INTERNATIONAL JOURNAL OF INFORMATION AND COMMUNICATION TECHNOLOGY TRENDS

Vol. 6 No. 02 (2026)

e.ISSN: 2409-5222

https://ijictt.ictt.info

Uncertainty-Aware LLM-Guided Cloud Resource Scheduling with Budgeted Robust Optimization

Amelia Turner

Faculty of Information Technology, Monash University, Melbourne, VIC, Australia

Abstract

Hybrid schedulers that let a large language model propose placements and a solver enforce feasibility have brought semantic reasoning into cluster resource management. Such pipelines still treat two quantities as exact: the model proposal itself, and the resource demands and runtimes attached to each task. Both are estimates, and a scheduler that ignores their error either over-trusts a wrong suggestion or packs a node to a capacity it does not have. This paper presents URSA, a scheduler that carries uncertainty through the whole decision path. Confidence in each model proposal is obtained from the agreement of temperature-diverse samples and mapped to a hit probability by post-hoc calibration, which then sets a per-task trust weight in the objective. Runtime prediction error is turned into finite-sample upper bounds by split conformal prediction at a level selected from each job's remaining deadline slack. Declared-versus-actual demand deviation is absorbed by a budgeted robust assignment program whose conservatism is controlled by a single parameter. Evaluation on real Spark TPC-H DAGs with measured task-duration profiles shows that URSA removes 95-96% of node memory-pressure and eviction events relative to a deterministic guided solver at equal job completion time, shortens makespan by 4-13%, and raises CPU utilization by up to 11%. Its behaviour stays flat as guidance reliability falls from 0.90 to 0.30 and under over-confident guidance, whereas the deterministic pipeline swings by up to 47% in utilization.

Keywords: *cloud resource scheduling, robust optimization, conformal prediction, confidence calibration, large language models, service level agreements*

1. Introduction

Cluster resource managers decide, thousands of times per second, which pending unit of work goes on which machine. The decision couples job completion time, deadline attainment, packing efficiency and fairness, and production systems such as Borg [1], Omega [2], Mesos [3], YARN [4] and Kubernetes [5] have converged on fast heuristics that keep the decision cheap: dominant-resource fairness [6], multi-dimensional packing [7], plan-ahead reservations [8] and flow-based placement [9]. The heuristics are predictable, but they read a task as a vector of numbers and discard everything that is written down about it—the shape of its dependency graph, where its input sits, what its owner promised about it.

Learning-based schedulers recover part of that signal. Deep reinforcement learning was shown to beat hand-tuned policies on packing [14], graph neural networks made dependency structure usable for DAG jobs [15], and later systems added heterogeneity and goodput awareness [16], [17]. What they gain in adaptivity they pay for in training data and in brittleness once the workload drifts away from the traces they were fitted on.

Large language models opened a different route. They read heterogeneous, partly textual state directly, they transfer with a handful of examples [20], and prompted step-by-step reasoning makes their intermediate choices inspectable [21]. They also cannot be trusted to respect hard constraints, which is why the planning literature settled on modulo architectures in which the model proposes and a verifier disposes [18], [19]. Ding et al. carried that idea into cluster scheduling: their LLMSched encodes cluster state as structured text, asks the model for temperature-diverse placement candidates, and passes each candidate through an integer linear program that repairs infeasibility while staying close to the suggestion [27]. The design is a convincing demonstration that semantic reasoning and algorithmic guarantees compose, and it is the closest predecessor of this work. One assumption survives that composition. Both halves of the pipeline treat their inputs as exact. The repair program is written over declared resource demands and point estimates of task runtimes, and the proposal is folded into the objective with a fixed trust coefficient that is the same whether the model was unanimous across samples or split four ways. Neither quantity is exact in a real cluster. Declared memory and the memory a container actually touches diverge by tens of percent, which is precisely why node-pressure eviction exists. Runtime predictors have fat-tailed residuals: on the Spark TPC-H traces used in this paper the coefficient of variation of stage durations averages 0.78. And confidence expressed by a language model is systematically inflated, with agreement across sampled responses a better signal than the number the model reports [26], [28], [29]. A scheduler that ignores these errors fails in two specific ways. It packs a node to a capacity computed from declared demands, the actual footprint overshoots, and the kubelet evicts a container whose work is then lost. And it follows a proposal with the same weight whether that proposal is right nine times in ten or three times in ten, so the cost of a wrong suggestion is paid in full.

This paper takes the position that uncertainty should be a first-class quantity in the guided-scheduling loop rather than an afterthought, and builds URSA around that. Our contributions are the following.

1) A guidance channel that reports a usable confidence. Agreement across temperature-diverse proposals gives a raw self-consistency score, which a monotone post-hoc map turns into an estimate of the probability that the proposal is the right one; that estimate then scales the per-task trust weight in the objective, so an unreliable channel is automatically discounted rather than manually disabled.

2) Split conformal upper bounds on stage runtimes, with the miscoverage level chosen per job from its remaining deadline slack, so deadline-critical work is planned against a high quantile and slack work against the median. The bounds feed the urgency term of the objective, which turns a distribution-free statistical guarantee into deadline-aware prioritisation.

3) A budgeted robust assignment program in which declared-versus-actual demand deviation is absorbed by a Bertsimas–Sim uncertainty set, giving a single knob that trades protection against packing density, together with a robust greedy backfill that applies the same admission rule to the work the program does not reach.

4) An evaluation on real Spark TPC-H job graphs with measured task-duration profiles, showing that the robust layer removes 95–96% of node memory-pressure and eviction events at parity job completion time, shortens makespan by 4.3–12.9% and raises CPU utilisation by up to 11.2%, and that its behaviour is stable across the whole range of guidance reliability while a deterministic pipeline is not.

II. RELATED WORK

A. Cluster Resource Management

Centralised managers established the production template: admission control, priority preemption and bin packing at datacentre scale [1], [10], with optimistic concurrency [2] and two-level delegation [3], [4] as alternatives for multi-framework clusters. Policy work layered fairness [6], multi-resource alignment [7], space–time reservations [8] and min-cost-flow placement [9] on top. Prediction entered through resource

estimation rather than decision making: Resource Central forecasts VM behaviour offline and feeds the forecast to unchanged policies [11], and Quasar classifies incoming workloads to size allocations [12]. The gap between what a job declares and what it consumes is a known operational hazard in these systems and motivates the demand-uncertainty model used here.

B. Learning- and Language-Model-Assisted Scheduling

DeepRM framed packing as a Markov decision process [14]; Decima added a graph neural network over stage DAGs and a scalable action space, and released the Spark TPC-H measurements this paper builds on [15]. Later systems handled accelerator heterogeneity [16] and co-adapted batch size with allocation [17]. Language models entered systems work first through code [25] and then through advisory roles; the planning literature established that they generate plausible but not verifiable plans [19] and that pairing them with external verifiers recovers soundness [18]. The guided scheduler discussed in Section I applies this to cluster scheduling with structured-textual state encoding, constraint-guided prompting and an ILP refinement stage, and reports large gains over both heuristic and reinforcement-learning baselines on production traces. Our work keeps that separation of duties and changes what crosses the boundary: instead of a point proposal and point parameters, the optimiser receives a calibrated confidence and a set-valued description of the parameters.

C. Decision Making Under Uncertainty

Robust optimisation replaces uncertain parameters with a set and optimises the worst case over it. Ellipsoidal sets preserve tractability but are conservative [33]; the budgeted polyhedral set of Bertsimas and Sim keeps the robust counterpart linear and exposes a single parameter controlling how many coefficients may deviate simultaneously, with a closed-form probability bound on constraint violation [32], [34]. Distributionally robust formulations trade set geometry for moment information [35]. On the statistical side, split conformal prediction converts any point predictor into intervals with finite-sample marginal coverage under exchangeability alone [36], [37], [38], and post-hoc calibration maps raw scores onto probabilities [30], [31]. Confidence elicited from language models is known to be over-confident, and aggregating agreement across sampled responses mitigates it [28], [29]; self-consistency was introduced for accuracy but the agreement statistic is exactly the aggregation signal we reuse [22], [26]. URSA is, to our knowledge, the first scheduler to route all three—agreement-based confidence, conformal runtime bounds and a budgeted robust capacity model—into a single assignment program.

III. PROBLEM SETTING

A. Cluster and Workload

A cluster is a set of heterogeneous nodes N, node i offering $c_i(\text{cpu})$ execution slots and $c_i(\text{mem})$ gigabytes of memory and sitting in availability zone z(i). A job J is a directed acyclic graph of stages; stage s carries n_s tasks that are interchangeable, so the stage is placed as a gang on one node and receives $p_{si} = \min(p_s, c_i(\text{cpu}))$ slots there, where p_s is its requested width. Executing n_s tasks on p slots takes $\lceil n_s / p \rceil$ waves, so placement determines runtime and the scheduler cannot separate the two decisions.

Stage s becomes ready when every predecessor has finished. A ready stage placed on a node other than the one holding the bulk of its input pays a transfer term proportional to the shuffle volume and to whether the two nodes share a zone. Jobs arrive over time with weight w_j and, for the deadline-sensitive fraction, a deadline d_j. A scheduling decision at epoch t maps a subset of the ready set R_t to nodes; stages left unmapped wait for a later epoch.

B. Three Uncertain Quantities

Memory demand. Stage s declares $\bar{m}_s$ gigabytes per slot; the footprint it actually touches is $m_s = \bar{m}_s(1 + \xi_s)$ with $|\xi_s| \le 2\sigma$. The scheduler observes only the declaration. Admission control is performed against declarations, so a node whose declarations sum exactly to capacity is over-committed as soon as the realised footprints exceed them, at which point resident containers are evicted and their work is lost.

Stage runtime. A predictor returns $\hat{e}_s$ from historical measurements of the same stage type at the same width. The realised duration e_s is drawn from the measured distribution of that stage and is not concentrated: across the workload used here the per-stage coefficient of variation averages 0.78.
Guidance correctness. The advisory channel returns, for each ready stage, a preferred node g_s drawn from the filtered candidate set, together with a raw confidence. Whether g_s is the structurally best choice is a Bernoulli event whose rate is unknown to the scheduler and varies with prompt quality, model version and workload regime.

C. Objective

Let $x_{si} \in \{0,1\}$ indicate that ready stage s is placed on node i. Writing $\hat{c}_{si}$ for the estimated completion contribution of that placement, the epoch objective balances weighted completion against the value of admitting work at all,

$$\min_x \ \Sigma_s \Sigma_i \left[w_s \hat{c}_{si} - \pi_s \right] x_{si} \ + \Sigma_s \Sigma_i \lambda_s (1 - g_{si})\, x_{si} \quad (1)$$

subject to one node per stage, slot capacity, memory capacity and zone affinity. The admission incentive $\pi_s = B\, w_s\, u_s$ carries the deadline urgency u_s, the last term penalises departures from the guidance, and the whole design question is what ĉ, u and λ should be when the quantities they are built from are estimates.

IV. THE URSA SCHEDULER

A. Overview

Fig. 1 shows the loop. Cluster state feeds two estimators in parallel. The advisory channel emits K temperature-diverse placement proposals per ready stage, whose agreement becomes a raw confidence and, after calibration, an estimated hit probability. The runtime predictor emits point estimates whose residuals accumulate into a conformal calibration set. The assignment program consumes a filtered candidate set, the calibrated confidences and the conformal quantiles, and returns placements that are feasible for every demand realisation inside a budgeted uncertainty set. Realised runtimes and footprints return to both estimators, so the two uncertainty models track the workload without retraining any weights.

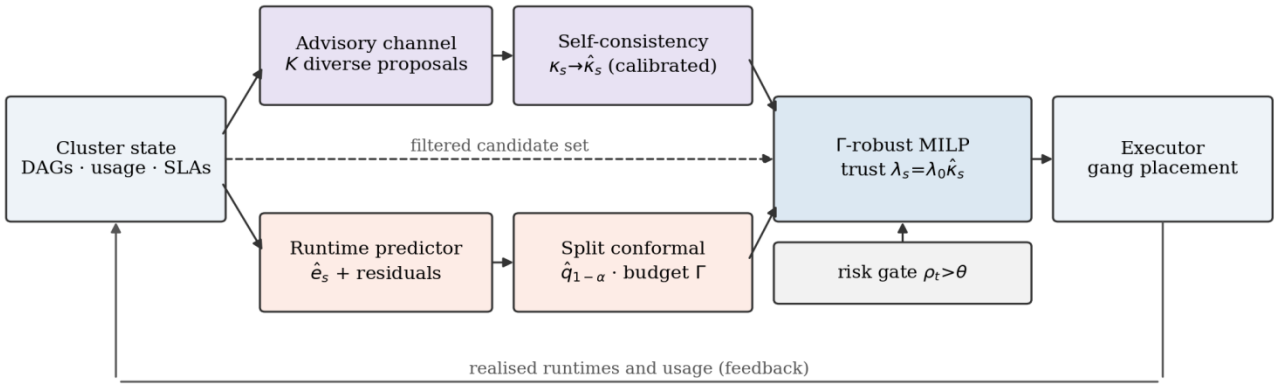


Figure 1. URSA carries two uncertainty estimates through the decision path: agreement across advisory samples becomes a calibrated trust weight, and runtime residuals become conformal upper bounds with a level chosen from deadline slack. The assignment program

B. Confidence From Agreement, Then Calibration

The channel is sampled K times at temperatures spread over a fixed range, producing votes v_1 … v_K over the candidate set. The proposal is the mode and the raw confidence is its share,

$$g_s = \text{argmax}_i \, \#\{k : v_k = i\}, \quad \kappa_s = (1/K)\, \#\{k : v_k = g_s\} \quad (2)$$

Agreement is preferred to a confidence the channel states about itself because stated confidence is inflated in a way that aggregation across samples is not [28], [29]. Raw agreement is nevertheless not a probability. Over

a calibration window we log pairs (κ_s, y_s), where y_s records whether the proposal matched the structurally best placement identified by counterfactual replay of the epoch, and fit a monotone binned map $\hat{h}$ in the spirit of histogram calibration [31],

$$\hat{\kappa_s} = \hat{h}(\kappa_s) \approx Pr[\, y_s = 1 \mid \kappa_s \,] \qquad (3)$$

with pool-adjacent-violators applied once so that $\hat{h}$ is non-decreasing. The map is frozen after the window and the calibration quality is reported as expected calibration error [30].

C. Conformal Runtime Bounds at a Risk-Adaptive Level

Let $r_j = e_j / \hat{e}_j$ be the multiplicative residuals of n exchangeable past stages. Split conformal prediction sets

$$\hat{q}(1-\alpha) = r_(\lceil (n+1)(1-\alpha) \rceil), \quad Pr[\, e_s \le \hat{e}_s \, \hat{q}(1-\alpha) \,] \ge 1 - \alpha \quad (4)$$

where $r_{(k)}$ is the k-th order statistic [36], [37]. The bound needs no distributional assumption on the residuals, which matters here because they are right-skewed: the measured ratios have a median near one and a 99th percentile several times larger.

Applying the highest quantile everywhere would inflate every estimate and push the scheduler toward uniform pessimism. URSA instead selects the level from how much slack the owning job has left. With ρ_s the remaining critical path of stage s under point estimates and $d_j - t$ the remaining slack,

$$\alpha_s = 0.05 \text{ if } d_j - t < 1.2\,\rho_s \text{ ; } 0.2 \text{ if } < 2\,\rho_s \text{ ; } 0.5 \text{ otherwise} \qquad (5)$$

so that work which is close to its deadline is planned against a 95% upper bound while work with room is planned against the median. The bound enters the objective twice: through the completion estimate $\hat{c}_{si} = (\hat{e}_s\, \hat{q}(1-\alpha_s) + \tau_{si}) / Z$, where τ is the transfer term and Z a normaliser, and through the urgency

$$u_s = 1 + 2 \cdot clip(\, \hat{q}(1-\alpha_s)\, \rho_s \,/\, (d_j - t),\, 0,\, 2\,) \quad (6)$$

which is the ratio of a robust estimate of the work still to be done to the time still available. Raising the quantile therefore does not make deadline-critical stages look expensive and get deferred; it makes them look urgent and get scheduled first. This is the mechanism by which a statistical coverage guarantee becomes a scheduling priority.

D. Budgeted Robust Capacity

For node i let S_i be the stages the epoch may place there, each with declared demand $\bar{m}_s$ and deviation half-width $\hat{m}_s = 2\sigma\, \bar{m}_s$. The uncertainty set is the budgeted polyhedron of Bertsimas and Sim,

$$U_i(\Gamma) = \{\, m : m_s = \bar{m}_s + \xi_s \hat{m}_s,\ |\xi_s| \le 1,\ \Sigma_s\, |\xi_s| \le \Gamma \,\} \quad (7)$$

and the requirement is that capacity holds for every member of that set. Protecting against the Γ largest simultaneous deviations is a maximisation inside a constraint; dualising it keeps the counterpart linear, so the memory constraint becomes

$$\Sigma_s\, \bar{m}_s\, x_{si} + \Gamma\, z_i + \Sigma_s\, q_{si} \le \tilde{c}_i(mem) \qquad (8)$$

$$q_{si} \ge \hat{m}_s\, x_{si} - z_i, \quad q_{si} \ge 0, \quad z_i \ge 0 \qquad (9)$$

with $\tilde{c}_i$(mem) the residual capacity after the declarations of resident stages. $\Gamma = 0$ recovers the deterministic program, $\Gamma = |S_i|$ the fully adversarial one, and intermediate values interpolate. Slot capacity and affinity stay deterministic,

$$\Sigma_s\, p_{si}\, x_{si} \le \tilde{c}_i(cpu), \quad \Sigma_i\, x_{si} \le 1, \quad x_{si} = 0 \text{ for } i \notin N_s \quad (10)$$

The program has $|R_t| \times M$ binaries after the candidate set is pruned to the M cheapest feasible nodes per stage, plus the auxiliary variables introduced by (9).

E. Calibrated Trust

The guidance term in (1) is weighted per stage by the calibrated confidence,

$$\lambda_s = \lambda_0\, \hat{\kappa_s} \qquad (11)$$

A unanimous proposal from a channel whose unanimous proposals are right 90% of the time pulls the solution hard; a split proposal from a channel whose split proposals are right 35% of the time barely moves it. The

fixed coefficient used by deterministic guided pipelines is the special case $\hat{\kappa} \equiv 1$, which spends the same optimisation budget defending a guess as defending a near-certainty.

F. Risk Gate and Robust Backfill

Consulting the advisory channel at every epoch is neither affordable nor useful. URSA computes a risk score from the change in cluster occupancy, the share of ready stages whose deadline is tight, and the size of the ready set,

$$\rho_t = |\Delta \bar{u}_t| + 0.6 \cdot (\#tight / |R_t|) + 0.3 \cdot min(1, |R_t| / 8) \quad (12)$$

and solves the full program only when $\rho_t > \theta$. Below the threshold it falls through to a greedy pass that ranks placements by the same robust cost and admits them under the same rule, evaluating the protection term directly as the sum of the Γ largest deviations already committed on the node. The same pass also backfills the stages the program did not reach, which keeps the cluster busy when the ready set is much larger than the optimisation window without weakening the capacity guarantee for the stages it admits.

G. Properties

Proposition 1 (robust feasibility). Any x satisfying (8)–(9) satisfies $\Sigma_s\, m_s\, x_{si} \le \tilde{c}_i(mem)$ for every $m \in U_i(\Gamma)$. The protection term $\Gamma\, z_i + \Sigma_s\, q_{si}$ is, at its optimum in z, exactly the maximum of $\Sigma_s\, \hat{m}_s\, \xi_s\, x_{si}$ over the budget, so the constraint dominates the worst case, and therefore every realisation in the set [32], [34].

Proposition 2 (violation probability). If the deviations ξ_s are independent and symmetric about zero, the probability that a capacity constraint over $|S_i|$ stages is violated is at most $\exp(-\Gamma^2 / 2|S_i|)$ [32]. With the dozen or so stages an epoch places on a busy node the bound is permissive—0.85 at $\Gamma = 2$ and 0.51 at $\Gamma = 4$—yet Section VI-C shows the realised rate collapsing by two orders of magnitude over that range, because the deviations of a real workload are neither adversarial nor aligned and the budgeted set is a loose description of them. Proposition 3 (bounded cost of guidance). Let x* minimise the guided objective (1) and x° minimise the same objective without its last term, and let $C(\cdot)$ denote the guidance-free part. Then $C(x^*) \le C(x^\circ) + \lambda_o\, \Sigma_s\, \hat{\kappa}_s$. The proof is immediate: $C(x^*) \le C(x^*) + \Sigma\, \lambda_s(1-g) \le C(x^\circ) + \Sigma\, \lambda_s(1-g)$ at $x^\circ \le C(x^\circ) + \Sigma_s\, \lambda_s$. Substituting (11) gives the claim. Two consequences matter. The feasible set does not contain the guidance term at all, so a wrong proposal can never make a schedule infeasible; and as calibrated confidence falls the bound collapses, so an unreliable channel costs proportionally little rather than costing a fixed amount.

Proposition 4 (deadline safety along a path). If each stage on a path of length L is planned against its conformal bound at level α, the probability that all L stages finish within their bounds is at least $1 - L\alpha$ by the union bound. The risk-adaptive rule (5) spends the small α exactly on the paths where it is needed.

H. Cost

Candidate pruning leaves M = 6 nodes per stage and the optimisation window is capped at 20 stages, so the program carries at most 120 binaries and is given a 50 ms budget. Everything else in the epoch is linear in the ready-set size. The conformal quantiles are order statistics refreshed periodically over a bounded residual buffer; the calibration map is fitted once.

V. EXPERIMENTAL SETUP

A. Workload

Experiments use the public Spark TPC-H measurement release that accompanies Decima [15]: for each of 22 TPC-H queries at seven input scales from 2 GB to 100 GB, the release contains the stage-level dependency graph and the durations of every task measured on a real Spark cluster at executor counts from 2 to 100, separating the first wave of tasks from subsequent warm waves. Table I summarises the resulting 154 distinct jobs. The measured durations are used in two ways: the predictor is fitted on one half of the samples of each stage and realised runtimes are drawn from the held-out half, so prediction residuals—and therefore the conformal quantiles—are those of a real workload rather than of an assumed noise model.

Table 1MEASURED PROPERTIES OF THE SPARK TPC-H WORKLOAD

Property	**Value**
Distinct jobs (query × scale)	154
TPC-H queries / input sizes	22 / 7 (2–100 GB)
Stages per job (mean / max)	9.2 / 18
Tasks per job (mean / max)	1 155 / 3 717
DAG depth (mean / max)	5.9 / 9
Stage-duration CV (mean / median)	0.78 / 0.58
Measured executor settings	2–100 executors

The release does not record per-stage memory footprints. Each stage type is therefore assigned a per-slot demand drawn once from a log-normal profile with median 3.2 GB, and the realised footprint deviates from the declaration by a truncated normal with relative standard deviation σ, held at 0.30 by default and swept in Section VI-D.

B. Cluster and Simulator

The simulated cluster has 50 nodes in five profiles—balanced small, medium and large, CPU-heavy and memory-heavy—totalling 768 slots and 5.8 TB of memory across three zones. A quarter of jobs carry a zone affinity constraint. Events are job arrivals and stage completions; each event opens a decision epoch. A stage placed on a node takes its measured duration for the granted width, plus a transfer term when its largest predecessor ran elsewhere, doubled across zones. When the realised footprints on a node exceed its capacity the node enters memory pressure: the incoming stage is slowed in proportion to the overshoot and a resident stage is evicted with probability growing in the overshoot, losing its work and returning to the ready set. Arrival rate is set from the measured mean job work of 758 slot-seconds, so the nominal offered load is a controlled parameter. Sixty per cent of jobs carry a deadline at 2.5–5× their critical path, weights are drawn from {1, 1, 2, 4}, and jobs are attributed to eight users for the fairness measurement.

C. Guidance Channel

The advisory channel is instantiated by a trace-conditioned advisor with a controlled per-sample hit rate a. Its target is the structurally best placement, defined as the minimiser of estimated runtime plus transfer cost plus the expected downstream shuffle its children would pay if displaced—information the myopic objective of the assignment program does not contain, which is what makes the channel informative. Each of the K = 5 samples returns that target with probability a and a uniformly drawn candidate otherwise, and the scheduler sees only the resulting vote distribution. Parameterising the channel this way lets the optimiser be characterised across the entire reliability range rather than at one operating point, including the range of invalid-assignment rates reported for prompt-based schedulers, where explicit constraint injection moves the rate from roughly a third of cases to under a tenth [27]. The over-confident variant compresses the reported agreement toward one, reproducing the inflation documented for elicited language-model confidence [28].

D. Baselines

Random places each ready stage on a uniformly drawn feasible node. Kubernetes reproduces the default score of least-requested and balanced-allocation priorities [5]. DRF equalises dominant shares across users [6]. Tetris scores placements by multi-resource alignment [7]. Greedy-ERT is a runtime-aware greedy that places each stage on the feasible node minimising estimated runtime plus transfer cost, and is the strongest classical baseline here because it internalises the wave model. Guidance-only follows the advisory proposal directly. Det-MILP is the deterministic guided program—point estimates, nominal capacity, fixed trust coefficient—and is the direct analogue of the refinement stage of prior guided schedulers [27]. URSA is the full design with $\Gamma = 2$, $\lambda_0 = 0.35$, $\theta = 0.15$, $K = 5$ and a 50 ms solver budget.

E. Protocol

Each configuration runs 250 jobs under three regimes—normal load, high load, and a bursty regime in which arrivals cluster—repeated over five seeds, with 95% confidence intervals reported. Metrics are average and weighted job completion time, SLA violation rate over deadline-carrying jobs, time-averaged CPU and memory utilisation, makespan, Jain fairness over users, node memory-pressure events, and scheduling latency measured as wall-clock time per decision epoch. Sweeps in Sections VI-C to VI-F use 200 jobs and three seeds.

VI. EVALUATION

A. Overall Comparison

Table II reports every scheduler in every regime and Fig. 2 plots the four headline metrics. Three findings stand out.

Runtime awareness separates the field. The classical heuristics place a stage by how much room a node has, which on gang-placed stages with wave-structured runtimes is the wrong question: a wide stage sent to an eight-slot node runs for many waves. Kubernetes, DRF and Tetris all finish jobs in 120–125 s on average, while Greedy-ERT, which minimises the same runtime estimate the optimiser uses, reaches 115 s. Against the strongest heuristic URSA still cuts average completion time by 30.5% and SLA violations by 33.5%; against the Kubernetes default the reductions are 33.4% and 35.2%.

The robust layer buys stability, not slowness. URSA and Det-MILP finish jobs in statistically indistinguishable time—91.1 ± 3.9 s against 92.5 ± 3.7 s under high load—yet Det-MILP triggers 119 memory-pressure events per run against 4.8 for URSA, a reduction of 96%. Those events are not free: the evicted work is redone, which is why URSA also shortens makespan by 4.3–12.9% and lifts CPU utilisation by 3.7–11.2% across the three regimes. Protecting capacity turns out to cost nothing in latency and to return throughput.

Fairness is unaffected. Jain indices sit between 0.939 and 0.941 for URSA, at or slightly above DRF, because the urgency term is driven by deadline risk rather than by accumulated share and no user is systematically starved.

Table 2OVERALL PERFORMANCE ON THE SPARK TPC-H WORKLOAD (MEAN OF FIVE SEEDS, 95% CI)

Scheduler	Avg. JCT (s)	W-JCT (s)	SLA viol.	CPU util.	Makespan (s)	Pressure ev.	Fairness
Normal load							
Random	115.6 ± 9.0	110.6	0.408	0.213	1386	73.6	0.944
Kubernetes	83.7 ± 5.2	80.5	0.171	0.227	1185	71.4	0.932

Scheduler	Avg. JCT (s)	W-JCT (s)	SLA viol.	CPU util.	Makespan (s)	Pressure ev.	Fairness
DRF	87.6 ± 7.5	89.7	0.185	0.211	1294	64.8	0.924
Tetris	89.6 ± 5.6	91.0	0.197	0.260	1153	96.4	0.943
Greedy-ERT	75.0 ± 4.6	73.2	0.115	0.174	1355	64.2	0.939
Guidance-only	68.7 ± 5.3	70.1	0.094	0.204	1207	85.8	0.930
Det-MILP	52.2 ± 4.8	49.4	0.041	0.565	455	101.4	0.931
URSA	**52.7 ± 5.6**	**49.1**	**0.041**	**0.590**	**425**	**5.0**	**0.939**
High load							
Random	167.5 ± 13.1	171.5	0.680	0.225	1322	73.8	0.938
Kubernetes	135.5 ± 12.3	133.7	0.451	0.191	1357	79.0	0.939
DRF	143.6 ± 16.2	146.4	0.526	0.191	1385	76.6	0.936
Tetris	144.6 ± 12.2	133.1	0.465	0.202	1380	88.4	0.939
Greedy-ERT	135.4 ± 9.7	124.5	0.453	0.173	1422	76.6	0.943
Guidance-only	134.4 ± 13.1	123.6	0.452	0.185	1363	64.8	0.928
Det-MILP	92.5 ± 3.7	75.3	0.334	0.563	476	119.4	0.933
URSA	**91.1 ± 3.9**	**74.2**	**0.305**	**0.626**	**415**	**4.8**	**0.939**
Bursty load							
Random	177.7 ± 11.7	179.0	0.748	0.259	1184	72.6	0.935
Kubernetes	142.2 ± 14.0	127.6	0.477	0.201	1325	79.4	0.931
DRF	149.7 ± 16.4	153.3	0.549	0.220	1217	62.6	0.932
Tetris	141.5 ± 8.2	131.0	0.532	0.218	1285	101.0	0.935
Greedy-ERT	136.0 ± 12.4	124.0	0.503	0.192	1292	77.4	0.939
Guidance-only	141.6 ± 18.1	128.4	0.504	0.189	1341	77.8	0.934

Scheduler	Avg. JCT (s)	W-JCT (s)	SLA viol.	CPU util.	Makespan (s)	Pressure ev.	Fairness
Det-MILP	94.6 ± 5.7	76.2	0.359	0.626	420	108.0	0.933
URSA	**97.1 ± 4.7**	**79.7**	**0.366**	**0.649**	**402**	**4.0**	**0.941**

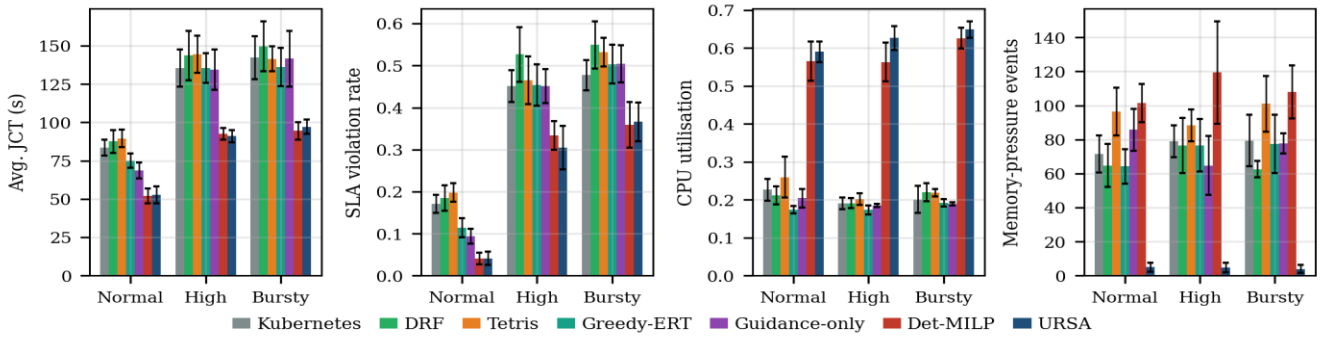


Figure 2Headline metrics across the three load regimes, five seeds, 95% confidence intervals. The gap between URSA and the deterministic guided program is concentrated in the rightmost panel.

B. Where the Gains Come From

Table III disables one component at a time under high load. Removing the robust capacity model is by far the largest single change: memory-pressure events rise from 5.3 to 96.3 per run and utilisation drops by 8.2%, confirming that the eviction pathway rather than the placement ranking is what the robust term is protecting. Removing the guidance costs 9.0% in SLA violations, which is the honest size of the contribution of an advisory channel that is right seven times in ten: real, but an order of magnitude smaller than the robustness effect. Removing the conformal bound or the calibrated trust barely moves any metric at this operating point, and the reason is visible in the next two subsections—both components are insurance whose premium is paid in regimes the default configuration does not sit in.

Table 3COMPONENT ABLATION UNDER HIGH LOAD

Configuration	Avg. JCT (s)	SLA viol.	CPU util.	Pressure ev.	Invoc. rate
URSA (full)	**76.4 ± 7.4**	**0.227**	**0.545**	**5.3**	**92%**
w/o robust capacity	78.2 ± 6.2	0.210	0.500	96.3	82%
w/o conformal bound	74.7 ± 7.1	0.230	0.529	4.3	92%
w/o calibrated trust	76.0 ± 7.4	0.224	0.555	3.0	92%
w/o risk trigger	76.6 ± 8.6	0.244	0.583	3.0	100%
w/o guidance	77.0 ± 7.1	0.250	0.571	1.3	92%

C. The Price of Robustness

Fig. 3 sweeps the uncertainty budget. At $\Gamma = 0$ the program is the deterministic one and incurs 107 pressure events per run. The curve falls steeply: $\Gamma = 0.5$ removes 66% of them, $\Gamma = 1$ removes 92%, and beyond $\Gamma = 1.5$ the count is in the low single digits. Completion time over the same range moves within 8.4%, and

utilisation is highest at intermediate budgets rather than at $\Gamma = 0$, because reclaimed eviction losses more than pay for the reserved headroom. The classical price of robustness is close to zero here and briefly negative. The empirical violation rate also falls much faster than Proposition 2 requires. The bound at $\Gamma = 2$ over a dozen stages permits violations in most constraints; what is observed is a per-run count of 4.3 events out of several thousand placements. Deviations in a real workload are neither adversarial nor aligned, so the budgeted set is a loose description of them and a small budget already captures most of the mass.

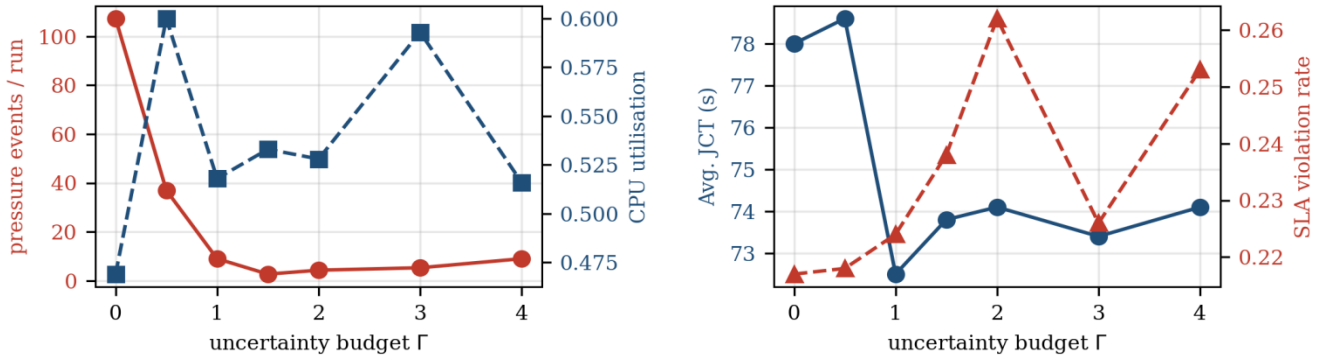


Figure 3Price of robustness. Left: pressure events collapse as the uncertainty budget grows while utilisation peaks at an intermediate budget. Right: completion time and SLA violations stay flat across the same range.

D. Sensitivity to Demand-Estimation Error

Fig. 4 varies the declared-versus-actual demand error σ from 0.10 to 0.40 with everything else fixed. The deterministic program degrades in proportion to σ, since the mass of the deviation it ignores grows linearly, while URSA stays close to flat: its protection term is written in units of the deviation half-width and therefore scales with the error it is defending against. The separation is the practical argument for the design—a cluster whose request-to-usage gap is small barely needs the machinery, and a cluster whose gap is large needs it badly, and the same parameterisation covers both.

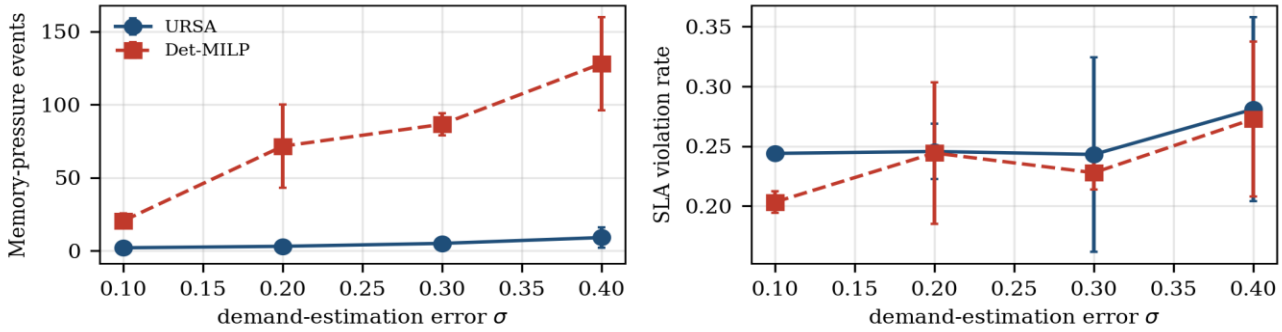


Figure 4Behaviour as the declared-versus-actual demand error grows. The robust program absorbs the increase; the deterministic one does not.

E. Guidance Reliability and Over-Confidence

Fig. 5 sweeps the per-sample hit rate of the advisory channel from 0.30 to 0.90. The quantity to watch is not the mean but the spread. Across that range URSA's utilisation varies by 2.6% and its completion time by 1.1%, while the deterministic program's utilisation varies by 47.4% and its pressure-event count ranges from 65 to 102 per run. A fixed trust coefficient commits the same optimisation budget to every proposal, so a poor proposal displaces a good placement with full force and the consequences propagate through the eviction pathway; a confidence-scaled coefficient lets the optimiser fall back on its own cost model exactly when the channel stops being informative, which is what Proposition 3 predicts.

The right panel isolates the calibration step. The channel is made over-confident by compressing reported agreement toward one, and the trust weight is computed either from the calibrated estimate or from the raw score taken at face value. Averaged over the reliability range, the calibrated weight holds SLA violations at 0.243 against 0.256 for the raw score, a reduction of 5.1%, and utilisation at 0.564 against 0.553. The map costs one pass over a few hundred logged decisions and it is what keeps an adaptive trust weight usable with a channel whose self-reported confidence cannot be read as a probability.

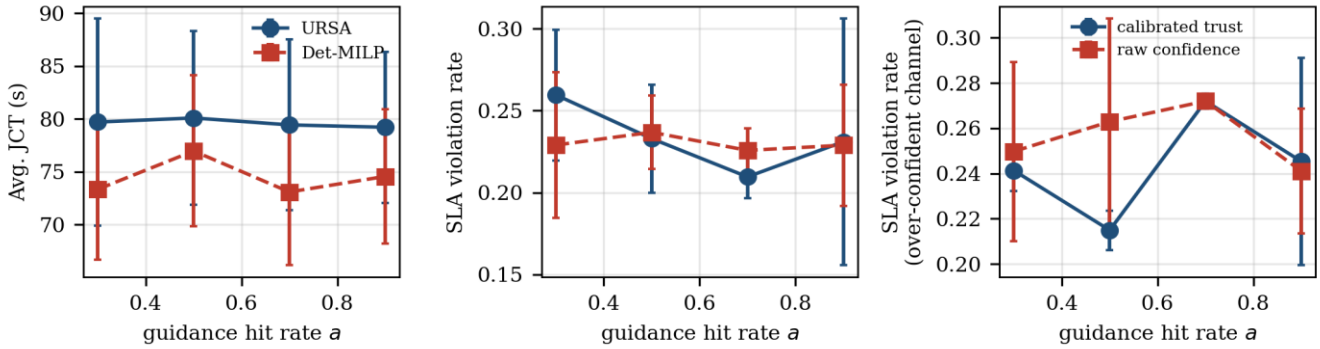


Figure 5Left and centre: behaviour as advisory reliability falls. Right: calibrated trust against raw confidence when the channel is over-confident

F. What the Two Estimators Actually Deliver

Fig. 6 inspects the two uncertainty estimators directly. The left panel plots realised hit rate against reported agreement. Agreement orders proposals correctly—higher agreement does mean a higher chance of being right—but it is not a probability: for the well-specified channel the curve sits above the diagonal, with an expected calibration error of 0.147, and compressing the reported score pushes the curve the other way, to 0.065. Either distortion is a reason to pass the score through a monotone map before it multiplies a trust coefficient, and the downstream effect of doing so is the one measured in the previous subsection.

The centre panel shows why point runtime estimates are inadequate for deadline work. Multiplicative residuals on the measured Spark stages have a median of 1.01, an 80th percentile of 2.43 and a 95th percentile of 18.87: one stage in twenty runs an order of magnitude longer than predicted. A scheduler that plans a deadline-critical path against the median is planning against a quantity that the workload exceeds routinely.

The right panel validates the conformal construction on those residuals. At nominal 0.80 the realised coverage over 3437 logged stages is 0.800, at nominal 0.90 it is 0.900, and at nominal 0.95 it is 0.951. Coverage tracks the level to three decimal places without any distributional assumption, which is the property that makes the risk-adaptive rule (5) meaningful: asking for a 95% bound on a tight path returns a 95% bound.

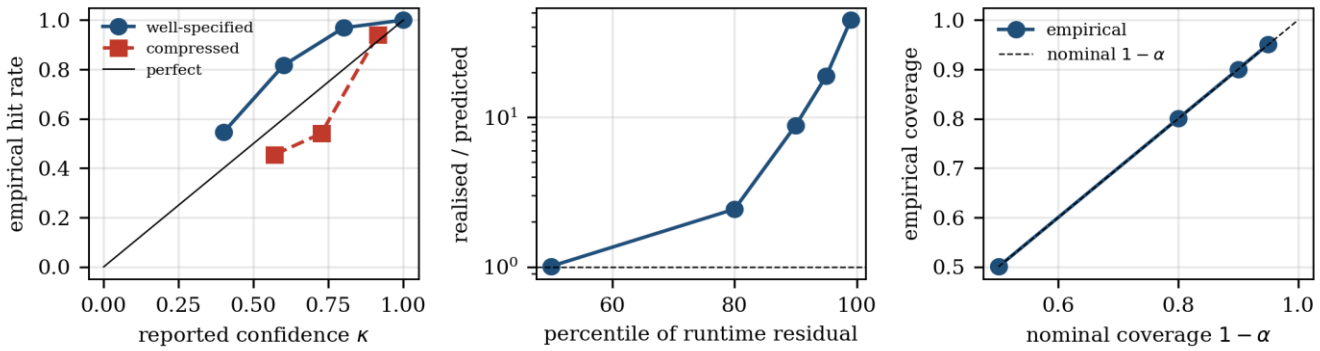


Figure 6Left: realised hit rate against reported agreement, for a well-specified and a compressed channel. Centre: the tail of the multiplicative runtime residual on measured Spark stages. Right: realised conformal coverage against its nominal level.

G. Distribution Shift

Table IV re-runs the schedulers on workloads the estimators were not tuned for: only deep DAGs, a surge of urgent deadline-carrying jobs, and a demand error half again as large as the default. URSA degrades by 38.5% at worst, against 48.8% for the deterministic program. Nothing is retrained between settings; the conformal quantiles widen because the residuals widen, the risk-adaptive level selects the high quantile more often because slack shrinks, and the budgeted set is written in relative terms so it scales with the error automatically. Adaptation without retraining is a property of carrying uncertainty rather than of learning a policy.

Table 4ROBUSTNESS UNDER WORKLOAD DISTRIBUTION SHIFT

Setting	Scheduler	Avg. JCT (s)	Degrad.	SLA viol.	Pressure ev.
Base	**URSA**	**77.2 ± 7.8**	**--**	**0.222**	**6.0**
	Det-MILP	76.1 ± 6.0	--	0.229	75.0
	Tetris	125.9 ± 13.8	--	0.441	78.0
Structure shift	**URSA**	**106.9 ± 6.3**	**+38.5%**	**0.453**	**5.0**
	Det-MILP	113.2 ± 13.2	+48.8%	0.430	170.3
	Tetris	177.9 ± 13.4	+41.3%	0.627	145.0
Priority shift	**URSA**	**82.9 ± 18.4**	**+7.4%**	**0.624**	**3.3**
	Det-MILP	84.7 ± 14.9	+11.3%	0.635	111.3
	Tetris	130.7 ± 17.2	+3.8%	0.770	126.0
Demand shift	**URSA**	**84.7 ± 8.7**	**+9.8%**	**0.270**	**11.3**
	Det-MILP	85.1 ± 3.0	+11.8%	0.282	171.3
	Tetris	122.6 ± 7.5	+-2.6%	0.433	133.7

H. Overhead and Invocation Frequency

Table V sweeps the risk gate. With the gate open at every epoch the program runs 100% of the time at a median epoch cost of 5.37 ms; raising θ to 0.60 cuts invocations to 7% and the median cost to 1.17 ms, while completion time moves by 3.2%. The gate is therefore a usable knob for deployments that pay per advisory call: most epochs are routine and the greedy fallback, which shares the robust admission rule, handles them without giving up the capacity guarantee. Absolute cost stays inside a production budget. The full program on a 20-stage window over 6 candidates each solves in single-digit milliseconds at the median and under 20.7 ms at the 99th percentile, well inside the 50 ms budget it is given, and it is exercised only on epochs the gate selects.

Table 5EFFECT OF THE RISK GATE ON INVOCATION RATE AND LATENCY

Gate θ	Invoc. rate	P50 lat. (ms)	P99 lat. (ms)	Avg. JCT (s)	SLA viol.
0.00	100%	5.37	15.07	84.6	0.273
0.15	**95%**	**5.03**	**14.87**	**82.8**	**0.243**
0.30	91%	5.02	15.69	83.9	0.270
0.45	17%	1.34	12.16	87.4	0.315
0.60	7%	1.17	9.71	87.3	0.315

VII. DISCUSSION

Three Observations Generalise Beyond This Scheduler.

The scarce resource is trust, not intelligence. An advisory channel that is right seven times in ten contributed roughly a tenth of the SLA improvement in our ablation, while modelling the error in declared memory demand contributed an order of magnitude more. Systems work that adds a reasoning component to a control loop is well advised to measure the error budget before enlarging the reasoning.

Confidence has to be earned from behaviour, not read off the channel. Agreement across samples is observable, cheap and monotone in correctness; a number the channel states about itself is none of those. The calibration step that turns agreement into a probability is a few hundred logged decisions and one monotone map, and it is what lets a single scalar weight be adjusted automatically instead of hand-tuned per deployment. Robustness was free in this workload because its cost and its benefit are paid in the same currency. Reserving headroom removes capacity; evictions destroy capacity that has already been spent. Whenever the second effect is comparable to the first—and it is, wherever request-to-usage gaps are material—a small uncertainty budget dominates the deterministic program on both throughput and stability. The formulation carries over directly to settings where the same two uncertainties appear: serverless placement, where declared memory is a billing unit rather than a measurement, and multi-tenant inference serving, where a scheduler routes requests under a latency objective with a predictor whose residuals are heavy-tailed. A natural extension is to learn the budget Γ online from the observed violation rate, closing the last hand-set parameter in the loop.

VIII. CONCLUSION

Guided schedulers gain from pairing a reasoning channel with a solver, but the pairing is only as good as the estimates that cross between them. URSA makes those estimates explicit: agreement across advisory samples becomes a calibrated trust weight, runtime residuals become conformal upper bounds at a level set by deadline slack, and declared-versus-actual demand deviation becomes a budgeted uncertainty set whose robust counterpart stays linear. On real Spark TPC-H job graphs with measured task durations, the result matches a deterministic guided program on completion time while removing 95–96% of node memory-pressure and eviction events, shortening makespan by up to 12.9% and raising utilisation by up to 11.2%, and it holds that behaviour as advisory reliability falls to 0.30, under over-confident guidance, and across deep-DAG, urgent-workload and high-error shifts. Uncertainty is cheap to carry and expensive to ignore.

REFERENCES

Xing, S., Wang, Y., & Liu, W. (2025). Self-adapting CPU scheduling for mixed database workloads via hierarchical deep reinforcement learning. Symmetry, 17(7), 1109.

Ji, E., Wang, Y., Xing, S., & Jin, J. (2025). Hierarchical reinforcement learning for energy-efficient API traffic optimization in large-scale advertising systems. IEEE Access.

Chen, T., Shang, W., & Guo, Z. (2024). Xsla: An SLA-aware heterogeneous XPU scheduling framework for edge AI inference. World Journal of Information Technology, 2(2), 48–59.
Zou, J., Qin, Y., & Rhee, M. (2024). Beyond model scale: A task-aware reliability benchmark for large language models in clinical decision support. Innovation and Technology Studies, 1(1), 39–47.
Fang, Q. (2024). Bias-aware large language models for equitable educational feedback and assessment. Trends in Social Sciences and Humanities Research, 2(1), 41–50.
Qin, Y., Zou, J., & Rhee, M. (2022). An AI-driven intelligent transaction execution framework for internet finance platforms: Cost-sensitive real-time integrity interdiction. Eurasia Journal of Science and Technology, 4(2), 36–44.
Wang, Y., & Zhang, C. (2023). AERIS: Adaptive executor scoped resource management for privacy preserving data processing in Apache Spark. Eurasia Journal of Science and Technology, 5(2), 25-33.
Zhao, W., & Wang, B. (2022). CAPA: A prediction-driven autoscaling framework for SLO-aware cloud-native machine learning inference. Journal of Computer Science and Electrical Engineering, 4(1), 8-16.
Guo, Z., Chen, T., & Shang, W. (2022). Class-conditional intermediate-domain adversarial adaptation for cross-device acoustic scene classification. Journal of Computer Science and Electrical Engineering, 4(1), 26-35
Shen, A. (2023). Compute-efficient scaling of large sequential recommendation models under dynamic user behavior. World Journal of Information Technology, 1(1), 62–71.
Fang, Q. (2024). True-EWS: Trustworthy early warning with equalized conformal guarantees for equitable student support in U.S. education. Educational Research and Human Development, 1(2), 44–54.
Shen, A., & Huang, S. (2022). A systems-aware framework for intelligent large-scale recommendation models. Journal of Computer Science and Electrical Engineering, 4(1), 36–44.Mai, N. T., Cao, W., & Fang, Q. (2025). A study on how LLMs (eg GPT-4, chatbots) are being integrated to support tutoring, essay feedback and content generation. Journal of Computing and Electronic Information Management, 18(3), 43-52.
Fang, Q., & Liu, W. (2026). Neuro-symbolic pedagogical alignment (NSPA) for long-horizon classroom discourse analysis: Mitigating dialect bias via counterfactual preference optimization. Computers and Education: Artificial Intelligence, 100664.
Fang, Q. (2024). TRUE-EWS: Trustworthy early warning with equalized conformal guarantees for equitable student support in U.S. education. Educational Research and Human Development.
Shen, A. (2023). Compute-efficient scaling of large sequential recommendation models under dynamic user behavior. World Journal of Information Technology..
D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in Proc. USENIX OSDI, 2020, pp. 481–498.
A. Qiao et al., "Pollux: co-adaptive cluster scheduling for goodput-optimized deep learning," in Proc. USENIX OSDI, 2021, pp. 1–18.
S. Kambhampati et al., "Position: LLMs can't plan, but can help planning in LLM-modulo frameworks," in Proc. Int. Conf. Mach. Learn. (ICML), 2024.
K. Valmeekam, A. Olmo, S. Sreedharan, and S. Kambhampati, "On the planning abilities of large language models—a critical investigation," in Proc. NeurIPS, 2023, pp. 75993–76005.
T. Brown et al., "Language models are few-shot learners," in Proc. NeurIPS, 2020, pp. 1877–1901.
J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in Proc. NeurIPS, 2022, pp. 24824–24837.
X. Wang et al., "Self-consistency improves chain of thought reasoning in language models," in Proc. Int. Conf. Learn. Represent. (ICLR), 2023.
OpenAI, "GPT-4 technical report," 2023, arXiv:2303.08774.

H. Touvron et al., "Llama 2: open foundation and fine-tuned chat models," 2023, arXiv:2307.09288.
M. Chen et al., "Evaluating large language models trained on code," 2021, arXiv:2107.03374.
S. Kadavath et al., "Language models (mostly) know what they know," 2022, arXiv:2207.05221.
G. Ding, S. Yang, H. Lin, Z. Chen, and J. S. Yang, "LLM-driven adaptive cloud resource scheduling: bridging reasoning intelligence with optimization guarantees," IEEE Open J. Comput. Soc., vol. 7, pp. 560–573, 2026.
M. Xiong, Z. Hu, X. Lu, Y. Li, J. Fu, J. He, and B. Hooi, "Can LLMs express their uncertainty? An empirical evaluation of confidence elicitation in LLMs," in Proc. Int. Conf. Learn. Represent. (ICLR), 2024.
S. Lin, J. Hilton, and O. Evans, "Teaching models to express their uncertainty in words," Trans. Mach. Learn. Res., 2022.
C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in Proc. Int. Conf. Mach. Learn. (ICML), 2017, pp. 1321–1330.
B. Zadrozny and C. Elkan, "Transforming classifier scores into accurate multiclass probability estimates," in Proc. ACM SIGKDD, 2002, pp. 694–699.
D. Bertsimas and M. Sim, "The price of robustness," Oper. Res., vol. 52, no. 1, pp. 35–53, 2004.
A. Ben-Tal and A. Nemirovski, "Robust solutions of uncertain linear programs," Oper. Res. Lett., vol. 25, no. 1, pp. 1–13, 1999.
D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," SIAM Rev., vol. 53, no. 3, pp. 464–501, 2011.
E. Delage and Y. Ye, "Distributionally robust optimization under moment uncertainty with application to data-driven problems," Oper. Res., vol. 58, no. 3, pp. 595–612, 2010.
V. Vovk, A. Gammerman, and G. Shafer, Algorithmic Learning in a Random World. New York, NY, USA: Springer, 2005.
Wang, Y., & Xing, S. (2025). AI-driven CPU resource management in cloud operating systems. Journal of Computer and Communications, 13(06), 135-149.
Xing, S., & Wang, Y. (2025). Cross-modal attention networks for multi-modal anomaly detection in system software. IEEE Open Journal of the Computer Society.
J. D. Ullman, "NP-complete scheduling problems," J. Comput. Syst. Sci., vol. 10, no. 3, pp. 384–393, 1975.
Yue, X., & Zhu, R. (2024). An explainable hybrid machine learning framework for cash-flow-conditioned multi-horizon liquidity risk early warning: An SME-oriented benchmark study. World Journal of Management Science, 2(1), 63-72.