. https://docs.cloud.google.com/gemini/docs [Online; accessed Mar-2-2026].

[28] Junhao Hu, Jiang Xu, Zhixia Liu, Yulong He, Yuetao Chen, Hao Xu, Jiang Liu, Jie Meng, Baoquan Zhang, Shining Wan, Gengyuan Dan, Zhiyu Dong, Zhihao Ren, Changhong Liu, Tao Xie, Dayun Lin, Qin Zhang, Yue Yu, Hao Feng, Xusheng Chen, and Yizhou Shan. 2025. DEEPSERVE: Serverless Large Language Model Serving at Scale. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. 57–72.

[29] Saurabh Jha, Rohan Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, et al. 2025. ITBench: Evaluating AI agents across diverse real-world IT automation tasks. In *Proceedings of the 42nd International Conference on Machine Learning*. 27134–27197.

[30] Artjom Joosen. 2025. Huawei Cloud Production FaaS Trace Data Releases. https://github.com/sir-lab/data-release [Online; accessed May-1-2026].

[31] Artjom Joosen, Ahmed Hassan, Martin Asenov, Rajkarn Singh, Luke Darlow, Jianfeng Wang, Qiwen Deng, and Adam Barker. 2025. Serverless cold starts and where to find them. In *Proceedings of the Twentieth European Conference on Computer Systems*. 938–953.

[32] Kostis Kaffes, Neeraja J Yadwadkar, and Christos Kozyrakis. 2022. Hermod: principled and practical scheduling for serverless functions. In *Proceedings of the 13th Symposium on Cloud Computing*. 289–305.

[33] kagent. 2026. kagent | Bringing Agentic AI to Cloud Native. https://kagent.dev/ [Online; accessed Mar-2-2026].

[34] Knative. 2026. About Autoscaling. https://knative.dev/docs/serving/autoscaling/ [Online; accessed May-4-2026].

[35] Knative. 2026. About Revisions. https://knative.dev/docs/serving/revisions/ [Online; accessed May-14-2026].

[36] Knative. 2026. Additional autoscaling configuration for Knative Pod Autoscaler. https://knative.dev/docs/serving/autoscaling/kpa-specific/ [Online; accessed February-16-2026].

[37] Knative. 2026. Collecting Metrics in Knative. https://knative.dev/docs/serving/observability/metrics/collecting-metrics/ [Online; accessed May-4-2026].

[38] Knative. 2026. Knative Technical Overview. https://knative.dev/docs/ [Online; accessed May-4-2026].

[39] Stavros Kontopoulos. 2025. Demystifying Activator on the data path. https://knative.dev/blog/articles/demystifying-activator-on-path/ [Online; accessed May-9-2026].

[40] Kubernetes. 2026. Annotations. https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/ [Online; accessed May-4-2026].

[41] Kubernetes. 2026. Horizontal Pod Autoscaling. https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/ [Online; accessed May-9-2026].

[42] Grafana Labs. 2026. prometheus.scrape | Grafana Agent documentation. https://grafana.com/docs/agent/latest/flow/reference/components/prometheus.scrape/ [Online; accessed May-9-2026].

[43] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2025. GPTuner: An LLM-based database tuning system. *ACM SIGMOD Record* 54, 1 (2025), 101–110.

[44] Changyuan Lin, Yuanzhi Ma, and Mohammad Shahrad. 2026. Demystifying serverless costs on public platforms: Bridging billing, architecture, and OS scheduling. In *Proceedings of the 21st European Conference on Computer Systems*. 1964–1981.

[45] Changyuan Lin, Nima Mahmoudi, Caixiang Fan, and Hamzeh Khazaei. 2022. Fine-grained performance and cost modeling and optimization for FaaS applications. *IEEE Transactions on Parallel and Distributed Systems* 34, 1 (2022), 180–194.

[46] John DC Little and Stephen C Graves. 2008. Little's law. In *Building intuition: insights from basic operations management models and principles*. Springer, 81–100.

[47] Yuetian Mao, Junjie He, and Chunyang Chen. 2025. From prompts to templates: A systematic prompt template analysis for real-world LLMapps. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. 75–86.

[48] Microsoft. 2026. Azure SRE Agent. https://azure.microsoft.com/en-us/products/sre-agent [Online; accessed Mar-2-2026].

[49] Arshia Moghimi, Joe Hattori, Alexander Li, Mehdi Ben Chikha, and Mohammad Shahrad. 2023. Parrotfish: Parametric regression for optimizing serverless functions. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*. 177–192.

[50] Nima Nasiri, Nalin Munshi, Simon Daniel Moser, Marius Pirvu, Vijay Sundaresan, Daryl Maier, Thatta Premnath, Norman Böwing, Sathish Gopalakrishnan, and Mohammad Shahrad. 2026. In-production characterization of an open source serverless platform and new scaling strategies. In *Proceedings of the 21st European Conference on Computer Systems*. 2054–2072.

[51] Chanh Nguyen, Erik Elmroth, and Monowar Bhuyan. 2025. Silent Failures in Stateless Systems: Rethinking Anomaly Detection for Serverless Computing. In *2025 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 8–19.

[52] OpenRouter. 2026. Models | OpenRouter. https://openrouter.ai/models [Online; accessed May-18-2026].

[53] Prometheus. 2026. Alerting rules. https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/ [Online; accessed May-16-2026].

[54] Amazon Web Services. 2026. AWS Lambda Pricing. https://aws.amazon.com/lambda/pricing/ [Online; accessed May-1-2026].

[55] Amazon Web Services. 2026. Serverless Function, FaaS Serverless - AWS Lambda. https://aws.amazon.com/lambda/ [Online; accessed May-2-2026].

[56] Amazon Web Services. 2026. Web Optimizer - Workload Rightsizing - AWS Compute Optimizer - AWS. https://aws.amazon.com/compute-optimizer/ [Online; accessed May-4-2026].

[57] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. 2019. Architectural implications of function-as-a-service computing. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 1063–1075.

[58] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 205–218.

[59] Manish Shetty, Yinfang Chen, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Xuchao Zhang, Jonathan Mace, Dax Vandevoorde, Pedro Las-Casas, Shachee Mishra Gupta, Suman Nath, Chetan Bansal, and Saravan Rajmohan. 2024. Building AI agents for autonomous clouds: Challenges and design principles. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*. 99–110.

[60] Kubernetes SIGs. 2026. kubernetes-sigs/metrics-server: Scalable and efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines. https://github.com/kubernetes-sigs/metrics-server [Online; accessed May-4-2026].

[61] Prasoon Sinha, Kostis Kaffes, and Neeraja J Yadwadkar. 2025. ALAP: Intent-Based Serverless Computing via Delayed Decision-Making. In *Proceedings of the 2025 ACM Symposium on Cloud Computing*. 831–846.

[62] Jaehyun Song, Bumsuk Kim, Minwoo Kwak, Byoungyoung Lee, Euiseong Seo, and Jinkyu Jeong. 2024. A Secure, Fast, and Resource-Efficient Serverless Platform with Function REWIND. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 597–613.

[63] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the curtains of serverless platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 133–146.

[64] Yilun Wang, Pengfei Chen, Hui Dou, Yiwen Zhang, Guangba Yu, Zilong He, and Haiyu Huang. 2024. FaaSConf: QoS-aware hybrid resources configuration for serverless workflows. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 957–969.

[65] Xingda Wei, Fangming Lu, Tianxia Wang, Jinyu Gu, Yuhan Yang, Rong Chen, and Haibo Chen. 2023. No provisioned concurrency: Fast RDMA-codesigned remote fork for serverless computing. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 497–517.

[66] Shuaiyu Xie, Jian Wang, Hanbin He, Zhihao Wang, Yuqi Zhao, Neng Zhang, and Bing Li. 2026. TVDiag: a task-oriented and view-invariant failure diagnosis framework for microservice-based systems with multimodal data. *ACM Transactions on Software Engineering and Methodology* 35, 2 (2026), 1–39.

[67] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

[68] Xiao Zhang, Qi Wang, Mingyi Li, Yuan Yuan, Mengbai Xiao, Fuzhen Zhuang, and Dongxiao Yu. 2025. TAMO: Fine-Grained Root Cause Analysis via Tool-Assisted LLM Agent with Multi-Modality Observation Data in Cloud-Native Systems. *IEEE Transactions on Services Computing* 18, 6 (2025), 4221–4233.