Compass: SLO-aware Query Planner for Compound AI Serving at Scale

Banruo Liu
University of Illinois Urbana-Champaign
Urbana, IL
banruol2@illinois.edu

Wei-Yu Lin
Unaffiliated
Urbana, IL
weiyulin68@gmail.com

Minghao Fang
University of Illinois Urbana-Champaign
Urbana, IL
mf46@illinois.edu

Yihan Jiang
University of Illinois Urbana-Champaign
Urbana, IL
hunterj3@illinois.edu

Fan Lai
University of Illinois Urbana-Champaign
Urbana, IL
fanlai@illinois.edu

ABSTRACT

The rise of compound AI serving that integrates multiple operators in a pipeline enables end-user applications such as generative AI-powered meeting companions, autonomous driving, and immersive gaming. These workloads span diverse deployment spaces, from cloud-only queries to edge-assisted ones across infrastructure tiers, often including both within an application. Achieving high service goodput—i.e., meeting service level objectives (SLOs) for pipeline latency, accuracy, and costs—requires joint planning of operators' placement, configuration, and resource allocation. However, diverse SLOs, varying runtime environments (e.g., heterogeneous device speeds), and a large volume of queries competing for shared infrastructure explode the planning space, making real-time serving and cost-efficient deployment intractable with existing advances.

This paper presents Compass, the first SLO-aware query planner that optimizes large-scale compound AI workloads across diverse deployment spaces. Compass decomposes the many-query, multi-SLO planning problem into tractable subproblems while preserving global decision quality, exploiting plan similarities within and across queries to slash the search steps. It further improves per-step efficiency with a plan profiler that performs selective profiling to achieve high-fidelity performance estimates at a fraction of the profiling cost. At runtime, Compass performs query-plan bipartite matching to maximize SLO goodput under resource contentions. Real-world evaluations show that Compass improves service goodput by 2.4–5.1×, reduces deployment costs by 3.8–4.5×, and accelerates planning by 4.2–10.5×, achieving service responsiveness within seconds and near-optimal decision quality.

PVLDB Reference Format:
Banruo Liu, Wei-Yu Lin, Minghao Fang, Yihan Jiang, and Fan Lai. Compass: SLO-aware Query Planner for Compound AI Serving at Scale. PVLDB, 19(9): 1921-1934, 2026.
doi:10.14778/3819518.3819524

Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.
doi:10.14778/3819518.3819524

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/UIUC-MLSys/Compass.

1 INTRODUCTION

Machine learning (ML) serving is shifting from deploying monolithic models to composing pipelines of multiple operators, such as video denoisers, person and pose detectors, feature extractors, large language model (LLM)-based reasoning modules, and speech and avatar renderers for a live video chat query with AI agents like *"track and imitate user's body movements while performing push-ups, provide instant corrective feedback on muscle engagement"*. By adding and distributing augmented operators, compound AI serving enables better accuracy [9], access control (e.g., via local data processing [29]), efficiency (e.g., reduced data transfer [59]), and operational flexibility (e.g., autoscaling). Increasingly, compound AI applications span diverse use cases, including AI companions in live video conferences [65], ChatGPT's live video conversation [11, 39], speech-to-text streaming [40], autonomous driving [61], and immersive gaming [16], with deployments across cloud machines or even infrastructure tiers with varying capabilities (e.g., from edge devices and near-edge clusters to the cloud [62]).

As with many advances in serving single ML models [2, 20, 34, 58], maximizing the number of queries meeting their service level objectives (i.e., SLO goodput) is critical for both service providers and users. However, compound AI serving poses unique challenges: query planning must holistically consider the *configuration* of operators (e.g., video sampling rates, Qwen3 vs. Llama3 agents), their physical *placements* (e.g., cloud-machine selection and edge offload choices that reduce intermediate traffic), and *resource allocation* (e.g., GPU or bandwidth provisioning). These decisions directly affect SLO metrics such as latency, accuracy, and cost, making query planning a fundamental systems challenge [13].

However, SLO requirements vary across applications [1, 7, 11] and even among users of the same application (e.g., free vs. paid users, or varying speaking speeds in GenAI chat [34]). Worse yet, pipelines with identical operators can exhibit performance variations due to heterogeneous input (e.g., accuracy changes under different lighting conditions [10]) and system capabilities (e.g., edge-cloud network bandwidth). Moreover, real deployments face high

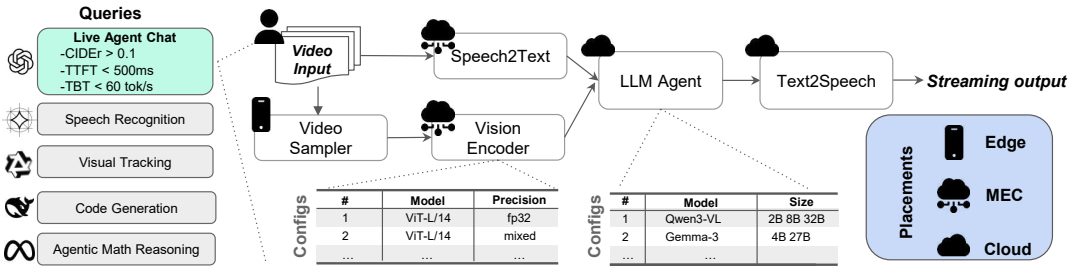


Figure 1: As compound AI services increasingly cater to end users, their deployment can span multi-tier, heterogeneous infrastructure.

query volumes (e.g., tens of thousands of concurrent ChatGPT live video chat sessions [39]), resource-intensive operators, and dynamic runtime environments. These factors make it essential to plan many pipelines efficiently, minimizing deployment costs under resource contention while sustaining SLOs in real time (§2).

Unfortunately, the design space of existing ML query planners remains fragmented, targeting either single, long-running queries [55, 62] or homogeneous cloud-only environments [57] (Table 1). Yet, compound AI applications span diverse deployment spaces, with queries spreading both cloud-only and edge-cloud tiers to maximize SLO goodput (§2.2). Worse, they falter under the combinatorial explosion of candidate plans in compound AI pipelines. For example, an agentic video chat query can yield thousands of candidate plans, involving various placements, configurations, and resource allocations. Planning overheads of existing advances can stretch to tens of minutes, taking 40%-62% total time including serving, becoming the major bottleneck, outlasting the lifetime of many queries and making real-time serving infeasible, while still incurring substantial SLO violations and inflated costs (§2.3).

We introduce Compass (§3), the first SLO-aware query planner for Compound AI serving at scale, capable of supporting diverse deployment spaces (e.g., cloud-only, multi-tier, and multi-query settings). It simultaneously serves the needs of three key stakeholders: (i) high SLO attainment and real-time responsiveness for users, (ii) cost-efficient deployment for service providers, and (iii) high resource utilization for infrastructure operators. It novelly decomposes the many-query, multi-SLO planning problem into efficient single-query planning, and performs query-plan bipartite matching (combination) under contention to maintain global decision quality.

Searching for a cost-efficient plan, even for a single query, is challenging due to the vast search space, multi-dimensional SLOs (e.g., accuracy, latency, and cost), and the heterogeneous plan profiling costs in each search step. For example, profiling a plan's final accuracy that involves LLaMA3-70B inference is 9× more expensive, in both GPU time and cost, than that with LLaMA3-8B. To minimize search steps, we design a search optimizer that decomposes multi-SLO objectives into separate surrogate optimizers, maximizing knowledge reuse across similar plans (e.g., defined by their performance proximity in the learned latent feature space), both within and across queries (§4.1). The optimizer prioritizes high-potential candidates at each step while accounting for heterogeneous profiling costs. To cut per-step overhead, we introduce a profiler that adaptively adjusts profiling amount, reuses prefix caches from prior runs, saving cost while ensuring provable profiling correctness (§4.2).

Maximizing SLO service goodput, however, requires joint planning across many concurrent queries, intensifying the combinatorial explosion. Planning queries independently ignores resource contention in shared infrastructure, yielding globally inefficient allocations and SLO violations (e.g., each query's optimal plan may prefer the cloud tier, leaving edge tiers idle). Compass addresses this with an integral two-stage approach that preserves global decision quality: the single-query planner first identifies the Pareto-optimal set of candidate plans that meet each query's SLOs. Then, Compass efficiently performs bipartite matching between the query set and their Pareto-optimal set of plan candidates, selecting query-plan combinations that maximize SLO goodput (§4.3).

Compass is API-compatible with Kubernetes [24], supporting existing ML serving stacks with a few lines of code change (§5). Evaluations across realistic cloud-only and edge-cloud deployments—spanning five representative compound AI applications such as live agentic video chat and code generation—show that Compass delivers 2.4–5.1× higher service goodput than existing advances [57, 62], even in their targeted deployment space (e.g., cloud-only setting). Furthermore, Compass reduces deployment costs by 3.8–4.5×, and accelerates planning by 4.2–10.5×, enabling second-level query responsiveness and *near-optimal* decision quality across diverse deployment spaces (§6).

Overall, we make three contributions in this paper. First, we develop the first generic SLO-aware query planner supporting compound AI serving at scale across deployment spaces. Second, we propose a novel search mechanism that effectively decomposes global planning, leveraging plan similarities and adaptive profiling to navigate multi-SLO planning. Third, we evaluate Compass in various real-world settings, showing significant SLO improvements and cost savings.

2 BACKGROUND AND MOTIVATION

We begin with an overview of compound AI (§2.1), followed by the challenges it faces (§2.2), and conclude with the limitations of current advances that motivate our work (§2.3).

2.1 Compound AI Serving

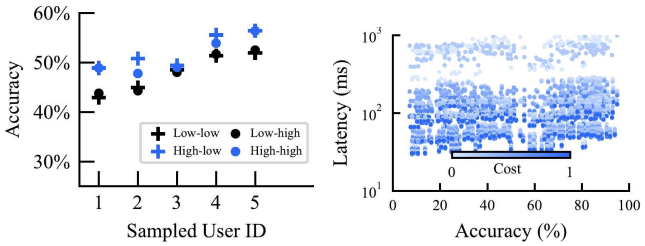
Compound AI serving relies on multi-stage pipelines to generate the final output. For example (Figure 1), a video chat with the assistive AI agent may require operators that sample video frames, encode images, and then invoke a multi-modality model [41], and convert text output back to audio. Each operator typically exposes multiple configuration options (e.g., frame sampling rates and model sizes).

These configurations interact with placement and resource allocation decisions. Cloud machines are capable yet expensive and easily overloaded due to the prevalence of load burstiness, while edge devices are resource-constrained yet free and close to the data source. Near-edge clusters (e.g., MEC [5]) sit in between, providing moderate cost and moderate capacity as an attractive offload layer. We show that using multi-tier resources can significantly improve service goodput (§ 6.3). However, placement decision of each operator becomes complicated; for example, placing operators closer to the data source (e.g., on end devices) can reduce traffic over the Internet but may increase compute latency, ultimately impacting the pipeline's overall latency and accuracy (§2.2).

Design Space. Satisfying SLOs demands an efficient query planner to identify the optimal configuration, placement, and resource allocation for each operator. Practical compound AI queries primarily fall into two categories:

- *Queries with Continuous Interaction*: These involve continuous input streams, such as assistive video AI agents (often lasts minutes) [10, 11], AI-powered mobile cloud gaming [18], and video conferencing [65]. They require fast planning to launch and adapt real-time services under dynamic conditions [44], while identifying cost-effective plans to scale for many queries and users. Even throughput-intensive workloads, like offline video captioning [22], can strain current planners, which often require hours of search (§6.2; additional results are in our technical report [33]) and demand minimizing costly GPU-based accuracy profiling for each configuration.
- *Queries with Ephemeral Interaction*: These pipelines handle numerous short-lived requests, such as speech recognition or agentic code generation from many users, and aim to optimize average serving performance. Diverse applications, SLO tiers (e.g., free vs. paid users [4]), and runtime dynamics—e.g., changes in input distributions, resource availability, and bursty arrivals—require real-time, cost-efficient (re-)planning to sustain performance.

As shown in Table 1, with the growing diversity and scale of applications and users, queries from different or within applications increasingly span deployment spaces for efficiency, cost-effectiveness, and data governance. For example, video understanding queries may process live chat streams directly from end-user devices (e.g., live assistive AI video agents [10]), distributing operators across edge, near-edge, and cloud infrastructure [11, 65], or may run entirely on offline video corpora in the cloud [22]. This widening operational footprint makes a unified, cross-space query planner indispensable.



(a) Heterogeneous Data. (b) Vast search space.

Figure 2: (a) Input distributions vary across users, causing accuracy variance for the same plan. "Low-high": resource-light configurations for operator 1 and resource-intensive configurations for operator 2. (b) Each query can have thousands of plans.

2.2 Challenges in Compound AI Serving

While optimizing SLO service goodput (i.e., cost-effectiveness) remains the key as prior advances for single ML models [43, 58], compound AI serving introduces unique challenges due to its wide span of operators and deployment scenarios.

Diverse SLO Requirements and Runtime Environments. SLO requirements vary not only across applications but also among users within the same application [1, 4, 43]. For example, speaking speeds differ by language and age in live agent chat [34]; user playstyles and reaction times vary widely in AI-powered gaming [15]; and users interacting with ChatGPT's live video chat for real-world assistance exhibit varying responsive needs [10].

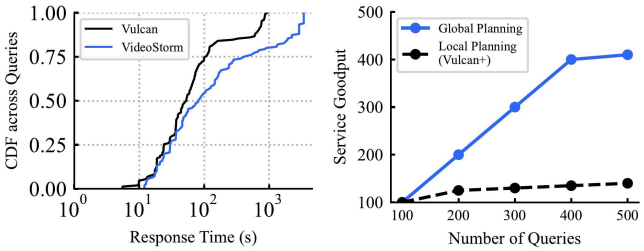
Even under identical SLO targets, queries often encounter heterogeneous runtime environments and input data characteristics. Operators may be distributed across machines with varying capabilities for cost efficiency (e.g., mixing high- and low-end GPUs in the cloud) or even span heterogeneous tiers such as edge devices, MEC, and cloud platforms (e.g., for live edge inputs). At the same time, input distributions are query- and user-specific. For example, our experiments on real-world video chats [52] (Figure 2(a)) reveal that the accuracy of a fixed pipeline varies widely depending on the user's video input, such as content and lighting conditions [10]. Importantly, more resource-intensive configurations do not always yield higher accuracy, making pipeline planning intractable. All these necessitate customized pipelines.

Enormous Plan Search Space and Multi-query at Scale. The interplay of diverse SLOs, heterogeneous system speeds, and data characteristics results in a vast search space. For a pipeline with M operators, each with N configuration options and K placement choices—where $K = 1$ under homogeneous machines, but cost-optimal deployments often prefer heterogeneous machines, thus placements, even within a tier [13]—the number of plan candidates grows exponentially, $O(K^M \cdot N^M)$, yielding thousands of candidates (Figure 2(b)).

Moreover, practical deployments must serve large volumes of queries that share underlying infrastructure resources—e.g., today's live assistive AI video application supporting thousands of users concurrently [39, 65]—to maximize utilization and SLO goodput.

System	Multi-query	Multi-tier	Heter. Aware.	Contention Aware.	Response Time	Near-Optimal Time
Exhaustive Search	Yes	Yes	Yes	Yes	> 2 hours	> 24 hours
Videostorm [57]	Yes	No	No	Yes	~1.6 min	~34 min
Vulcan [62]	No	Yes	Yes	No	~1.2 min	~7 min
Compass	Yes	Yes	Yes	Yes	*< 5s*	*< 20s*

Table 1: Compass enables real-time planning for compound AI workloads across deployment spaces. Unlike prior systems that take minutes to hours, Compass finds the first SLO-compliant plan ("Response time.") in under 5s and near-optimal plans ("Near-Optimal.") in under 20s (§6.2), while comprehensively supporting multiple features. Such speed is essential for powering user-facing services [19, 35, 40, 44].



(a) Time to first feasible plan. (b) Global planning is needed.

Figure 3: (a) Existing advances, Vulcan [62] and VideoStorm [57], spend tens of seconds to identify the first feasible plan that satisfies SLO; and (b) are fundamentally limited for multi-query planning.

This requires contention-aware joint planning (e.g., co-locating pipelines on the same machine), which further expands the planning space far beyond single-query scenarios. Worse, bursty workloads and performance fluctuations (e.g., network fluctuations) complicate trade-offs between SLO compliance and deployment cost.

2.3 Limitations of Existing Advances

Existing ML serving systems have focused on optimizing either server-side metrics, such as serving throughput [2, 22, 31], or SLO-aware request scheduling for individual ML models [58, 63]. As summarized in Table 1, a few recent works target live ML analytics (e.g., video analytics), but are fundamentally limited to single, long-running pipelines [55, 62] and/or homogeneous cloud deployments [57]. Beyond their piecemealed support, we next show how they fail to meet the demands of practical compound AI serving.

Infeasibility for Single-Query Deployment. Existing efforts suffer from slow responsiveness and high costs in planning, even in their targeted single-query planning scenario. Figure 2(b) shows that an agentic video chat query can have thousands of possible plans with widely varying performance and deployment costs. Navigating this explosive search space, existing advances such as Vulcan [62] and VideoStorm [57] take minutes to identify the first SLO-compliant plan (Figure 3(a)), long before any effort to optimize for the plan's deployment cost.

Such sluggish responsiveness renders real-time or short-lived queries infeasible—e.g., brief speech recognition queries often last only a dozen seconds [33]—and prevents rapid replanning to sustain services under runtime dynamics [40]. Even for queries without strict responsiveness demands (e.g., offline video captioning), they cause exploring large plan spaces, imposing extensive, costly GPU-based plan profiling (§6.2).

Inability for Multi-Query Deployment. In practical settings with many queries, existing approaches incur substantial SLO goodput loss due to their limited design scope. For example, we extend Vulcan to support multi-query deployment by allocating one hour per query to identify its local optimal plan and pack queries across machines. Even with this augmentation and highly relaxed responsiveness requirements, the SLO goodput of Vulcan+ remains far below that of an oracle performing joint global planning (Figure 3(b)). This gap arises because single-query planners ignore cross-query resource contention: when multiple queries share infrastructure, (1) latency deviations caused by contention invalidate previously selected local plans, triggering widespread SLO violations; and (2) per-query optimal plans may overload one tier while leaving others idle (e.g., edge devices). Addressing this, without global multi-query planning, can require aggressive resource overprovisioning, which further reduces overall efficiency, especially under runtime dynamics. Unfortunately, global planning is intractable, with the exhaustive searching time ballooning to many hours for a dozen queries (§6.2). Such planning latency makes it impractical to scale online deployments, co-locate queries, or adapt to runtime changes in a timely manner.

3 COMPASS OVERVIEW

We next introduce Compass, the first SLO-aware query planner for compound AI serving that enables real-time planning across diverse deployment spaces, including cloud-only, multi-tier, and settings where these coexist with multiple queries. Compass simultaneously achieves high SLO attainment and fast responsiveness for users, cost-efficient deployments for service providers, and maximized resource utilization for infrastructure operators.

System Components. In practical deployments, queries arrive online and exit once completed. For each query, Compass determines the query plan, specifying operator placement, configuration, and resource allocation, where it accounts for resource contention and may collocate pipelines (e.g., sharing machines or models). The resource orchestrator (e.g., Kubernetes [24]) then deploys the plan across machines and potentially tiers. Compass enables effective multi-query planning in seconds with three core components (Figure 4):

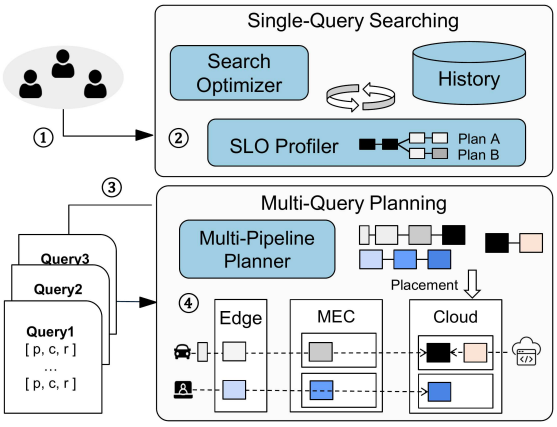


Figure 4: Compass overview for compound AI serving at scale.

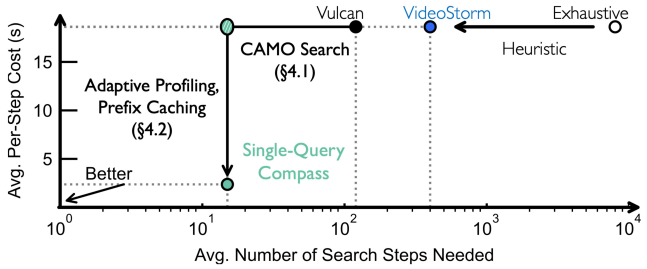


Figure 5: Compass optimizes single-query search by reducing the number of search steps (§4.1) and per search step cost (§4.2).

- *Search Optimizer (§4.1):* Minimizes search steps (i.e., plans proposed and profiled) to find SLO-compliant plans. It optimizes the configuration (c), placement (p), and resource allocation (r) of operators, enabling flexible downstream scheduling.
- *SLO Profiler (§4.2):* Efficiently profiles each plan's performance (e.g., accuracy) and identifies a set of Pareto-optimal plans to guide the search optimizer.
- *Multi-query Scheduler (§4.3):* Given a list of candidate plans per query, it selects query-plan combinations to maximize goodput.

Workflow. Compass employs a two-stage workflow for high efficiency and decision quality (Figure 4). ① When a user submits a new query, the Search Optimizer initiates an iterative single-query search process to generate (propose) candidate plans. ② The SLO Profiler evaluates the candidate's performance (e.g., accuracy and latency), filtering out infeasible plans that fail to meet SLO requirements. ③ The Search Optimizer refines feasible candidates by optimizing their Pareto-optimal resource cost. This ②-③ repeats until a predefined termination condition is met (e.g., response-time limit) or the search space is exhausted. ④ The resulting candidate set is passed to the Multi-query Scheduler, which selects the best combination of query plans across all queries to maximize service goodput while avoiding SLO violations due to resource contention. The finalized deployment plan is then executed by the cluster resource orchestrator. If runtime dynamics are detected, the resource orchestrator will trigger Compass to start a replanning phase.

4 COMPASS DESIGN

Compass decomposes the intractable multi-query planning into a two-stage planning process for scalability and adaptability. This design isolates performance fluctuations within a single query, avoiding costly system-wide replanning, while, for the first time, enabling real-time query planning for various deployment scenarios using a unified planner. However, such decomposition is non-trivial, introducing three unique challenges in balancing planning efficiency and decision quality:

- *Search Efficiency:* Each query may yield thousands of potential plan candidates with varying performance. How to minimize search steps to identify SLO-compliant candidates, improving responsiveness and planning costs (due to profiling) (§4.1)?
- *Profiling Efficiency:* Each proposed plan requires costly performance profiling (e.g., measuring final model accuracy using GPUs [62]). How to reduce the per-candidate profiling time and the overall monetary cost due to high query volumes (§4.2)?
- *Resource Efficiency:* Independent query planning overlooks cross-query resource contention, potentially overloading cloud tiers while underutilizing edge resources. How to maintain high decision quality due to decomposition to maximize goodput (§4.3)?

Next, as shown in Figure 5, we start by describing how Compass enables real-time query planning by simultaneously reducing the per-step cost and the total number of steps required.

4.1 Search Optimizer: Leverage Similarities

Even for a single query, identifying the optimal plan that satisfies latency and accuracy SLOs while minimizing deployment costs is challenging due to the presence of thousands of candidate plans. Worse, per-step search costs vary widely due to heterogeneous profiling costs; for example, searching (i.e., proposing and profiling) a plan involving LLaMA3-70B costs 9× more GPU time than its LLaMA3-8B counterpart. Consequently, minimizing the number of search steps is critical for both single- and multi-query planning.

To this end, Compass employs a novel cost-aware search optimizer built on two key insights. First, reducing resource allocations to operators increases latency but does not affect pipeline accuracy. By *tentatively* assuming resource over-provisioning in the plan (e.g., dedicating a cloud GPU to the query), Compass defers fine-grained resource allocation to the multi-query scheduling stage (§4.3) after resource trimming (§4.2). This over-provisioning adds no extra profiling overhead, as the same data is processed to assess accuracy. This preserves optimality, as over-provisioned plans form a superset that contains the global optimum: (1) they include plans meeting

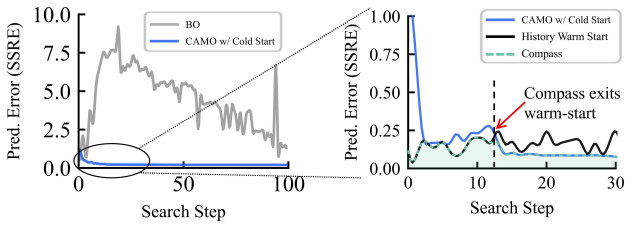


Figure 6: Prediction accuracy of f_l. Left: CAMO achieves accurate latency predictions, whereas standard BO shows significant errors. Right: Compass leverages a committee of past models for warm start, transitioning to its own CAMO as it matures.

accuracy SLOs regardless of high resource costs; and (2) because resources are overcommitted, these plans represent the best possible latency of their counterparts with fewer resource allocations (e.g., when sharing the resource). Plans that still fail to meet latency or accuracy are naturally pruned without compromising decision quality. We defer the resource trimming design to the SLO Profiler (§4.2), which finds the Pareto-optimal counterparts of these over-provisioned plans. We empirically validate this close-to-optimal performance across various settings (§6.4).

Second, instead of exploring plans independently, we can exploit their latent similarities to prioritize candidates with higher potential. However, plans are heterogeneous, comprising diverse operators and configurations, making explicit similarity metrics infeasible.

Leveraging Intra-query Plan Similarity. Compass introduces a Cost-Aware Multi-Objective (CAMO) search optimizer that exploits the latent similarity across a query's candidate plans. CAMO builds on Bayesian Optimization (BO)'s core strength—learning from sparse, noisy feedback without requiring explicit performance models—but fundamentally departs from BO's traditional formulation. Classic BO assumes a single optimization objective and uniform evaluation costs. In contrast, compound AI pipelines exhibit inherently multi-dimensional objectives (accuracy, latency, and cost) and highly heterogeneous search costs, where profiling different plans can differ by orders of magnitude. As shown in Figure 6 (left) and further validated in Section 6.2, these violations cause standard BO to mis-rank candidates, over-explore expensive regions, and ultimately become ineffective for compound AI serving.

To address these limitations, CAMO maintains two independent surrogate BO models: f_a predicts the accuracy of a plan p, and f_l predicts its latency. Compass proposes new plans that maximize the acquisition function (utility) U:

$$U(p) = \mathbf{Pr}[f_a(p) \geq A_{slo}] \cdot \mathbf{Pr}[f_l(p) \leq L_{slo}] \times \frac{1}{\mathbf{C}(f_l(p))}$$

where $\mathbf{Pr}[\cdot]$ is the probability derived from the surrogate predictions.

$\mathbf{C}(f_l(p))$ is the profiling cost function, which is essentially the latency to run the inference, so we can estimate the cost using predicted latency from our CAMO formulation.

Algorithm 1: Compass Single-Query Search.

```
1:  Function SingleQuerySearch(A_slo, L_slo, dataset D):
2:      plans ← [ ]
3:      while current_time < response_time_requirement do
            /* CAMO models of similar history queries vote for a new
               proposal if the new CAMO is immature.                 */
4:          plan ← Propose(C, A_slo, L_slo, f_A, f_L)
5:          if Profile_SLO(plan, D, A_slo, f_A, f_L) then
                /* If the plan is SLO-compliant, then generate its
                   Pareto-optimal variants.                          */
6:              plans += ParetoOptimize(plan, L_slo)
            /* Update CAMO model with profiling feedback.            */
7:          CAMO_Update (plans, f_A, f_L)
8:      return plans
9:  Function Propose(C_r, A_slo, L_slo, f_A, f_L):
        /* When the CAMO model of that query matures, it proposes
           the plan with the highest utility.                        */
10:     if pred. error of C_r is lower than committee's then
11:         c_0 ← argmax(C_r, x : utility(x, A_slo, L_slo, f_A, f_L))
12:     else
13:         c_0 ← committee_propose(C_r, A_slo, L_slo)
14:     return c_0
15: Function ParetoOptimize(c, p, L_slo):
        /* Multi-dimensional binary search to tighten resource usage
           until the latency SLO is violated.                        */
16:     r ← over_provisioned(c)
17:     candidates ← multi_dim_binary_search(c, p, r, L_slo)
18:     return candidates
19: Function CAMO_Update(plans, f_A, f_L):
20:     f_A, f_L ← fit_new_points(plans.acc, plans.lat)
21:     update_committee_similarity(plans.acc, plans.lat)
```

This design provides three key advantages. First, by modeling accuracy and latency as separate objectives, CAMO evaluates each SLO's feasibility directly rather than collapsing them into a single scalar that obscures user requirements. Second, by explicitly incorporating profiling cost, CAMO not only prioritizes high-potential candidates but also prefers those that can be evaluated efficiently (benefit-to-cost ratio). This enables a more principled exploration-exploitation trade-off, while avoiding spending excessive time on marginally beneficial but costly configurations for responsiveness.

Third, decomposing objectives at the SLO level enables model reuse: CAMO can apply the latency surrogate learned from one query and the accuracy surrogate from another, as we will introduce shortly.

As shown in Algorithm 1, at each search step, the plan (i.e., the configuration and placement pair of operators) with the highest utility is proposed and sent to the SLO Profiler (Line 3-8). The profiling results serve as feedback to update CAMO's surrogate models. Figure 6 (left) illustrates the effectiveness of our CAMO design over the standard BO solution. Note that our CAMO optimizer is pretty lightweight, requiring 40 ms per proposing step (§6.3).

Leveraging Inter-query Plan Similarity. Despite the rich heterogeneity in system conditions, input data, and QoE requirements, as well as the complex interplay among them, different queries

may share latent structural patterns. Exploiting such inter-query similarity has been largely overlooked, yet it can dramatically reduce search time. However, identifying similar past queries is challenging. Application-level metadata is frequently unavailable; even when present, it is impractical to encode diverse features for various queries (e.g., multimodal inputs, operator graphs, or model-configuration knobs). Compass leverages historical CAMO models, which have already learned these latent structures, to warm-start the search process for new, similar queries.

Compass uses CAMO's latent representation of candidate plans to define similarity by prediction error, i.e., the gap between a past surrogate model's prediction and the current query's real profiled performance. This requires no additional query information and lets CAMO reuse latently similar accuracy and latency models from different prior queries.

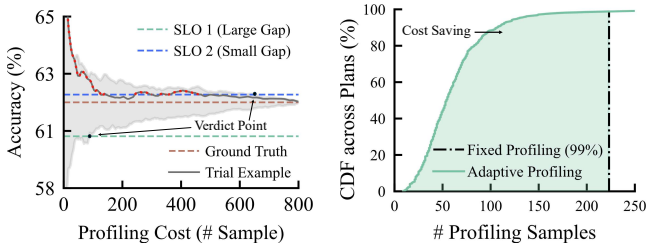
Algorithm 1 realizes this idea through a *consensus-guided warm-start*. Instead of selecting a single most similar query, which is fragile in the presence of noise, dynamics, or partial alignment, Compass assembles a committee of past surrogate models whose errors are lower than the current CAMO optimizer. Each committee member votes for the next candidate plan to propose, with a weight proportional to its similarity to the new query (Lines 10–14), effectively pooling cross-query inductive biases. Then, the selected plan is profiled, and similarity scores are updated based on prediction errors, allowing the committee to reorganize itself on-the-fly.

A novel emergent behavior of our consensus-guided design is its self-terminating transfer. As shown in Figure 6 (right), once the new CAMO optimizer matures—i.e., when its prediction error is lower than the committee—Compass automatically exits the warm-up phase, allowing the search to focus on the new query's unique context (Line 11). As CAMO is lightweight in proposing plans, we further empirically validate that the warm-start design adds little overhead while delivering substantial speedups, even with only a dozen prior queries in history (§6.4).

4.2 SLO Profiler: Ensure Correctness Efficiently

For every candidate plan proposed by the Search Optimizer, verifying its SLO compliance through profiling is essential for steering the search process. Prior ML serving advances [55] have streamlined profiling via two well-established insights. First, a plan's accuracy is determined by its operators' configuration, not by placement or resource allocation, and can therefore be measured efficiently in the cloud (e.g., using proxy inputs or replayed recent inputs [57]). Second, pipeline latency can be decomposed into operator runtimes and their communication costs. Since operator outputs are invariant to placement and resource settings, existing deployments [23, 36, 62] reuse operator-level latency from accuracy profiling and estimate end-to-end latency by applying resource- and input-dependent offsets, such as adjusting data transfer time based on network bandwidth. Recent work [6, 27, 42, 54] validates the effectiveness of this approach. So both accuracy and latency profiling can be consolidated into the accuracy profiling phase.

However, as ML models and pipelines grow more complex, accuracy profiling has become a major bottleneck. Profiling requires running GPU inference over many data samples: profiling *a single* speech recognition plan can take 5-10 minutes on an A100



(a) Shaded area: 1-99% range of estimated accuracy over 100 profiling runs for a plan. Solid line: one trial run. Red-dotted line shows overestimation from insufficient profiling. Verdict depends on the gap between ground truth and the SLO.

(b) Adaptive profiling ensures the same correctness level (e.g., 99%) while adjusting the number of samples per plan, significantly reducing the amount of profiling compared to fixed profiling designs (e.g., using a uniform Y=99% cutoff).

Figure 7: Adaptive profiling ensures correctness and efficiency.

GPU, driving the total profiling cost to hundreds of dollars when scaled to thousands of plan candidates for a single query (§6.3). Unfortunately, accuracy is highly input-dependent, varying across input distribution and user context (§2). Even minor configuration changes (e.g., switching video resolution) can alter the input data, propagate through the pipeline, and impact the overall accuracy.

Compass reduces this increasingly dominant cost and enables real-time planning with two complementary designs: (i) adaptive profiling, which minimizes required profiling data size while preserving correctness; and (ii) dependency-aware caching, which eliminates redundant profiling by reusing overlapping operator profiles.

Adaptive Profiling for Correctness with Minimal Work. Profiling accuracy using the full dataset $\mathcal{D}$ yields ground truth but is prohibitively expensive. Prior efforts [22, 55, 57] typically rely on a fixed, manually specified number of profiling samples (e.g., 500 video frames). As shown in Figure 7(a), this ad-hoc design creates a tradeoff between profiling cost and correctness: (1) too few samples jeopardize correctness, mislabeling invalid plans as SLO-compliant and ultimately causing SLO violations and misleading the search optimizer; but (2) too many samples waste GPUs and slow down search steps. In Figure 7(a), stopping in the red dotted part of the profiling trial leads to an incorrect verdict for SLO target 2, while stopping after the verdict point is a waste.

Compass resolves this tradeoff with insights that profiling only needs to determine whether a plan meets the SLO (i.e., $\mathbb{I}(\text{acc} \geq \text{acc}_{\text{slo}})$), and the required sample size depends on the gap between the plan's true accuracy and the SLO target. As shown in Figure 7(a), different SLO targets for the same plan require different amounts of profiling data (thus time): a large gap allows an early verdict with few samples (SLO 1), while a small gap demands many more samples for a confident decision (SLO 2) due to sample variance.

Compass proposes adaptive profiling that terminates once the SLO verdict is statistically certain, minimizing overhead. Specifically, Compass adopts the Student's t-test [64], continuously estimates confidence using the accuracy sample mean and its variance on the profiled data's distribution thus far, which reflects the likelihood that the plan satisfies the SLO threshold. Profiling stops once

confidence exceeds a target level (e.g., 99%), with theoretical guarantees. As shown in Figure 7(b), this adaptive profiling achieves both efficiency and strong correctness guarantees.

Dependency-shared Profiling. Candidate-plan search often revisits overlapping pipeline segments, introducing opportunities for incremental profiling. Specifically, changes to downstream operators do not affect upstream outputs. This structural dependency creates opportunities for incremental profiling, but existing advances [55, 62] treat accuracy profiling as a monolithic end-to-end procedure and therefore repeatedly profile identical segments.

Compass introduces dependency-shared profiling, a profiling-time optimization that identifies and reuses shared execution across candidate plans. The SLO Profiler maintains operator-level dependency information and selectively caches intermediate outputs at strategic boundaries, especially for GPU-expensive stages. When a new plan is proposed, the profiler performs a fast prefix match against cached subgraphs and skips redundant execution for all reused upstream portions, profiling only the operators that change.

The cache involved in dependency-shared profiling is scoped to a single query's planning session and discarded immediately after, keeping memory overhead *transient* and small (e.g., about 60 MB throughout a code-generation query). Note that GPU computation, not caching I/O, is the primary bottleneck during profiling, and storage can be scaled more cost-effectively across machines than expensive GPUs (§6.3); thus dependency-shared profiling delivers savings without introducing new bottlenecks.

Resource Trimming: Identify Resource-Pareto Plans. As the Search Optimizer tentatively over-commits resources, the resulting plans may not lie on the cost-optimal Pareto frontier preferred by service providers. To address this, for each identified SLO-compliant plan, the SLO Profiler refines the plan's resource costs via a greedy reduction process (Line 16–Line 18 in Algorithm 1). Based on the monotone property that reducing resources increases latency without affecting accuracy, Compass performs a multi-dimensional binary search to iteratively trim operator-level resources (e.g., reducing GPU share from 100% to 50% supported by MPS [38]). The trimming procedure halts once further reductions would violate latency constraints. This process produces a rich set of low-cost, SLO-compliant plans along the Pareto frontier, which supply the multi-pipeline planner for maximizing goodput.

4.3 Multi-Query Planner: Global Scheduling

So far, Compass efficiently identifies cost-effective plans for individual queries. However, when multiple queries share the same infrastructure, naively deploying each query's locally optimal plan can lead to severe global inefficiencies (§2.3): some machines or tiers become overloaded while others remain underutilized, ultimately reducing overall SLO goodput. To address this, Compass extends its single-query planning to multi-query global scheduling, targeting two practical deployment objectives: (i) maximize SLO service goodput under resource-constrained environments (e.g., private clusters), and (ii) minimize total deployment costs when resources can scale elastically (e.g., public clouds), efficiently serving all queries.

Maximizing SLO Goodput. In resource-constrained deployment spaces, it is often infeasible to serve all queries concurrently, so maximizing SLO goodput becomes critical. To preserve high-quality planning decisions due to decomposition, Compass goes beyond selecting the locally optimal plan for each query (i.e., the lowest-cost SLO-compliant plan). Instead, it leverages the full set of Pareto-optimal plans identified by the single-query planner. These candidates span different resource footprints and infrastructure tiers while meeting SLO requirements.

However, deciding query-plan combinations to maximize goodput is notoriously difficult, resembling a *graph bipartite matching* problem between the query set and plan set, combined with a *multi-dimensional maximum cardinality bin packing* problem to allocate operators into machines. In fact, even a relaxed version—where each query is restricted to its single best plan—is still NP-hard [12]. One potential formulation is as an integer linear program (ILP): deciding which queries to serve, selecting plans from available options, and assigning them to machines, subject to per-tier resource constraints, with the objective of maximizing total SLO goodput. Our technical report presents the full ILP [33]. Unfortunately, for N queries, each with P feasible plans, across T tiers and M machines per tier, this formulation yields $O(N \cdot P \cdot T \cdot M)$ binary variables, intractable for large-scale deployments.

To scale, Compass leverages a greedy heuristic. Each plan i has an *aggregate* cross-tier and machine resource demand cr_i and an associated SLO benefit w_i. Compass ranks all plans across queries based on their benefit-to-resource ratio (i.e., w_i/cr_i). The planner then iteratively allocates resources to the highest-ranked plans, adding a plan to the allocation if its corresponding query has not yet been scheduled. This process continues until available resources are exhausted.

Minimizing Total Deployment Costs. Unlike the setup with limited resources, service providers may rely on public clouds with elastic resource scaling to serve all queries, where minimizing deployment costs becomes the key. To address so, our planner extends our previous solution by modifying the sorting criteria. Instead of prioritizing based on the benefit-to-resource ratio, plans are ranked and selected based on their SLO benefit-to-monetary cost ratio.

Our greedy heuristic is flexible and can incorporate various practical considerations. For example, service providers can dynamically assign higher weights to queries as their pending time increases to prevent starvation and ensure fairness. Additional costs (e.g., migration overheads) can be injected into the cost term too. In fact, our multi-query planner offers a theoretical guarantee:

> THEOREM 1. *Let $A(t)$ and $C(t)$ denote the goodput and deployment cost achieved by our greedy planner at time t. $A_{\text{opt}}(t)$ and $C_{\text{opt}}(t)$ denote the optimal values. Then:*
>
> *(1) In cloud-only setups, $A(t) \geq \frac{1}{2} A_{opt}(t)$;*
>
> *(2) Let d be the number of tiers, $C(t) \leq \frac{11}{9} \times C_{opt}(t) + O(d)$.*

Our technical report provides the detailed proof [33]. It shows that the deployment cost from our greedy planner is within a competitive ratio of $\frac{11}{9}$ to the optimal. Indeed, our empirical evaluations show Compass achieves comparable performance with oracle

Table 2: Compass supports representative, real-world compound AI applications.

Task	Model Family	Space
Live Chat (LVC)	MLLM + LLM	~ 7K
Code Gen. (ACG)	LLM agents	~ 1K
Video Caption. (DVC)	ASR + captioner	~ 23K
Tracking (VT)	Detector + classifier	~ 6.5K
Speech Rec. (SR)	ASR models	~ 6.4K

Table 3: Our evaluation uses realistic multi-tier infrastructure, including 16 A100s and 16 V100s across tiers.

Tier	GPU	Price
Cloud	4× A100 (80GB)	20.3$/h
Near-Cloud	4× V100 (16GB)	8.0$/h
Near-Edge	1× T4 (16GB)	1.3$/h
Edge	Commodity GPUs/SoCs	Free

method (i.e., ILP with unlimited search time), while preserving scalability, even in large-scale deployments (§6.2 and §6.4).

Online Adaptation. In real-world deployments, query input distributions and runtime environments may change over time (e.g., lighting changes or network fluctuations in GenAI chat [11]), risking SLO violations. Despite various potential dynamic sources (adaptation needs) such as multi-tier resource changes and data distribution shifts, online adaptation overhead primarily boils down to three sources: (i) single-query search adaptation, (ii) multi-query scheduling adaptation, and (iii) execution-level migration. Building on existing infrastructure support for pipeline migration [17, 24, 62], the core challenge becomes how to efficiently replan in the wild. When triggered by the service monitor [17] or user intervention, Compass warm-starts the previously learned CAMO model, reusing prior knowledge to quickly correct stale estimations and adapt to the new environment. Our component-wise evaluations show that Compass's robust responsiveness in seconds (§6.4).

5 IMPLEMENTATION

We implemented a Compass prototype in roughly 2,800 lines of Python, fully compatible with Kubernetes [24] and LangChain [30], and supporting existing ML stacks with minimal API changes.

Interface, Runtime and Performance Isolation. Compass exposes simple APIs for specifying SLOs, covering the majority of compound AI workflows; an interface example is available in our technical report [33]. Our current implementation uses Kubernetes to manage the cluster and containers for each operator. For the underlying inference engine of LLM workflows, we use vLLM [28]. We leverage Linux cgroups and virtualization technologies like NVIDIA MPS and MIG and AMD MxGPU to enforce performance isolation, thus avoiding performance degradation from resource contention. Our implementations show only about 7% performance interference with temporal GPU sharing. Note that to further mitigate potential contention effects, we enable conservative resource partitioning during profiling. Rather than relying on fine-grained fractional allocations, we restrict GPU slices to coarse levels (e.g., 25%, 50%, 75%, 100%). Our Kubernetes implementation supports distributing operators across machines, connected via gRPC. Following existing advances [17, 62], we maintain standby machines to minimize latency. We provide a detailed runtime latency breakdown, including state migration (§6.3).

Fault Tolerance. Compass periodically and asynchronously records profiled configurations to a checkpoint file. In case of failure, Compass can resume execution from the last saved state, reducing redundant profiling efforts. For backend fault tolerance, we rely on cluster management tools such as Kubernetes to handle failures.

6 EVALUATION

We evaluate Compass on five representative compound AI applications (e.g., agentic video chat) across realistic deployment spaces (e.g., cloud-native and edge-cloud infrastructures), showing:

- Compass achieves over 2.4–5.1× better service goodput, and slashes deployment costs by 3.8–4.5× (§6.2).
- Compass accelerates single-query planning by 4.2–10.5×, achieving second-level responsiveness (§6.2).
- For multiple-query planning, Compass effectively scales to thousands of queries while achieving *close-to-optimal* global planning performance (§6.2-§6.3).
- Compass improves performance across varied settings and reacts to dynamics in seconds (§6.4).

6.1 Methodology

Realistic Infrastructure Setup. Compass targets large-scale deployments, but full-scale experiments are costly and hard to reproduce. We therefore use a mid-scale, multi-tier setup that captures cross-tier effects (Table 3): cloud, near-cloud, near-edge, and edge devices with commodity GPUs/SoCs. This heterogeneous deployment allows us to realistically emulate multi-tier workloads.

By default, we report on a medium deployment with 4 cloud and 4 near-cloud machines (16 A100 and 16 V100 GPUs). We also simulate larger deployments with 32 cloud and 32 near-cloud machines (128 A100 and 128 V100 GPUs) using traces collected above, and evaluate cloud-only and four-tier variants in §6.4.

Additional setup details, including edge devices and bandwidths, are available in our technical report [33].

Datasets, Queries, SLOs, and Loads. We evaluate Compass using five popular compound AI applications with real-world workloads (Table 2). Real-time applications, like live agent chat, visual tracking, and speech recognition, desire fast responsiveness, thus being *latency-critical* in responsiveness. While agentic code generation and video captioning are batch applications targeting the most cost-effective plans, thus *throughput-critical*.

We use realistic datasets with diverse input distributions and popular models; our technical report provides additional details [33].

We follow query workloads and SLOs inspired by recent advances in model-serving planning [43, 57, 58], such as *"track and imitate a user's body movements during push-ups and provide instant corrective feedback on muscle engagement"* in live agentic video chat. To construct realistic SLO requirements, we first identify the Pareto-optimal plans for each query via exhaustive search. We then relax and sample the accuracy and latency performance of plans on the Pareto frontier to generate diverse SLOs: by default, accuracy is set to 85% and latency to 1.5× the sampled points. Uniform sampling

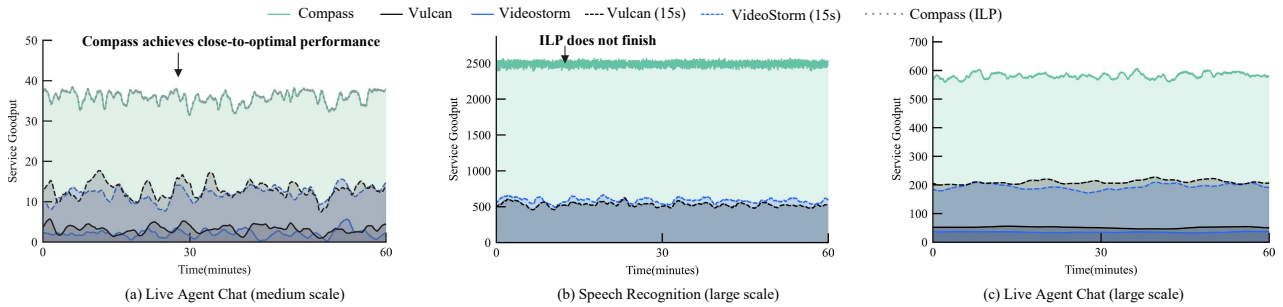


Figure 8: In resource-constrained deployments, Compass improves service goodput across scales and applications, achieving performance close to the ILP optimal. It outperforms baselines that require 3× longer response times—15 seconds render real-time service impractical.

of these Pareto-optimal plans ensures coverage of a broad search space. We further evaluate varying SLO settings (§6.4).

User query arrivals follow Microsoft's ML-serving traces [53], scaled proportionally to our setup's total resource capacity. More specific, to simulate realistic system load, we use a Pareto-optimal plan to control query arrival rates. We define the cluster capacity as the maximum number of Pareto-optimal plans that can be served concurrently. System load **1** corresponds to an arrival rate matching this capacity, ensuring the cluster is fully utilized without overload. In practice, baselines cannot achieve Pareto-optimal plans, resulting in slight overload, which provides a meaningful comparison point. We validate the impact of different system loads in Section 6.4.

Baselines. State-of-the-art methods focus on either single-tier or single-query settings. We extend them into *stronger baselines* that operate in multi-tier, multi-query environments:

- *Vulcan* [62]: A state-of-the-art query planner designed for cross-tier, single-query live ML analytics. It encodes operator placement and configuration search into an aggregate utility function.
- *VideoStorm*: a cloud-only multi-query planner that uses hill climbing [57].

We augment Vulcan with our multi-query planner to support many queries, and extend VideoStorm to support cross-tier settings by incorporating placement search into its hill-climbing method, alongside our multi-pipeline planner. Our results are consistent with prior reports of the baselines.

Metrics. We measure response time (time to first SLO-compliant plan), profiling cost, SLO service goodput under limited resource capacity, and deployment cost for sustaining all queries.

We report the mean values over five runs per experiment.

6.2 End-to-End Performance

We evaluate Compass in online multi-query settings. For *latency-critical* queries, we use a 5-second planning deadline, matching live-service startup constraints [19, 35, 43]. Baselines also get a relaxed 15-second timeout. For *throughput-critical* queries, we use a 10 A100-GPU-hour profiling budget (~$37 in GCP) per query. We later present ablation studies under different response time and profiling cost budgets (§6.4).

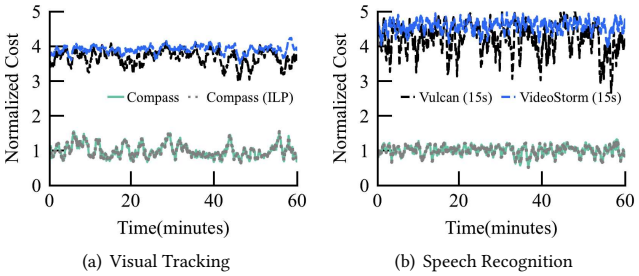


Figure 9: In cost-constrained deployment, Compass reduces resource monetary costs in supporting all queries.

Compass achieves SLO service goodput close to optimal. Figure 8 shows that, over a 60-minute service window, Compass achieves 15–61× higher average service goodput compared to Vulcan and VideoStorm in both medium- and large-scale deployments. Improvements are consistent across tasks and sustained over time. Even when compared to Vulcan (15s) and VideoStorm (15s), Compass improves average goodput by 2.4–5.1×, respectively. Note that 15 seconds have rendered many real-time services impractical [43].

Crucially, Compass delivers performance comparable to ILP-based scheduling, showing that our greedy scheduler approximates the global optimum. Unlike ILP solvers, which do not scale to large deployments, Compass's scheduling algorithm scales sub-linearly; our technical report reports detailed scalability results [33]. We later validate Compass's comparable performance with an exhaustive search-based oracle (Figure 13(d)).

Compass slashes deployment costs. We next evaluate the resource-abundant deployment scenario (e.g., utilizing cloud scaling services). Monetary cost is measured using Google Cloud Platform prices. For *latency-critical* tasks, we report the real-time normalized monetary cost over 60 minutes; for *throughput-critical* tasks, we report the unit (per hour) deployment cost of a single query pipeline. Figure 9 shows that Compass lowers the monetary cost for serving all queries by 3.8–4.5× compared to baselines. Figure 11 shows that Compass finds a 2.0–6.0× cheaper deployment plan in time.

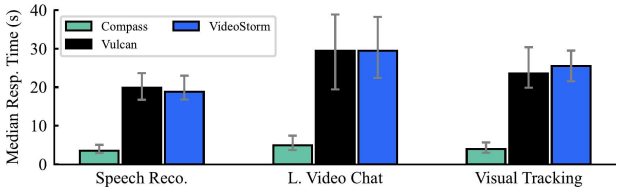
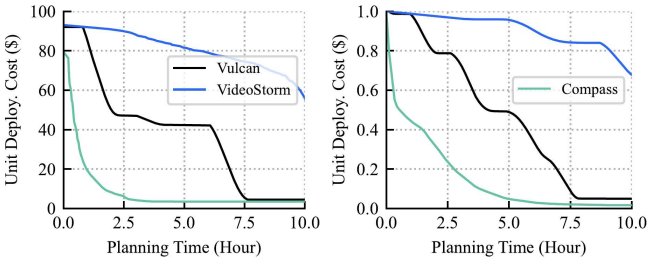


Figure 10: Compass improves query responsiveness in single-query planning. Error bars show 25% - 75% percentile.



(a) Dense Video Captioning (b) Agentic Code Generation

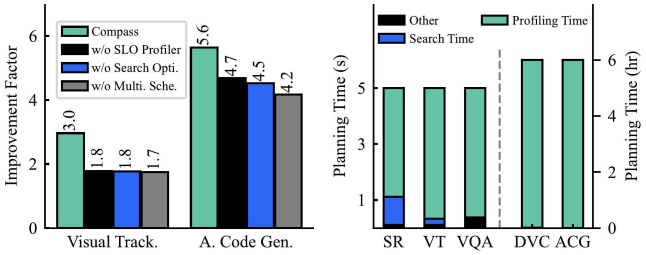
Figure 11: Compass reduces planning (profiling) costs for throughput-critical applications, in A100 GPU hours.

Compass enables second-level responsiveness while reducing planning expense. Figure 10 shows Compass identifies the first SLO-compliant plan in five seconds, 4.2–10.5× faster than Vulcan and 6.0–12.3× faster than VideoStorm. This fast responsiveness is important for user engagement in real-time services [53]. Moreover, Compass spends 6× less time finding a comparable plan (Figure 11) for *throughput-critical* queries, meaning substantial profiling cost savings considering high query volumes.

6.3 Performance Breakdown

Breakdown by system components. We analyze the improvement factor of Compass's each component: (i) *Compass w/o SLO Profiler*: removes *SLO Profiler*, using fixed sampling without prefix cache, (ii) *Compass w/o Search Optimizer*: removes history warm start and CAMO, using BO in Figure 6 (left), and (iii) *Compass w/o Multi-query Planner*: uses Yarn [50] scheduling (admits queries in a first-come-first-served fashion) for many queries, using their locally optimal plans. We use VideoStorm-based local planning as the 1× baseline, with the same planning time budgets. Figure 12(a) shows each component of Compass achieves 1.7–4.7× comparable improvements, contributing to 3.0–5.6× end-to-end improvements.

Breakdown by planning runtime. We next decompose the single-query planning overhead, primarily consisting of three aspects: (i) *Profiling Time*: Time spent on SLO profiler, profiling the plan's SLO performance on GPUs (e.g., model inference overhead); (ii) *Search Time*: Time spent on the Search Optimizer, including CAMO updates and plan proposing; and (iii) *Other*: Miscellaneous system overhead, such as setting up connections within the Kubernetes backend. This includes time to spin up pre-warmed containers.



(a) End-to-end Perf. Breakdown. (b) Planning Runtime Breakdown.

Figure 12: Performance breakdown of Compass.

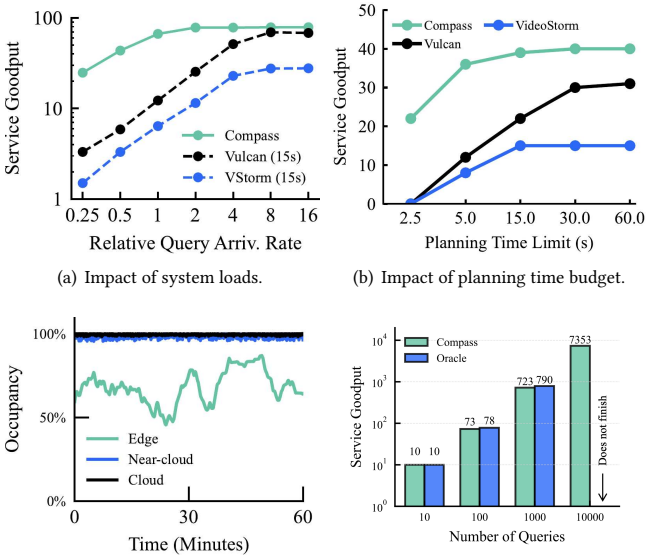
Figure 12(b) shows that for latency-critical tasks, profiling dominates the time budget, accounting for over 80% of the total planning overhead. For throughput-critical tasks, profiling consumes over 99% of the planning time. These results reinforce the importance of minimizing profiling cost. They also highlight the efficiency of CAMO, which requires only 40-100 ms per proposing step.

Breakdown by resource occupancy. We next break down the scheduling decision. We measure **resource occupancy**, representing the percentage of queries using different tiers (e.g., 50% of the edge occupancy means, at that time, running queries are using 50% of edge devices). As shown in Figure 13(c), Compass is able to keep near-cloud and cloud machines occupied to make sure SLO is met, while also allocating edge devices when possible, saving cloud resource to potentially accommodate more queries.

6.4 Ablation Study

Impact of system loads. We deploy a visual tracking task and report the service goodput to study the impact of system load (by default **1**, detailed in §6.1). Loads > 1 indicate overload (queuing), while < 1 indicate underutilization. Figure 13(a) shows that (i) Compass consistently delivers higher goodput than baselines, and (ii) as load exceeds 1, the improvement narrows due to resource saturation (i.e., goodput capped by capacity but not Compass's capability), with baselines appearing to catch up because we are sampling queries uniformly, thus it can prioritize simpler, less resource-intensive queries. (iii) the gap between Compass and baselines is largest when the load is light (i.e., 0.25). We delve into that and find out that, despite having a 15s budget (3× that of Compass), baselines cannot find a feasible solution for many requests. As a result, even if the cluster is idle, baselines are not able to deliver goodput because all its plans will violate SLO. This is consistent with Figure 10.

Impact of planning time limits. We evaluate Compass's performance under varying planning time budgets (i.e., the responsive requirement for real-time tasks) using the live agent chat task. Figure 13(b) shows that: (i) Compass consistently outperforms baselines across different planning time budgets; and (ii) Compass achieves comparable performance to Vulcan and VideoStorm with nearly 10× less search time, making real-time serving practical in seconds. The performance gap narrows as the planning time increases because Compass identifies near-optimal plans early.



(a) Impact of system loads.

(b) Impact of planning time budget.

(c) Resource occupancy breakdown.

(d) Compass is near-optimal.

Figure 13: Compass's performance and scheduling behavior in a wide range of settings.

Compass empirically achieves near-optimal performance. Figure 13(d) shows that Compass closely matches the near-optimal global planning oracle (exhaustive search + ILP packing), while scaling to tens of thousands of requests. Constructing the oracle itself is extremely expensive—we spend weeks profiling all candidate plans, and the ILP solver alone takes over 4 hours for just 1K queries—impractical for online serving. Our technical report includes the scheduling scalability experiment for Compass [33].

Impact of SLO requirements. We evaluate Compass under varying SLO latency and accuracy tightness. For latency SLO, we define easy, medium, and hard SLOs as 2.0×, 1.5×, and 1.1× the Pareto distribution mean. We evaluate Compass on a large-scale Speech Recognition task and observe that Compass achieves 2.5–5.0×, 1.5–3.5× higher goodput compared to the 15s-baseline.

Handling runtime dynamics. Our technical report shows that Compass can handle runtime latency and accuracy dynamics within seconds [33]. In the latency-dynamics experiment, we run an agent chat task in a scenario where the network outage drops the bandwidth between the edge and the cloud to 0.2 Gbps at 26s. Compass reacts swiftly, finding a new SLO-compliant plan within seconds.

Compass across deployment spaces. Compass's design generalizes to various deployment spaces. Our technical report includes both a single-tier (cloud-only) setup with 4 cloud machines, corresponding to VideoStorm's design space, and a four-tier setup [33]. In the single-tier setup, VideoStorm slightly outperforms Vulcan, consistent with their design goals: VideoStorm is optimized for cloud-only deployments, while Vulcan is for edge-cloud setups. Compass consistently improves performance in the four-tier.

7 RELATED WORK

ML Inference Optimization. Optimizing ML inference efficiency has gained significant attention recently. Existing advances span model parallelism [31, 37], quantization [14, 32], memory efficiency [28], and scheduling [45, 56] for in-cluster model serving [48]. However, they focus on serving single ML models, or query pipelines in the cloud [57]. Extending these ML serving efforts toward the edge will only grow more important (e.g., distributing data preprocessing closer to the edge [9]). Compass complements these advancements by providing multi-tier support (§5).

Edge-cloud AI Serving. Recent studies have explored moving computation to the edge to improve efficiency and privacy [25, 29], particularly in video analytics [8]. For instance, Chameleon [23] exploits spatiotemporal correlations in video streams to achieve optimal resource-accuracy tradeoffs. EFL [60] partitions video clips and then dynamically assigns these partitions to edge servers. Similarly, JellyBean [55] and Vulcan [62] optimize the placement and execution of cost-efficient pipeline operators across infrastructure tiers but are restricted to handling single queries. Compass serves for generic, compound AI workloads where multiple queries share infrastructure resources, outperforming existing advances (§6).

Efficient Configuration Search. Efficient configuration search has been extensively studied in the system domain, such as database [49], cloud hardware [3, 51], or ML hyper-parameter tuning [47]. Selecta [26] applies collaborative filtering to identify optimal cloud machine configurations. VideoStorm [57] searches query configurations and resource demands in cloud environments. Many learned algorithms have been used to facilitate the search process, such as Bayesian optimization [3, 46, 62], greedy hill climbing [57], and multi-armed bandit [21]. We propose a novel CAMO optimizer to tackle the considerably large search space for many queries.

8 CONCLUSION

This paper presents Compass, a novel SLO-aware query planner for large-scale compound AI serving deployment. By introducing a two-stage planning framework and leveraging plan similarity with precision-aware profiling, Compass efficiently determines operator placements, configurations, and resource allocations for many queries with diverse SLO requirements, system, and data characteristics. Our evaluations using real-world workloads show that Compass improves service goodput by 2.4–5.1× and reduces deployment costs by 3.8–4.5×, enabling real-time compound AI deployments.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive and insightful feedback. This work was supported in part by gifts from Cisco and Google, and by an award from NVIDIA Academic Program. It also utilized the Delta system at the National Center for Supercomputing Applications (NCSA) through allocation CIS240236 from the ACCESS program.

REFERENCES

[1] Neil Agarwal, Rui Pan, Francis Y Yan, and Ravi Netravali. 2024. Tarzan: Passively-Learned Real-Time Rate Control for Video Conferencing. *arXiv preprint arXiv:2410.03339* (2024).

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 117–134.

[3] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 469–482.

[4] Apidog. 2023. ChatGPT Free vs Paid: What's the Difference? https://apidog.com/blog/chatgpt-free-vs-paid/?utm_source=chatgpt.com. *Apidog* (2023). Accessed: May 17, 2026.

[5] azuremec 2023. Azure public multi-access edge compute (mec). https://azure.microsoft.com/en-us/solutions/public-multi-access-edge-compute-mec/. Accessed: May 17, 2026.

[6] Jehyeon Bang, Yujeong Choi, Myeongwoo Kim, Yongdeok Kim, and Minsoo Rhu. 2023. vTrain: A Simulation Framework for Evaluating Cost-effective and Compute-optimal Large Language Model Training. arXiv:2312.12391 [cs.LG]

[7] Marcus Basalla, Johannes Schneider, Martin Luksik, Roope Jaakonmäki, and Jan Vom Brocke. 2021. On latency of e-commerce platforms. *Journal of Organizational Computing and Electronic Commerce* 31, 1 (2021), 1–17.

[8] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. 2022. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 119–135.

[9] Dongqi Cai, Shangguang Wang, Chen Peng, Zeling Zhang, and Mengwei Xu. 2024. Recall: Empowering Multimodal Embedding for Edge Devices. *arXiv preprint arXiv:2409.15342* (2024).

[10] Ruei-Che Chang, Rosiana Natalie, Wenqian Xu, Jovan Zheng Feng Yap, and Anhong Guo. 2025. Probing the Gaps in ChatGPT's Live Video Chat for Real-World Assistance for People who are Blind or Visually Impaired. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. 1–14. https://doi.org/10.1145/3663547.3746319

[11] chatgpt-videochat 2023. ChatGPT can now see, hear, and speak. https://openai.com/index/chatgpt-can-now-see-hear-and-speak/. Accessed: May 17, 2026.

[12] Chandra Chekuri and Sanjeev Khanna. 2004. On multidimensional packing problems. *SIAM journal on computing* 33, 4 (2004), 837–851.

[13] compoundai-blog 2024. The Shift from Models to Compound AI Systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/. Accessed: May 17, 2026.

[14] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In *Advances in Neural Information Processing Systems*, Vol. 35. 30318–30332.

[15] Matthew W. G. Dye, Shawn C. Green, and Daphné Bavelier. 2009. Increasing Speed of Processing With Action Video Games. *Current Directions in Psychological Science* (2009).

[16] gcp-genai-gaming 2024. The evolution of play: From live to living games. https://cloud.google.com/blog/products/gaming/generative-ai-fuels-next-gen-living-games. Accessed: May 17, 2026.

[17] gcp-iot 2024. IoT platform product architecture on Google Cloud. https://cloud.google.com/architecture/connected-devices/iot-platform-product-architecture. Accessed: May 17, 2026.

[18] Google Developers Blog. n.d.. Google AI for game developers. https://developers.googleblog.com/en/google-ai-for-game-developers/. Accessed: May 17, 2026.

[19] Google Workspace. n.d.. Gemini in Google Meet. https://workspace.google.com/resources/ai-for-meetings/. Accessed: May 17, 2026.

[20] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 443–462.

[21] Daniel N Hill, Houssam Nassif, Yi Liu, Anand Iyer, and SVN Vishwanathan. 2017. An efficient bandit algorithm for realtime multivariate optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1813–1821.

[22] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. 2018. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 269–286.

[23] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*. 253–266.

[24] k8s 2026. Kubernetes: Production-Grade Container Scheduling and Management. https://kubernetes.io/. Accessed: May 17, 2026.

[25] Peter Kairouz, H. Brendan McMahan, Brendan Avent, AurÃllien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, AdriÃă GascÃşn, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub KoneÄŀnÃ¡, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, TancrÃĺde Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer ÃŰzgÃijr, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian TramÃĺr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2021. Advances and Open Problems in Federated Learning. *Foundations and Trends in Machine Learning* 14, 1–2 (2021), 1–210. https://doi.org/10.1561/2200000083

[26] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. 2018. Selecta: heterogeneous cloud storage configuration for data analytics. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 759–773.

[27] Ferdi Kossmann, Ziniu Wu, Alex Turk, Nesime Tatbul, Lei Cao, and Samuel Madden. 2024. CascadeServe: Unlocking Model Cascades for Inference Serving. arXiv:2406.14424 [cs.DC] https://arxiv.org/abs/2406.14424 Accessed: May 17, 2026.

[28] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*. 611–626. https://doi.org/10.1145/3600006.3613165

[29] Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient Federated Learning via Guided Participant Selection. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. 19–35.

[30] langchain 2026. LangChain. https://www.langchain.com/. Accessed: May 17, 2026.

[31] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 663–679.

[32] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, Vol. 6. 87–100.

[33] Banruo Liu, Wei-Yu Lin, Minghao Fang, Yihan Jiang, and Fan Lai. 2025. Circinus: Efficient Query Planner for Compound ML Serving. *arXiv preprint arXiv:2504.16397* (2025). https://doi.org/10.48550/arXiv.2504.16397 arXiv:2504.16397 [cs.DB] Accessed: May 17, 2026.

[34] Jiachen Liu, Zhiyu Wu, Jae-Won Chung, Fan Lai, Myungjin Lee, and Mosharaf Chowdhury. 2024. Andes: Defining and Enhancing Quality-of-Experience in LLM-Based Text Streaming Services. *arXiv preprint arXiv:2404.16283* (2024).

[35] Chiheng Lou, Sheng Qi, Chao Jin, Dapeng Nie, Haoran Yang, Yu Ding, Xuanzhe Liu, and Xin Jin. 2025. HydraServe: Minimizing Cold Start Latency for Serverless LLM Serving in Public Clouds. *arXiv preprint arXiv:2502.15524* (2025). https://arxiv.org/abs/2502.15524 Accessed: May 27, 2026.

[36] Chengfei Lv, Chaoyue Niu, Renjie Gu, Xiaotang Jiang, Zhaode Wang, Bin Liu, Ziqi Wu, Qiulin Yao, Congyu Huang, Panos Huang, Tao Huang, Hui Shu, Jinde Song, Bin Zou, Peng Lan, Guohuan Xu, Fei Wu, Shaojie Tang, Fan Wu, and Guihai Chen. 2022. Walle: An End-to-End, General-Purpose, and Large-Scale Production System for Device-Cloud Collaborative Machine Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 249–265.

[37] Yixuan Mei, Yonghao Zhuang, Xupeng Miao, Juncheng Yang, Zhihao Jia, and Rashmi Vinayak. 2025. Helix: Serving Large Language Models over Heterogeneous GPUs and Network via Max-Flow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25)*. 586–602. https://doi.org/10.1145/3669940.3707215

[38] NVIDIA. n.d.. NVIDIA Multi-Process Service. https://docs.nvidia.com/deploy/mps/index.html#/. Accessed: May 17, 2026.

[39] OpenAI. n.d.. ChatGPT Voice mode with live video. https://chatgpt.com/features/voice-with-video/. Accessed: May 17, 2026.

[40] openai-tts 2025. OpenAI: Introducing next-generation audio models in the API. https://openai.com/index/introducing-our-next-generation-audio-models/. Accessed: May 17, 2026.

[41] Kanchana Ranasinghe, Xiang Li, Kumara Kahatapitiya, and Michael Ryoo. 2025. Understanding Long Videos with Multimodal Language Models. In *International Conference on Learning Representations (ICLR)*. 11810–11835. https://proceedings.iclr.cc/paper_files/paper/2025/hash/1f591824a3bd840f74947fcda304c945-Abstract-Conference.html Accessed: May 17, 2026.

[42] Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. 2020. ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 81–92. https://doi.org/10.1109/ISPASS48437.2020.00018

[43] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 397–411.

[44] Junhee Ryu, Dongeun Lee, Kang G. Shin, and Kyungtae Kang. 2023. Fast Application Launch on Personal Computing/Communication Devices. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. 425–440.

[45] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. 2024. Fairness in Serving Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 965–988.

[46] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems* 25 (2012).

[47] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. 2015. Scalable Bayesian Optimization Using Deep Neural Networks. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, Vol. 37. 2171–2180.

[48] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. In *Proceedings of the 30th Symposium on Operating Systems Principles (SOSP '24)*. 590–606. https://doi.org/10.1145/3694715.3695964

[49] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. 1009–1024. https://doi.org/10.1145/3035918.3064029

[50] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing (SoCC '13)*. 1–16. https://doi.org/10.1145/2523616.2523633

[51] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 363–378.

[52] videommeds 2019. lmms-lab/VideoMME. https://huggingface.co/datasets/lmms-lab/Video-MME. Accessed: May 17, 2026.

[53] Jaylen Wang, Daniel S. Berger, Fiodar Kazhamiaka, Celine Irvene, Chaojie Zhang, Esha Choukse, Kali Frost, Rodrigo Fonseca, Brijesh Warrier, Chetan Bansal, Jonathan Stern, Ricardo Bianchini, and Akshitha Sriraman. 2024. Designing Cloud Servers for Lower Carbon. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 452–470. https://doi.org/10.1109/ISCA59077.2024.00041

[54] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Dan Li, Li Chen, Heyang Zhou, Linkang Zheng, Sen Zhang, Yikai Zhu, Yang Liu, Pengcheng Zhang, Kun Qian, Kunling He, Jiaqi Gao, Ennan Zhai, Dennis Cai, and Binzhang Fu. 2025. SimAI: Unifying Architecture Design and Performance Tuning for Large-Scale Large Language Model Training with Scalability and Precision. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX, 541–558. https://www.usenix.org/conference/nsdi25/presentation/wang-xizheng-simai Accessed: May 17, 2026.

[55] Yongji Wu, Matthew Lentz, Danyang Zhuo, and Yao Lu. 2022. Serving and optimizing machine learning workflows on heterogeneous infrastructures. *arXiv preprint arXiv:2205.04713* (2022).

[56] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 521–538.

[57] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live video analytics at scale with approximation and {Delay-Tolerance}. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 377–392.

[58] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. 2023. {SHEPHERD}: Serving {DNNs} in the wild. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 787–808.

[59] Li Zhang, Zhe Fu, Boqing Shi, Xiang Li, Rujin Lai, Chenyang Yang, Ao Zhou, Xiao Ma, Shangguang Wang, and Mengwei Xu. 2024. More is Different: Prototyping and Analyzing a New Form of Edge Server with Massive Mobile SoCs. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 285–302.

[60] Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. 2021. Elf: Accelerate High-resolution Mobile Deep Vision with Content-aware Parallel Offloading. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking (MobiCom '21)*. 201–214. https://doi.org/10.1145/3447993.3448628

[61] Xumiao Zhang, Anlan Zhang, Jiachen Sun, Xiao Zhu, Y. Ethan Guo, Feng Qian, and Z. Morley Mao. 2021. EMP: edge-assisted multi-vehicle perception. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking (MobiCom '21)*. 545–558. https://doi.org/10.1145/3447993.3483242

[62] Yiwen Zhang, Xumiao Zhang, Ganesh Ananthanarayanan, Anand Iyer, Yuanchao Shu, Victor Bahl, Z Morley Mao, and Mosharaf Chowdhury. 2024. Vulcan: Automatic Query Planning for Live {ML} Analytics. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1385–1402.

[63] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 193–210.

[64] Zhi-Jian Zhou, Jie Ni, Jia-He Yao, and Wei Gao. 2023. On the Exploration of Local Significant Differences for Two-Sample Test. In *Advances in Neural Information Processing Systems*, Vol. 36. 5262–5300. https://doi.org/10.52202/075280-0233

[65] zoom-aicompanion 2024. Zoom: AI Companion 2.0. https://news.zoom.us/ai-companion-2-0-launch/. Accessed: May 17, 2026.