An Edge–Cloud Cooperative Inference System for Video Analytics with Adaptive Resolution and Early-Exit Routing

Faisal Mansouri[1] and Hassan Mahroos[2]

[1]Applied Science University, Department of Computer Science and Engineering, Samaheej Street 42, Al Eker 623, Bahrain
[2]Gulf University, Department of Computer Science and Engineering, Avenue 12, Sanad District, Riffa 905, Bahrain

Abstract

Video analytics pipelines increasingly span resource-constrained edge devices and elastic cloud backends, motivated by the need to reduce end-to-end latency while maintaining accuracy under bursty visual complexity and fluctuating network conditions. At the same time, modern deep models are over-provisioned for many frames, and naïvely transmitting full-resolution video to the cloud wastes bandwidth and energy. This paper presents an edge–cloud cooperative inference system for streaming video analytics that jointly adapts spatial resolution and performs early-exit routing across a multi-exit neural network, while deciding whether to complete inference on the edge or offload intermediate representations to the cloud. The system treats resolution selection, exit selection, and placement as a coupled control problem under latency, energy, and bandwidth budgets. A unified formulation integrates rate–distortion intuitions for adaptive encoding, uncertainty-calibrated confidence gating for early exits, and placement-aware feature compression for communication efficiency. The design includes an online policy that uses embedding similarity to exploit temporal redundancy, a cost model grounded in operator-level performance measurements, and a distributed execution runtime that supports pipelined micro-batching and backpressure. The paper details optimization strategies spanning constrained stochastic gradient methods, Lagrangian relaxation, and Pareto-style trade-offs between accuracy and resource usage, and provides an evaluation methodology emphasizing trace-driven reproducibility, error decomposition, and stress testing under network variability.

1 Introduction

Edge inference has become a default architectural choice for interactive visual applications because transmitting raw video to a remote datacenter often incurs latency and cost that are disproportionate to the value of each frame [1]. Yet purely on-device inference is bounded by thermal envelopes, limited memory bandwidth, and the heterogeneity of embedded accelerators, which frequently forces the use of smaller models or lower frame rates. Cloud inference, by contrast, enables large models and scalable throughput but is gated by network round-trip time, jitter, congestion, and privacy or policy constraints. The tension is acute in continuous video analytics where the workload is nonstationary: a static hallway may yield many redundant frames that can be processed cheaply, while a crowded street corner triggers heavy computation due to multiple objects, occlusions, and long-range temporal dependencies. A practical system must therefore spend resources proportional to difficulty, rather than per-frame uniformly [2].

Two observations motivate an integrated edge–cloud cooperative approach. The first is that resolution is not merely a pre-processing knob but a primary determinant of both computation and information content. Lowering resolution reduces arithmetic intensity in convolutional backbones and decreases memory traffic, but it can destroy small objects and fine-grained cues needed for detection or action recognition. Resolution should thus be selected adaptively and with awareness of downstream model behavior rather than as a fixed global setting. The second observation is that deep networks often support early termination without catastrophic accuracy loss [3]. Multi-exit architectures expose intermediate classifiers that can correctly predict easy frames after fewer layers. When coupled with confidence estimates, early exits can dramatically reduce average latency and energy. However, early exiting interacts with resolution: a frame that is easy at high resolution may become ambiguous at low resolution, and

a frame that is hard on the edge may become easy if offloaded to a larger cloud model or completed after deeper layers [4].

Edge–cloud cooperation extends beyond a binary decision of whether to offload raw pixels. A spectrum exists: the edge may compute a prefix of the network, producing intermediate feature maps that are then compressed and transmitted; the cloud may compute a suffix, potentially using a larger backbone or performing refinement [5]. This co-inference setting raises fundamental systems questions about where to cut the model, how to represent and compress intermediate activations, how to schedule pipelined execution across heterogeneous hardware, and how to guarantee stable throughput under bursty conditions. It also raises modeling questions: the system must decide, for each frame or short clip, how to choose resolution, which exit to use, and whether to offload at which cut, all while respecting budgets and maintaining accuracy [6].

This paper presents a technical design for an edge–cloud cooperative inference system for video analytics with adaptive resolution and early-exit routing. The core contribution is not a single heuristic but a cohesive end-to-end architecture that treats resolution control, exit gating, and placement as a joint constrained optimization problem [7]. The system is designed around a performance model that decomposes latency into compute, memory, and communication components, and it couples this with an accuracy model based on uncertainty-calibrated confidence scores and temporal redundancy signals derived from embeddings. The runtime provides operator-level instrumentation, supports micro-batched execution for GPU-like accelerators, and exposes a control loop that adjusts decisions online based on observed queueing delay and network conditions.

A key theme is that online control must be backprop-friendly during training yet robust during deployment. The decision variables are discrete, but differentiable surrogates can be used to train gating networks and resolution policies. The system adopts a hybrid strategy: it trains soft policies with continuous relaxations and then deploys hard decisions with safeguards, including conservative thresholds and drift detectors [8]. Another theme is communication efficiency: intermediate representations are compressed using quantization and entropy-aware coding principles, and the system exploits temporal similarity to reuse computations when successive frames share semantically similar embeddings. This reduces bandwidth consumption and amortizes compute across time.

The paper proceeds by formalizing the cooperative inference problem as a constrained stochastic control objective, then describing adaptive resolution mechanisms grounded in rate–distortion trade-offs and compute models. It then develops early-exit routing with calibrated uncertainty and placement-aware gating, followed by a systems section on distributed execution mechanics and storage internals that enable stable streaming performance. The final technical section integrates optimization, evaluation methodology, and reproducibility considerations, with emphasis on error analysis for approximate decisions and sketches [9]. The paper concludes by synthesizing the practical implications and the remaining open engineering considerations for robust deployment.

2 Formal Model and Cooperative Architecture

Consider a video stream as a sequence of frames or short clips indexed by time t. Let $x_t \in \mathbb{R}^{H \times W \times C}$ denote the raw input, and let y_t denote the target output for a given analytics task such as detection, classification, segmentation, or action recognition. A deep model is parameterized by θ and organized as a sequence of blocks, producing hidden states $h_t^{(\ell)}$ at depth ℓ. A multi-exit architecture attaches classifiers or heads at a subset of depths $\mathcal{E} \subset \{1, \ldots, L\}$, producing predictions $\hat{y}_t^{(e)}$ for $e \in \mathcal{E}$. The system can terminate at an exit e or continue deeper to a later exit. Additionally, the system can execute a prefix on the edge and a suffix on the cloud by selecting a cut layer $c \in \{0, \ldots, L\}$, where $c = 0$ corresponds to sending pixels and $c = L$ corresponds to fully edge execution. When $c \in (0, L)$, the edge computes $h_t^{(c)}$ and transmits a compressed representation to the cloud, which then completes computation to an exit $e \geq c$.

Adaptive resolution is modeled as a transformation T_r that maps the raw frame to a chosen resolution parameter r, which may encapsulate spatial scaling, region-of-interest cropping, and possibly temporal subsampling over clips. For simplicity, let r denote a discrete set of supported scales, where $T_r(x_t) \in \mathbb{R}^{H_r \times W_r \times C}$. The decision at time t is a tuple $a_t = (r_t, c_t, e_t)$, where r_t is the resolution, c_t is the cut placement, and e_t is the exit depth. The system observes state s_t that may include recent embeddings, network statistics, queue lengths, and hardware utilization [10]. The system then selects a_t via a policy $\pi_\phi(a_t \mid s_t)$ parameterized by ϕ, which can be trained offline and adapted online.

Accuracy is measured by a loss $\ell(y_t, \hat{y}_t^{(a_t)})$ where $\hat{y}_t^{(a_t)}$ is the prediction produced under decision a_t. Resource costs include latency $\tau_t(a_t)$, energy $E_t(a_t)$, and bandwidth $B_t(a_t)$. Latency decomposes into computation time on the edge for the prefix, communication time for transmitting either pixels or features, and computation time on the cloud for the suffix, plus queueing and scheduling overhead. A concrete decomposition uses [11]

$$\tau_t(a_t) = \tau^{\text{edge}}(r_t, c_t) + \tau^{\text{tx}}(r_t, c_t) + \tau^{\text{cloud}}(c_t, e_t) + \tau_t^{\text{queue}} \quad . \tag{1}$$

Here $\tau^{\text{edge}}(r, c)$ depends on resolution and cut because earlier layers have different compute profiles and memory

footprints, while $\tau^{\text{tx}}(r, c)$ depends on whether pixels or features are sent and the chosen compression. Queueing delay τ_t^{queue} captures contention and is observable at runtime via measured backlog.

The system objective can be expressed as constrained risk minimization over the stream distribution:

$$\min_{\phi,\theta} \quad \mathbb{E}\big[\ell(y_t, \hat{y}_t^{(a_t)})\big] \tag{2}$$

$$\text{s.t.} \quad \mathbb{E}\big[\tau_t(a_t)\big] \le \bar{\tau}, \quad \mathbb{E}\big[E_t(a_t)\big] \le \bar{E}, \quad \mathbb{E}\big[B_t(a_t)\big] \le \bar{B} \quad , \tag{3}$$

where expectation is taken over the stream and policy-induced actions. A Lagrangian relaxation yields a multi-objective unconstrained form:

$$\min_{\phi,\theta} \quad \mathbb{E}\Big[\ell(y_t, \hat{y}_t^{(a_t)}) + \lambda\, \tau_t(a_t) + \mu\, E_t(a_t) + \nu\, B_t(a_t)\Big] \quad , \tag{4}$$

where (λ, μ, ν) trade off accuracy and costs. In deployment, λ, μ, ν can be interpreted as tunable knobs to navigate a Pareto frontier, while in training they can be adjusted via dual ascent to meet budgets [12]. To avoid instability under nonstationary workloads, the system can use an exponentially weighted moving average for costs and apply a proximal update to dual variables:

$$\lambda_{k+1} = \Big[\lambda_k + \eta_\lambda\big(\widehat{\tau}_k - \bar{\tau}\big)\Big]_+ \quad , \quad \mu_{k+1} = \Big[\mu_k + \eta_\mu\big(\widehat{E}_k - \bar{E}\big)\Big]_+ \quad , \quad \nu_{k+1} = \Big[\nu_k + \eta_\nu\big(\widehat{B}_k - \bar{B}\big)\Big]_+ \quad , \tag{5}$$

where $[\cdot]_+$ denotes projection onto nonnegative reals and $\widehat{\tau}_k$ is the smoothed observed latency at control step k.

A central architectural concern is the representation transmitted when $c \in (0, L)$. Let $z_t = h_t^{(c)}$ be the activation tensor produced at the cut. Direct transmission is bandwidth-prohibitive, so the system applies a compressor Q_ψ parameterized by ψ to produce a bitstream $b_t = Q_\psi(z_t)$, and a decoder D_ψ reconstructs $\tilde{z}_t = D_\psi(b_t)$ on the cloud. For compatibility with end-to-end training, Q_ψ is designed with differentiable surrogates such as additive uniform noise for quantization during training and hard rounding during deployment. The cloud then computes the suffix using $\tilde{z}_t$. This yields an additional distortion term in the loss due to feature reconstruction error. A practical modeling approach is to incorporate a reconstruction penalty: [13]

$$\ell_{\text{total}} = \ell(y_t, \hat{y}_t) + \beta\, \|z_t - \tilde{z}_t\|_2^2 \quad , \tag{6}$$

with β tuned to prevent the compressor from discarding task-relevant information.

Temporal redundancy is a defining characteristic of video. The system maintains an embedding function g_θ that maps a frame or a lightweight feature to an embedding vector $u_t \in \mathbb{R}^d$. Similarity between frames can be measured via cosine similarity $\text{sim}(u_t, u_{t-1}) = \frac{\langle u_t, u_{t-1}\rangle}{\|u_t\|\|u_{t-1}\|}$ or via Euclidean distance after normalization. If similarity is high, the system can reuse past results, skip expensive computation, or choose lower resolution. To make embedding storage efficient and queryable at high throughput, embeddings can be projected into a lower-dimensional subspace via PCA or SVD. If $U \in \mathbb{R}^{d\times n}$ is a matrix of recent embeddings, an SVD $U \approx P\Sigma V^\top$ yields a rank-k approximation $U_k = P_k \Sigma_k V_k^\top$. The projected embedding is $\bar{u}_t = P_k^\top u_t$, reducing memory and accelerating similarity search. This projection also reduces noise and can stabilize gating decisions [14].

The cooperative architecture is therefore comprised of an edge pipeline that performs resolution transformation, optionally computes a model prefix and embedding, applies feature compression when offloading, and maintains local caches and performance monitors. The cloud pipeline receives either pixels or compressed features, completes inference possibly with a larger model variant, and can return outputs and optional feedback signals such as refined confidence calibration parameters. The control plane spans both sides, using measured throughput and latency to adjust π_ϕ. Importantly, the system is designed so that decisions can be made per frame or per clip, but the runtime can also amortize overhead by executing decisions in short windows when stability is desired [15].

3 Adaptive Resolution and Resource-Aware Encoding

Resolution adaptation is approached as a joint information and systems problem. On the information side, scaling alters the mutual information between input and label and affects the separability of classes in feature space. On the systems side, scaling changes arithmetic intensity, memory bandwidth demands, cache behavior, and encoding rates for transmission. A naive strategy that reduces resolution whenever bandwidth drops may cause catastrophic accuracy loss on small-object scenes, while a naive accuracy-first strategy may exceed latency budgets during bursts. The goal is to map observed state s_t to a resolution r_t that is justified by both predicted difficulty and available resources [16].

A useful abstraction is a rate–distortion view where the system chooses an operating point that balances expected task loss and resource costs. Let $R(r)$ be the expected number of transmitted bits when using resolution

r and a given encoding strategy, and let $D(r)$ be an expected task distortion induced by resolution, which can be approximated by the increase in loss relative to a reference resolution. Although classical rate–distortion theory is defined for reconstruction distortion, the same conceptual trade-off applies to task loss, with the caveat that $D(r)$ is label-dependent and model-dependent. A pragmatic approach is to learn a surrogate predictor $\widehat{D}(s_t, r)$ using offline profiling data, where s_t includes a lightweight difficulty proxy such as edge embedding norms, motion magnitude estimates, or entropy of a shallow classifier's logits. The system then chooses r_t by minimizing an estimated cost:

$$r_t = \arg\min_{r \in \mathcal{R}} \quad \widehat{D}(s_t, r) + \nu\, \widehat{R}(s_t, r) + \lambda\, \widehat{\tau}^{\text{edge}}(s_t, r) \quad , \tag{7}$$

where $\widehat{R}$ and $\widehat{\tau}^{\text{edge}}$ are predicted bitrate and edge compute time under resolution r.

To connect bitrate to compressibility, consider that transmitted content may be either pixels or features [17]. For pixel transmission, the encoder can be configured with a quantization parameter that trades rate for visual fidelity. For feature transmission, the compressor Q_ψ produces a latent code whose entropy controls expected bit rate. If the compressor outputs discrete symbols c_t with distribution $p_\psi(c)$, the expected code length under an entropy coder is bounded by the entropy:

$$\mathbb{E}[\text{bits}(c_t)] \geq H(c_t) = -\sum_i p_\psi(c = i) \log_2 p_\psi(c = i) \quad . \tag{8}$$

In practice, the system uses learned probability models for latents, enabling arithmetic coding close to entropy. Resolution affects entropy because lower-resolution inputs often yield smoother feature maps and narrower latent distributions, reducing $H(c_t)$ [18].

From a compute perspective, resolution affects the per-layer cost. For a convolutional layer with kernel size $k \times k$, input channels C_{in}, output channels C_{out}, and output spatial size $H_r \times W_r$, the multiply-add count scales as $O(H_r W_r k^2 C_{\text{in}} C_{\text{out}})$. Memory traffic and cache reuse depend on layout and blocking, but a coarse scaling law suggests compute roughly proportional to $H_r W_r$. For transformer-like architectures with patch embeddings, the token count scales similarly with resolution, and attention costs can scale quadratically in token count unless sparse approximations are used. Hence resolution choice has a first-order impact on both edge and cloud compute [19].

A systems-grade resolution controller should also incorporate the overhead of resizing and decoding. Downsampling itself costs time, and repeated resizes can be expensive for high frame rates. The pipeline therefore treats resizing as an operator with measured latency $\tau^{\text{resize}}(r)$ and integrates it into cost. Additionally, the controller can prefer resolution values that align with hardware-friendly dimensions, reducing padding and enabling vectorized kernels. On some accelerators, certain multiples yield better memory coalescing, and the controller can incorporate a discrete feasibility set $\mathcal{R}_{\text{hw}} \subset \mathcal{R}$ to avoid pathological sizes.

Adaptive resolution can be improved by region-aware approaches. Many analytics tasks depend on localized content such as objects or faces [20]. Instead of uniformly scaling the entire frame, the system can apply a two-stage scheme: a low-resolution scan produces coarse proposals, and then selected regions are processed at higher resolution. The difficulty is that region selection must be cheap and must not require the very computation it is trying to save. A practical compromise is to use a lightweight edge model to generate candidate regions at low cost, then crop and optionally super-resolve those regions before feeding them to the heavy backbone. The compute model must include the cost of cropping, potential GPU kernel launch overhead, and any increased memory fragmentation due to variable-sized inputs. Because variable shapes complicate batching, the runtime can bucket crops into a small set of canonical sizes, trading slight resampling distortion for stable throughput [21].

The control loop for resolution can be cast as a stochastic decision problem with partial observability. Let m_t denote a motion statistic derived from optical flow approximations or block matching, and let e_t denote the entropy of a shallow classifier on the edge. These signals correlate with difficulty but are imperfect. The system can model the latent difficulty d_t as a hidden variable with a Bayesian-ish update: [6]

$$p(d_t \mid s_{1:t}) \propto p(s_t \mid d_t) \int p(d_t \mid d_{t-1})\, p(d_{t-1} \mid s_{1:t-1})\, \mathrm{d}d_{t-1} \quad . \tag{9}$$

In deployment, a full Bayesian filter is often too heavy, but its structure motivates a simple exponential smoothing of difficulty proxies and a conservative decision rule that avoids rapid oscillation. The system can also learn a small recurrent network as a proxy filter, producing a smoothed difficulty score $\hat{d}_t$ used in resolution selection.

Temporal redundancy enables additional compression through reuse. If the embedding similarity between consecutive frames exceeds a threshold, the system may reuse the previous model output for a short horizon or run only a cheap verifier. To formalize, let $\Delta_t = \|\bar{u}_t - \bar{u}_{t-1}\|_2$. For small Δ_t, the output is likely stable. A Lipschitz-like approximation in embedding space suggests [22]

$$\|\hat{y}_t - \hat{y}_{t-1}\| \leq K\, \Delta_t \quad , \tag{10}$$

for some task-dependent constant K that can be estimated empirically. This motivates a skip rule when Δ_t is below a learned threshold that depends on the current uncertainty and scene context. Because mistakes accumulate when skipping too aggressively, the system enforces periodic refreshes and uses drift detection based on embedding statistics to trigger higher-resolution processing.

Low-rank approximations appear both in model execution and in state representation. For feature tensors transmitted at a cut layer, if z_t is shaped as $(C \times H \times W)$, one can reshape to a matrix $Z_t \in \mathbb{R}^{C\times(HW)}$ and approximate it by rank-k factors $A_t \in \mathbb{R}^{C\times k}$ and $B_t \in \mathbb{R}^{k\times(HW)}$, where $Z_t \approx A_t B_t$. Transmitting A_t and B_t can reduce bits when $k \ll \min(C, HW)$, but the factorization cost can be large [23]. The system therefore prefers learned linear projections that approximate low-rank structure implicitly: a 1×1 convolution or linear layer maps z_t to a bottleneck latent $q_t \in \mathbb{R}^{C'\times H\times W}$ with $C' < C$, which is then quantized and entropy-coded. This is essentially a learned low-rank compression in channel space, and it aligns with accelerator-friendly operations.

The controller must incorporate not only expected compute but also worst-case constraints for tail latency. Streaming analytics is often sensitive to high percentiles, not just mean. A system that occasionally selects a very high resolution might violate tail latency budgets even if average is acceptable. This motivates a risk-sensitive objective, for example minimizing a conditional value-at-risk proxy for latency: [24]

$$\min \quad \mathbb{E}[\ell] + \lambda\, \mathrm{CVaR}_\alpha(\tau) \quad , \tag{11}$$

where α is a high quantile. In practice, CVaR can be estimated online by tracking high-percentile latency via sketches such as t-digests, and the controller can penalize decisions that increase the upper tail. The error analysis of such sketches is relevant because approximate quantile estimation can mislead the controller; therefore the system maintains confidence intervals for percentile estimates and reduces aggressiveness when estimation uncertainty grows due to low sample counts.

Finally, resolution adaptation must be integrated with codec-level concerns when pixels are transmitted [25]. If the edge streams video segments to the cloud, the system can adjust quantization parameters and keyframe intervals. Keyframes reduce prediction drift but increase bitrate, while longer GOPs reduce bitrate but may worsen cloud-side analytics if compression artifacts align with important edges. The controller can treat codec settings as part of r_t or as a coupled decision variable. The system also benefits from encoding features instead of pixels whenever possible, because feature coding avoids spending bits on nuisance details. However, feature coding shifts complexity to the compressor and may reduce portability across model updates [26]. To address this, the system version-controls compressor parameters and supports gradual rollout where the cloud can decode multiple compressor versions during transitions.

4 Early-Exit Routing and Uncertainty-Calibrated Decisioning

Early-exit networks attach intermediate heads to a backbone, allowing inference to terminate early when predictions are sufficiently confident. For video analytics, early exits can yield large savings because many frames are easy, particularly when the scene is static or when the task is coarse. Yet naive confidence gating based on max softmax probability is often miscalibrated, especially under domain shift or when resolution changes. Moreover, early exit interacts with edge–cloud placement: a frame that would be uncertain at a shallow edge exit might become confident at a deeper cloud exit, but offloading introduces communication and queueing costs [27]. The system thus requires a routing policy that is both uncertainty-aware and cost-aware.

Let the multi-exit model produce logits $z_t^{(e)}$ at exit e and probabilities $p_t^{(e)} = \mathrm{softmax}(z_t^{(e)})$. A standard confidence proxy is $c_t^{(e)} = \max_i p_t^{(e)}(i)$, but this can be overconfident. The system employs calibration mechanisms that can be trained on held-out data. One simple method is temperature scaling, where calibrated probabilities are $\mathrm{softmax}(z/T)$ with temperature $T > 0$ learned to minimize negative log-likelihood. More expressive calibration uses a small monotone network mapping logits to calibrated confidences. Calibration must be resolution-conditioned, because downsampling can alter logit distributions. The system can therefore maintain calibration parameters per resolution bucket or condition calibration on a low-dimensional summary of r_t [28].

A more robust uncertainty estimate is predictive entropy $H(p_t^{(e)}) = -\sum_i p_t^{(e)}(i) \log p_t^{(e)}(i)$, which increases when the distribution is diffuse. For dense tasks, analogous measures exist such as pixel-wise entropy aggregated over regions. Bayesian-ish uncertainty can also be approximated using dropout at inference or ensembles, but these are often too expensive for edge constraints. A compromise is to use a single network with an auxiliary head that predicts expected error, trained with a regression target derived from correctness. Let $q_t^{(e)}$ denote a predicted failure probability. The exit decision then compares expected loss at the current exit with expected cost of continuing [29].

A cost-aware early-exit rule can be derived by comparing expected downstream improvement against additional cost. Let $\mathcal{L}_t^{(e)}$ be the expected loss if exiting at e, approximated by a function of uncertainty, and let $\Delta\tau_t^{(e\to e')}$

be the incremental latency to continue from e to a later exit e', potentially on a different device if offloading. An optimal stopping view suggests exiting when

$$\mathbb{E}[\mathcal{L}_t^{(e)} \mid s_t] + \lambda\, \mathbb{E}[\tau_t^{(e)} \mid s_t] \le \mathbb{E}[\mathcal{L}_t^{(e')} \mid s_t] + \lambda\, \mathbb{E}[\tau_t^{(e')} \mid s_t] \quad , \tag{12}$$

for all $e' > e$, where $\tau_t^{(e)}$ denotes total latency if exiting at e. Because evaluating all future exits is expensive, the system uses a learned value function $V_\omega(s_t, e)$ approximating the expected downstream objective from state (s_t, e), enabling a one-step decision:

$$\text{exit at } e \quad \text{if} \quad \widehat{\mathcal{L}}_t^{(e)} + \lambda\, \widehat{\tau}_t^{(e)} \le \widehat{V}_\omega(s_t, e+1) \quad . \tag{13}$$

This resembles reinforcement learning but can be trained using supervised traces generated by profiling multiple exits on recorded video, enabling off-policy estimation of the objective without online exploration in production.

Early-exit routing must also contend with error propagation in video tasks [30]. If a detector exits early and produces a slightly inaccurate bounding box, subsequent tracking may drift. To address this, the system introduces task-aware gating that considers not only per-frame confidence but also the stability of temporal state. Let T_t denote a tracker state and let ρ_t be a stability metric such as IoU overlap between predicted boxes across frames or the variance of keypoint estimates. When stability is low, the system increases the probability of deeper exits or higher resolution to re-anchor the state. This can be expressed as a gating threshold that adapts to ρ_t: [31]

$$\text{exit if} \quad c_t^{(e)} \ge \gamma(\rho_t) \quad , \tag{14}$$

where γ is decreasing in stability so that low stability demands higher confidence to exit early. Because ρ_t is itself noisy, it can be smoothed and bounded to prevent oscillation.

Placement-aware early-exit routing integrates the edge–cloud split. Suppose the edge can compute up to cut c and optionally exit at an edge-attached head $e = c$ if that head exists. If not, the edge must offload to continue [32]. The offload decision is thus intertwined with the exit decision. A unified policy can compare three options at a given depth: exit now on edge, continue on edge to a later edge exit, or offload features to cloud and continue. Let the three candidate actions have predicted objective costs J_{exit}, J_{edge}, and J_{cloud}. The policy chooses the minimum, but because the predictors are approximate, the system introduces hysteresis margins to avoid frequent switching. If network conditions degrade, J_{cloud} increases via τ^{tx} and the policy naturally shifts toward edge exits.

To make routing trainable, discrete decisions are relaxed [33]. Let α_e be soft weights over exits, and let the final prediction be a mixture $\hat{y}_t = \sum_{e \in \mathcal{E}} \alpha_e \hat{y}_t^{(e)}$ with $\alpha_e \ge 0$ and $\sum_e \alpha_e = 1$. A differentiable surrogate for latency is $\tilde{\tau} = \sum_e \alpha_e \tau^{(e)}$. Training minimizes $\ell(y_t, \hat{y}_t) + \lambda \tilde{\tau}$, encouraging mass on early exits when accuracy permits. At deployment, α is replaced by a hard exit selection derived from confidence and predicted cost. A similar relaxation can be applied to offloading and cut selection, though care is required because sending features and computing locally are not linearly composable in the same sense as mixing exits. A Gumbel-Softmax relaxation can be used during training for discrete choices, with annealing to sharpen decisions.

Backprop-friendly derivatives for quantized feature transmission are required when the exit decision depends on cloud-side computation that uses reconstructed features. If quantization is modeled as $\tilde{q} = \text{round}(q)$, gradients are zero almost everywhere. The system uses a straight-through estimator where during backward pass $\frac{\partial \tilde{q}}{\partial q} \approx 1$ within a clipping range, enabling end-to-end optimization. For entropy models, the loss includes a rate term approximated by $-\log p_\psi(\tilde{q})$, which has well-defined gradients through p_ψ. The joint training objective thus includes accuracy, latency surrogate, and rate penalty, yielding a tri-objective optimization that can be navigated via weighted sums or adaptive weighting [34]. A practical method is to normalize each term by a running estimate of its scale to avoid one term dominating gradients.

Early-exit routing introduces a structured error profile. Errors arise when the system exits too early on hard frames, when calibration is wrong under shift, or when resolution reduction makes a frame ambiguous. An explicit error decomposition helps engineering decisions. Let $e_t^\star$ denote the deepest exit, treated as a reference [35]. The excess error at time t can be decomposed into an exit-induced component and a resolution-induced component by evaluating counterfactuals on logged data:

$$\Delta \ell_t = \ell(y_t, \hat{y}_t^{(r_t, e_t)}) - \ell(y_t, \hat{y}_t^{(r_{\text{ref}}, e_t^\star)}) \tag{15}$$

$$= \Big(\ell(y_t, \hat{y}_t^{(r_t, e_t)}) - \ell(y_t, \hat{y}_t^{(r_t, e_t^\star)})\Big) + \Big(\ell(y_t, \hat{y}_t^{(r_t, e_t^\star)}) - \ell(y_t, \hat{y}_t^{(r_{\text{ref}}, e_t^\star)})\Big) \quad . \tag{16}$$

The first term isolates early-exit harm at a fixed resolution, while the second isolates resolution harm at the deepest exit. This decomposition can guide whether to tune thresholds or to adjust resolution policy. Because evaluating counterfactuals online is expensive, the system uses periodic sampling and offline replay.

Routing must also consider queueing dynamics [36]. Even if a deeper exit provides better accuracy, it may increase queue lengths and cause downstream frames to miss deadlines. Therefore the policy incorporates a deadline-aware component. Let d_t be a per-frame deadline relative to capture time, and let $\widehat{\tau}_t(a)$ be predicted completion time for action a. The policy can impose a penalty for deadline miss probability, modeled as a soft constraint:

$$J(a) = \widehat{\mathcal{L}}(a) + \lambda\, \widehat{\tau}(a) + \kappa\, \sigma\big(\widehat{\tau}(a) - d_t\big) \quad , \tag{17}$$

where σ is a smooth increasing function such as $\log(1 + \exp(\cdot))$ [37]. This encourages low-latency actions when close to deadlines without requiring hard infeasible constraints that could destabilize the control loop.

Finally, early-exit routing benefits from embedding-based reuse. If the embedding similarity indicates the scene is unchanged, the policy can bias toward early exits or even reuse outputs. This can be formalized by adding a term to the value function that discounts the expected benefit of deeper computation when Δ_t is small. The embedding itself can be derived from early layers, meaning that computing it is cheap and aligns with early-exit structure [38]. To prevent adversarial failure cases where embeddings are stable but labels change due to small but important motion, the system includes a verifier based on motion magnitude or sparse feature tracking that detects subtle changes in regions of interest.

5 Distributed Execution Mechanics and Storage Internals

A cooperative inference system is only as effective as its runtime. Decisions about resolution and early exits must translate into stable throughput on heterogeneous devices under variable network conditions. The runtime must support pipelined execution, manage memory for variable-shaped tensors, compress and transmit data efficiently, and provide accurate performance feedback to the controller. The design is best understood as a streaming dataflow with control-plane hooks [39].

On the edge, the pipeline stages include capture, decode, optional resize or crop, model prefix execution, optional early exit, feature compression, and transmit. Each stage is represented as an operator with bounded queues. Backpressure is essential: if transmit stalls due to congestion, upstream stages must slow down or drop frames according to policy. The system supports two drop modes: semantic dropping, where frames with high redundancy are skipped based on embedding similarity, and deadline dropping, where frames that would miss deadlines are discarded early to avoid wasting compute [40]. Dropping decisions must be accounted for in evaluation, as they change the effective frame rate and can bias accuracy metrics if not reported carefully.

The cloud pipeline receives data, decodes or decompresses it, runs model suffix or full model variants, and returns results. To minimize tail latency, the cloud side can use a warm pool of model replicas and a load balancer that routes requests based on observed service times. Because feature tensors vary with resolution and cut, batching must be handled carefully. A naive batching strategy that mixes shapes can cause padding overhead and memory waste [41]. The runtime therefore employs shape bucketing: resolutions are discretized to a small set, and cut points are limited to a small subset that yields a small set of feature shapes. This discretization slightly reduces policy expressiveness but greatly improves performance predictability.

Data structures for buffering and caching matter. The edge maintains a ring buffer for recent embeddings and outputs, enabling reuse and skip decisions. For fast similarity queries, the system can use locality-sensitive hashing on the projected embeddings [42]. Let $\bar{u}_t \in \mathbb{R}^k$ be the PCA-projected embedding. A random hyperplane hash uses m random vectors r_i and computes bits $b_i = \text{sign}(\langle r_i, \bar{u}_t \rangle)$. The resulting m-bit signature indexes buckets of similar frames. This provides approximate nearest-neighbor lookup in expected sublinear time for large windows, which is valuable when multiple streams are multiplexed on one edge gateway. The approximation error can be bounded in terms of angle preservation, and the system uses conservative thresholds to avoid incorrect reuse. Because hashing introduces false positives, reuse is gated by a secondary exact similarity check on a small candidate set, maintaining robustness.

Intermediate feature transmission requires careful memory layout [43]. Activations often reside in device memory in formats optimized for compute kernels, such as channels-last layouts or blocked formats. Converting to a contiguous CPU buffer for compression can be costly. The runtime therefore integrates compression kernels that operate directly on device tensors when possible, producing quantized latents without staging through host memory. When staging is unavoidable, pinned memory is used to accelerate DMA transfers [44]. The compressor outputs a byte stream and a small header encoding shape, cut version, and codec parameters, enabling the cloud decoder to reconstruct tensors correctly.

The communication protocol supports prioritization. Frames with imminent deadlines are prioritized in transmit queues, and the cloud responds with acknowledgments and optional control feedback. To reduce overhead, control feedback is piggybacked on inference responses. Network variability is handled by an adaptive bitrate mechanism for feature coding, where the compressor adjusts quantization step sizes based on measured throughput [45]. This ties back to the rate term in the objective. The runtime measures effective throughput and round-trip time and feeds it to the controller.

Scheduling across edge and cloud resources resembles query planning in databases. The analytics pipeline can be seen as a sequence of operators, some of which may execute on edge, some on cloud, with optional branching due to early exits. Choosing a plan per frame is a dynamic programming problem when the space of options is structured [46]. If operator i can run on edge or cloud, with costs that depend on prior choices due to data locality and transmission, a DP recurrence can compute the minimum expected cost:

$$C_i(\text{edge}) = \min\left(C_{i-1}(\text{edge}) + \tau_i^{\text{edge}}, \quad C_{i-1}(\text{cloud}) + \tau_{i-1\to i}^{\text{tx}} + \tau_i^{\text{edge}}\right) \tag{18}$$

$$C_i(\text{cloud}) = \min\left(C_{i-1}(\text{cloud}) + \tau_i^{\text{cloud}}, \quad C_{i-1}(\text{edge}) + \tau_{i-1\to i}^{\text{tx}} + \tau_i^{\text{cloud}}\right) \quad , \tag{19}$$

where $\tau_{i-1\to i}^{\text{tx}}$ depends on whether intermediate representations are transferable and on their size. In streaming, the DP is not solved from scratch per frame; instead, the system uses precomputed plan templates for each resolution and cut and selects among them using the controller. Nonetheless, the DP view clarifies that placement decisions are path-dependent, and it motivates plan caching and incremental update when conditions change.

Graph algorithms arise when multiple streams and multiple cloud replicas are involved. The system can model routing of requests as a bipartite assignment between incoming frames and cloud replicas with costs reflecting expected completion time and congestion [47]. A min-cost flow formulation can allocate requests to replicas to minimize total latency subject to capacity constraints. In practice, approximate greedy methods are used because exact solutions may be too slow, but the graph formulation guides the design of heuristics that avoid pathological imbalance. The runtime maintains per-replica service time estimates and uses power-of-two-choices routing with latency-aware weights.

The optimization of cut points and exit points can be computationally hard in general [48]. If one models each candidate cut as an item with a cost in bandwidth and a value in accuracy gain, selecting a set of decisions under constraints resembles a knapsack-like problem. Extending to multiple constraints and temporal coupling yields NP-hardness by reduction from partition-style problems. A proof sketch can be conveyed by mapping each item in a partition instance to a frame type where choosing cloud offload yields a particular bandwidth and latency pair, and selecting which frames to offload to satisfy a bandwidth budget while minimizing accuracy loss mirrors subset selection. This theoretical hardness motivates the use of heuristics, relaxations, and learned policies rather than exact optimization in real time.

Storage internals on the cloud side also affect latency [49]. Model weights are large, and loading them on demand is infeasible. The runtime keeps weights resident and uses memory-mapped files only for cold start. For multi-exit models, the early heads share backbone parameters, and the runtime ensures parameter sharing does not lead to cache thrashing when different exits are used. For feature decoding, small codec tables and probability models are cached in CPU L3, and decoding is parallelized across cores to overlap with GPU compute. The system also uses persistent CUDA graphs or equivalent mechanisms to reduce kernel launch overhead for repeated shapes, which is one reason shape bucketing is important [50].

A frequent systems challenge is observability. The controller requires accurate latency and bandwidth estimates, yet measurement overhead must be low. The runtime therefore uses lightweight tracing with sampled spans, measuring per-operator service time and queueing time. For quantile tracking, sketches are used, and their approximation error is tracked. Let $\hat{q}_\alpha$ be an estimated α-quantile from a sketch. The system maintains an error bound ϵ such that the true quantile lies in an interval with high probability, and it reduces the aggressiveness of latency-sensitive decisions when ϵ increases, thereby preventing feedback loops based on noisy estimates [51].

Finally, the runtime must be robust to failures and version changes. Compressor and model versions can drift between edge and cloud. The protocol includes version negotiation, and the cloud can accept multiple compressor versions during rolling updates. If decoding fails, the system can fall back to pixel offload at reduced frame rate, preserving service availability [52]. Such fallbacks must be integrated into the controller as high-cost actions with guaranteed correctness.

6 Optimization, Evaluation, and Reproducibility

Training and tuning a cooperative inference system involves two intertwined problems: optimizing model parameters for multi-exit behavior under compression and resolution variation, and optimizing the control policy for routing under resource constraints. The training objective must align with deployment behavior, but deployment decisions are discrete and depend on runtime measurements. This section develops optimization strategies that connect differentiable training with discrete control, then outlines evaluation methodology, error analysis, and reproducibility practices appropriate for systems that adapt online.

The model parameters θ include backbone weights and exit head weights [53]. The compressor parameters ψ include encoder and decoder networks and entropy models. The policy parameters ϕ include resolution and routing

logic, which may be partially neural. A joint objective can be written as

$$\min_{\theta,\psi,\phi} \quad \mathbb{E}\Big[\ell\big(y_t, \hat{y}_t^{(a_t)}\big) + \beta \, \|z_t - \tilde{z}_t\|_2^2 + \nu \, R_\psi(z_t) + \lambda \, \tilde{\tau}(a_t)\Big] \quad , \tag{20}$$

where $R_\psi(z_t)$ is a differentiable estimate of rate, often computed as $-\log p_\psi(\tilde{q}_t)$ under the entropy model, and $\tilde{\tau}$ is a differentiable latency surrogate. The expectation is over training clips and over policy actions, which may be sampled via relaxed distributions. The presence of multiple exits implies multiple losses; the system uses a weighted sum across exits during training so that early exits learn meaningful representations rather than being starved [54]. A common stability issue is that shallow exits may dominate gradients and harm deeper exits, or vice versa. A practical remedy is to weight exit losses by a schedule that approximates expected exit frequencies under the target policy, and to use gradient normalization to balance magnitudes.

Resolution augmentation is incorporated during training. Inputs are randomly resized to supported resolutions, and the model learns to be robust [55]. However, training on all resolutions uniformly may over-emphasize rare resolutions. Instead, the training distribution can mimic expected deployment frequencies. Because deployment frequencies depend on the policy, which depends on model behavior, the system can iterate: initialize with a heuristic policy, collect traces of chosen resolutions and exits, then reweight training samples accordingly. This is akin to dataset reweighting and can be stabilized using clipping to avoid extreme weights.

The routing policy can be optimized by treating it as a constrained bandit or reinforcement learning problem [56]. Direct RL online on real streams is risky because exploration can degrade service. The system therefore uses offline policy optimization on logged data. Logged traces include state s_t, action a_t taken by a baseline policy, and outcomes such as loss and latency. Offline evaluation requires counterfactual reasoning because the new policy may choose different actions. Importance sampling can correct bias but has high variance [57]. A system-friendly approach is to record rich counterfactuals by occasionally running multiple exits or multiple resolutions on a subsample of frames, effectively collecting supervised labels for cost and accuracy under alternative actions. This increases logging overhead but enables direct supervised learning of predictors $\widehat{\mathcal{L}}(s,a)$ and $\widehat{\tau}(s,a)$, turning policy learning into cost-sensitive classification:

$$\phi^\star = \arg\min_{\phi} \; \mathbb{E}\big[\widehat{\mathcal{L}}(s_t, \pi_\phi(s_t)) + \lambda \, \widehat{\tau}(s_t, \pi_\phi(s_t)) + \mu \, \widehat{E}(s_t, \pi_\phi(s_t)) + \nu \, \widehat{B}(s_t, \pi_\phi(s_t))\big] \quad . \tag{21}$$

This reduces variance and aligns with engineering constraints.

Gradient descent variants matter because the objective is multi-term and may be ill-conditioned. Adaptive optimizers such as Adam can accelerate convergence but may interact poorly with entropy-model training and quantization surrogates. The system can use decoupled weight decay and separate learning rates for backbone, exits, and compressor [58]. Dual variables for constraints can be updated with a slower timescale to avoid oscillation. If the policy is trained with relaxed discrete variables, temperature annealing must be tuned to avoid premature hardening, which can lock into suboptimal discrete choices. A practical method is to anneal temperature based on validation constraint satisfaction rather than a fixed epoch schedule.

Multi-objective optimization can be approached through weighted sums, but selecting weights is nontrivial [59]. An alternative is to approximate a Pareto front by training multiple policies with different (λ, μ, ν) and selecting one at deployment. Another approach is to learn a conditional policy that takes desired budgets as input, enabling one model to cover multiple trade-off points. Let $b = (\bar{\tau}, \bar{E}, \bar{B})$ be a budget vector. The policy becomes $\pi_\phi(a \mid s, b)$, and training samples budgets from a distribution. This yields a controllable system where operators can dial budgets based on service tier.

Evaluation must reflect both accuracy and systems performance [60]. For video analytics, accuracy metrics depend on task, but the system should report not only final accuracy but also accuracy conditioned on latency and on resource usage. Because the policy can drop frames or reuse outputs, the effective sampling rate may change; metrics should be normalized appropriately, for example reporting accuracy at a fixed processed-frame budget and separately reporting recall over time. Latency evaluation should include percentiles, not just mean, and should separate compute from communication and queueing to diagnose bottlenecks.

Error analysis is essential because approximate components can fail in subtle ways. Approximation enters through feature compression, quantile sketches for latency tracking, approximate nearest-neighbor lookup for embedding reuse, and surrogate predictors for cost and difficulty [61]. Each approximation introduces a statistical error that can be bounded or at least characterized. For example, if embedding reuse is based on LSH, the probability of collision depends on angle; false reuse occurs when embeddings collide but semantics differ. The system can estimate false reuse rate by auditing a random sample where reuse was applied and recomputing full inference, then adjusting thresholds to achieve a target false reuse probability. Similarly, quantile sketches have an error that can be measured by comparing to exact quantiles on a sample. These measured errors should be incorporated into controller safeguards, such as adding margins to thresholds [62].

Feature compression introduces distortion that depends on quantization and entropy coding. To analyze, consider a uniform scalar quantizer with step size Δ. For a latent scalar q with approximately smooth density,

quantization error has variance approximately $\Delta^2/12$ under standard assumptions. When latents are vector-valued, the distortion accumulates [63]. The effect on task loss can be approximated by a first-order Taylor expansion of the loss with respect to latents:

$$\ell(y, \hat{y}(\tilde{z})) \approx \ell(y, \hat{y}(z)) + \nabla_z \ell^\top (\tilde{z} - z) + \frac{1}{2}(\tilde{z} - z)^\top H (\tilde{z} - z) \quad , \tag{22}$$

where H is a Hessian proxy. The first-order term has zero mean if quantization noise is unbiased, while the second-order term scales with distortion. This suggests that controlling $\|\tilde{z} - z\|_2^2$ via β and choosing Δ based on observed gradient magnitudes can mitigate loss increase. In practice, gradient norms can be monitored during training to tune quantization ranges.

The complexity of routing decisions must be low [64]. The controller runs per frame and cannot afford heavy optimization. If it evaluates a small set of candidate actions, complexity is $O(|\mathcal{A}|)$ per frame, where $\mathcal{A}$ is the set of resolution–cut–exit combinations allowed. By limiting resolutions and cut points to small sets, $|\mathcal{A}|$ can be kept manageable. Approximate similarity search in an embedding cache can be made expected $O(1)$ per query with hashing, though worst-case can be larger if buckets are imbalanced. The runtime should therefore bound bucket sizes, for example by aging out old entries or limiting per-bucket entries.

Reproducibility in adaptive systems requires more than code availability. Because decisions depend on timing and network conditions, experiments must be trace-driven and environment-controlled [65]. A reproducible evaluation methodology logs input videos, timestamps, network traces including bandwidth and RTT over time, hardware configurations, model and compressor versions, and random seeds for any stochastic components. The runtime should support deterministic replay where network and queueing are simulated from logs, enabling comparison across policy variants without requiring identical live conditions. When true determinism is impossible due to nondeterministic GPU kernels, the system should at least report variability across repeated runs and isolate sources of nondeterminism.

The evaluation should include stress tests under nonstationary conditions. Network degradation events can be injected, and the controller's response should be measured in terms of recovery time and constraint violations [66]. Workload bursts can be simulated by increasing scene complexity, and the system should demonstrate stable behavior without oscillatory switching between edge and cloud. Because policies can be brittle under domain shift, evaluation should include shifts such as different lighting, motion blur, and camera viewpoints, and should report calibration drift for confidence estimates. Online recalibration can be evaluated by tracking expected calibration error over time.

A final engineering dimension is safety of adaptation [67]. Aggressive adaptation can lead to rapid switches that harm throughput due to cache misses and reinitialization overhead. The controller therefore uses smoothing, hysteresis, and rate limits on action changes. These mechanisms can be analyzed as adding regularization on action variation, effectively penalizing $\mathbb{E}[\mathbf{1}\{a_t \neq a_{t-1}\}]$ in the objective. While this term is not differentiable, it can be approximated with soft penalties during training and enforced as a hard rate limit during deployment. Such stability constraints improve predictability and simplify capacity planning.

7 Conclusion

This paper presented a cohesive design for an edge–cloud cooperative inference system for streaming video analytics that jointly adapts input resolution and performs early-exit routing under resource constraints [68]. The system was formulated as a constrained optimization problem over per-frame decisions that couple accuracy, latency, energy, and bandwidth. Adaptive resolution was developed through resource-aware cost models and rate–distortion intuitions, including feature-level compression with entropy-aware coding and temporal redundancy exploitation via embedding similarity. Early-exit routing was treated as an uncertainty-calibrated decision problem that must be placement-aware, deadline-aware, and stable under nonstationary workloads, motivating calibrated confidence gating, value-function style predictors, and conservative safeguards.

A systems-centric runtime perspective was emphasized throughout. The distributed execution design relied on pipelined operators, backpressure, shape bucketing for predictable batching, device-resident compression, and protocol mechanisms for prioritization and version negotiation [69]. The paper connected runtime control to algorithmic tools such as approximate nearest-neighbor search via hashing, low-rank projection for embedding storage, dynamic programming views of placement planning, and graph-inspired load balancing heuristics. Optimization strategies included Lagrangian relaxation for constraints, differentiable relaxations for discrete decisions, and practical training considerations for multi-exit models and quantized feature transmission. The evaluation methodology stressed error decomposition, approximate-sketch uncertainty, trace-driven reproducibility, and stress testing under network and workload variability. Taken together, the design shows how adaptive resolution and early-exit routing can be made operationally meaningful when integrated with communication-efficient co-inference and a robust runtime. The remaining engineering considerations largely concern calibration under evolving domains, maintaining

stability under aggressive adaptation, and managing versioned compatibility between edge and cloud components without sacrificing performance predictability [70].

References

[1] K. Marszałek, A. Domański, R. Cupek, and M. Drewniak, "Testing quality of service of communication system for agv fleet with software-defined network," in *2023 IEEE International Conference on Big Data (BigData)*, IEEE, Dec. 15, 2023, pp. 5078–5086. DOI: 10.1109/bigdata59044.2023.10386380

[2] N. Benmoussa, M. F. Amr, S. Ahriz, K. Mansouri, and E. Illoussamen, "Outlining a model of an intelligent decision support system based on multi agents," *Zenodo (CERN European Organization for Nuclear Research)*, Jun. 20, 2018. DOI: 10.5281/zenodo.1400511

[3] *IM - Tofino + P4: A Strong Compound for AQM on High-Speed Networks?* May 17, 2021.

[4] C. Yogatama and R. D. Wibisono, "Investigating quantum approximation techniques for tackling np-hard problems in distributed systems," *ALCOM: Journal of Algorithm and Computing*, vol. 1, no. 1, pp. 33–44, Feb. 10, 2025. DOI: 10.63846/emqcw506

[5] A. Moumen, M. Jammoukh, L. Zahiri, and K. Mansouri, "Mechanical and morphological study of a polymer reinforced by an animal bio load using the finite element method," in Springer International Publishing, Feb. 7, 2022, pp. 281–292. DOI: 10.1007/978-3-030-90633-7_24

[6] J. C. G. Vargas, R. Fabregat, A. Carrillo-Ramos, and T. Jové, "Survey: Using augmented reality to improve learning motivation in cultural heritage studies," *Applied Sciences*, vol. 10, no. 3, pp. 897–, Jan. 29, 2020. DOI: 10.3390/app10030897

[7] R. Albarrak and D. A. Menascé, "Trust but verify: A framework for the trustworthiness of distributed systems," *IEEE Transactions on Dependable and Secure Computing*, no. 01, pp. 1–1, Dec. 30, 2020.

[8] D. Temnikov and R. Dubinin, "Application of ai for enhancing the performance of distributed systems," *The American Journal of Engineering and Technology*, vol. 7, no. 7, pp. 180–185, Jul. 31, 2025. DOI: 10.37547/tajet/volume07issue07-17

[9] S. Dahdal, A. Gilli, F. Poltronieri, M. Tortonesi, C. Stefanelli, and N. Suri, "Roamml platform: Enabling distributed continual learning for disaster relief operations," in *2024 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, Jun. 26, 2024, pp. 1–6. DOI: 10.1109/iscc61673.2024.10733586

[10] R. Malik et al., "Mlr-index: An index structure for fast and scalable similarity search in high dimensions," in *International Conference on Scientific and Statistical Database Management*, Springer, 2009, pp. 167–184.

[11] A. Slo, S. Bhowmik, and K. Rothermel, "State-aware load shedding from input event streams in complex event processing," *IEEE Transactions on Big Data*, vol. 8, no. 5, pp. 1340–1357, Oct. 1, 2022. DOI: 10.1109/tbdata.2020.3047438

[12] H. Ni, R. van Renesse, and G. Morrisett, *Functional reasoning for distributed systems with failures*, Oct. 14, 2025. DOI: 10.48550/arxiv.2510.12131

[13] K. Erciyes, "Distributed graph algorithms," in Springer International Publishing, Apr. 14, 2018, pp. 117–136. DOI: 10.1007/978-3-319-73235-0_5

[14] L. Kroll, *Compile-time safety and runtime performance in programming frameworks for distributed systems*, Mar. 14, 2020.

[15] R. M. Albarrak and D. A. Menasce, "Trust but verify: A framework for the trustworthiness of distributed systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 2105–2121, May 1, 2022. DOI: 10.1109/tdsc.2020.3048301

[16] P. Schmidt, A. Jaust, H. Steeb, and M. Schulte, "Simulation of flow in deformable fractures using a quasi-newton based partitioned coupling approach," *Zenodo (CERN European Organization for Nuclear Research)*, Apr. 8, 2021. DOI: 10.5281/zenodo.4671542

[17] "Distributed system based on correlation sequencing algorithm," *Distributed Processing System*, vol. 2, no. 3, Sep. 16, 2021. DOI: 10.38007/dps.2021.020303

[18] P. Han, H. Li, G. Xue, and C. Zhang, "Distributed system anomaly detection using deep learning-based log analysis," *Computational Intelligence*, vol. 39, no. 3, pp. 433–455, Apr. 22, 2023. DOI: 10.1111/coin.12573

[19] C. Zhu et al., "Blockchain-enhanced federated learning for secure and intelligent consumer electronics : An overview," *IEEE Consumer Electronics Magazine*, pp. 1–12, Jan. 1, 2025.

[20] F. D. Cosmo, *Verification of sometimes termination of lazy-bounded declarative distributed systems*, Jan. 1, 2023. DOI: 10.48550/arxiv.2308.10007

[21] F. Poltronieri, M. Tortonesi, and C. Stefanelli, "A chaos engineering approach for improving the resiliency of it services configurations," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, IEEE, Apr. 25, 2022, pp. 1–6. DOI: 10.1109/noms54207.2022.9789887

[22] M. Maćkowski, M. Kawulok, P. Brzoza, and D. Spinczyk, "Method and tools to supporting math learning in inclusive education of blind students," in Germany: Springer Nature Switzerland, May 22, 2023, pp. 42–53. DOI: 10.1007/978-3-031-32883-1_4

[23] S. Gouraguine, I. Salhi, M. Riad, M. Qbadou, and K. Mansouri, "Towards a humanoid teaching assistant-robot-primitives knowledge modeling," in Springer International Publishing, Nov. 18, 2022, pp. 802–811. DOI: 10.1007/978-3-031-20601-6_66

[24] R. Chandrasekar, R. Suresh, and S. Ponnambalam, "Evaluating an obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 International Conference on Advanced Computing and Communications*, IEEE, 2006, pp. 628–629.

[25] P. Gaj, I. Smołka, and J. Stój, "Reliability analysis of m2m cyclic data transfer based on ad-hoc wireless p2p link," in *2023 IEEE International Conference on Big Data (BigData)*, IEEE, Dec. 15, 2023, pp. 5048–5056. DOI: 10.1109/bigdata59044.2023.10386287

[26] X. Fu, B. Lin, and H. Cai, "Distfax," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ACM, May 21, 2022, pp. 51–55. DOI: 10.1145/3510454.3516859

[27] V. V. Kulba, S. Somov, and Y. Merkuryev, "Placement of data array replicas in a distributed system with unreliable communication channels," *Applied Computer Systems*, vol. 24, no. 1, pp. 69–74, May 1, 2019. DOI: 10.2478/acss-2019-0009

[28] H. Zhang, N. Lu, R. Song, and J. Li, "Modelling and optimizing for distributed systems with limited resource migration capacity," *Journal of Physics: Conference Series*, vol. 1670, no. 1, pp. 012 026–, Nov. 1, 2020. DOI: 10.1088/1742-6596/1670/1/012026

[29] R. Sannholm and H. Risku, "Situated minds and distributed systems in translation," *Target. International Journal of Translation Studies*, vol. 36, no. 2, pp. 159–183, Apr. 23, 2024. DOI: 10.1075/target.22172.san

[30] M. Farhadi, D. Miorandi, and G. Pierre, *Blockchain enabled fog structure to provide data security in iot applications*, Dec. 10, 2018.

[31] P. Domanski, D. Pflüger, and R. Latty, "Learn to tune: Robust performance tuning in post-silicon validation," in *2023 IEEE European Test Symposium (ETS)*, IEEE, May 22, 2023, pp. 1–4. DOI: 10.1109/ets56758.2023.10174123

[32] L. Alberto, "Pseudospheres: Combinatorics, topology and distributed systems," *Journal of Applied and Computational Topology*, vol. 8, no. 4, pp. 1023–1052, Feb. 10, 2024. DOI: 10.1007/s41468-023-00162-5

[33] E. Becks, M. Josten, V. Matkovic, and T. Weis, "Revising poor man's eye tracker for crowd-sourced studies," in *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, IEEE, Mar. 13, 2023, pp. 328–330. DOI: 10.1109/percomworkshops56833.2023.10150386

[34] I. S. Ochôa et al., "Prichain: A partially decentralized implementation of ubipri middleware using blockchain.," *Sensors (Basel, Switzerland)*, vol. 19, no. 20, pp. 4483–, Oct. 16, 2019. DOI: 10.3390/s19204483

[35] L. Ciuffreda, *Distributed systems for neural network models*, Oct. 26, 2018.

[36] O. Okusanya, *Consensus in distributed systems: Raft vs crdts*, May 28, 2019.

[37] L. Nenov, "Reinforcement learning for key management in distributed systems," in *2024 32nd National Conference with International Participation (TELECOM)*, IEEE, Nov. 21, 2024, pp. 1–5. DOI: 10.1109/telecom63374.2024.10812245

[38] R. Chandrasekar and T. Srinivasan, "An improved probabilistic ant based clustering for distributed databases," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, 2007, pp. 2701–2706.

[39] J. Pennekamp, "Evolving the industrial internet of things: The advent of secure collaborations," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, IEEE, May 6, 2024, pp. 1–6. DOI: 10.1109/noms59830.2024.10575325

[40] R. Mayerhofer, A. Morichetta, A. Furutanpey, and S. Dustdar, "Hpaqt: Adaptive and interpretable high-level slo-aware autoscaling with reinforcement learning," in *Proceedings of the 18th IEEE/ACM International Conference on Utility and Cloud Computing*, ACM, Dec. 31, 2025, pp. 1–9. DOI: 10.1145/3773274.3774274

[41] *On Strict Homogeneous Lyapunov Function for Generalized Homogeneous PI Controller*, Sep. 20, 2021.

[42] J. Hecking-Harbusch and N. O. Metzger, "Atva - efficient trace encodings of bounded synthesis for asynchronous distributed systems," in Germany: Springer International Publishing, Oct. 21, 2019, pp. 369–386. DOI: 10.1007/978-3-030-31784-3_22

[43] S.-M. Choi, J. Park, Q. H. Nguyen, and A. Cronje, *Fantom: A scalable framework for asynchronous distributed systems*, Oct. 22, 2018.

[44] O. Sukhoroslov and A. Makogon, "Fast and flexible framework for simulation of distributed systems," in Germany: Springer Nature Switzerland, Jan. 31, 2025, pp. 90–106. DOI: 10.1007/978-3-031-78462-0_7

[45] *Reasoning about knowledge in byzantine distributed systems*, Jul. 1, 2020. DOI: 10.55776/p33600

[46] K. K. Rout, D. P. Mishra, and S. R. Salkuti, "Deadlock detection in distributed system," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 24, no. 3, pp. 1596–1603, Dec. 1, 2021.

[47] D. Heye, S. Islam, J. Pennekamp, and K. Wehrle, "Poster: Transport security orchestration using dns," in *2025 IEEE 33rd International Conference on Network Protocols (ICNP)*, IEEE, Sep. 22, 2025, pp. 1–3. DOI: 10.1109/icnp65844.2025.11192410

[48] M. H. Hilman, M. A. Rodriguez, and R. Buyya, *Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions*, Sep. 14, 2018.

[49] A. A. T. Thulnoon, *Efficient runtime security system for decentralised distributed systems*, Aug. 11, 2018.

[50] J. Stein et al., *Introducing reduced-width qnns, an ai-inspired ansatz design pattern*, Jan. 1, 2023. DOI: 10.48550/arxiv.2306.05047

[51] X. Wei et al., "Phoenixos: Concurrent os-level gpu checkpoint and restore with validated speculation," in *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, ACM, Oct. 12, 2025, pp. 996–1013. DOI: 10.1145/3731569.3764813

[52] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, "An obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 IEEE Conference on Cybernetics and Intelligent Systems*, IEEE, 2006, pp. 1–6.

[53] A. Charapko, A. Ailijiang, M. Demirbas, and S. S. Kulkarni, "Retroscope: Retrospective monitoring of distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 11, pp. 2582–2594, Nov. 1, 2019. DOI: 10.1109/tpds.2019.2911944

[54] "Three-layer distributed system based on bayesian classifier," *Distributed Processing System*, vol. 3, no. 4, Oct. 15, 2022. DOI: 10.38007/dps.2022.030409

[55] J. Zerwick, *Improving a distributed system post-incident*, Jan. 21, 2020.

[56] M. García-Valls and L. L. Ferreira, "Introduction to the special issue," *ACM SIGBED Review*, vol. 14, no. 4, pp. 7–7, Jan. 4, 2018. DOI: 10.1145/3177803.3177804

[57] S. Sadiq and S. R. M. Zeebaree, "Distributed systems for machine learning in cloud computing: A review of scalable and efficient training and inference," *Indonesian Journal of Computer Science*, vol. 13, no. 2, Apr. 1, 2024. DOI: 10.33022/ijcs.v13i2.3814

[58] D. Le-Phuoc and M. Hauswirth, "Semantic stream processing and reasoning," in Springer International Publishing, Mar. 16, 2022, pp. 1–10. DOI: 10.1007/978-3-319-63962-8_287-2

[59] I. Kunze, C. Sander, L. Tissen, B. Bode, and K. Wehrle, "Spintrap: Catching speeding quic flows," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, IEEE, May 6, 2024, pp. 1–10. DOI: 10.1109/noms59830.2024.10575719

[60] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and A. Manikandan, "Localized tree change multicast protocol for mobile ad hoc networks," in *2006 International Conference on Wireless and Mobile Communications (ICWMC'06)*, IEEE, 2006, pp. 44–44.

[61] P. Foytik and S. Shetty, *Blockchain for Distributed Systems Security - Blockchain Evaluation Platform*. Wiley, Mar. 15, 2019. DOI: 10.1002/9781119519621.ch13

[62] S. Agarwal, M. A. Rodriguez, and R. Buyya, "Input-based ensemble-learning method for dynamic memory configuration of serverless computing functions," in *2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC)*, IEEE, Dec. 16, 2024, pp. 346–355. DOI: 10.1109/ucc63386.2024.00055

[63] A. Hermann, N. Trkulja, P. Wachter, B. Erb, and F. Kargl, "Quantification methods for trust in cooperative driving," in *2025 IEEE Vehicular Networking Conference (VNC)*, IEEE, Jun. 2, 2025, pp. 1–8. DOI: 10.1109/vnc64509.2025.11054209

[64] B. Bollig and P. Gastin, *Non-sequential theory of distributed systems*, Jan. 1, 2019. DOI: 10.48550/arxiv.1904.06942

[65] S. Silalahi, R. M. Ijtihadie, T. Ahmad, and H. Studiawan, "A survey on logging in distributed system," in *2022 1st International Conference on Information System & Information Technology (ICISIT)*, IEEE, Jul. 27, 2022, pp. 7–12. DOI: 10.1109/icisit54091.2022.9873095

[66] D. I. Sukhoplyuev and A. N. Nazarov, "Monitoring fault tolerance in distributed systems," *Computational nanotechnology*, vol. 11, no. 4, pp. 94–106, Sep. 24, 2024. DOI: 10.33693/2313-223x-2024-11-4-94-106

[67] R. Malik, C. Ramachandran, I. Gupta, and K. Nahrstedt, "Samera: A scalable and memory-efficient feature extraction algorithm for short 3d video segments.," in *IMMERSCOM*, 2009, p. 18.

[68] A. Almen and D. Dentcheva, "Fair risk optimization of distributed systems," *Annals of Operations Research*, Nov. 5, 2025. DOI: 10.1007/s10479-025-06917-w

[69] S. Banerjee, P. Mukherjee, S. Kanrar, and N. Chaki, "A novel symmetric algorithm for process synchronization in distributed systems," in Germany: Springer Singapore, Apr. 8, 2018, pp. 51–66. DOI: 10.1007/978-981-10-8102-6_4

[70] "Enterprise distributed system based on raft algorithm," *Distributed Processing System*, vol. 1, no. 2, Jun. 10, 2020. DOI: 10.38007/dps.2020.010207