Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

TANGO: Inter-Job and Intra-Job Orchestration for Distributed Learning on Heterogeneous Wireless IoT Clusters

Teh-Jen Sun
Kyung Hee University
Md Nahid Sultan
Kyung Hee University
Eui-Nam Huh
johnhuh@khu.ac.kr

Kyung Hee University

Article

Keywords: Distributed deep learning, Cluster scheduling, Wireless edge computing, Heterogeneous IoT, Resource orchestration

Posted Date: November 18th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11264552/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

TANGO: Inter-Job and Intra-Job Orchestration for Distributed Learning on Heterogeneous Wireless IoT Clusters

Teh-Jen Sun[1], Md Nahid Sultan[2], Eui-Nam Huh[2,3*]

[1]Department of Artificial Intelligence, Kyung Hee University, Yongin, 17104, Republic of Korea .
[2]Department of Computer Science and Engineering, Kyung Hee University, Yongin, 17104, Republic of Korea .
[3]Artificial Intelligence Research Center, Kyung Hee University, Yongin, 17104, Republic of Korea .

*Corresponding author(s). E-mail(s): johnhuh@khu.ac.kr;
Contributing authors: dlwmznzl1@khu.ac.kr; nahid@khu.ac.kr;

Abstract

Distributed learning on wireless Internet of Things (IoT) clusters requires coordination of shared communication resources and heterogeneous device capacities. Concurrent synchronous training jobs can generate overlapping communication bursts, while memory limits and thermal throttling alter their execution rates. We present TANGO, a hierarchical closed-loop co-scheduler that couples inter-job synchronization control with intra-job runtime adaptation. The inter-job controller coordinates collective timing and adjusts gradient accumulation to regulate synchronization frequency using congestion feedback. The intra-job controller adapts batch size, learning rate, and input-pipeline settings within an admissible configuration region, while reporting telemetry that updates inter-job decisions. We evaluate TANGO on a 12-node heterogeneous Raspberry Pi 5 testbed with nine vision and text workloads. Under a 50-job Poisson trace, TANGO reduces average job completion time by 16.8%, makespan by 18.0%, and P99 queueing delay by 19.7% relative to Sia. Differences in final accuracy from Sia remain below 1% point for the three reported workloads at the evaluated training budgets. Trace-driven simulations extend the evaluation to 2,048 nodes. These results support the usefulness of coupled communication and execution control for the evaluated contention-heavy edge-learning setting.

Keywords: Distributed deep learning, Cluster scheduling, Wireless edge computing, Heterogeneous IoT, Resource orchestration

1 Introduction

Deep learning at the network edge brings model execution closer to the environments in which data are generated [1–3]. The increasing computational complexity and memory demand of deep neural networks also place substantial pressure on resource-constrained edge platforms [4]. Beyond inference, models may require periodic or event-triggered updates as operating conditions and available data change [5]. Moving every update to a remote datacenter is not always compatible with limited uplink capacity, intermittent connectivity, or requirements to retain data within a local deployment [6, 7]. These considerations motivate investigating how available edge resources can support local model training and adaptation [8–10].

In this work, we consider a co-located wireless IoT cluster in which multiple training requests share a managed pool of embedded devices. The intended workload is local model refresh or adaptation, rather than large-scale model pretraining. Several models may require updates within the same deployment, creating concurrent jobs that compete for both computing resources and wireless communication capacity. The purpose of using embedded devices is therefore not to claim higher raw training throughput than a dedicated GPU server, but to investigate coordination within a resource-constrained setting where training is performed on the available local infrastructure. Our Raspberry Pi testbed provides a controlled realization of this setting. It is not intended to represent all IoT devices, applications, or deployment conditions.

Coordinating such a cluster requires accounting for interactions between communication and device execution [11, 12]. In synchronous data-parallel training, workers periodically exchange gradients through collective operations such as all-reduce [13, 14]. When independent jobs synchronize over a shared wireless medium, their communication phases can overlap and compete for the same transmission opportunities. The resulting synchronization time depends not only on the communication volume of an individual job, but also on when other jobs communicate. This motivates treating collective timing and synchronization frequency as scheduling variables, rather than considering communication only through a per-job iteration-time estimate.

Device heterogeneity introduces a second source of variability [15, 16]. Memory capacity constrains feasible local batch sizes, while differences in execution speed affect progress across synchronous workers. Sustained execution can also trigger thermal throttling on embedded devices [17]. These conditions change the intervals between collective operations and can disturb an initially useful communication schedule. Conversely, changing synchronization frequency through gradient accumulation changes the effective batch size and the interval over which local computation proceeds before communication. Communication coordination and local training adaptation are therefore coupled: decisions at one level change the runtime conditions observed at the other.

Existing training schedulers provide important foundations for resource allocation and adaptive execution. Pollux jointly considers system throughput and statistical efficiency, Lucid addresses the scheduling of deep-learning training jobs, and Sia extends goodput-oriented scheduling to heterogeneous clusters [18–20]. The focus of this work is the coupling of synchronization timing, synchronization frequency, and device-side adaptation through runtime feedback in a shared wireless edge environment.

We present TANGO, a hierarchical closed-loop co-scheduling framework for this setting. At the inter-job level, TANGO coordinates collective timing through phase assignments and regulates synchronization frequency by adjusting gradient accumulation steps. These decisions use feedback describing synchronization behavior and wireless congestion. At the intra-job level, per-job coordinators adapt batch size, learning rate, and, when supported, input-pipeline parameters according to device and execution telemetry. Training-configuration changes are restricted to an admissible region around the baseline configuration. This restriction is an operational safeguard for adaptation, rather than a theoretical guarantee that all configurations preserve convergence or final accuracy.

The two control levels exchange information throughout execution. Intra-job coordinators report synchronization footprints, execution-time proxies, and congestion indicators. The inter-job scheduler uses these summaries to update phase and accumulation decisions, which in turn change the execution conditions observed by the coordinators. This feedback structure allows communication coordination to respond to device-side drift and allows local adaptation to account for the communication schedule. TANGO's contribution lies in this integrated control structure and its evaluation on a wireless embedded-device testbed, rather than in phase shifting, gradient accumulation, or batch-size adaptation considered individually.

We evaluate TANGO on a 12-node Raspberry Pi 5 testbed with heterogeneous memory capacities, mixed 2.4 GHz and 5 GHz wireless connections, and passive cooling. The evaluation uses nine vision and text workloads and a 50-job Poisson arrival trace designed to exercise a contention-heavy operating regime. Under this trace, TANGO reduces average job completion time (JCT) from 2.38 h to 1.98 h relative to Sia, corresponding to a 16.8% reduction. Makespan decreases from 5.00 h to 4.10 h, and P99 queueing delay decreases from 4.11 h to 3.30 h. We separately report queueing and service-time outcomes to distinguish improvements in end-to-end completion from changes in execution time. Accuracy results are reported for three workloads at their specified training budgets, and a separate 25-job ablation compares the inter-job, intra-job, and combined configurations. Trace-driven simulations extend the evaluation to 2,048 nodes and are interpreted as an extrapolatory study rather than evidence from a physical deployment at that scale.

The main contributions of this paper are as follows:

- We formulate a coupled scheduling problem for multi-job wireless edge training, relating collective timing and synchronization frequency to heterogeneous device execution and effective-batch configuration.
- We design and implement TANGO, a hierarchical feedback controller that combines inter-job synchronization coordination with telemetry-driven intra-job adaptation through explicit runtime interfaces.

- We evaluate TANGO on a heterogeneous 12-node wireless testbed and through trace-driven simulations, reporting end-to-end performance, queueing and service-time decomposition, workload-dependent trade-offs, component ablations, and representative accuracy outcomes.

2 Related Work

2.1 Cluster Scheduling for Deep Learning

Cluster schedulers for deep learning address resource sharing, job placement, and completion-time objectives in multi-tenant environments [21, 22]. Gandiva exploits the repetitive execution patterns of training jobs to support GPU time-sharing and migration across resources [23]. Tiresias combines scheduling and placement policies to reduce job completion time without requiring precise knowledge of job duration [24]. Themis introduces finish-time fairness and accounts for placement sensitivity when allocating resources among competing jobs [25]. Lucid takes a non-intrusive approach, using job profiling and interpretable models to guide job packing, prioritization, and resource orchestration [19].

These systems establish the importance of coordinating jobs rather than optimizing each training process independently. TANGO addresses a related runtime coordination problem: how concurrently executing jobs should share a wireless communication medium while their device-side execution rates change. Its distinguishing control variables are collective timing and synchronization frequency, coupled with local training adaptation. This focus differs from deciding only which jobs run and where they are placed, but does not imply that resource-allocation policies or placement-sensitive schedulers disregard communication costs.

2.2 Adaptive Distributed Training

Adaptive training systems coordinate resource allocation with training configuration [26, 27]. Pollux models goodput by combining system throughput with statistical efficiency and jointly adapts cluster resources and per-job training configurations [18]. Sia extends goodput-oriented scheduling to heterogeneous resources, matching jobs and their configurations to different GPU types and counts as cluster conditions change [20]. These systems demonstrate that both per-job adaptation and coordination between job-level and cluster-level decisions are established approaches.

At the runtime level, KungFu provides adaptation policies that use monitored training statistics to modify hyperparameters and system parameters during execution [28]. AdaScale addresses learning-rate adaptation when the batch size changes during distributed training [29]. Sailor explores training configurations for dynamic, heterogeneous, and geo-distributed resources using runtime and memory-footprint estimation [30]. At the device level, self-supervised instance filtering and error-map pruning reduce the computational cost of on-device CNN training [10]. Together, these studies show that training efficiency depends on interactions between resource availability, execution configuration, and optimization behavior [31].

TANGO focuses on these interactions within shared wireless contention domains. Changing local batch sizes or input-pipeline settings changes the interval between collective operations, while adjusting gradient accumulation changes both synchronization frequency and effective batch size. Accordingly, TANGO exchanges synchronization footprints and execution telemetry between its inter-job and intra-job controllers. The contribution is not adaptive batch sizing or hierarchical coordination alone, but their integration with congestion-driven control of collective timing and frequency in a wireless embedded-device cluster.

2.3 Mitigating Communication Bottlenecks

Communication-efficient training can reduce the data exchanged during synchronization or change how workers coordinate their updates. QSGD uses gradient quantization and encoding to reduce communication volume [32]. Efficient sparse collective communication has also been developed for distributed deep learning [33]. These approaches address the representation or execution of gradient exchange. TANGO instead focuses on coordinating the timing and frequency of exchanges across concurrent jobs. Reducing communication volume and coordinating communication schedules are therefore distinct, potentially complementary approaches. The present study does not evaluate their joint optimization.

Synchronization semantics provide another design dimension. The stale synchronous parallel parameter server allows workers to access bounded-staleness parameter values, reducing waiting while providing correctness guarantees under its stated assumptions [34]. TANGO takes a different approach: workers retain a coordinated synchronous update schedule, while gradient accumulation changes how many local micro-steps are performed before synchronization. Maintaining this coordination does not, by itself, establish equivalence to the optimization trajectory of a fixed-batch baseline.

2.4 Distributed Learning on the Wireless Edge

Federated learning provides an important approach to learning from data distributed across edge devices [35, 36]. Federated averaging aggregates locally computed model updates without requiring the underlying training data to be collected centrally [37]. Oort improves federated training through participant selection that considers both the utility of a client's data and its ability to complete training quickly [38]. These studies address the coordination of participating devices and local learning [39, 40], rather than assuming that all edge-learning deployments follow the same communication pattern.

Wireless distributed learning has also been studied through coded computation. Joint optimization of coding and edge-node scheduling addresses uncertain wireless connections, heterogeneous computing capacities, and straggling [41]. This approach uses redundancy and node selection to reduce the impact of delayed participants. TANGO addresses a different coordination mechanism: the scheduling of collective operations generated by multiple synchronous DDP jobs sharing a managed wireless cluster.

TinyMOA supports resource-aware stream analytics by configuring machine-learning operators for local and remote edge resources [42].

The intended scope of TANGO is therefore co-located, multi-job edge training, such as repeated local model updates on an available pool of embedded devices. It does not replace federated participant selection or coded-computing schemes, nor does it imply that synchronous DDP is preferable for all wireless deployments. Within its target setting, TANGO connects domain-level communication coordination with device-level runtime adaptation. This positions the framework as a systems approach to managing concurrent edge-learning workloads, with applicability bounded by the workload and deployment conditions examined in the evaluation.

3 Problem Formulation

We consider multi-job synchronous distributed training on a heterogeneous wireless IoT cluster. The scheduling objective is to reduce average job completion time by coordinating collective communication across jobs and adapting execution configurations within each job. The following formulation describes the interactions that motivate TANGO's runtime controls. It is a control-oriented abstraction, rather than an exact model of wireless channel behavior or a claim of globally optimal scheduling.

System notation and control variables. Let $\mathcal{J}$ denote the set of submitted jobs, $\mathcal{J}_{\text{act}}(t)$ the jobs executing at wall-clock time t, and $\mathcal{D}$ the set of wireless contention domains. For an active job k, $\mathcal{N}_k$ denotes its assigned workers and $d(k) \in \mathcal{D}$ its associated contention domain. The formulation uses this job-to-domain association to describe shared-medium contention at the scheduling level. It does not model individual wireless transmissions. We use k to index a job under consideration, i to index jobs in summations, n to index workers, and d to index contention domains.

TANGO controls three groups of variables. First, $\Phi = \{\phi_k\}$ contains normalized phase assignments, where $\phi_k \in [0, 1)$ specifies a relative synchronization offset. Second, $\mathbf{G} = \{G_k\}$ contains gradient accumulation settings, where $G_k \in \mathbb{Z}_{\geq 1}$ is the number of local micro-steps performed before a scheduled gradient synchronization. A micro-step denotes one local forward/backward pass. Third, Θ_k contains intra-job settings, including the per-worker batch sizes $\{B_{k,n}\}$, the job-level learning rate η_k, and the supported input-pipeline parameters. The controls are updated during execution. Their explicit time arguments are omitted where the current settings are understood.

Communication and shared-medium demand. Let $I_i(t) \in \{0, 1\}$ indicate whether active job i is in its collective communication phase at time t under the current controls. Let $b_i(t) \geq 0$ denote its offered communication demand while in that phase, expressed in bytes per second. The aggregate offered load in contention domain d is represented as

$$L_d(t) = \sum_{\substack{i \in \mathcal{J}_{\text{act}}(t) \\ d(i)=d}} I_i(t)\, b_i(t). \tag{1}$$

The notation $L_d(t)$ suppresses its dependence on Φ, $\mathbf{G}$, and the intra-job configurations. Phase assignments affect the timing of the indicators $I_i(t)$, while accumulation and execution settings affect how frequently communication phases occur.

Let $D_k(\Theta_k)$ denote the communication volume associated with one synchronization round of job k, and let $C_d(t) > 0$ denote a reference effective service rate for domain d, in bytes per second. We represent the communication duration of a synchronization round as

$$T_{\text{comm}}^{(k)}(t; \Phi, \mathbf{G}, \Theta_k) \approx \frac{D_k(\Theta_k)}{C_{d(k)}(t)} \, g\left(\frac{L_{d(k)}(t)}{C_{d(k)}(t)}\right), \tag{2}$$

where $g(\cdot)$ is a dimensionless, non-decreasing congestion penalty. This penalty represents additional delay associated with competing communication demand, including shared-medium access and retransmission effects. Its monotonicity is a modeling assumption. No particular nonlinear or parametric form is assumed.

Equation (2) separates the communication volume of a job from the shared-medium conditions under which that volume is exchanged. The dependence $D_k(\Theta_k)$ allows for configuration changes that affect the communicated payload. It does not imply that every batch-size or learning-rate change alters the number of gradient bytes. Such changes can instead affect communication indirectly by changing the execution interval between synchronization rounds.

Under this abstraction, overlapping communication phases can increase aggregate demand relative to available service. This motivates coordinating synchronization timing and frequency. The equation does not, by itself, establish how much a particular phase assignment reduces collective latency or end-to-end JCT.

Compute variability and synchronization intervals. For worker $n \in \mathcal{N}_k$, let $T_{\text{comp}}^{(k,n)}(t; \Theta_k)$ denote the local compute duration of a micro-step. We represent the job's straggler-limited compute duration by

$$T_{\text{comp}}^{(k)}(t; \Theta_k) = \max_{n \in \mathcal{N}_k} T_{\text{comp}}^{(k,n)}(t; \Theta_k). \tag{3}$$

This abstraction captures the influence of heterogeneous and time-varying worker execution rates. Let $\tau_k(t; \Theta_k)$ denote a recent job-level estimate of the representative micro-step compute duration. For an accumulation window containing G_k micro-steps, the corresponding local compute duration is approximated by

$$T_{\text{local}}^{(k)}(t) \approx G_k \, \tau_k(t; \Theta_k). \tag{4}$$

The local compute duration in Eq. (4) is not the complete wall-clock interval between synchronization rounds. We denote the measured interval between the starts of consecutive synchronization rounds by $\widehat{T}_k(t)$. This interval reflects the realized execution schedule, including the effects of input stalls, collective communication, and any injected waiting. It is distinct from $T_{\text{comm}}^{(k)}$, which denotes the duration of the communication phase itself. In particular, a delay applied before every synchronization round contributes recurrent execution cost and can change $\widehat{T}_k(t)$. It cannot be treated solely as a one-time phase displacement.

Over a recent execution window in which the communication volume per round is approximately stable, the average communication demand of job k can be represented

as

$$\bar{\lambda}_k(t) \approx \frac{D_k(\Theta_k)}{\widehat{T}_k(t)}. \tag{5}$$

The compute-only expression $D_k(\Theta_k)/(G_k\tau_k(t;\Theta_k))$ omits communication, input, and pacing effects and is therefore not used as an exact estimate of realized wireless load. The coupling remains important: changing G_k changes the amount of local work between synchronizations, while device-side adaptation changes the rate at which that work is completed. Both can change the measured synchronization interval and the communication demand observed by other jobs.

3.1 Joint Objective

Let A_k and E_k denote the arrival and completion times of job k, respectively. Its job completion time is

$$\mathrm{JCT}_k = E_k - A_k. \tag{6}$$

Because JCT includes queueing as well as execution, a reduction in JCT need not arise entirely from shorter collective communication. Admission and placement determine which jobs execute concurrently and which contention domains they share. The formulation below focuses on the runtime coordination variables applied to those jobs. Queueing and queue-excluded service time are reported separately in the evaluation.

Gradient accumulation couples the communication controls to the training configuration. Let $B_k = \sum_{n\in\mathcal{N}_k} B_{k,n}$ denote the aggregate batch size across workers for one micro-step. When each worker's batch size is held fixed within an accumulation window, the effective global batch size is

$$B_k^{\mathrm{eff}} = G_k B_k = G_k \sum_{n\in\mathcal{N}_k} B_{k,n}. \tag{7}$$

This quantity describes the number of sample contributions accumulated before a synchronized update. It does not by itself specify gradient normalization when local batch sizes differ across workers.

Let $\Omega \subset [0,1)$ denote the finite set of candidate phase assignments, $G_{\max} \geq 1$ the configured accumulation cap, and $\mathcal{C}_k$ the admissible effective-batch/learning-rate region for job k. The region $\mathcal{C}_k$ is defined around the workload's baseline training configuration, as described in Section 4.4. The scheduling objective can then be expressed as

$$\min_{\Phi,\mathbf{G},\{\Theta_k\}} \quad \frac{1}{|\mathcal{J}|}\sum_{k\in\mathcal{J}}(E_k - A_k) \tag{8}$$

$$\text{s.t.} \quad \phi_k \in \Omega, \tag{9}$$

$$G_k \in \mathbb{Z}_{\geq 1}, \qquad G_k \leq G_{\max}, \tag{10}$$

$$\left(B_k^{\mathrm{eff}}, \eta_k\right) \in \mathcal{C}_k, \tag{11}$$

where the constraints apply to each active job whenever the corresponding controls are updated. Per-worker settings are additionally subject to the resource-feasibility checks described in Section 4.3.

These constraints describe the available control choices, rather than guarantees about wireless utilization or final model accuracy. Restricting $(B_k^{\text{eff}}, \eta_k)$ to $\mathcal{C}_k$ provides an operational safeguard against unrestricted training-configuration changes. Whether the resulting execution maintains the desired training quality remains an empirical question. Likewise, the communication controller seeks to relieve congestion without assuming that an instantaneous load bound can always be satisfied under changing wireless conditions.

Connection to runtime control. TANGO pursues Eq. (8) through the feedback rules in Section 4, rather than solving the full scheduling problem analytically. The reported synchronization footprints $\hat{p}_k(\cdot)$ and periods $\widehat{T}_k$ inform phase assignment. Normalized RTT, retransmission-proxy, and synchronization-duration measurements contribute to the congestion score $\rho_d(t)$, which guides changes to G_k. Worker telemetry guides changes to Θ_k within the admissible configuration region.

The score $\rho_d(t)$ is an operational congestion indicator, not an identified value of the abstract ratio $L_d(t)/C_d(t)$. Similarly, the controller does not require an explicit fitted function $g(\cdot)$. The communication model identifies the quantities that are coupled, while the implementation uses measured execution and congestion signals to adjust the available controls. The resulting performance is evaluated through observed completion times, service times, and training outcomes rather than an optimality or convergence guarantee derived from the formulation.

4 TANGO: System Design

TANGO implements the coupled control structure introduced in Section 3. The inter-job controller coordinates collective timing and synchronization frequency across jobs sharing a wireless contention domain. The intra-job controller adapts execution settings using worker telemetry and reports the resulting synchronization behavior. Together, these controllers address the interaction between shared communication demand and time-varying device execution rates.

The design separates the coordination objective from the runtime mechanisms used to pursue it. Phase assignment provides a timing policy, gradient accumulation provides a synchronization-frequency control, and intra-job adaptation changes the local execution configuration. Their combined effect is assessed through the measured behavior of the resulting execution, rather than through an assumption that each control independently improves job completion time.

4.1 Architecture: Layers and Interfaces

Figure 1 organizes TANGO into three layers connected by explicit control and telemetry interfaces.

Layer 1: Orchestration. The Dynamic Dispatcher tracks job arrivals, resource availability, and contention-domain membership. It maintains the active-job set and determines feasibility and placement with respect to the available resource headroom. The Interference Orchestrator operates on the resulting active jobs and computes their phase assignments ϕ_k and gradient accumulation settings G_k.

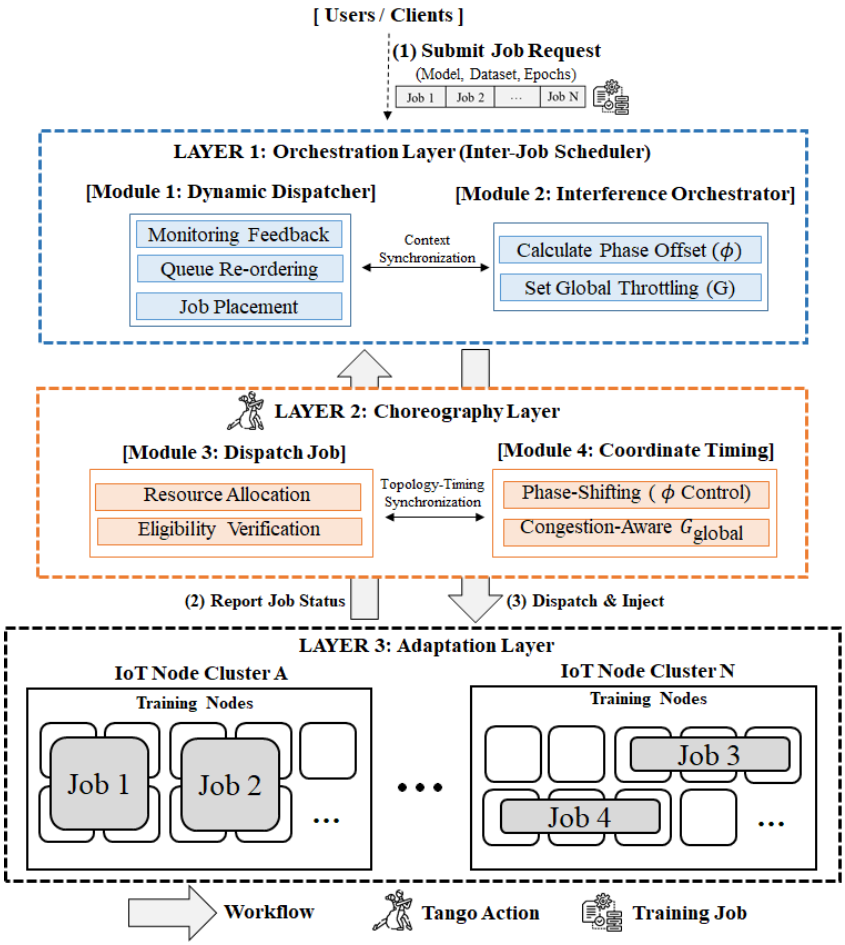


Fig. 1 TANGO architecture. The orchestration layer maintains cluster state and computes inter-job coordination signals. The choreography layer provisions workers and conveys the coordination decisions to job runtimes. Per-job coordinators in the adaptation layer adjust execution settings and return summarized telemetry.

These functions have different effects on end-to-end completion. Admission and placement determine which jobs execute concurrently, whereas phase and accumulation control modify the execution of those jobs. Accordingly, changes in queueing time cannot be attributed to collective pacing alone without separating these effects.

Layer 2: Choreography. The Dispatch Job module provisions the selected workers and enforces the eligibility constraints associated with their placement. The Coordinate Timing module conveys the inter-job decisions to the job runtime. All participating ranks of a job receive the common synchronization policy, including its accumulation setting and assigned timing parameters.

Layer 3: Adaptation. Each job has a coordinator that collects worker telemetry and adapts the supported execution settings. These settings include per-worker

batch sizes, the job-level learning rate, and, when supported by the data pipeline, data-loader parallelism and shard ratios. The coordinator reports a synchronization footprint $\hat{p}_k(\cdot)$, a measured synchronization interval $\widehat{T}_k$, a micro-step compute proxy $\tau_k(t)$, and congestion-related indicators.

The cluster-level controller receives summarized runtime information rather than model gradients. This separation limits the information exchanged by the scheduling layer, but does not by itself establish the throughput or scalability of the coordinator implementation.

Worker telemetry is sampled continuously at the configured sampling interval, while control decisions are updated at epoch boundaries using measurements collected during the preceding epoch.

4.2 Inter-Job Scheduler: Communication Timing and Intensity

The inter-job controller uses synchronization footprints and congestion indicators to coordinate the active jobs in each contention domain. Timing control distributes their communication phases, while accumulation control changes the amount of local work performed between synchronizations.

Phase orchestration. Let t_0 denote a control-decision time and $H > 0$ a scheduling horizon. For this decision, let $p_k(u) \in \{0, 1\}$ denote the nominal synchronization pulse train associated with job k. It represents the timing pattern estimated from the reported footprint $\hat{p}_k(\cdot)$ and interval $\widehat{T}_k$. The periodic representation is a local scheduling approximation. The measured runtime is allowed to depart from it.

A normalized phase assignment $\phi_k \in \Omega$ is converted into a nominal time shift by

$$\Delta_k = \phi_k \, \widehat{T}_k. \tag{12}$$

Let

$$\mathcal{J}_d(t_0) = \{i \in \mathcal{J}_{\text{act}}(t_0) : d(i) = d\}$$

denote the active jobs associated with domain d at the decision time. The aggregate nominal synchronization occupancy is

$$P_d(u) = \sum_{i \in \mathcal{J}_d(t_0)} p_i(u - \Delta_i). \tag{13}$$

For a candidate assignment to job k, define

$$P_{d(k),-k}(u) = \sum_{\substack{i \in \mathcal{J}_{d(k)}(t_0) \\ i \neq k}} p_i(u - \Delta_i).$$

The desired phase assignment can then be expressed as

$$\phi_k^\star = \arg\min_{\phi \in \Omega} \int_{t_0}^{t_0+H} w\big(P_{d(k),-k}(u)\big)\, p_k\Big(u - \phi\widehat{T}_k\Big)\, du, \tag{14}$$

where $w(\cdot)$ is a non-decreasing overlap penalty. We use the linear penalty $w(x) = x$ in this scheduling formulation.

Equation (14) describes the objective of reducing predicted overlap. The prototype uses a lightweight phase-slot heuristic: active jobs are assigned uniformly spaced phase slots, with assignments updated using the observed synchronization periods and occupancy when jobs arrive or the execution regime changes. This implementation should be distinguished from an exact evaluation and minimization of every candidate in Eq. (14).

The shifted pulse train is therefore a scheduling surrogate, not a guarantee that the runtime will reproduce the predicted timing. Different job periods, worker skew, and injected waiting can change the realized pattern. Updated footprints allow subsequent decisions to use the behavior observed under the applied policy.

Congestion-aware synchronization control. Phase staggering does not directly reduce the communication volume generated by each synchronization. TANGO therefore also adjusts G_k using a domain-level congestion indicator:

$$\rho_d(t) = \alpha\, \widetilde{\mathrm{RTT}}_d(t) + \beta\, \widetilde{\mathrm{Retry}}_d(t) + \gamma\, \widetilde{T}_{\mathrm{comm},d}(t). \tag{15}$$

Here, the three normalized terms represent RTT inflation, a retransmission-related proxy, and collective-duration inflation, respectively. The coefficients α, β, and γ determine their contributions to the score.

The communication-duration term in Eq. (15) is distinct from the start-to-start synchronization interval $\widehat{T}_k$. Likewise, $\rho_d(t)$ is an operational feedback score rather than a direct measurement of the abstract load ratio $L_d(t)/C_d(t)$ in Section 3.

Consistent with Eq. (5), a job's average communication demand is approximated by

$$\bar{\lambda}_k(t) \approx \frac{D_k(\Theta_k)}{\widehat{T}_k(t)}. \tag{16}$$

Increasing G_k increases the number of local micro-steps between scheduled synchronizations. Its effect on realized communication demand depends on the resulting execution interval, including computation, input stalls, communication, and pacing. TANGO therefore uses congestion feedback rather than assuming that doubling G_k necessarily halves the measured wireless load.

Sensitivity-aware candidate selection. When a domain exhibits high congestion, TANGO preferentially selects jobs with lower latency-sensitivity proxies for additional throttling. Let $\mathcal{J}_{\mathrm{norm}}(t)$ denote the active-job set used to normalize these proxies. For positive normalization denominators, the score is

$$\begin{aligned} \pi_k(t) = \omega_a & \frac{a_k(t)}{\max_{i\in\mathcal{J}_{\mathrm{norm}}(t)} a_i(t)} \\ & + \omega_r \left(1 - \frac{\widehat{R}_k(t)}{\max_{i\in\mathcal{J}_{\mathrm{norm}}(t)} \widehat{R}_i(t)}\right) \\ & + \omega_s \frac{\widehat{S}_k(t)}{\max_{i\in\mathcal{J}_{\mathrm{norm}}(t)} \widehat{S}_i(t)}. \end{aligned} \tag{17}$$

Here, $a_k(t)$ denotes job age, $\widehat{R}_k(t)$ estimated remaining work, and $\widehat{S}_k(t)$ recent slowdown. The non-negative weights satisfy $\omega_a + \omega_r + \omega_s = 1$. A larger score gives greater protection to older, nearly completed, or already slowed jobs when throttling candidates are selected.

The score is a scheduling proxy, not a measured derivative of JCT with respect to a control variable. Its role is to rank candidates within the configured policy.

Multiplicative control and safeguards. TANGO uses separated congestion thresholds and bounded accumulation:

$$\rho_{\mathrm{hi}} > \rho_{\mathrm{lo}}, \qquad G_{\min}(d) = 1, \qquad G_{\min}(d) \le G_k \le G_{\max}. \tag{18}$$

For jobs eligible for the corresponding control action, the proposed accumulation update is

$$G_k^+ = \begin{cases} \min(2G_k, G_{\max}), & \rho_{d(k)}(t) > \rho_{\mathrm{hi}}, \\ \max(\lfloor G_k/2 \rfloor, 1), & \rho_{d(k)}(t) < \rho_{\mathrm{lo}}, \\ G_k, & \text{otherwise.} \end{cases} \tag{19}$$

The high-congestion branch is applied to selected throttling candidates, whereas the low-congestion branch relaxes accumulation when congestion subsides.

This rule is a bounded heuristic controller. Doubling and halving provide coarse changes in synchronization frequency, and the interval between the thresholds leaves the setting unchanged. The rule is not derived as an optimizer of Eq. (8), and threshold separation alone is not a proof of closed-loop stability.

TANGO additionally caps G_k, exempts jobs with $\widehat{R}_k(t) \le R_{\mathrm{short}}$ from further throttling unless congestion persists, and limits the cumulative throttled duration by $T_{\max}^{\mathrm{th}}$ before temporarily removing a job from the candidate set. Here, R_{short} is the near-completion threshold and $T_{\max}^{\mathrm{th}}$ is the configured throttling-duration budget. These safeguards reduce repeated throttling but do not establish strict per-class fairness or a general starvation-free guarantee.

The proposed accumulation setting must also be compatible with the training-configuration constraints discussed in Section 4.4.

4.3 Intra-Job Control: Telemetry-Driven Adaptation and Reporting

The intra-job controller adapts local execution settings and supplies the runtime information required by the inter-job controller. Its purpose is to respond to device-side pressure without treating each job as independent of the shared communication schedule.

Telemetry and capacity estimation. Each worker reports temperature, CPU utilization, memory pressure, compute duration, input-wait duration, collective duration, and RTT-related indicators. The coordinator smooths these signals and maintains a worker-capacity proxy $\hat{s}_{k,n}(t)$ together with a job-level compute estimate $\tau_k(t)$.

The prototype described here uses a Kalman-style nonlinear smoother (CKF) with a nine-dimensional state constructed from thermal, compute, memory, training-progress, and network-related telemetry. The scheduling interface consumes the

resulting execution proxies rather than the filter state itself. Consequently, the controller structure does not require the smoother to be regarded as a separate scheduling contribution.

Pressure-driven execution adaptation. For worker n of job k, the coordinator computes

$$\begin{aligned} q_{k,n}(t) = \lambda_T \, \widetilde{T}^{\mathrm{temp}}_{k,n}(t) + \lambda_M \, \widetilde{M}_{k,n}(t) \\ + \lambda_C \, \widetilde{T}^{\mathrm{comp}}_{k,n}(t) + \lambda_I \, \widetilde{T}^{\mathrm{input}}_{k,n}(t), \end{aligned} \tag{20}$$

where the normalized terms represent temperature, memory pressure, compute duration, and input-wait duration. The coefficients specify their contributions to the local pressure score.

When high pressure persists across consecutive control intervals, the coordinator decreases $B_{k,n}$ within its configured bounds. When pressure remains low and memory headroom is available, it can increase $B_{k,n}$. These adjustments change the configuration of subsequent micro-steps. They do not resize a forward/backward pass already in progress. The learning-rate setting is coordinated at the job level, and candidate updates are restricted by the admissible configuration region.

The controller also adjusts the input pipeline when input waiting remains a substantial fraction of the observed iteration duration. It increases the data-loader worker count $W_{k,n}$ when CPU and memory headroom permit. Where adaptive sharding is supported, the shard ratio $\pi_{k,n}$ is adjusted using $\hat{s}_{k,n}(t)$ so that slower workers receive a smaller share, subject to the configured ratio bounds. The shard ratio $\pi_{k,n}$ is distinct from the job-level throttling score π_k.

These actions target the measured sources of local execution delay. Their effect on communication is indirect: changing the execution rate also changes the synchronization interval and footprint reported to the inter-job controller.

Update boundaries and training semantics. Runtime configuration changes are applied only after completion of the current accumulation window and optimizer update. A new batch-size or accumulation configuration therefore takes effect before the first micro-step of the next accumulation window, and partially accumulated gradients are not carried across a configuration change.

Because workers can use different local batch sizes, TANGO performs sample-weighted gradient aggregation. Let $B_{k,n}$ denote the local mini-batch size of worker n and let $|\mathcal{N}_k|$ denote the number of workers assigned to job k. Before DDP gradient averaging, the local loss of worker n is scaled by

$$s_{k,n} = \frac{|\mathcal{N}_k| B_{k,n}}{\sum_{i \in \mathcal{N}_k} B_{k,i}}. \tag{21}$$

Since DDP subsequently averages gradients across ranks, this scaling yields a sample-weighted gradient over the aggregate mini-batch rather than giving every worker equal weight regardless of its local batch size.

For gradient accumulation with G_k micro-steps, the scaled loss is additionally divided by G_k before backpropagation. The local batch sizes and G_k remain fixed

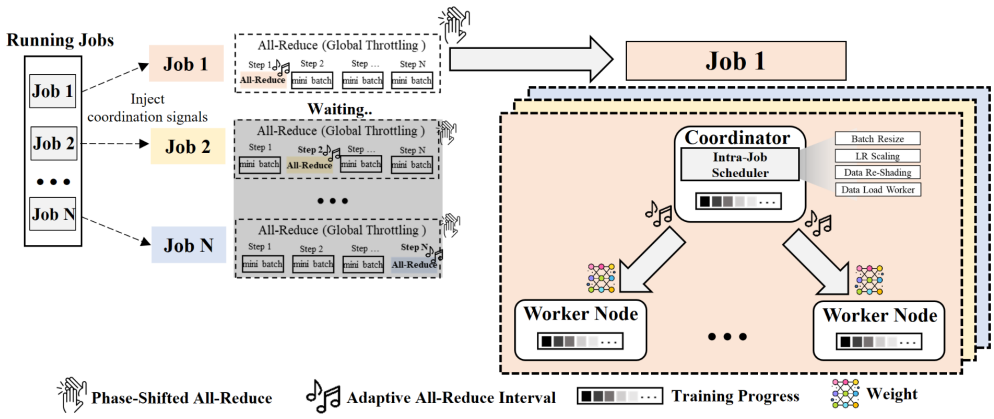


Fig. 2 Inter-job and intra-job interaction in TANGO. The inter-job controller returns phase assignments and gradient accumulation settings. Per-job coordinators apply the resulting synchronization policy, adapt supported execution settings, and report synchronization footprints, execution proxies, and congestion indicators for the next control decision.

within an accumulation window. Accordingly, the effective sample count associated with one synchronized optimizer update is $B_k^{\text{eff}} = G_k \sum_{n \in \mathcal{N}_k} B_{k,n}$, consistent with Eq. (7).

Runtime injection in PyTorch DDP. The phase decision is job-specific. Different jobs can therefore receive different phase offsets, while all ranks participating in the same job receive the same timing decision.

TANGO updates the phase policy at epoch boundaries. During an epoch, worker and network telemetry are continuously monitored, while the current phase assignment remains unchanged. At the end of the epoch, the coordinator summarizes the observed execution behavior, including the synchronization interval $\widehat{T}_k$, and the inter-job controller uses these measurements to determine the phase assignment for the next epoch.

Using Eq. (12), the phase offset for job k is computed from the current phase assignment and the synchronization interval observed during the preceding epoch. The resulting offset is applied once when the job enters the next epoch and remains unchanged throughout that epoch. It is not repeatedly inserted before individual DDP collective operations. All ranks of the same job receive the same epoch-level phase decision. The coordinator then observes the resulting execution behavior during the epoch and uses the collected telemetry for the next epoch-level control decision.

4.4 Inter–Intra Job Interaction

Figure 2 summarizes the feedback exchanged between the two control levels. The inter-job controller returns (ϕ_k, G_k), while the intra-job controller adapts Θ_k and reports the execution resulting from those decisions.

The two directions of this interaction are important. Changing batch size or input-pipeline settings changes the compute proxy and synchronization footprint used

for subsequent phase assignments. Conversely, changing the accumulation setting or injected delay changes the execution interval observed by the coordinator. The controller therefore updates its timing estimates from the applied runtime policy instead of assuming that a job retains its original period.

Joint feasibility of accumulation and local adaptation. For a proposed accumulation setting G_k^+ and local batch sizes $B_{k,n}^+$, the effective global batch becomes

$$B_k^{\mathrm{eff},+} = G_k^+ \sum_{n \in \mathcal{N}_k} B_{k,n}^+. \tag{22}$$

The corresponding job-level learning rate η_k^+ is selected together with the local batch configuration, subject to

$$\left(B_k^{\mathrm{eff},+}, \eta_k^+\right) \in \mathcal{C}_k. \tag{23}$$

Thus, an inter-job proposal to increase accumulation is not an unrestricted command: the resulting training configuration must remain feasible under the local resource bounds and the admissible region.

The job-level learning-rate adjustment is shared by all ranks and applied with a short warmup over W_{warm} iterations. Optimizer states and the gradient-clipping configuration are retained. These measures constrain the transition between configurations, but do not establish equivalence to the optimization trajectory of the fixed-configuration baseline.

Operational construction of the admissible region. For a recurring workload, the described design constructs $\mathcal{C}_k$ through a short sweep around its baseline batch-size and learning-rate configuration. A candidate pair is retained when the calibration run remains stable and its accuracy reduction relative to the baseline is within a configured tolerance δ_{acc}.

The resulting region can be cached and reused for subsequent instances of the same workload. For a previously unseen workload, the design starts with a conservative batch–learning-rate rule with caps and refines the admissible region when the calibration sweep becomes available. Calibration is therefore distinct from the recurring runtime control-loop work.

Membership in $\mathcal{C}_k$ reflects the conditions under which the configuration was screened. A short calibration run does not establish final-accuracy equivalence at convergence, and the reported training outcomes must be interpreted at their evaluated budgets.

4.5 Algorithm: Closed-Loop Co-Scheduling

Algorithm 1 summarizes the controller-level workflow. Coordinators first update the execution summaries. The inter-job controller then assigns phase slots and proposes accumulation changes using domain congestion and throttling safeguards. Each coordinator combines these proposals with feasible local adaptation before applying the resulting runtime configuration.

Algorithm 1 TANGO closed-loop co-scheduling

Require: Domains $\mathcal{D}$, phase slots Ω

Initialize phase assignments, accumulation settings, and per-job configurations
loop
  Obtain the active-job set and placement from the dispatcher
  Coordinators collect telemetry and update $\hat{s}_{k,n}$ and τ_k
  Report $\hat{p}_k(\cdot)$, $\widehat{T}_k$, and normalized congestion indicators
  Compute each ρ_d using Eq. (15)
  for each newly active job or job requiring phase reassignment **do**
    Assign $\phi_k \in \Omega$ using the uniform phase-slot heuristic
    Compute $\Delta_k = \phi_k \widehat{T}_k$
  end for
  for each contention domain $d \in \mathcal{D}$ **do**
    if $\rho_d > \rho_{\text{hi}}$ **then**
      Form eligible throttling candidates using π_k and the safeguards
      Propose increased G_k for selected candidates using Eq. (19)
    else if $\rho_d < \rho_{\text{lo}}$ **then**
      Propose accumulation relaxation for eligible jobs using Eq. (19)
    else
      Retain the current accumulation settings
    end if
  end for
  for each active job k **do**
    Determine feasible local batches and job-level LR consistent with the accumulation proposal
    Restrict the joint update using Eq. (23)
    Adjust data-loader parallelism and shard ratios where supported and indicated by telemetry
    Apply feasible updates at coordinated iteration boundaries
    Enforce the common pacing and accumulation policy
    Refresh footprints and intervals under the applied policy
  end for
  Continue execution until the next epoch boundary
end loop

Both directions of the congestion rule are included: high congestion can increase accumulation for selected candidates, whereas low congestion can relax it. The intermediate region retains the current setting. The algorithm describes the coordination flow without assuming that the phase heuristic solves Eq. (14) exactly.

The resulting loop connects measured synchronization behavior to subsequent timing and frequency decisions. It also keeps the communication policy coupled to the local training configuration through the effective-batch constraint. The evaluation examines the behavior of this combined system and its inter-job and intra-job variants. The algorithm itself does not imply an optimality, fairness, or convergence guarantee.



Fig. 3 Physical 12-node Raspberry Pi 5 testbed used for the experimental evaluation. The devices are arranged in three four-node groups with heterogeneous memory capacities and wireless connections.

5 Experimental Evaluation

We evaluate TANGO on a heterogeneous wireless IoT testbed and through trace-driven simulations calibrated from testbed measurements. The evaluation examines end-to-end completion times, queueing and queue-excluded service time, workload-dependent outcomes, and the behavior of the inter-job and intra-job controller variants. We additionally report representative training-time and accuracy summaries, simulated scaling results, and prototype control-loop latency.

The physical experiments and simulations provide different forms of evidence. The testbed results characterize the implemented system under the specified workloads and arrival trace, whereas the simulations extend the comparison to larger modeled worker populations. The interpretation below distinguishes these outcomes from direct measurements of individual wireless-contention mechanisms.

5.1 Experimental Setup

Testbed. We deploy a 12-node Raspberry Pi 5 cluster organized into three wireless contention domains. Figure 3 shows the physical testbed, and Table 1 summarizes the hardware and software configuration. The domains differ in memory capacity and wireless band, while passive cooling introduces the possibility of time-varying execution capacity under sustained load. Unless otherwise stated, the experiments use PyTorch 2.1.0 and Python 3.9.

Workloads and arrival model. We use nine vision and text workloads, grouped as lightweight (G1), balanced (G2), and intensive (G3), as shown in Table 2. The workloads include training from scratch and configurations identified as `Freeze`. The table reports the baseline per-worker mini-batch size and training-epoch budget for each workload.

Table 1 Heterogeneous wireless IoT testbed configuration.

Component	Specification
Nodes	12 × Raspberry Pi 5
Cluster A	4 nodes, 8 GB RAM per node, 5 GHz WiFi
Cluster B	4 nodes, 16 GB RAM per node, 5 GHz WiFi
Cluster C	4 nodes, 16 GB RAM per node, 2.4 GHz WiFi
Software	PyTorch 2.1.0, Python 3.9

Table 2 Distributed learning workloads. Batch denotes the baseline per-worker mini-batch size. Epochs specify the evaluated training budget. The 50-job trace uses the G1/G2/G3 mix of 50/30/20%.

Group	ID	Model	Dataset	Mode	Batch	Epochs
G1 Lightweight	J1	SimpleCNN	F-MNIST	Scratch	32	50
	J2	LeNet-5	MNIST	Scratch	16	50
	J3	TextCNN	SMS-Spam	Scratch	32	50
G2 Balanced	J4	MobileNetV3	CIFAR-10	Freeze	32	20
	J5	FastText	MR	Scratch	64	20
	J6	SqueezeNet	CIFAR-10	Freeze	32	20
G3 Intensive	J7	DenseNet121	CIFAR-100	Freeze	32	10
	J8	ResNet18	CIFAR-100	Freeze	16	10
	J9	EffNet-B0	CIFAR-100	Freeze	16	10

The principal evaluation uses a 50-job Poisson arrival trace with a G1/G2/G3 workload mix of 50/30/20%. The configured mean inter-arrival time is 1 s, corresponding to an arrival-rate parameter $\lambda_{\text{arr}} = 1\,\text{s}^{-1}$.

Trace representativeness and scope. The arrival trace is intended to exercise a contention-heavy multi-job operating regime rather than reproduce the measured request distribution of a production deployment. The 1 s mean inter-arrival time concentrates submissions relative to the reported training durations and allows a backlog to develop. The experiment therefore examines how the scheduling configurations handle a demanding finite workload on the available cluster.

The workload groups cover different model and training configurations, but their names do not imply an ordering of end-to-end job duration. In particular, a lightweight model can still have a substantial training budget or experience a long queueing delay. The conclusions are restricted to the evaluated workload mix, training budgets, and hardware configuration.

Baselines and comparison scope. We compare TANGO with FIFO and shortest-job-first (SJF) as queueing-policy reference points, and with Pollux, Lucid, and Sia as representative multi-job deep-learning schedulers. These systems were selected

because they address scheduler-level decisions for concurrent training workloads, including resource allocation, job coordination, adaptive training configurations, or heterogeneous resources, which are the closest comparison axis to TANGO's multi-job orchestration objective. Although Pollux, Lucid, and Sia were not designed specifically for wireless IoT clusters, evaluating them on the same heterogeneous wireless testbed provides a common system-level basis for assessing whether TANGO's communication-aware and device-adaptive coordination provides additional benefit in the target edge setting. The results compare the reported scheduling configurations as complete systems and should therefore be interpreted as end-to-end scheduler comparisons rather than experiments that isolate collective timing while holding admission, placement, and all training settings fixed.

The component study in Section 5.5 compares inter-only, intra-only, and full TANGO configurations. These variants examine the two controller levels. They are not substitutes for separate comparisons with randomized phase assignments, fixed accumulation, or domain-level admission caps.

Metrics and statistical interpretation. For job k, JCT is measured from arrival to completion, $\mathrm{JCT}_k = E_k - A_k$. Let Q_k denote its recorded queueing delay. The queue-excluded service component is

$$\mathrm{Service}_k = \mathrm{JCT}_k - Q_k. \tag{24}$$

This service component represents elapsed execution time after excluding the recorded queueing component. It is not a measurement of computation or collective communication in isolation.

We report average JCT, average queueing delay, makespan, and the P95/P99 values of the specified latency metrics. Individual workload configurations occur multiple times within the 50-job trace, providing repeated workload-level observations under different queueing, contention, and co-running-job conditions. These within-trace occurrences are used to characterize workload-level variability, including the median and range summaries reported in Section 5.4. Because jobs within a trace share the evolving cluster state, however, these observations are not treated as statistically independent cluster-level replications. Accordingly, the percentile results characterize the recorded job outcomes rather than confidence intervals across independent trace repetitions, and the comparisons below are interpreted descriptively without a statistical-significance claim.

Training and runtime parameters. Tables 3 and 4 list the reported training, runtime, and trace defaults. The control mechanisms using these settings are described in Section 4.

Wireless conditions. The testbed includes both 2.4 GHz and 5 GHz links and uses online congestion feedback for runtime control. These conditions expose the scheduler to different device and wireless configurations within the evaluated cluster. However, evaluating this combined setup is distinct from independently varying wireless band, channel occupancy, and thermal state. The results below therefore characterize the combined operating regime rather than an isolated sensitivity curve for each factor.

Table 3 Reported default training and runtime parameters.

Item	Value
Optimizer	Adam, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, no weight decay
Initial learning rate	$\eta_0 = 10^{-3}$
Baseline batch size	Workload-specific values in Table 2
DDP bucket cap	50 MB
All-reduce compression	FP16 communication hook

Table 4 Reported monitoring and trace parameters.

Item	Value
Telemetry sampling	Every 2 s
RTT probing	ICMP ping every five ticks (approximately 10 s), timeout 1 s
Intra-job batch bound	$B_{k,n} \in [16, 128]$
Intra-job LR bound	$\eta_k \in [10^{-5}, 10^{-2}]$
Shard-ratio mapping	Sigmoid: range $(0.1, 0.5)$, mapping parameter $k_{\text{sig}} = 10$, threshold 0.5
Telemetry smoother	CKF, nine-dimensional state, diagonal Q and R
Mean inter-arrival time	1 s for the 50-job Poisson trace

5.2 End-to-End Performance on a 50-Job Trace

Table 5 reports the end-to-end results for the 50-job trace. TANGO achieves an average JCT of 1.98 h and a makespan of 4.10 h. Relative to Sia, average JCT decreases from 2.38 h to 1.98 h, corresponding to a 16.8% reduction, and makespan decreases from 5.00 h to 4.10 h, corresponding to an 18.0% reduction. P99 queueing delay decreases from 4.11 h to 3.30 h, a 19.7% reduction.

These measurements show improved aggregate completion and queueing outcomes under the evaluated scheduling configuration. They do not imply that every workload class benefits equally, as examined in Section 5.3.

Figure 4 presents the empirical distributions of JCT and queueing delay for Pollux, Lucid, Sia, and TANGO. FIFO and SJF remain included in the tabulated comparison. The distributions complement the aggregate and percentile results by showing variation across the recorded jobs. They should not be interpreted as uncertainty estimates across independent runs.

Queueing and service-time decomposition. Table 6 reports queue-excluded service time and tail JCT. Queueing accounts for approximately 76–79% of the reported JCT across the compared configurations. Relative to Sia, TANGO reduces average service time from 0.55 h to 0.45 h and P99 JCT from 4.93 h to 4.02 h.

Table 5 End-to-end performance on the 50-job Poisson trace. All latency values are in hours.

Scheduler	Avg JCT	Avg Queue	P95 Queue	P99 Queue	Makespan
FIFO	3.48	2.73	5.39	5.64	6.70
SJF	3.29	2.54	5.38	5.61	6.57
Pollux	2.64	2.02	4.20	4.33	5.38
Lucid	2.53	1.95	4.14	4.22	5.20
Sia	2.38	1.83	3.89	4.11	5.00
TANGO	**1.98**	**1.53**	**3.13**	**3.30**	**4.10**

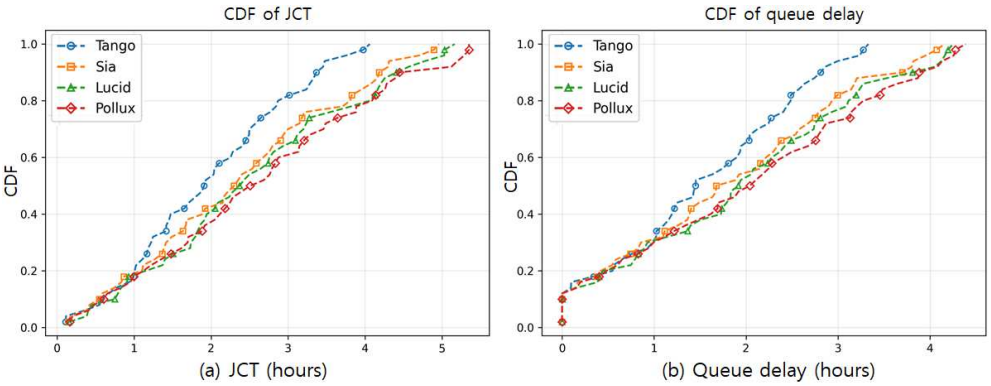


Fig. 4 Empirical CDFs of JCT and queueing delay for the 50-job testbed trace. The plotted systems are Pollux, Lucid, Sia, and TANGO.

Using the rounded values in the two tables, the difference in average JCT can be decomposed as

$$\underbrace{2.38 - 1.98}_{\text{JCT: 0.40 h}} = \underbrace{1.83 - 1.53}_{\text{Queue: 0.30 h}} + \underbrace{0.55 - 0.45}_{\text{Service: 0.10 h}}. \tag{25}$$

Thus, approximately 75% of the observed average-JCT difference is associated with the queueing component and 25% with the service component. The service-time reduction itself is approximately 18.2% relative to Sia.

This is an accounting decomposition, not a causal separation of scheduling mechanisms. Shorter execution can release resources earlier and thereby reduce the waiting time of subsequent jobs. Changes in admission or placement can also affect queueing. Consequently, the 75% queueing contribution neither establishes that the gain comes exclusively from queue reordering nor quantifies how much is caused by reduced collective contention.

Table 6 Queue-excluded service time, reported queueing share, and tail JCT on the 50-job trace. Time values are in hours.

Scheduler	Avg Service	Queue Share (%)	P95 JCT	P99 JCT
FIFO	0.75	78.5	6.19	6.56
SJF	0.75	77.2	6.21	6.48
Pollux	0.62	76.6	5.25	5.35
Lucid	0.58	77.0	4.91	5.10
Sia	0.55	76.8	4.53	4.93
TANGO	**0.45**	77.5	**3.63**	**4.02**

Table 7 Performance by workload group on the 50-job trace. JCT and queueing values are in hours.

Scheduler	G1: Lightweight		G2: Balanced		G3: Intensive	
	JCT	Queue	JCT	Queue	JCT	Queue
FIFO	4.08	3.32	1.92	1.04	4.32	3.79
SJF	4.84	4.08	1.91	1.08	1.47	0.85
Pollux	2.69	2.04	2.38	1.75	2.89	2.37
Lucid	2.57	2.00	2.34	1.67	2.75	2.26
Sia	**2.44**	**1.86**	2.09	1.57	2.64	2.14
TANGO	2.69	2.25	**1.49**	**1.02**	**0.92**	**0.50**

5.3 Performance by Workload Regime

Table 7 shows the results for the three workload groups. Relative to Sia, the reduction in average JCT is larger for G2 and G3 than for G1. For G2, average JCT decreases from 2.09 h to 1.49 h and average queueing delay from 1.57 h to 1.02 h, corresponding to reductions of 28.7% and 35.0%, respectively. For G3, average JCT decreases from 2.64 h to 0.92 h and average queueing delay from 2.14 h to 0.50 h, corresponding to reductions of 65.2% and 76.6%.

G1 outcomes and the aggregate-performance trade-off. For G1, TANGO increases average JCT from 2.44 h to 2.69 h relative to Sia, a 10.2% increase. Average queueing delay rises from 1.86 h to 2.25 h. Subtracting the reported queueing values from JCT gives a queue-excluded service component of approximately 0.58 h for Sia and 0.44 h for TANGO. The longer G1 completion time therefore reflects increased waiting despite a lower service-time component in this aggregate decomposition.

The G1 result highlights that the benefit of TANGO is workload dependent. In the evaluated lightweight regime, the queue-excluded service component decreases from approximately 0.58 h under Sia to 0.44 h under TANGO, indicating that execution

itself is not slower. However, the increase in queueing delay outweighs this service-time reduction and leads to the higher end-to-end JCT.

This pattern is consistent with a regime in which lightweight jobs provide less headroom for communication- and execution-aware coordination to improve completion time. By contrast, the larger gains for G2 and G3 indicate greater benefit when the workload regime offers more opportunity for such coordination to affect job progress. The target setting of TANGO is a mixed edge-learning cluster containing workloads with different execution and communication characteristics, rather than a homogeneous stream composed only of lightweight jobs. Accordingly, the G1 result reflects a workload-dependent trade-off rather than a claim that every workload class benefits equally. The current configuration does not provide a per-class fairness guarantee, and the group-level results do not isolate the contribution of individual admission, placement, pacing, or adaptation decisions.

5.4 Per-Job Execution Time and Accuracy

Table 8 reports training-time and accuracy summaries for J1, J5, and J8, representing the three workload groups. These workloads occur 9, 5, and 3 times, respectively, within the 50-job trace. Accordingly, the reported median and range values summarize repeated executions of the same workload configuration under different runtime conditions within the trace. Training time excludes the recorded queueing delay.

Compared with Sia, TANGO reduces the median training time of J1 from 4,160 s to 2,932 s, J5 from 139 s to 102 s, and J8 from 1,456 s to 1,284 s. These correspond to reductions of 29.5%, 26.6%, and 11.8%, respectively.

Accuracy at the evaluated training budgets. Across the repeated occurrences, TANGO achieves median accuracies of 91.64%, 62.03%, and 26.47% for J1, J5, and J8, respectively. Relative to Sia, the corresponding differences are -0.77, -0.56, and $+0.27$ percentage points. Thus, the absolute accuracy differences remain below one percentage point for all three representative workloads, while TANGO provides the training-time reductions reported above. These results indicate that, at the evaluated training budgets, the observed execution-time improvements are achieved without large changes in training quality for the representative workloads.

The comparisons reflect the epoch budgets and training modes in Table 2 and are intended to characterize the training outcomes observed under the evaluated configurations rather than establish accuracy equivalence after full convergence.

5.5 Controller-Level Ablation: Inter-Job and Intra-Job Coordination

Table 9 evaluates whether the two controller levels provide complementary benefits when used together. It compares three TANGO configurations on a separate 25-job mixed trace. The inter-only configuration enables phase assignments, adaptive accumulation, and sensitivity-based throttling selection while keeping the intra-job training knobs at their reported baseline settings. The intra-only configuration enables telemetry-driven intra-job adaptation while disabling inter-job phase and throttling updates. Full TANGO combines the two controller levels.

Table 8 Training-time and accuracy summaries for representative workloads from the 50-job trace. Count denotes occurrences of each workload in the trace. Values are reported as median [minimum, maximum]. Training time excludes queueing.

Job (Count)	Scheduler	Training time (s)	Accuracy (%)
J1 (9)	Pollux	4,253 [4054,4663]	92.16 [92.0,92.3]
	Lucid	3,656 [3278,3967]	92.01 [91.8,92.3]
	Sia	4,160 [3944,4376]	92.41 [92.1,92.5]
	TANGO	**2,932** [2829,3080]	91.64 [91.3,91.7]
J5 (5)	Pollux	177 [151,579]	62.03 [61.2,62.5]
	Lucid	399 [157,400]	62.03 [61.0,62.4]
	Sia	139 [138,153]	62.59 [61.2,63.6]
	TANGO	**102** [90,274]	62.03 [61.6,62.7]
J8 (3)	Pollux	1,344 [1317,1546]	26.31 [26.2,26.3]
	Lucid	1,380 [1355,1546]	26.61 [26.5,26.7]
	Sia	1,456 [1414,1491]	26.20 [26.1,26.3]
	TANGO	**1,284** [1268,1315]	26.47 [26.2,26.5]

Table 9 Ablation results on the 25-job mixed trace. All latency values are in hours.

Metric	Intra-only	Inter-only	Full TANGO
Makespan	3.76	3.72	**2.14**
Average JCT	1.85	1.81	**1.26**
P95 JCT	2.91	3.03	**1.93**
P99 JCT	3.54	3.55	**2.06**

The comparison is made within this 25-job experiment. The 25-job ablation trace was generated and evaluated independently from the 50-job trace used in Table 7. Its absolute completion times should therefore not be directly compared with the 50-job results as though the only difference were the controller configuration.

Full TANGO achieves lower values for all four reported metrics than either partial configuration. Average JCT decreases from 1.85 h under intra-only and 1.81 h under inter-only to 1.26 h under full TANGO. Calculated from these tabulated values, the reductions are approximately 31.9% and 30.4%, respectively. Full TANGO also achieves a makespan of 2.14 h and a P99 JCT of 2.06 h.

These results support the complementarity of the inter-job and intra-job controller levels in this experiment: the combined configuration outperforms either controller level used alone across all four reported completion-time metrics. The purpose of this ablation is to assess the interaction between the two controller levels rather than to attribute the end-to-end gain to every individual mechanism. Because the table does not include a configuration with both controllers disabled, it does not separately quantify the gain of each controller over an otherwise identical uncontrolled runtime.

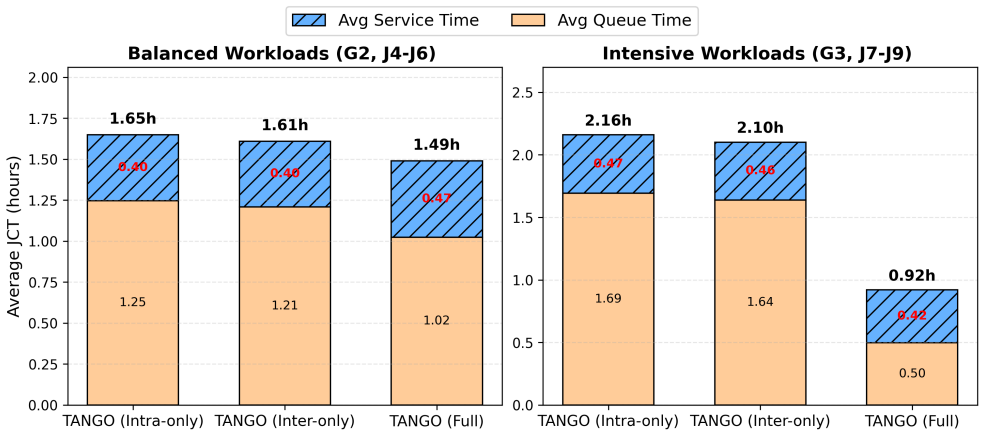


Fig. 5 Reported group-level decomposition of average JCT into queueing and queue-excluded service time for the 25-job ablation trace. The panels show G2 (J4–J6) and G3 (J7–J9) under intra-only, inter-only, and full TANGO.

Nor does it isolate the effects of phase assignment, accumulation, and sensitivity-based selection within the inter-job controller.

Figure 5 decomposes average JCT into queueing and service time for G2 and G3. For G2, full TANGO reduces queueing while service time increases from 0.40 h under either partial configuration to 0.47 h. The reduction in queueing outweighs this increase, yielding a lower JCT. For G3, full TANGO reduces both queueing and service time. The apparent equality of some two-decimal full-TANGO values with the 50-job results arises from rounded summaries of independently generated experimental data.

5.6 Scalability and Runtime Overhead

Simulation setup and metrics. We complement the physical testbed evaluation with a scheduler-level scalability simulation in which the worker population is varied from 64 to 2,048 nodes. The simulation abstracts heterogeneous compute capacity, memory constraints, network capacity, communication overhead, and training-efficiency effects in order to examine how the compared scheduling policies behave as the available resource pool grows. Its purpose is to evaluate the scalability of the orchestration and scheduling logic rather than to reproduce packet-level wireless behavior or emulate the Raspberry Pi testbed at larger scale. Accordingly, the simulated results are interpreted as a comparative scheduler-level analysis under the modeled resource conditions.

Table 10 defines the reported simulation metrics. Let $\mathcal{N}$ denote the full simulated worker population, distinct from the worker set assigned to any individual job. Simulation times are expressed in seconds.

At a fixed cluster size, WJCT is a constant multiple of average JCT and therefore gives the same scheduler ranking. Across cluster sizes, it combines completion latency

Table 10 Simulation metrics and definitions. A_k and E_k denote job arrival and completion times, respectively. $|\mathcal{N}|$ is the simulated worker count.

Metric	Definition
JCT	$\mathrm{JCT}_k = E_k - A_k$
Average JCT	$\mathrm{AvgJCT} = \frac{1}{\lvert\mathcal{J}\rvert} \sum_{k \in \mathcal{J}} \mathrm{JCT}_k$
Makespan	$\mathrm{Mksp} = \max_{k \in \mathcal{J}} E_k - \min_{k \in \mathcal{J}} A_k$
Cluster-size-weighted average JCT	$\mathrm{WJCT} = \mathrm{AvgJCT}\,\lvert\mathcal{N}\rvert$

Table 11 Cluster-size-weighted average JCT in the scalability simulation, reported in node·s/job.

Nodes	Pollux	Lucid	Sia	TANGO
64	54,316	96,057	79,737	**50,944**
128	63,219	114,880	100,979	**59,622**
256	68,352	139,904	113,382	**63,539**
512	94,054	179,916	133,529	**91,187**
1,024	151,756	300,236	183,091	**146,022**
2,048	367,616	831,078	331,980	**274,636**

with the available worker count. It is not a measurement of the worker-time actually consumed by each job, energy consumption, or resource utilization.

Scalability results. Table 11 reports WJCT for the six evaluated cluster sizes. TANGO has the lowest tabulated WJCT at each size. At 2,048 nodes, the reported value decreases from 331,980 node·s/job under Sia to 274,636 node·s/job under TANGO.

Using these tabulated values, the relative reduction is

$$\frac{331{,}980 - 274{,}636}{331{,}980} \times 100\% \approx 17.3\%. \tag{26}$$

This is a comparison at the same simulated cluster size and, by the definition of WJCT, is also the relative reduction in average JCT for that size.

Figure 6 presents average JCT and makespan for the simulated configurations. The comparisons are made separately at each evaluated size. The results support lower completion metrics for TANGO within these simulations, rather than a general claim that increasing the worker count must monotonically improve every scheduler.

Prototype control-loop overhead. The prototype reports approximately 1 ms average control-loop latency in steady state, compared with the multi-second training

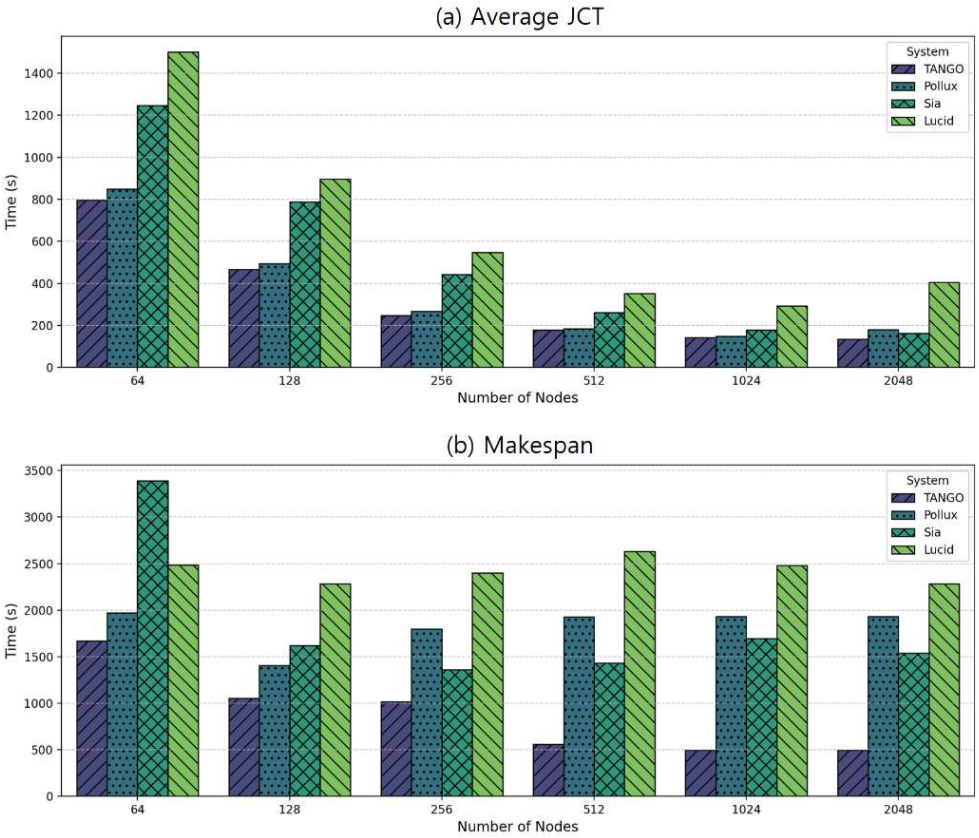


Fig. 6 Simulation results for cluster sizes from 64 to 2,048 nodes: (a) average JCT and (b) makespan.

iterations observed on the Raspberry Pi testbed. The controller exchanges execution summaries, synchronization footprints, and congestion indicators rather than gradients.

This reported latency is distinct from the worker telemetry-sampling period, the RTT-probing interval, and the injected worker-side pacing delay. In particular, a short controller latency does not imply that repeated pre-collective waiting has no execution cost. It also does not establish the same controller latency for the larger simulated worker populations.

Scope of the experimental evidence. The evaluation supports lower aggregate completion metrics for the reported TANGO configuration on the specified testbed trace, a workload-dependent trade-off for G1, and better outcomes for full TANGO than either partial controller configuration in the 25-job ablation. The simulations extend this comparison under the modeled execution conditions.

The reported results do not separately establish the shape of the collective-latency response to increasing synchronization overlap, the contribution of admission control

under otherwise fixed execution settings, or training-quality equivalence after convergence. These distinctions bound the interpretation of the measured gains without replacing the reported end-to-end comparison.

6 Conclusion

We presented TANGO, a hierarchical closed-loop co-scheduler for multi-job distributed learning on heterogeneous wireless IoT clusters. TANGO couples inter-job coordination of collective timing and synchronization frequency with telemetry-driven intra-job adaptation. Phase assignments and gradient accumulation provide communication controls, while batch-size, learning-rate, and supported input-pipeline adjustments respond to device-side conditions. The central contribution is the feedback connecting these controls: local adaptation changes the synchronization behavior observed by the inter-job controller, and inter-job decisions constrain the execution configurations used within each job.

On a 12-node heterogeneous Raspberry Pi 5 testbed with nine vision and text workloads, TANGO reduces average job completion time by 16.8%, makespan by 18.0%, and P99 queueing delay by 19.7% relative to Sia under the evaluated 50-job Poisson trace. The completion-time decomposition reports lower average queueing and queue-excluded service times, while the separate 25-job controller-level ablation yields lower completion metrics for full TANGO than for either the inter-only or intra-only configuration, supporting the benefit of coupling the two control levels in the evaluated setting.

For the three workloads with reported accuracy results, absolute differences from Sia remain below one percentage point at the specified training budgets. These observations characterize training quality under the evaluated configurations rather than establish equivalent accuracy after convergence. The scheduler-level scalability study extends the comparison to 2,048 nodes, with TANGO achieving the lowest cluster-size-weighted average JCT among the compared systems at each evaluated size. These results characterize comparative scaling behavior under the modeled resource conditions rather than performance of a physical deployment at that scale.

Limitations. The reported results are bounded by the evaluated hardware, workload mix, and contention-heavy arrival regime. The controller-level ablation evaluates the benefit of combining the inter-job and intra-job control levels, but does not isolate the causal contribution of every individual scheduling mechanism. The accuracy analysis covers three representative workloads at their specified training budgets, and the scalability study evaluates a modeled scheduler-level setting rather than a physical deployment at large scale. The G1 results further show that TANGO's benefit is workload dependent: although its queue-excluded service component decreases, the increase in queueing outweighs this improvement in end-to-end JCT for the lightweight group.

Future work. Future work will evaluate broader workload and arrival distributions, more diverse edge platforms, and mixed WiFi–cellular backhauls. We will also investigate explicit fairness and priority-aware objectives for lightweight and latency-sensitive workloads, and extend the coordination framework to local-SGD and hierarchical collective settings.

Acknowledgements

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2025-02263869, Development of AI Semiconductor Cloud Platform Establishment and Optimization Technologies, 50%) and was supported by Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2026-25521081, Development of AI Semiconductor Cloud Resource managment and Service Provision Platform Techonology, 50%).

Author Contributions

Teh-Jen Sun conceived and designed the study, developed the proposed method, implemented the system, conducted the experiments, analyzed the results, and wrote the original manuscript. Md Nahid Sultan reviewed the manuscript and contributed to its revision. Eui-Nam Huh supervised the research, acquired funding, contributed to the conceptualization of the study, and critically reviewed and revised the manuscript. All authors reviewed and approved the final manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The datasets used in this study are publicly available from their respective repositories. The raw experimental logs and simulation outputs generated during this study are not publicly available due to institutional security restrictions. Certain portions of these data may be made available from the corresponding author upon reasonable request, subject to institutional approval and applicable security restrictions. The aggregated results supporting the findings of this study are reported in the article.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the authors used ChatGPT (OpenAI) to improve the English grammar and readability of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

[1] Satyanarayanan, Mahadev. "The emergence of edge computing." Computer 50.1 (2017): 30–39.

[2] Mao, Yuyi, et al. "A survey on mobile edge computing: The communication perspective." IEEE Communications Surveys & Tutorials 19.4 (2017): 2322–2358.

[3] Singh, Raghubir, and Sukhpal Singh Gill. "Edge AI: a survey." Internet of Things and Cyber-Physical Systems 3 (2023): 71–92.

[4] Xu, X., Ding, Y., Hu, S. X., Niemier, M., Cong, J., Hu, Y. & Shi, Y. Scaling for edge inference of deep neural networks. *Nature Electronics* 1, 216–222 (2018).

[5] Parisi, German I., et al. "Continual lifelong learning with neural networks: A review." Neural Networks 113 (2019): 54–71.

[6] Kairouz, Peter, et al. "Advances and open problems in federated learning." Foundations and Trends® in Machine Learning 14.1–2 (2021): 1–210.

[7] Chen, Jingxue, et al. "When federated learning meets privacy-preserving computation." ACM Computing Surveys 56.12 (2024): 1–36.

[8] Duan, Sijing, et al. "Distributed artificial intelligence empowered by end-edge-cloud computing: A survey." IEEE Communications Surveys & Tutorials 25.1 (2022): 591–624.

[9] Walia, Guneet Kaur, Mohit Kumar, and Sukhpal Singh Gill. "AI-empowered fog/edge resource management for IoT applications: A comprehensive review, research challenges, and future perspectives." IEEE Communications Surveys & Tutorials 26.1 (2023): 619-669.

[10] Wu, Y., Wang, Z., Shi, Y. & Hu, J. Enabling on-device CNN training by self-supervised instance filtering and error map pruning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 3445–3457 (2020).

[11] Hua, Haochen, et al. "Edge computing with artificial intelligence: A machine learning perspective." ACM Computing Surveys 55.9 (2023): 1-35.

[12] Zhu, Guangxu, et al. "Toward an intelligent edge: Wireless communication meets machine learning." IEEE communications magazine 58.1 (2020): 19-25.

[13] Sergeev, Alexander, and Mike Del Balso. "Horovod: fast and easy distributed deep learning in TensorFlow." arXiv preprint arXiv:1802.05799 (2018).

[14] Keuper, Janis, and Franz-Josef Preundt. "Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability." 2016 2nd workshop on machine learning in HPC environments (MLHPC). IEEE, 2016.

[15] Shi, Weisong, et al. "Edge computing: Vision and challenges." IEEE Internet of Things Journal 3.5 (2016): 637–646.

[16] Ye, Mang, et al. "Heterogeneous federated learning: State-of-the-art and research challenges." ACM Computing Surveys 56.3 (2023): 1–44.

[17] Benoit-Cattin, Théo, Delia Velasco-Montero, and Jorge Fernández-Berni. "Impact of thermal throttling on long-term visual inference in a CPU-based edge device." Electronics 9.12 (2020): 2106.

[18] Qiao, Aurick, et al. "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning." In 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2021.

[19] Hu, Qinghao, et al. "Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs." In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 2, 2023.

[20] Jayaram Subramanya, Suhas, et al. "Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling." Proceedings of the 29th Symposium on Operating Systems Principles. 2023.

[21] Weng, Qizhen, et al. "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters." In 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2022.

[22] Narayanan, Deepak, et al. "Heterogeneity-Aware cluster scheduling policies for deep learning workloads." In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020.

[23] Xiao, Wencong, et al. "Gandiva: Introspective cluster scheduling for deep learning." In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2018.

[24] Gu, Juncheng, et al. "Tiresias: A GPU cluster manager for distributed deep learning." 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). 2019.

[25] Mahajan, Kshiteej, et al. "Themis: Fair and efficient GPU cluster scheduling." In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2020.

[26] Gu, Diandian, et al. "Elasticflow: An elastic serverless training platform for distributed deep learning." Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2023.

[27] Mo, Zizhao, Huanle Xu, and Chengzhong Xu. "Heet: Accelerating elastic training in heterogeneous deep learning clusters." Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2024.

[28] Mai, Luo, et al. "KungFu: Making training in distributed machine learning adaptive." In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020.

[29] Johnson, Tyler, et al. "AdaScale SGD: A user-friendly algorithm for distributed training." In International Conference on Machine Learning (ICML). PMLR, 2020.

[30] Strati, Foteini, et al. "Sailor: Automating distributed training over dynamic, heterogeneous, and geo-distributed clusters." Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles. 2025.

[31] McCandlish, Sam, et al. "An empirical model of large-batch training." arXiv preprint arXiv:1812.06162 (2018).

[32] Alistarh, Dan, et al. "QSGD: Communication-efficient SGD via gradient quantization and encoding." In Advances in Neural Information Processing Systems (NeurIPS), 2017.

[33] Fei, Jiawei, et al. "Efficient sparse collective communication and its application to accelerate distributed deep learning." In Proceedings of ACM SIGCOMM, 2021.

[34] Ho, Qirong, et al. "More effective distributed ML via a stale synchronous parallel parameter server." In Advances in Neural Information Processing Systems (NeurIPS), 2013.

[35] Hu, Shuyan, et al. "Distributed machine learning for wireless communication networks: Techniques, architectures, and applications." IEEE Communications Surveys & Tutorials 23.3 (2021): 1458-1493.

[36] Wen, Jie, et al. "A survey on federated learning: challenges and applications." International Journal of Machine Learning and Cybernetics 14.2 (2023): 513–535.

[37] McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." In Artificial Intelligence and Statistics (AISTATS). PMLR, 2017.

[38] Lai, Fan, et al. "Oort: Efficient federated learning via guided participant selection." In 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2021.

[39] Liu, Yi-Jing, et al. "Adaptive Clustering-Based Straggler-Aware Federated Learning in Wireless Edge Networks." IEEE Transactions on Communications 72.12 (2024): 7757-7771.

[40] Hu, Junyan, et al. "Performance optimization of split federated learning in heterogeneous edge computing environments." IEEE Transactions on Industrial Informatics (2025).

[41] Van Huynh, Nguyen, et al. "Joint coding and scheduling optimization for distributed learning over wireless edge networks." IEEE Journal on Selected Areas in Communications 40.2 (2021): 484-498.

[42] Petri, I., Chirila, I., Gomes, H. M., Bifet, A. & Rana, O. F. Resource-aware edge-based stream analytics. *IEEE Internet Computing* 26(4), 79–88 (2022).