

Fast Tokens, Slow Agents: Budgeting the Entire AI Workflow

Veera Ravindra Divi
Independent Researcher
Austin, TX
ravi3divi@gmail.com

Abstract

An agent can receive fast model responses and still miss its task deadline. Tool pauses, sandbox startup, repeated reasoning, cache restoration, and verification all consume the same end-to-end budget, but are often measured by different systems. Recent work on workflow-aware serving, agentic workload characterization, and elastic prefill-decode capacity makes this a timely control problem. This white paper proposes a workflow budget contract that carries a deadline, quality floor, resource envelope, and recovery reserve across orchestration and inference boundaries. Its contribution is an implementable control specification and evaluation protocol, not a new serving engine or measured performance result. A worked example shows how locally acceptable calls can exhaust a workflow budget. The central recommendation is to optimize correct, policy-compliant tasks completed on time while retaining engine-level latency and tenant-fairness measurements.

The latency paradox

Consider a support agent that reads an order, checks a policy, queries inventory, drafts a remedy, validates the proposed change, and submits it. Each model call may satisfy its time-to-first-token target. Yet the user waits through database queues, a slow partner endpoint, and a final verification call. Improving token throughput does not remove those dependencies. The relevant service is the completed remedy, with its correctness and policy conditions, rather than any single model response.

This distinction is now measurable rather than hypothetical. AgentSysBench examines ten agentic applications and reports non-LLM components as the dominant latency source in five of them [1]. XPerf reconstructs application execution graphs and replays them to reduce workload variation during serving comparisons [2]. These findings support measuring beyond individual calls; they do not establish that every deployment is tool-bound. A coding workload and a conversational support workflow can require opposite resource decisions.

The decision addressed here is narrower than designing a universal scheduler: what information must cross the orchestrator-engine boundary so that a system can decide whether to wait, reroute, shorten optional work, or decline a task? The proposed contract makes those decisions reviewable. It assumes that the application can define an admissible output and identify mandatory checks. Neither token speed nor a model-generated confidence score supplies those definitions.

What the closest systems already solve

DistServe separates prefill and decode and allocates resources against their distinct latency objectives [3]. Sarathi-Serve provides a strong colocated alternative through chunked prefills and stall-free batching [4]. These are useful engine mechanisms, not interchangeable baselines. A comparison must match hardware, models, precision, context distributions, and latency targets before drawing conclusions about disaggregation.

InferCept studies interrupted inference, including whether temporarily unused context should be retained, swapped, or recomputed [5]. Preble jointly considers distributed prefix reuse and load balancing [6]. Both undermine a simplistic rule that agents should always remain on the same replica: preserving cache locality can save work, while a congested cache-rich replica can extend the critical path.

Workflow-aware optimization is already established prior art. Dyserve jointly chooses models, verification policies, and backends, and adapts the uncommitted workflow suffix as load changes [7]. Cascade coordinates per-request latency budgets with prefix-state movement [8]. Crossflow lends bounded decode-node capacity to local prefill through revocable leases [9]. This paper does not claim to invent workflow awareness, slack scheduling, or elastic capacity. It supplies a cross-layer contract that accounts for tool delays, quality obligations, recovery, and offered-load evaluation together.

Define the service before scheduling it

Represent a workflow as a materialized dependency graph whose nodes are model calls, tools, retrieval, validation, or external commits. A graph can grow during execution; estimates are valid only for the currently exposed structure and a stated allowance for unknown continuation. Distinguish read-only speculation from actions that change external state. A scheduler may cancel unused read branches, but an already committed refund cannot be removed from the graph as if it never happened.

For workflow w , record its arrival time a , absolute deadline d , tenant, task class, maximum authorized spend, and mandatory checks. At time t , estimate the remaining critical-path duration R , including queueing, transfers, tools, verification, and expected continuation. Define planning slack as $S = d - t - R - H$, where H is a separately reserved recovery and estimation allowance. The timestamps d and t must share a clock domain; their difference, R , H , and S are durations in the same units. Negative slack identifies predicted infeasibility, not proof of a future miss.

Use an upper predictive quantile for R when calibration permits, rather than summing component p95 latencies and calling the result a workflow p95. Dependencies, parallel branches, correlated congestion, and shared networks invalidate that shortcut. Estimate the joint critical path from held-out trajectories or a calibrated dependency model. Report the empirical fraction of realized durations exceeding the estimate, separately by task class and load band. Under distribution shift the quantile can cease to provide its intended coverage.

Define eligible workflow goodput as the number of offered tasks completed correctly, on time, and within required policy conditions divided by the observation duration. Also report that count divided by all offered tasks as an attainment rate. Keep rejected, canceled, and timed-out requests in the offered-task denominator. Conditional success among admitted tasks is useful diagnostically, but a controller must not improve its headline score merely by rejecting difficult work. For finite experiments, fix an arrival cohort and follow it through every task deadline before scoring; report the arrival-window duration and drain interval separately so boundary censoring cannot favor a slower policy.

The workflow budget contract

The contract contains five groups of fields. Identity fields bind a workflow identifier, tenant, parent-child relationships, and trace version. Objective fields specify the absolute deadline, quality acceptance procedure, mandatory checks, and spend ceiling. Resource fields describe predicted prompt and output sizes, authorized cache scope, sandbox needs, and backend capabilities. State fields carry committed actions, reusable context identifiers, and cancellation status. Uncertainty fields carry estimator version, prediction coverage, continuation allowance, and recovery reserve.

These fields should be typed metadata, not instructions embedded in retrieved prose. A tool response must not be able to raise its own deadline or waive verification. The application owns quality obligations; the serving layer supplies resource and queue information; the controller chooses only among authorized alternatives. Sensitive prompts and raw business data need not accompany a scheduling hint. An opaque, tenant-scoped cache handle is preferable to exposing prompt contents across services.

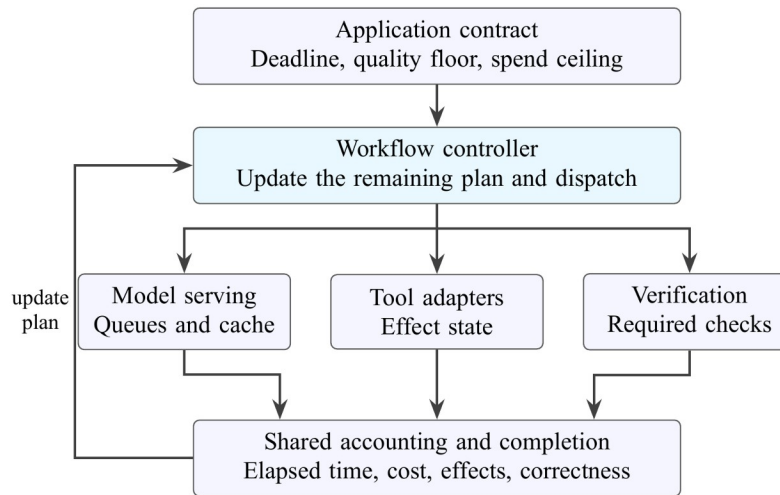
At each dispatch boundary, the controller first validates authorization and remaining spend, then updates the exposed graph and its predicted critical path. It excludes plans that omit mandatory checks or depend on unavailable resources. Among remaining plans, it selects a feasible quality-latency-cost alternative using documented priorities. If none is predicted feasible, it can offer a smaller authorized scope, defer with an explicit expectation, or reject before committing external effects. Degradation is a product decision and must be agreed in advance.

A completed node consumes elapsed time and measured resource usage. A future node receives a revised budget rather than inheriting its predecessor's original allowance. Parallel children share the parent's deadline and aggregate spend ceiling; they do not each receive the entire remaining budget as independent spending authority. A cancellation propagates to queued descendants, while in-flight tools return explicit cancellation or unresolved-outcome states. Late completions still enter accounting.

Separate an external tool's elapsed-time budget from permission to retry. A predicted deadline miss is never grounds to duplicate a non-idempotent action. When a commit outcome is unknown, the controller suspends dependent irreversible steps and requests reconciliation through the application. Recovery reserve H buys time for

this path but cannot guarantee that the remote provider will respond. Deadline compliance and external-state correctness remain different properties.

Figure 1 places the budget controller between the application contract and the execution components. Its feedback loop carries elapsed time, resource consumption, and unresolved outcomes across service boundaries.



One deadline covers model work, tools, validation, state movement, and recovery. Parallel branches share the parent's spending ceiling.

Figure 1. Workflow-level control across model serving, tools, and verification. The feedback loop updates the remaining plan under a shared deadline and aggregate spending ceiling.

Table 1. Proposed ownership boundaries for a workflow budget contract.

Owner	Supplies	Cannot infer alone
Application	Quality floor, required checks	Queue or transfer time
Orchestrator	Dependencies, committed state	GPU capacity
Serving engine	Queues, cache, feasible plans	Business correctness
Tool adapter	Effect and outcome semantics	Workflow quality
Controller	Budget updates, dispatch choices	Unknown remote outcomes

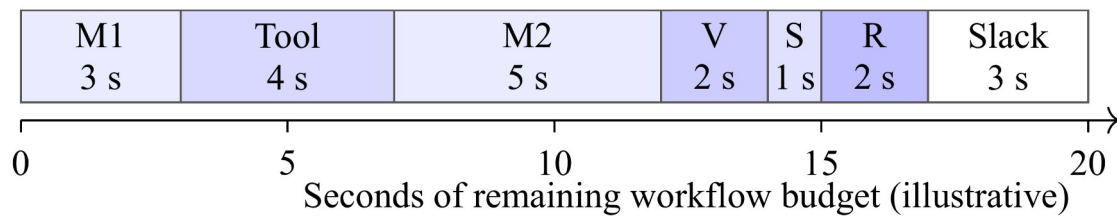
A worked example: where the missing seconds go

The following values are hypothetical design inputs, not observations from a deployed system. Suppose a task has 20 seconds remaining. Its exposed path contains a model call estimated at 3 seconds, a stock tool at 4, a final model call at 5, a mandatory validator at 2, and state restoration at 1. With a 2-second reserve, total planned demand is 17 seconds and slack is 3. Each estimate includes its relevant queueing allowance; none is a published benchmark number.

The stock tool then takes 7 seconds rather than 4. The workflow has consumed its initial slack, even if both model calls can still satisfy their own local targets. The controller recomputes the remaining plan. If a backend switch adds 2 seconds of cache transfer while saving only 1 second of queueing, that switch loses time on this path. If an optional explanation pass costs 2 seconds, omitting it can restore headroom only when the application's quality contract explicitly permits omission. The mandatory validator remains mandatory.

The example exposes a failure of independent budgets: giving each downstream component its usual local allowance does not constrain the sum of sequential delays. It also shows why always selecting the fastest model is inadequate. A cheaper, shorter answer that fails the remedy's correctness check contributes zero eligible workflow goodput. These arithmetic relationships motivate an experiment; they do not demonstrate a performance advantage for the proposed controller.

Figure 2 maps the hypothetical example to the shared 20-second budget; the segments are illustrative allocations, not measured execution times.



M1 / M2: model calls; V: mandatory validator; S: state restoration; R: recovery and estimation reserve. Slack is the unallocated remainder.

Figure 2. Illustrative accounting for the worked example: 17 seconds of planned work and reserve leave 3 seconds of slack. The strip summarizes budget categories rather than asserting a measured execution order.

Cache placement and phase elasticity under the same deadline

A cache-aware dispatch decision should compare predicted completion times, not cache-hit rate alone. For a cache-rich destination, include queueing and the opportunity cost of retaining memory. For an alternate destination, include transfer or recomputation, backend queueing, and interruption costs. Reuse is permissible only when model identity, weights, tokenization, positional assumptions, and tenant authorization are compatible. A cross-tenant hit is not an optimization if it violates isolation.

Crossflow's revocable resource leases illustrate a useful boundary: spare capacity is an offer with multiple resource limits, not permanent ownership [9]. At the workflow layer, borrowed capacity should therefore carry an expiry and a fallback plan. Admission must not assume that every decode-side prefill opportunity survives until a later agent turn. The proposed contract can transport that information without requiring the orchestrator to implement GPU scheduling.

Disaggregation should be a selectable configuration, not a conclusion built into the paper. It can isolate phase interference, but additional transfers and imbalanced pools can erase its benefit. The proposed study must include chunked-prefill colocation, static phase separation, and elastic separation where available. Likewise, retaining idle context should be evaluated against measured return-time distributions, not justified by labeling every session long-running.

A falsifiable evaluation plan

The central hypothesis is that exposing workflow budgets can increase correct, on-time completions under bursty mixed workloads without breaching prespecified quality and tenant-service constraints. The hypothesis fails if improvements disappear after rejection accounting, if a subgroup loses its service floor, or if the controller's overhead consumes its latency savings. No implementation or experiment is claimed in this white paper.

Use two complementary tracks. Fixed-graph replay holds dependency structure, tool outcomes, and token shapes constant to isolate infrastructure effects; XPerf motivates that reproducibility approach [2]. A separate live-agent track permits planning and output changes, scoring final task correctness and external state. Replay can measure a faster execution of the same shape but cannot show that changing models preserves meaning. The two tracks should not share a headline number.

Compare an engine-only baseline, an equal per-call budget baseline, deadline-aware scheduling without quality adaptation, and a workflow-aware optimizer such as Dyserve when a reproducible implementation is available. If a published system cannot be run, describe the alternative as an inspired baseline and document its differences. Ablate tool-time prediction, recovery reserve, cache-aware routing, quality-aware substitutions, and fairness constraints independently.

Sweep offered arrival rates from unsaturated operation through overload, including correlated bursts and slow-tool incidents. Vary graph depth, branching, cache reuse, output lengths, tool latency tails, tenant mix, and estimator error. Use identical arrivals across paired policies and reserve disjoint traces for predictor fitting and evaluation. Maintain separate hardware-measurement and simulator results; a simulated pool is not a production cluster.

Primary reports should include eligible goodput, offered-task attainment, per-tenant and per-class attainment, cost per eligible completion, deadline-miss distribution, correctness, and policy violations. Report TTFT, output-token latency, GPU occupancy, cache transfers, sandbox memory, and controller time as explanatory metrics. Publish absolute counts and rejected tasks alongside percentages. For correlated workloads, use run- or trace-block confidence intervals rather than treating every token as an independent sample.

Predeclare practical margins for quality noninferiority and tenant service, then test them separately from throughput gains. Report multiple workload seeds, independent repeated runs where hardware variance matters, and uncertainty with the unit of resampling stated. A single aggregate mean can conceal starvation of long-context or low-volume tenants. Include the settings where the controller loses, especially prediction failures and tool-dominated tasks with little scheduling headroom.

Deployment boundaries and open questions

The smallest useful deployment is an observation-only controller that logs contract fields, estimates, and counterfactual decisions. Check clock alignment, missing spans, tool-time coverage, and actual spend before enabling dispatch changes. Next enable decisions for read-only tasks with explicit deadlines. External commits require a separate authorization and recovery path; scheduling optimization alone cannot make them safe.

Several limitations remain. Future graph structure may be unknowable, external tools may provide no reliable latency estimate, and quality labels may arrive after the deadline. Fine-grained telemetry can itself expose customer behavior. Contract propagation and measurement consume resources, so their overhead belongs in the end-to-end result. A strategy that works on one model family or interconnect cannot be assumed to transfer.

Three research questions deserve priority: can continuation uncertainty be calibrated without exposing sensitive traces; can resource leases and workflow reservations avoid oscillation under synchronized bursts; and which quality obligations can be checked cheaply enough to influence online dispatch? The answers determine whether the contract is useful. Fast token generation remains valuable, but the service objective should describe the task the user actually receives.

Conclusion

Agentic serving connects dependent model calls to tools and persistent state. The resulting deadline is shared across those components. A workflow budget contract makes that shared constraint explicit and gives researchers a reproducible interface for testing scheduling, cache placement, quality adaptation, and overload behavior. Its value must be established through paired replay and live-agent evaluation, with offered-load denominators and tenant constraints intact. The immediate engineering step is to measure and account for the whole workflow before optimizing any individual stage.

Acknowledgment

Prepared as an independent research white paper. The views expressed do not represent an employer. Proposed mechanisms require implementation and empirical validation.

References

- [1] C. Chang et al., “From LLM Inference to Agentic Workloads: Characterization and Implications for Serving Systems,” arXiv:2608.15127v1, Aug. 15, 2026. Preprint. <https://arxiv.org/abs/2608.15127v1>
- [2] M. Wang, Y. Yue, S. Li, Y. E. Zhou, C. Wang, and J. Huang, “Benchmarking LLM Serving Systems for Agentic AI Workloads with XPerf,” arXiv:2608.20370v1, 2026. Preprint. <https://arxiv.org/abs/2608.20370v1>
- [3] Y. Zhong et al., “DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving,” OSDI, 2024; arXiv:2401.09670v3. <https://arxiv.org/abs/2401.09670v3>
- [4] A. Agrawal et al., “Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve,” OSDI, 2024; arXiv:2403.02310v3. <https://arxiv.org/abs/2403.02310v3>
- [5] R. Abhyankar, Z. He, V. Srivatsa, H. Zhang, and Y. Zhang, “InferCept: Efficient Intercept Support for Augmented Large Language Model Inference,” arXiv:2402.01869v2, May 30, 2024. <https://arxiv.org/abs/2402.01869v2>
- [6] V. Srivatsa, Z. He, R. Abhyankar, D. Li, and Y. Zhang, “Preble: Efficient Distributed Prompt Scheduling for LLM Serving,” arXiv:2407.00023v2, Oct. 3, 2024. <https://arxiv.org/abs/2407.00023v2>
- [7] J. Qian, Z. Wan, H. Yang, C. Tao, S. Kundu, and T. Krishna, “A Workflow-Aware Serving Layer for Agentic Applications,” arXiv:2607.02942v1, July 3, 2026. Preprint. <https://arxiv.org/abs/2607.02942v1>
- [8] M. Adnan et al., “Cascade: Exploiting SLO-Aware latency budget for fair and high goodput LLM inference serving,” arXiv:2608.06557v1, Aug. 6, 2026. Preprint. <https://arxiv.org/abs/2608.06557v1>
- [9] Y. Xu et al., “Crossflow: Prefill–Decode Elasticity for Agentic LLM Serving,” arXiv:2609.27085v1, Sept. 22, 2026. Preprint. <https://arxiv.org/abs/2609.27085v1>