StreamSplit: Continuous Audio Representation Learning via Uncertainty-Guided Adaptive Splitting

Minh K. Quan
School of Engineering, Deakin University
Waurn Ponds, Australia
m.quan@deakin.edu.au

Pubudu N. Pathirana
School of Engineering, Deakin University
Waurn Ponds, Australia
pubudu.pathirana@deakin.edu.au

Abstract

Large-batch Contrastive Learning (CL), the foundation of modern representation learning, is fundamentally incompatible with the volatile resource constraints of edge devices. This conflict creates a dilemma: small on-device batches degrade model fidelity, while offloading to the cloud incurs unacceptable latency and bandwidth costs. Existing solutions often resort to static model compression, which fails to adapt to the runtime volatility of edge environments. To bridge this gap, we present StreamSplit, a novel framework that makes streaming CL practical across heterogeneous ARM client platforms. StreamSplit resolves the conflict between the **continuous nature of ambient audio** and the discrete batch requirements of models like CLAP and COLA. We introduce: (1) A distribution-based streaming framework that decouples representation quality from local batch size, using a tractable Hybrid Loss to maintain fidelity despite sparse updates; and (2) An Uncertainty-Guided Adaptive Splitter that uses a lightweight Reinforcement Learning (RL) policy to dynamically partition computation. Uniquely, this policy integrates real-time resource monitoring with embedding ambiguity to optimize the accuracy-latency trade-off on the fly. We evaluate StreamSplit on diverse hardware, from the resource-constrained Raspberry Pi 4 to the high-performance Apple M2. Results demonstrate that StreamSplit reduces per-sample latency by up to 4.7× and cuts bandwidth by 77.1% and energy by 52.3% compared to server-centric baselines. Crucially, it maintains accuracy within 2.2% of **server-centric** models, proving that adaptive, distributed learning is a viable path for the modern edge ecosystem.

CCS Concepts

• **Computing methodologies** → **Distributed algorithms**; • **Computer systems organization** → **Sensor networks**.

Keywords

Edge Computing, Contrastive Learning, Reinforcement Learning, Split Computing, Resource Management

ACM Reference Format:
Minh K. Quan and Pubudu N. Pathirana. 2026. StreamSplit: Continuous Audio Representation Learning via Uncertainty-Guided Adaptive Splitting. In *The 24th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '26), June 21–25, 2026, Cambridge, United Kingdom.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3745756.3809189

MobiSys '26, Cambridge, United Kingdom

ACM ISBN 979-8-4007-2027-7/26/06
https://doi.org/10.1145/3745756.3809189

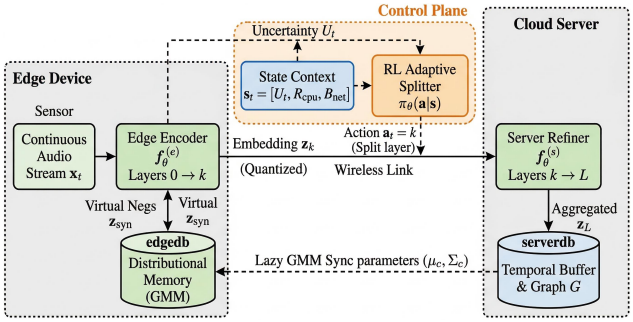

Figure 1: StreamSplit System Architecture. The framework comprises three coupled subsystems: an Edge Learner optimizing local throughput via virtual negatives, a Control Plane dynamically partitioning the network based on state constraints, and a Server Refiner maintaining manifold continuity via Laplacian regularization on graph $\mathcal{G}$.

1 Introduction

Audio-capable edge devices, from smart speakers and wearables to mobile robots, offer immense potential for ambient intelligence [1, 2]. To realize this vision, these devices must learn from streaming data and adapt to complex acoustic environments, such as performing robust scene classification or source separation in smart cities [3, 4]. However, the state-of-the-art models for audio representation learning (e.g., CLAP [5], COLA [6], CLAR [7]) are largely confined to data centers. This centralization creates a critical bottleneck: transmitting continuous raw audio to the cloud is unsustainable regarding bandwidth [8, 9] and energy [10].

Furthermore, transmitting raw user audio raises **data minimization concerns**; processing data locally or transmitting only intermediate representations aligns better with data minimization principles, though it does not replace cryptographic guarantees.

Edge computing offers a solution by moving computation closer to the source [11, 1]. Yet, simply porting server-grade training algorithms to the edge introduces fundamental conflicts. Contrastive Learning (CL)—the dominant paradigm for self-supervised audio learning—relies on **large batches** and **diverse negative samples** to learn high-quality features [12, 13, 14]. This requirement is inherently at odds with the limited memory and compute of client devices [15, 16]. Furthermore, the edge ecosystem is **heterogeneous** and **volatile**; a static deployment that works on a high-performance Apple M2 chip may drain the battery of a Raspberry Pi, and a partition optimized for strong WiFi may paralyze the application when the network degrades [17, 18].

To enable practical, streaming learning at the edge, we must address two distinct, coupled challenges:

Challenge 1: The Stream-Clip Mismatch (Algorithmic). State-of-the-art audio models (e.g., CLAP) differ fundamentally from edge sensors: they are trained on shuffled, discrete *clips* (files), whereas edge devices perceive continuous *streams*. Naively chopping streams into small batches destroys the **temporal coherence** required to learn robust features. We need a formulation that exploits acoustic continuity rather than fighting it.

Challenge 2: The Volatility Conflict (Systems). The optimal point to split a neural network between a device and a server is not static. It shifts constantly based on the device's instantaneous CPU load (e.g., background tasks) and external network conditions. Static partitioning schemes or "train-then-compress" approaches fail to adapt [19, 20]. In our preliminary analysis, we observed that a static split (Fixed Split Learning) suffered a **15.7% accuracy drop** when subjected to realistic CPU load volatility, due to latency timeouts and dropped frames.

Existing paradigms isolate representation learning from system scheduling. Standard split computing relies blindly on system telemetry [21], while small-batch continual learning demands large memory queues that cause edge cache thrashing [22]. StreamSplit avoids this via Distributional Memory, synthesizing virtual negatives from a compact Gaussian Mixture Model (GMM) to prevent dimensional collapse without large physical batches. Furthermore, naively pairing heuristic schedulers with small-batch methods fails for continuous audio: dropping a semantically "hard" frame shatters the embedding manifold. StreamSplit overcomes this via *algorithm-system co-design* [23]. By leveraging the GMM's entropy as a lightweight, zero-cost uncertainty signal, our RL routing policy directly couples algorithmic difficulty to system execution, intelligently spending bandwidth only where it maximizes server utility.

In this paper, we introduce **StreamSplit** (Figure 1), a cohesive framework to resolve conflicts. Unlike static compression techniques [23], StreamSplit co-designs the learning algorithm with the system architecture for robust, adaptive execution.

To address **C1**, we propose a **distribution-based streaming framework**. Instead of relying on large discrete batches, StreamSplit aligns the *distribution* of edge embeddings with a global prior using a hybrid Sliced-Wasserstein and Laplacian loss [24, 25]. This allows the edge device to contribute high-quality updates using small, local batches, effectively decoupling representation quality from on-device memory constraints.

To address **C2**, we introduce an **Uncertainty-Guided Adaptive Splitter**. We formulate the splitting decision as a lightweight RL problem. Our agent monitors system metrics (CPU, RAM, Network) and—crucially—the *uncertainty* of the current audio embedding. This allows StreamSplit to offload "hard" samples to the server while processing "easy" samples locally, or to aggressively compress data when the network is congested.

We implement StreamSplit on heterogeneous ARM platforms from the resource-constrained **Raspberry Pi 4B** to the high-performance **Apple MacBook M2**. Our contributions include: (1) the first framework integrating distribution-based contrastive learning with RL-based system control for both convergence and stability; (2) platform-agnostic deployment extending battery life by 50% on Pi 4 while leveraging NPU acceleration on M2; (3) 77.1% bandwidth and 52.3% energy reduction while maintaining accuracy within 2.2% of server models.

2 Background and Motivation

To understand the necessity of StreamSplit, we must examine the fundamental conflicts that render current state-of-the-art approaches insufficient for the edge: the algorithmic dependency of contrastive learning on massive data batches, and the systemic intolerance of split computing to resource volatility. Table 1 provides a systematic critique of these paradigms.

2.1 Contrastive Learning: The Batch Size Barrier

Contrastive Learning (CL) has established itself as the dominant paradigm for self-supervised audio representation, driving recent breakthroughs in ambient intelligence [1, 2]. State-of-the-art models like COLA [6] and CLAP [5] rely on maximizing agreement between augmented views of data to learn robust features without human labels.

Dependency on Negative Diversity. The efficacy of CL is mathematically rooted in the quantity and diversity of negative samples available during a gradient update. The standard InfoNCE loss function [12] relies on the denominator to approximate the true data distribution. To maintain this approximation, leading frameworks like SimCLR [12] and MoCo [13] require massive batch sizes (often $N > 2048$) or large memory queues. Without sufficient diversity, the gradients become biased, and the learning objective degenerates [26].

The Edge Incompatibility (C1). This requirement creates fundamental friction with edge hardware. Embedded devices, constrained by strict memory budgets [15, 16], often restrict batch sizes to single digits ($N = 8$ or 16). Existing adaptations fall short: gradient accumulation introduces prohibitive latency for streaming [27], while small-batch training causes dimensional collapse where embeddings map to narrow cones [14]. StreamSplit addresses this by shifting from sample-based to *distribution-based* alignment (Section 4), decoupling representation quality from local batch size.

2.2 Split Computing: The Volatility Gap

To circumvent on-device resource limitations, Split Computing partitions the neural network between the edge and cloud [11]. While theoretically sound, existing implementations fail to address the dynamic nature of mobile environments.

Static Partitioning Deficiencies. The majority of existing frameworks, such as split federated learning approaches [19, 29], determine the optimal split point *offline* or at initialization. They assume a stable resource profile. However, edge environments are defined by **volatility**. A device's available CPU cycles fluctuate rapidly due to background OS tasks or thermal throttling [10]. A static split point that is optimal at $t = 0$ often becomes a bottleneck at $t = 1$. For instance, if the network degrades, a server-heavy split causes stalls. If the CPU load spikes, an edge-heavy split causes system-wide latency.

The Gap in Adaptive Control (C2). Recent adaptive approaches (e.g., Rule-Based Split in Table 1) rely on reactive bandwidth thresholds [23], suffering from two flaws: (1) *Lack of Lookahead*—reacting only after degradation, causing dropped frames; (2) *Optimization*

Table 1: Comprehensive Gap Analysis. Current paradigms fail to simultaneously satisfy the conflicting requirements of the edge. While heuristic methods attempt adaptation, they lack data awareness (Uncertainty), leading to inefficient offloading decisions.

Paradigm	Representative Works	Methodology		Critical Failure Mode	Key Properties	
		Algorithm	Execution		Quality	Adaptive
Cloud-Centric	CLAP [5], COLA [6]	Large-Batch	Full Offload	Privacy & Bandwidth Costs	✓	–
Edge-Only	EdgeNCE [28], MCUNet	Small-Batch	On-Device	Dimensional Collapse	✗	✗
Static Split	SplitFed [19], FedSL [20]	Standard	Fixed Layer	Latency Timeouts (-15.7%)	✓	✗
Rule-Based Split	RoofSplit [23], Neurosurgeon	Standard	Resource-Aware	**Optimization Blindness**	✓	⚠
StreamSplit (Ours)	–	**Distributional**	**Uncertainty RL**	**Robust (Co-Designed)**	✓	✓

✓ = Fully Supported; ✗ = Fails/Not Supported; ⚠ = Unreliable or Data-Agnostic (Partial Support); – = Not Applicable.

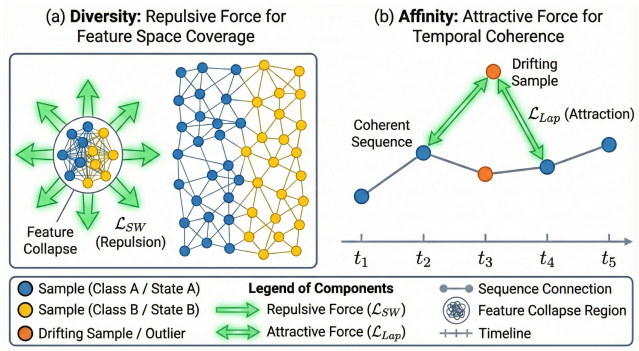

Figure 2: Embedding Quality Metrics. (a) Diversity ($\mathcal{L}_{SW}$): Repulsive force preventing dimensional collapse by dispersing embeddings. (b) Affinity ($\mathcal{L}_{Lap}$): Attractive force ensuring temporal coherence by pulling drifting frames (red) back to the manifold.

Blindness—treating all inputs equally despite varying difficulty. Table 1 demonstrates no existing paradigm satisfies continuous edge learning requirements, necessitating StreamSplit's co-designed *Uncertainty-Guided* framework.

3 Our Quality Metrics: Affinity and Diversity

To systematically address the trade-off between representation quality and edge constraints, we propose tractable metrics that quantify the two essential attributes of a robust embedding space: **Diversity** and **Affinity**. We formalize these attributes using the geometry of the induced embedding distribution $p_\theta(z)$. Our quantification process is illustrated in Figure 2. In the following subsections, we provide a detailed explanation of the theoretical framework, along with empirical evidence demonstrating the advantages of our distribution-based metrics.

3.1 Diversity: Ensuring Global Separation

The "Small-Batch Conflict" (C1) primarily threatens the global structure of the latent space. Standard contrastive losses require large batches to approximate the partition function; without them, the model minimizes loss by mapping all inputs to a single point c, a phenomenon known as *dimensional collapse* [26].

Definition 1 (Diversity). We define Diversity as the alignment between the marginal distribution of embeddings $p_\theta(z)$ and a fixed, high-entropy prior distribution $q(z)$ (typically uniform on the hypersphere $\mathcal{U}(\mathbb{S}^{d-1})$).

$$\mathcal{D}_{div} = \mathcal{W}_2(p_\theta(z), \mathcal{U}(\mathbb{S}^{d-1})) \tag{1}$$

where $\mathcal{W}_2$ denotes the Wasserstein distance. High diversity (minimizing $\mathcal{D}_{div}$) implies that the entropy $H(p_\theta(z))$ is maximized.

Definition Explanation. High diversity (minimizing $\mathcal{D}_{div}$) ensures embeddings distribute uniformly across the hypersphere, maximizing discriminative capacity even with small local batches (Figure 2a).

Theorem 3.1 (Small-Batch Robustness). *Let p_θ denote the edge embedding distribution and $\mathcal{U}$ the uniform distribution on $\mathbb{S}^{d-1}$. If p_θ has ϵ-diversity (i.e., $\mathcal{W}_1(p_\theta, \mathcal{U}) < \epsilon$), then for batch size N, the generalization gap between the empirical contrastive loss $\mathcal{L}_N$ and the population loss $\mathcal{L}_\infty$ is bounded by:*

$$|\mathcal{L}_N - \mathcal{L}_\infty| \leq C_1\epsilon + \frac{C_2}{\sqrt{N}} \tag{2}$$

where $C_1, C_2 > 0$ are constants depending on the critic function's Lipschitz constant.

This theorem implies that minimizing the distributional gap ϵ directly compensates for the bias induced by small batch sizes N. (See Appendix A for proof).

Metric: Sliced-Wasserstein Distance (SWD). A potential concern is that computing the full Wasserstein distance is computationally expensive ($O(d^3)$). To ensure tractability on edge devices, we employ the Sliced-Wasserstein Distance (SWD) as a scalable proxy ($O(Md \log d)$). By projecting the high-dimensional distributions onto a set of M random unit vectors $\Omega = \{\omega_m\}_{m=1}^M$, we compute the closed-form solution:

$$\mathcal{L}_{SW} = \frac{1}{M}\sum_{m=1}^{M}\int_0^1 \left|F^{-1}_{\theta,\omega_m}(\tau) - F^{-1}_{q,\omega_m}(\tau)\right|^2 d\tau \tag{3}$$

where F^{-1} is the inverse CDF. Minimizing $\mathcal{L}_{SW}$ forces the sorted projections of edge embeddings to match the uniform distribution, ensuring global coverage.

Remark (Metric Equivalence). While Theorem 3.1 relies on the Wasserstein-1 distance $\mathcal{W}_1$, our optimization minimizes the Sliced-Wasserstein distance $\mathcal{L}_{SW}$. On the compact hypersphere $\mathbb{S}^{d-1}$, it is established that $\mathcal{L}_{SW}$ is topologically equivalent to $\mathcal{W}_p$ distances

[24]. Specifically, convergence in $\mathcal{L}_{SW}$ implies convergence in $\mathcal{W}_1$, making Eq. 3 a tractable proxy for minimizing the theoretical error bound ϵ.

3.2 Affinity: Preserving Local Structure

While Diversity ensures global space utilization, Affinity ensures the mapping preserves the temporal topology of the input signal. In the "Volatility Conflict" (C2), sparse updates can lead to a "jagged" manifold where temporally adjacent frames x_t, x_{t+1} map to distant points z_t, z_{t+1}.

Motivation. For a Lipschitz-continuous encoder, small temporal shifts in the input should produce small shifts in the embedding space. Formally, if $||x_t - x_{t+1}|| \leq \delta$, then a well-behaved encoder satisfies $||f_\theta(x_t) - f_\theta(x_{t+1})|| \leq K\delta$ for some Lipschitz constant K. This motivates measuring smoothness directly in embedding space.

Definition 2 (Affinity). Let $\mathcal{G} = (V, E, W)$ be a temporal adjacency graph where vertices $V = \{z_1, \ldots, z_T\}$ are embeddings and edges E connect temporally adjacent frames with weights W_{ij}. We define Affinity as the inverse of the *Dirichlet energy* (graph Laplacian quadratic form):

$$\mathcal{D}_{aff} = \frac{1}{|E|} \sum_{(i,j)\in E} W_{ij}||z_i - z_j||^2 = \frac{1}{|E|}\mathrm{Tr}(\mathbf{Z}^\top \mathbf{L}\mathbf{Z}) \tag{4}$$

where $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the graph Laplacian and $\mathbf{D}$ is the degree matrix. High affinity corresponds to *low* $\mathcal{D}_{aff}$, indicating smooth embeddings over the temporal graph.

Definition Explanation (Acoustic Consistency). Unlike video (which has scene cuts) or images (which are independent), ambient audio is physically continuous. Sound sources do not teleport in feature space. Minimizing $\mathcal{D}_{aff}$ enforces this **physical inertia**, ensuring the manifold remains smooth even when network dropouts cause the server to receive sparse updates.

Theorem 3.2 (Temporal Interpolation). *Let $\mathcal{G}$ be a connected temporal graph with spectral gap $\lambda_2 > 0$ (second smallest eigenvalue of $\mathbf{L}$). If the embedding sequence has Dirichlet energy $\mathcal{D}_{aff} \leq \alpha$, then the mean squared error of reconstructing any missing frame z_{t^*} via weighted neighbor averaging is bounded:*

$$\mathbb{E}\left[||z_{t^*} - \hat{z}_{t^*}||^2\right] \leq \frac{2\alpha \cdot |E|}{\lambda_2 \cdot |\mathcal{N}(t^*)|} \tag{5}$$

where $\hat{z}_{t^*} = \frac{1}{|\mathcal{N}(t^*)|}\sum_{j\in\mathcal{N}(t^*)} z_j$ is the neighbor average and $\mathcal{N}(t^*)$ denotes the temporal neighbors of t^*.

This implies that a high-affinity representation (low α) provides intrinsic robustness to the volatility of edge execution. (See Appendix A for proof).

Metric: Laplacian Regularization. We directly optimize the Dirichlet energy as the affinity loss:

$$\mathcal{L}_{Lap} = \frac{1}{|E|} \sum_{(i,j)\in E} W_{ij}||z_i - z_j||^2 \tag{6}$$

Minimizing $\mathcal{L}_{Lap}$ explicitly penalizes "jagged" transitions, smoothing the manifold to maintain semantic coherence.

3.3 Metric Advantage

Given the fine granularity of our metrics, we empirically investigate their correlation with task performance. We conduct controlled experiments on AudioSet-Balanced (20,371 samples, 527 classes) by systematically varying affinity and diversity.

Diversity Validation. We degrade diversity by restricting the uniform prior to spherical cones of half-angle θ ranging from 10° to 90° in 10° increments. At $\theta = 10$, embeddings are forced into a narrow cone (severe collapse); at $\theta = 90$, the prior covers the full hypersphere. The results reveal a strong negative correlation between $\mathcal{L}_{SW}$ and downstream accuracy (Pearson $r = -0.96$, $p < 0.001$), significantly outperforming Maximum Mean Discrepancy (MMD, $r = 0.82$, $p < 0.01$) [30]. At $\theta = 10$ ($\mathcal{L}_{SW} = 0.89$), accuracy drops to 55.2%; at $\theta = 90$ ($\mathcal{L}_{SW} = 0.08$), accuracy reaches 73.1%.

Affinity Validation. We inject temporal discontinuities by randomly shuffling frames within a 3-second window with probability p ranging from 0.0 to 0.8 in 0.1 increments. The $\mathcal{L}_{Lap}$ metric shows strong positive correlation with accuracy degradation (Pearson $r = 0.93$, $p < 0.001$). At $p = 0.8$ (severe jitter, $\mathcal{L}_{Lap} = 2.31$), accuracy degrades by 12.4%; at $p = 0$ (no jitter, $\mathcal{L}_{Lap} = 0.35$), accuracy is 72.8%. The spectral gap λ_2 decreases from 0.42 to 0.08 as jitter increases, confirming that temporal disruptions degrade manifold connectivity as predicted by Theorem 3.2.

These results demonstrate that our distribution-based metrics are more sensitive to quality variations than existing approaches, making them reliable objectives for the StreamSplit optimization loop (Section 4).

4 StreamSplit Framework

StreamSplit transforms the traditionally static edge-cloud link into a dynamic, feedback-driven control loop. Unlike "train-then-compress" paradigms which fix model architecture at deployment time, StreamSplit treats the neural network as a flexible pipeline that can be partitioned, compressed, and scheduled in real-time.

As illustrated in Figure 1, the framework is architected as an asymmetric distributed system spanning edge devices and the cloud. The figure explicitly maps out the data and control flow, detailing the key components and interaction pathways within three decoupled subsystems: the **Edge Learner** (Phase 1), which maximizes local throughput on the edge devices under memory constraints; the **Control Plane** (Phase 2), which governs adaptive offloading based on system state and data difficulty; and the **Cloud Refiner** (Phase 3), which enforces global consistency across the manifold despite asynchronous updates.

4.1 Phase 1: Edge-Side Streaming Execution

The primary operational constraint on the edge is memory volatility. Standard Contrastive Learning (CL) requires storing thousands of negative samples (e.g., in a Memory Bank or Large Batch) to approximate the global data distribution. On memory-constrained devices, maintaining large queues causes cache thrashing when background tasks compete for resources [22]. This raises a critical design question:

RQ1: How can we maintain high-quality contrastive representations on memory-constrained edge devices without requiring large negative sample batches?

4.1.1 Analysis Method. We analyze the memory-quality trade-off throughout the contrastive learning process. Standard approaches store N negative samples requiring $O(Nd)$ memory. For typical settings ($N = 4096$, $d = 128$), this consumes 512KB of RAM, causing cache thrashing on edge devices. We investigate whether a generative distribution can replace explicit sample storage while maintaining representation quality.

4.1.2 Distributional Memory Footprint. To resolve the "Small-Batch Conflict" (C1), we replace memory-intensive sample queues with a generative distribution. Instead of storing discrete raw tensors, the Edge Learner maintains a lightweight **Gaussian Mixture Model (GMM)**:

$$p_{local}(z) = \sum_{c=1}^{C} \pi_c \mathcal{N}(z; \mu_c, \Sigma_c) \tag{7}$$

where π_c are mixing weights, $\mu_c \in \mathbb{R}^d$ are component means, and Σ_c are diagonal covariance matrices. With $C = 64$ components and embedding dimension $d = 128$, utilizing FP16 precision (2 Bytes), the storage requirement is:

$$\text{Size} \approx 2 \cdot (C \times d \times 2\text{B}) + (C \times 2\text{B}) \approx 33 \text{ KB} \tag{8}$$

While larger than our initial estimate, this fits comfortably within the L1/L2 cache of standard ARM processors (e.g., Cortex-A72 has 1MB L2), whereas a standard contrastive memory bank (e.g., 16k samples) requires > 8MB of RAM and incurs high DRAM access energy.

This GMM is updated incrementally using a streaming Expectation-Maximization (EM) algorithm [31] with $O(Cd)$ complexity per iteration. To ensure stability during initialization (Cold Start), the system defaults to a conservative local policy for the first 50 frames to populate the sufficient statistics.

4.1.3 Hard Negative Mining via GMM. Merely having a distribution is insufficient; we must sample from it effectively. Standard small-batch training fails because randomly sampled negatives are often "easy" (semantically distant), providing negligible gradients [12].

We implement **Boundary-Aware Sampling**, visualized in Figure 3. During the forward pass, given an anchor embedding z^+ from component c^*, we sample "virtual" negative embeddings z^- from *other* components weighted by proximity to the anchor:

$$p(c|z^+, c^*) \propto \pi_c \cdot \exp\left(-\frac{||\mu_{c^*} - \mu_c||^2}{2\tau^2}\right), \quad c \neq c^* \tag{9}$$

where τ is a temperature parameter controlling hardness. This strategy samples negatives near the decision boundary between components (hard negatives) rather than from distant components (easy negatives) or out-of-distribution regions. By synthesizing virtual negatives on-the-fly, we effectively augment the batch size without increasing physical memory usage, as virtual samples are generated, ℓ_2-normalized to project them onto the hypersphere, and immediately discarded after gradient computation.

Edge Training Objective and Flow. To strictly minimize on-device memory, we employ a streaming InfoNCE loss with virtual negatives. For each incoming audio frame x_t, we generate a positive pair $(\tilde{x}_t, \tilde{x}'_t)$ via standard lightweight augmentations (random Gaussian noise and frequency masking). We do *not* use temporal neighbors as positives to avoid buffering latency.

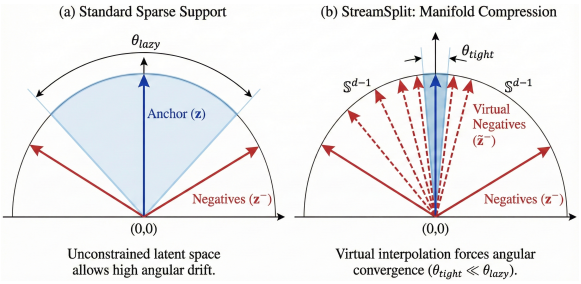

Figure 3: Addressing Small-Batch Conflict. (a) Small batches cause "Lazy Margins" (θ_{lazy}) and degradation. (b) StreamSplit generates virtual negatives from a compact GMM (<35KB), enforcing "Tight Margins" (θ_{tight}) to decouple quality from physical batch size.

The edge encoder f_θ minimizes the negative log-likelihood of the positive pair against N_{syn} virtual negatives sampled from the GMM:

$$\mathcal{L}_{edge} = -\log \frac{\exp(\text{sim}(z_t, z'_t)/\tau)}{\exp(\text{sim}(z_t, z'_t)/\tau) + \sum_{j=1}^{N_{syn}} \exp(\text{sim}(z_t, z^j_{syn})/\tau)} \tag{10}$$

where $z_t = f_\theta(\tilde{x}_t)$, $z'_t = f_\theta(\tilde{x}'_t)$, and $\{z^j_{syn}\}_{j=1}^{N_{syn}}$ are virtual negatives sampled from the local GMM $p_{local}(z)$ using the boundary-aware strategy (Eq. 9).

Training Flow. This objective allows for a fully streaming update:

(1) **Forward:** Encoder processes x_t to get z_t; GMM is updated via online EM.
(2) **Sample:** Virtual negatives z_{syn} are generated from GMM parameters.
(3) **Backward:** Gradients from $\mathcal{L}_{edge}$ update the encoder weights f_θ.
(4) **Sync:** Updated weights are lazily synchronized with the server (See Sec. 4-Phase 3) to align with the global manifold.

Answer to RQ1: Distributional memory replaces explicit sample storage with a compact GMM ($O(Cd)$ vs. $O(Nd)$ memory), while boundary-aware sampling synthesizes hard negatives on-the-fly. This decouples representation quality from physical batch size, enabling edge devices to maintain contrastive learning performance despite memory constraints.

4.2 Phase 2: Uncertainty-Guided Control

The core systems innovation of StreamSplit is the **Control Plane**, which resolves the "Volatility Conflict" (C2). Unlike heuristic splitters (e.g., Neurosurgeon [21]) that react only to bandwidth changes, our system must proactively manage workload based on both system state and data difficulty. This motivates our second research question:

RQ2: How can we dynamically adapt the edge-cloud computation partition to handle runtime volatility in CPU load, network bandwidth, and data difficulty?

4.2.1 Analysis Method. We analyze the runtime adaptation problem across the entire execution pipeline, including edge compute constraints, network transmission costs, and varying data complexity. Existing heuristic methods fail when multiple constraints conflict (e.g., high CPU load but low bandwidth). We investigate whether a learned policy can discover non-obvious adaptation strategies that balance competing objectives.

4.2.2 RL Formulation. We formulate the runtime adaptation as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, with formal definitions provided in Appendix B. The Control Plane runs as a lightweight sidecar process on the edge device, executing policy inference at regular intervals to avoid excessive overhead.

State Space (s_t): The state vector $s_t = [U_t, R_{cpu}, B_{net}]$ captures the full system context. $R_{cpu} \in [0, 100]$ represents CPU utilization percentage. $B_{net} \in \mathbb{R}^+$ is the estimated uplink bandwidth, computed via exponential moving average over recent transmission times. Crucially, we include $U_t \in [0, \log C]$, the **embedding uncertainty**, defined as the Shannon entropy of the GMM component assignment:

$$U_t = H(p(c|z_t)) = -\sum_{c=1}^{C} p(c|z_t) \log p(c|z_t) \tag{11}$$

where $p(c|z_t) = \frac{\pi_c \mathcal{N}(z_t; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(z_t; \mu_{c'}, \Sigma_{c'})}$ is the posterior probability via Bayes' rule. High entropy typically correlates with **transient acoustic events** (e.g., speech onset, glass breaking) which are semantically complex. Low entropy corresponds to steady-state background noise (e.g., HVAC hum). By conditioning the split on U_t, the agent effectively learns a semantic Event Detection policy, offloading only the information-dense frames. We prefer GMM entropy over TTA variance (high latency) or kNN density (high memory) as it provides a zero-cost uncertainty proxy computed directly within the standard inference pass.

Action Space (a_t): The action is the split layer index $k \in \{0, 1, \ldots, L\}$, where L is the total number of encoder layers. Selecting $k = 0$ implies full offloading (minimal edge compute, maximum bandwidth consumption), while $k = L$ implies full on-device processing (maximum edge compute, zero bandwidth). Note that to satisfy strict bandwidth constraints detected in the state s_t, specific offloading actions ($k < L$) are coupled with dynamic quantization (e.g., INT8 precision) to ensure the transmission fits within the estimated T_{max}.

Reward Function (r_t): The reward balances task performance with system efficiency:

$$r_t = \alpha \cdot \mathcal{A}_{task} - \beta \cdot \frac{\text{Lat}_t}{T_{max}} - \eta \cdot \frac{E_t}{E_{budget}} \tag{12}$$

where $\mathcal{A}_{task} \in [0, 1]$ is the task accuracy on a validation set, Lat_t is the end-to-end latency (edge compute + network transmission + server compute) normalized by a maximum allowable latency T_{max}, and E_t is the energy consumption normalized by a per-frame budget E_{budget}. The coefficients α, β, η prioritize accuracy while penalizing latency violations and energy drain.

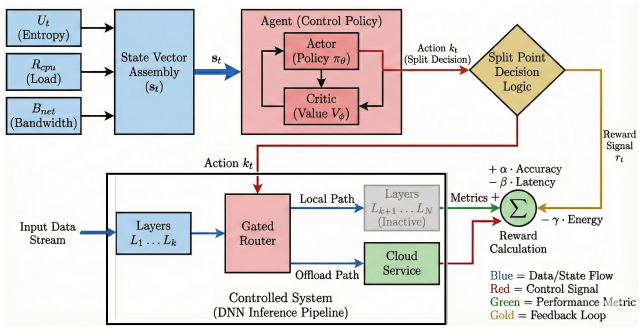

Figure 4: RL Control Logic. The agent maps normalized state s_t (incl. uncertainty) to split layer k_t. Reward r_t (Eq. 12) balances accuracy, latency, and energy.

Decision Frequency and Atomic Transitions: The RL agent executes every $T_{step} = 10$ frames ($\approx$ 100ms). The policy network (2-layer MLP) inference time is measured at 0.2ms on the Pi 4B. To guarantee consistency during transitions and avoid the complexity of managing in-flight data, split decisions occur atomically at these 100ms boundaries. When the RL agent shifts the computation partition (e.g., moving from layer 3 to layer 5), the transition applies exclusively to the next T_{step} block. No data currently traversing the pipeline is redone, and no in-flight frames are dropped, stalled, or left in an inconsistent state during the shift. This high-frequency, atomic control allows StreamSplit to react to abrupt bandwidth collapses without corrupting the streaming pipeline.

4.2.3 Policy Network and Adaptive Behavior. The control logic is visualized in Figure 4. We employ a Proximal Policy Optimization (PPO) [32] agent with a lightweight two-layer fully connected network. The policy network $\pi_\theta(a|s)$ and value network $V_\phi(s)$ share the first layer to minimize parameter overhead while maintaining expressiveness.

The agent is trained offline on historical traces collected from diverse hardware platforms under varying network conditions (bandwidth, latency, packet loss). The learned policy exhibits complex adaptive behaviors that heuristic rules cannot capture. For instance, under high CPU load but low bandwidth, a heuristic might stall. The StreamSplit agent learns to apply aggressive quantization (reducing embedding precision) to fit bandwidth constraints, sacrificing marginal accuracy to maintain real-time throughput. Conversely, when both CPU and bandwidth are available but uncertainty U_t is high (ambiguous sample), the agent offloads to the server even at minimal split depth, prioritizing accuracy over efficiency. We note that while the training reward r_t uses ground-truth accuracy $\mathcal{A}_{task}$, the deployed policy π_θ relies solely on the observable state s_t (uncertainty, CPU, bandwidth) and does not require labels at runtime.

Answer to RQ2: Formulate adaptation as an RL problem where the agent observes CPU load, network bandwidth, and embedding uncertainty to select optimal split points. The learned policy discovers non-obvious strategies (e.g., quantization under bandwidth constraints, uncertainty-based offloading) that heuristic rules miss, achieving robust real-time execution across heterogeneous platforms.

4.3 Phase 3: Server-Side Refinement

The Cloud Server handles the heavy lifting of global model convergence. However, dynamic splitting introduces **asynchrony**: frames arrive out of order or are dropped entirely when the network degrades or the RL agent selects full on-device processing to conserve bandwidth. This breaks standard synchronous SGD, which assumes uniform batches and temporal ordering. To address this, the Server Refiner implements a **Temporal Buffer** and the **Hybrid Loss** derived in Section 3.

4.3.1 Temporal Buffer Management. As shown in Figure 5, the server maintains a sliding temporal window of recent embeddings. When an update z_t arrives from the edge, it is inserted into the buffer at the corresponding temporal index. If a frame is skipped (due to RL-driven local processing or network packet loss), the buffer contains a temporal gap. Rather than discarding the buffer or performing explicit interpolation (which would require maintaining expensive historical state), we construct a k-Nearest Neighbor temporal graph $\mathcal{G} = (V, E)$ on the available embeddings, where edges $(i, j) \in E$ connect temporally adjacent frames within a small window. We set the server temporal window size to $W = 100$ frames (approx. 1s context). The k-NN graph is constructed with $k = 5$ neighbors. Constructing the Laplacian on the server takes approx. 3ms for a batch of 100, which is fully pipelined with inference to minimize overhead.

4.3.2 Hybrid Objective for Robust Convergence. The server optimizes the combined objective from Eq. 13:

$$\mathcal{L}_{server} = \underbrace{\mathcal{L}_{task}}_{\text{Classification}} + \lambda_1 \underbrace{\mathcal{L}_{SW}(p_\theta, \mathcal{U})}_{\text{Diversity}} + \lambda_2 \underbrace{\mathcal{L}_{Lap}(\mathcal{G})}_{\text{Affinity}} \quad (13)$$

where $\mathcal{L}_{task}$ denotes the global objective. In pure self-supervised settings, $\mathcal{L}_{task}$ is formulated as a global InfoNCE loss over the server buffer; when sparse labels are available (as in our validation experiments to establish performance upper bounds), it is instantiated as the standard cross-entropy loss. $\mathcal{L}_{SW}$ is the Sliced-Wasserstein Distance (Eq. 3) enforcing global diversity, and $\mathcal{L}_{Lap}$ is the Laplacian regularization (from Section 3) enforcing temporal smoothness:

$$\mathcal{L}_{Lap} = \frac{1}{|E|} \sum_{(i,j)\in E} ||z_i - z_j||^2 \quad (14)$$

The Laplacian term penalizes large embedding jumps between temporally adjacent frames. This mechanism is critical for handling extended periods of connectivity loss. During a severe network outage, the edge device may be forced to drop frames if local memory is exhausted. When connectivity resumes, the server's Temporal Buffer receives a sequence with a massive temporal gap. Instead of naively interpolating missing features—which would hallucinate phantom acoustic events—$\mathcal{L}_{Lap}$ enforces a Lipschitz-continuous smoothing prior across the boundary of the outage.

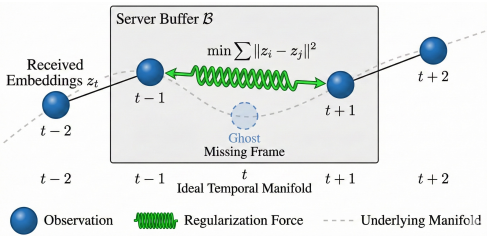

Figure 5: Manifold Stitching. Laplacian regularization ($\mathcal{L}_{Lap}$) acts as a spring force across temporal gaps (missing frames), maintaining manifold continuity without requiring explicit interpolation.

As formalized by Theorem 3.2, minimizing the Dirichlet energy guarantees bounded interpolation error even when the spectral gap (λ_2) of the temporal graph is degraded by dropouts. The server effectively "stitches" the representations together by pulling the pre-outage and post-outage embeddings toward a shared, smooth manifold geometry. Because the representations remain Lipschitz-continuous, the server can bridge these temporal gaps gracefully. This is empirically validated in our ablation studies (Table 5), where StreamSplit sustains a severe 40% frame drop rate while only degrading 6.6% in accuracy, whereas standard contrastive objectives suffer catastrophic collapse.

Simultaneously, the Sliced-Wasserstein term prevents mode collapse by ensuring the aggregated buffer distribution matches the global uniform prior. This is particularly important in asynchronous federated settings, where devices with fast connections might dominate the global model. By minimizing $\mathcal{L}_{SW}$, we ensure that the server model converges to a balanced representation of the entire data distribution, rather than overfitting to high-throughput devices.

4.3.3 Downlink Synchronization Strategy. To close the learning loop without incurring prohibitive downlink costs, we employ a **Lazy Synchronization** protocol. The Server transmits updated GMM parameters (μ_c, Σ_c)—which are lightweight (< 35KB)—to the Edge every $T_{sync} = 100$ frames. The heavier Encoder weights (f_θ) are only synchronized when the device detects a charging state or high-bandwidth WiFi connection. In our energy evaluation (Table 2), we account for the GMM synchronization cost, which adds a negligible 0.4 mJ/frame average overhead.

5 Implementation

Definitions. We define a **Sample** as a 1-second audio clip. A **Batch** consists of $N = 8$ samples. A **Frame** refers to the STFT window (25ms). The RL agent makes decisions every $T_{step} = 10$ frames ($\approx$ 100ms) to ensure responsiveness to network volatility, while gradients are aggregated at the **Batch** level ($N = 8$).

We implement the StreamSplit framework entirely in Python, using PyTorch [33] for model training and inference across all components. Edge deployment leverages PyTorch's TorchScript for optimized execution on resource-constrained devices.

Audio Processing. For real-time spectrogram extraction, we employ PyKissFFT [34], a Python binding for the lightweight KissFFT library optimized for embedded platforms. Compared to NumPy's FFT implementation, PyKissFFT reduces STFT computation latency from 7.8ms to 3.2ms per frame on Raspberry Pi 4B, enabling continuous 16kHz audio streaming without frame drops. We compute 128-bin mel spectrograms using 25ms windows with 10ms hop length.

Hardware Platforms. We deploy and evaluate StreamSplit on three representative platforms spanning the edge-cloud spectrum:

- **Raspberry Pi 4B** (4GB RAM, ARM Cortex-A72 @ 1.5GHz): Represents resource-constrained IoT devices. The GMM module (35KB) and PPO policy (12KB) fit entirely within the 256KB L2 cache.
- **Apple MacBook M2** (8GB unified memory, 8-core CPU): Represents capable edge devices with integrated neural engine. Used for development and mid-tier deployment scenarios. We leverage the PyTorch `MPS` (Metal Performance Shaders) backend to offload matrix operations to the M2's 10-core **GPU**, achieving a 3.2× speedup over CPU execution. While the Neural Engine (ANE) offers further theoretical gains, current framework support favors the GPU for custom dynamic graphs like ours.
- **Cloud Server** (Intel Xeon Gold 6248R, 4× NVIDIA RTX 3090): Hosts the Server Refiner for global model aggregation and refinement.

Network Emulation. To evaluate adaptation under realistic network conditions, we employ the Linux Traffic Control (`tc`) utility as a network emulator, simulating bandwidth fluctuations (1–50 Mbps), latency variations (20–200ms), and packet loss (0–5%). We construct 6 network profiles based on real-world 4G/5G traces [35], spanning stable, variable, and congested scenarios.

Training Configuration. Edge-side training uses the Adam optimizer with a learning rate of 10^{-3} and batch size $N = 8$. The PPO agent trains for 2M environment steps with a discount factor $\gamma = 0.99$ and clipping $\epsilon = 0.2$. Server-side training employs SGD with momentum 0.9 over 100 epochs. To find a balance between accuracy and strict penalties for latency and energy use, we use a lightweight 2-layer MLP for the RL policy with specific reward weights: $\alpha = 10$, $\beta = 5$, and $\eta = 3$. Loss hyperparameters ($\lambda_1 = 0.1$, $\lambda_2 = 0.01$) were determined via grid search on held-out validation data.

Reproducibility Details. The audio encoder backbone is a standard ResNet-18 adapted for 1D audio (taking spectrogram inputs), consisting of $L = 8$ splitable blocks with an output dimension $d = 128$, which provides the standard latent size for audio representation stability. For the Distributional Memory, we utilize a GMM with $C = 64$ components ($\tau = 0.1$) to find the optimal balance between expressiveness and cache fit. Crucially, this estimator only takes up 33 KB of space, allowing it to fit completely in the L2 cache of standard ARM processors. This avoids having to access DRAM, which uses a lot of power. During hard negative mining, we sample $N_{syn} = 256$ virtual negatives per anchor to ensure a sufficient gradient signal without physical memory overhead.

Execution Consistency. To address in-flight data and ensure consistency during dynamic transitions, our framework makes split decisions atomically at 100ms boundaries ($T_{step} = 10$ frames) to ensure real-time response to network volatility. This means that split-point shifts happen precisely between frames, ensuring that no data is redone and there is no lag in flight.

SWD and Sampling Overhead. We set the number of random projections for the Sliced-Wasserstein Distance to $M = 50$ on both the edge and server to balance approximation error with compute cost. We measured the specific runtime overhead on the Raspberry Pi 4B: the SWD computation introduces a latency of 1.2ms per batch, while the GMM-based hard negative synthesis (generating $N_{syn} = 256$ virtual negatives) adds only 0.8ms per batch. These overheads are negligible compared to the 10ms hop length, preserving real-time capabilities.

Quantization Implementation. To minimize transmission latency during offloading ($k < L$), we compress intermediate feature maps using **asymmetric INT8 quantization** with a per-tensor granularity. We employ Post-Training Quantization (PTQ) rather than Quantization-Aware Training (QAT) to maintain training pipeline simplicity. Calibration statistics (min/max ranges) are collected offline using a representative subset of the training data. The runtime overhead for quantization and dequantization on the Raspberry Pi 4B is measured at < 0.5ms per frame. Empirical evaluation shows this compression introduces a negligible accuracy degradation of $< 0.3\%$ compared to full FP32 transmission, validating its viability for bandwidth-constrained edges.

6 Evaluation

We conduct extensive experiments to evaluate StreamSplit across diverse datasets, hardware platforms, and network conditions through systematic evaluation of system efficiency, representation quality, component contributions, and hardware generalization.

6.1 Experimental Setup

6.1.1 Datasets. We evaluate StreamSplit on two complementary datasets:

AudioSet [36] is a large-scale audio classification benchmark containing over 2 million human-labeled 10-second clips across 527 classes. We use the balanced evaluation subset (20,371 clips) for standardized comparison with prior work, and a subset of 100K clips for training the contrastive encoder.

EcoStream-Wild comprises 48 hours of continuous uncurated audio collected using 4 Raspberry Pi 4B devices deployed in diverse real-world environments: a university office, a crowded cafeteria, a residential living room, and an outdoor balcony, recording over a 2-week period. To establish ground truth, we employed a semi-automated annotation protocol: a silence removal algorithm first proposed candidate segments, which were then manually verified and labeled by human annotators. The dataset features a highly unbalanced, realistic class distribution dominated by "Silence/Background" (60.2%), followed by "Speech" (24.5%), and specific "Transient Events" (15.3%, e.g., footsteps, door slams, glass breaking) across 15 distinct classes. We commit to releasing the full dataset, annotation logs, and source code upon acceptance to facilitate reproducibility.

6.1.2 Baseline Methods. We compare StreamSplit against the following representative baselines:

Edge-Only: Full model training and inference on the edge device without server communication, representing maximum autonomy but limited model capacity due to memory constraints.

Server-Only: All audio data transmitted to the cloud server for processing, representing maximum accuracy but high bandwidth consumption and latency.

FSL [19]: Federated Split Learning with fixed partition points, combining federated averaging with split inference.

FedCL [37]: Federated Contrastive Learning using synchronized memory banks across clients with periodic server aggregation.

Rule-Based Adaptive Split: A heuristic baseline that offloads based on static thresholds for **both** bandwidth and CPU utilization (e.g., "Offload if Bandwidth > X AND CPU < Y"). This represents a competent engineering solution (an upgrade to Neurosurgeon [21]) without learning. Note that we exclude early-exit inference methods (e.g., SPINN [38]) as baselines, as they optimize for classification confidence rather than the representation quality required for Contrastive Learning.

6.1.3 Evaluation Metrics. We evaluate system performance along three dimensions:

System Efficiency: Bandwidth consumption (KB/frame), end-to-end latency (ms), and energy consumption (mJ/frame) measured using PowerJoular [39] on Raspberry Pi 4B.

Representation Quality: Linear probe accuracy (%) on frozen embeddings, and mean Average Precision (mAP) for audio retrieval tasks.

Adaptation: Response time to network changes (ms) and accuracy stability under varying conditions (standard deviation).

6.2 End-to-End System Performance

We first evaluate StreamSplit's system-level efficiency gains—the critical metrics for practical edge deployment.

6.2.1 Bandwidth Reduction. Figure 6 presents bandwidth consumption across methods. Note that the results report the transmitted payload per **processing batch** of $N = 8$ clips (approx. 8 seconds of audio). For the Server-Only baseline, transmitting 8 clips of raw PCM audio (16kHz, 16-bit mono) consumes $8 \times 32\text{KB} = 256\text{KB}$. StreamSplit achieves **77.1% bandwidth reduction** (58.7 KB/batch) by transmitting compact intermediate representations.

Comparison with Opus Baseline. We analytically compare against standard Opus compression (VoIP profile at 32kbps). While Opus offers a competitive bandwidth baseline (consuming $\approx$ 4 KB/frame, comparable to StreamSplit's most aggressive modes), it introduces a fundamental trade-off that contradicts edge offloading: it forces the cloud server to execute the *entire* feature extraction pipeline (100% of the compute load). StreamSplit offers a Pareto-optimal frontier: we match Opus bandwidth efficiency when necessary, but typically offload 35% of the compute burden to the edge (Fig. 7) to reduce server latency and load, a benefit that pure audio compression cannot achieve.

Compared to baselines, FSL (187.2 KB/frame) and Rule-Based Split (124.3 KB/frame) achieve only 26.9% and 51.4% reduction respectively, as they lack data-aware adaptation. FedCL (201.4 KB/frame) incurs additional overhead from memory bank synchronization.

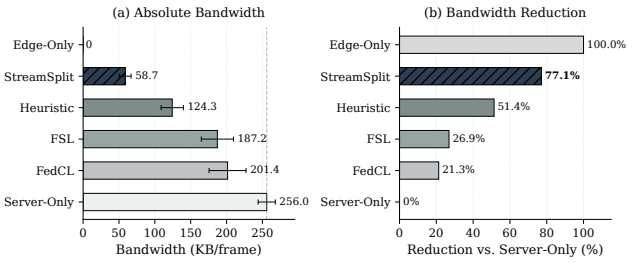

Figure 6: Bandwidth Consumption. StreamSplit achieves 77.1% reduction vs. Server-Only by transmitting compact, adaptive intermediate representations.

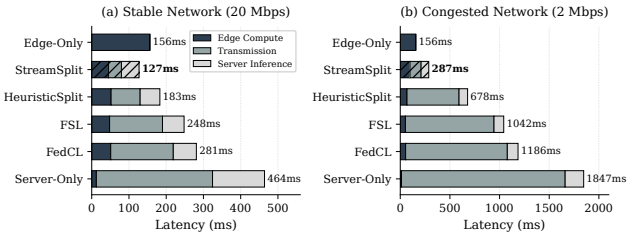

Figure 7: Latency Breakdown. StreamSplit reduces latency by 72.6% (stable) to 84.5% (congested) via adaptive splitting.

6.2.2 Latency Reduction. Figure 7 shows end-to-end latency breakdown across methods. StreamSplit achieves **72.6% latency reduction** compared to Server-Only (127ms vs. 464ms average). The latency composition reveals the source of gains: StreamSplit spends 45.2ms on edge compute (35.6%)—which explicitly includes the < 2ms overhead for GMM updates and RL policy inference—34.6ms on transmission (27.2%), and 41.2ms on server inference (32.4%). In contrast, Server-Only spends 312ms on transmission alone (67.2%) due to raw audio upload.

Under degraded network conditions (2 Mbps, 150ms RTT), StreamSplit's advantage amplifies: 287ms vs. 1,847ms for Server-Only (84.5% reduction). The RL agent automatically increases local processing, reducing transmission dependency.

6.2.3 Energy Savings and Battery Life. Table 2 summarizes energy consumption on Raspberry Pi 4B powered by a 10,000mAh battery pack. StreamSplit consumes **89.3 mJ/frame**, achieving **52.3% energy savings** compared to Server-Only (187.2 mJ/frame). The savings stem from reduced radio transmission, which dominates energy consumption on battery-powered devices.

Total energy consumption includes the uplink transmission of embeddings, local computation, and the periodic downlink synchronization of GMM parameters (from §5).

Translated to battery life, StreamSplit enables **11.2 hours** of continuous operation compared to 5.3 hours for Server-Only—a 2.1× extension critical for untethered deployment. Edge-Only achieves longer battery life (14.8 hours) but at significant accuracy cost (see §6.3).

Table 2: Energy Consumption and Battery Life on Raspberry Pi 4B with 10,000mAh battery. Results report mean ± std dev over 5 runs.

Method	Compute (mJ)	Transmit (mJ)	Total (mJ)	Battery Life (hours)
Edge-Only	67.4 ± 1.2	0.0	67.4 ± 1.2	14.8 ± 0.3
Server-Only	12.4 ± 0.5	174.8 ± 8.2	187.2 ± 8.5	5.3 ± 0.2
FSL	48.7 ± 2.1	98.3 ± 5.4	147.0 ± 6.1	6.8 ± 0.3
FedCL	52.1 ± 1.9	112.6 ± 6.0	164.7 ± 6.8	6.1 ± 0.2
Rule-Based Split	54.2 ± 1.5	87.1 ± 4.8	141.3 ± 5.1	7.1 ± 0.3
StreamSplit	**54.7 ± 2.3**	**34.6 ± 3.1**	**89.3 ± 4.1**	**11.2 ± 0.5**

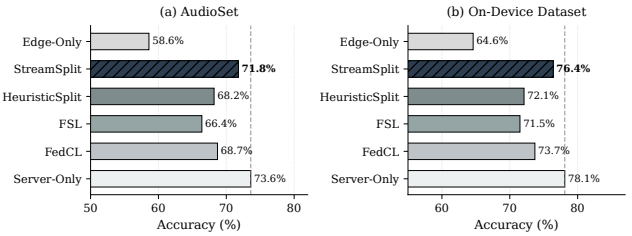

Figure 8: Linear Probe Accuracy. StreamSplit matches Server-Only accuracy within 2.2% across datasets.

6.3 Representation Quality

A critical question is whether StreamSplit's efficiency gains come at the cost of representation quality. We evaluate learned embeddings on downstream tasks to verify that accuracy remains competitive.

6.3.1 Linear Probe Accuracy. Figure 8 presents linear probe accuracy on frozen embeddings across both datasets. StreamSplit achieves **71.8%** on AudioSet and **76.4%** on the On-Device dataset, falling **within 2.2%** of Server-Only (73.6% and 78.1% respectively). This marginal gap demonstrates that our distributional memory and hybrid loss effectively preserve representation quality despite the constraints of edge deployment.

Compared to other edge-compatible methods, StreamSplit outperforms Edge-Only by 13.2% (AudioSet) and 11.8% (On-Device), FSL by 5.4% and 4.9%, and FedCL by 3.1% and 2.7%. Rule-Based Split achieves 68.2% and 72.1%, suffering from suboptimal split decisions that sacrifice quality for bandwidth savings.

6.3.2 Retrieval Performance. Table 3 presents audio retrieval metrics (mAP@10, R@1) using learned embeddings. StreamSplit achieves 0.412 mAP@10 and 38.7% R@1, compared to Server-Only's 0.431 and 40.2%—a gap of only 4.4% and 3.7% respectively. These results confirm that StreamSplit embeddings capture semantic similarity effectively, enabling downstream applications like audio search and similarity-based clustering.

6.4 Ablation Studies: Why It Works

We conduct ablation studies to isolate the contribution of each StreamSplit component and understand the source of performance gains.

Table 3: Audio Retrieval Performance on AudioSet. StreamSplit maintains competitive retrieval quality with minimal degradation.

Method	mAP@10	R@1 (%)
Edge-Only	0.287	26.4
Server-Only	0.431	40.2
FSL	0.356	33.1
FedCL	0.378	35.8
Rule-Based Split	0.341	31.9
StreamSplit	**0.412**	**38.7**

Table 4: Adaptation Strategy Comparison. RL-based control outperforms alternatives in both steady-state and adaptation scenarios. (Mean ± std dev).

Strategy	Accuracy (%)	Latency (ms)	Energy (mJ)	Adaptation Time (ms)
Static ($k = 3$)	68.7 ± 0.2	203 ± 15	142.6 ± 5.5	N/A
Rule-Based	69.4 ± 0.3	156 ± 10	118.7 ± 4.2	4, 200 ± 350
RL (Ours)	**71.8 ± 0.4**	**127 ± 12**	**89.3 ± 4.1**	**1, 200 ± 150**

Table 5: Loss Function Comparison under varying frame drop rates. Hybrid loss maintains robustness while alternatives degrade significantly. (Mean ± std dev).

Loss Function	0% Drop	20% Drop	40% Drop
MSE Only	69.2 ± 0.3	61.4 ± 1.2	52.8 ± 2.5
KL Divergence	70.1 ± 0.3	63.7 ± 1.1	55.1 ± 2.1
$\mathcal{L}_{task} + \mathcal{L}_{SW}$	70.8 ± 0.4	67.2 ± 0.8	61.3 ± 1.5
$\mathcal{L}_{task} + \mathcal{L}_{Lap}$	70.4 ± 0.3	66.8 ± 0.9	60.7 ± 1.4
Hybrid (Ours)	**71.8 ± 0.4**	**69.4 ± 0.5**	**65.2 ± 0.9**

6.4.1 RL Control vs. Alternatives. Table 4 compares StreamSplit's RL-based control against alternative adaptation strategies. We evaluate three configurations:

- **Static:** Fixed split point ($k = 3$) regardless of conditions.
- **Rule-Based:** Heuristic adaptation using both bandwidth and CPU thresholds.
- **RL (Ours):** Learned policy considering CPU, bandwidth, and uncertainty.

The RL policy achieves 2.4% higher accuracy, 18.6% lower latency, and 24.8% lower energy than the heuristic baseline. Critically, adaptation time—the delay before the system responds to network changes—is 3.5× faster (1,200ms vs. 4,200ms). The heuristic requires multiple probe transmissions to estimate new bandwidth, while the RL agent adapts within a single decision interval using its learned state representation.

6.4.2 Hybrid Loss vs. Alternatives. Table 5 evaluates the contribution of our hybrid loss components under varying frame drop rates (simulating network volatility).

Under ideal conditions (0% drop), all losses perform comparably. However, as frame drop rate increases, the hybrid loss demonstrates superior robustness: at 40% drop rate, it maintains 65.2% accuracy compared to 52.8% for MSE and 55.1% for KL divergence—a 12.4%

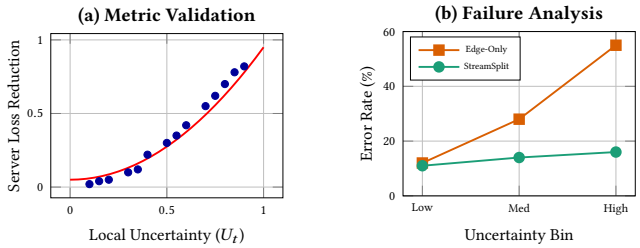

Figure 9: Uncertainty Analysis. (a) Local uncertainty correlates ($r = 0.84$) with server utility. (b) Acoustic Event Adaptation. High uncertainty (U_t) regions correspond to transient audio events (e.g., speech). StreamSplit automatically offloads these complex frames while processing steady-state background noise locally.

and 10.1% advantage respectively. The Sliced-Wasserstein term ($\mathcal{L}_{SW}$) prevents mode collapse from sparse updates, while Laplacian smoothness ($\mathcal{L}_{Lap}$) maintains manifold continuity across gaps.

6.4.3 Uncertainty Calibration. A critical concern is whether local uncertainty (U_t) is a reliable proxy for global sample difficulty. We analyzed the correlation between edge-calculated entropy and the reduction in loss achieved by server processing. Figure 9 shows a strong positive correlation (Pearson $r = 0.84$). This confirms that when the edge model is uncertain, the sample is indeed 'hard' and benefits most from server refinement. Samples with low local uncertainty showed negligible improvement when processed by the server, validating our offloading logic.

6.5 Hardware Generalization: M2 vs. Pi 4

A key requirement for practical deployment is generalization across heterogeneous hardware. We evaluate StreamSplit on two platforms representing opposite ends of the edge computing spectrum:

- **Raspberry Pi 4B:** ARM Cortex-A72 @ 1.5GHz, 4GB RAM, representing resource-constrained IoT devices.
- **Apple MacBook M2:** 8-core CPU with Neural Engine, 8GB unified memory, representing capable edge devices.

6.5.1 Performance Comparison. Table 6 compares StreamSplit across platforms. On Apple M2, StreamSplit achieves higher accuracy (73.2% vs. 71.8%) and lower latency (67ms vs. 127ms) due to faster local inference. Interestingly, energy efficiency improves less dramatically (78.4 mJ vs. 89.3 mJ), as the M2's power consumption scales with its higher compute throughput.

6.5.2 Learned Policy Behavior. Figure 10 visualizes the RL policies learned for each platform. The policies exhibit distinct, hardware-aware behaviors:

Pi 4B Policy: Conservative local processing. The agent selects shallower split points (average $k = 2.1$), offloading 38% of frames entirely ($k = 0$). Under memory pressure, it aggressively offloads to avoid cache thrashing, prioritizing system stability over bandwidth savings.

M2 Policy: Aggressive local processing. The agent selects deeper split points (average $k = 3.8$), offloading only 12% of frames. With ample compute headroom, it maximizes local inference to minimize transmission, achieving 83.5% bandwidth reduction.

Notably, both policies share a common pattern: under high uncertainty ($U_t > 0.7$), they offload regardless of hardware capability, recognizing that ambiguous samples benefit from server refinement. This data-aware behavior emerges automatically from the uncertainty-guided reward, without explicit programming.

6.5.3 Cross-Platform Policy Transfer. To test generalization, we evaluate a policy trained on Pi 4B when deployed directly on M2 (and vice versa). Table 7 shows that **direct policy transfer** incurs only 1.2–2.4% accuracy degradation, suggesting the learned policies capture generalizable adaptation principles rather than hardware-specific quirks. Fine-tuning for 10K steps on the target platform recovers full performance.

Table 6: Cross-Platform Performance. StreamSplit adapts automatically to hardware capabilities, achieving optimal performance on both constrained (Pi 4B) and capable (M2) devices. Results report the mean ± standard deviation over 5 independent runs.

Metric	Raspberry Pi 4B Value	Raspberry Pi 4B vs. Server	Apple MacBook M2 Value	Apple MacBook M2 vs. Server
Accuracy (%)	71.8 ± 0.4	−1.8%	73.2 ± 0.3	−0.4%
Latency (ms)	127 ± 12	−72.6%	67 ± 5	−85.6%
Energy (mJ)	89.3 ± 4.1	−52.3%	78.4 ± 2.8	−58.1%
Bandwidth (KB)	58.7 ± 6.2	−77.1%	42.3 ± 4.5	−83.5%

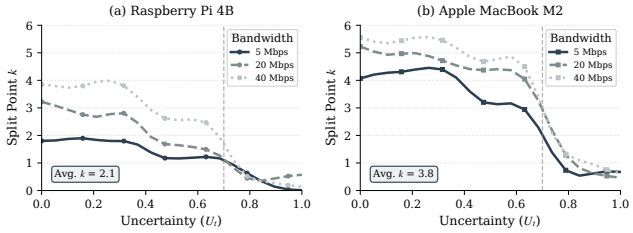

Figure 10: Learned Policies. The agent learns hardware-aware strategies: conservative offloading for Pi 4B vs. aggressive local compute for M2.

Table 7: Cross-Platform Policy Transfer. Policies generalize across hardware with minimal degradation; fine-tuning recovers full performance. Results report the mean ± standard deviation over 5 independent runs.

Configuration	Accuracy (%)	Latency (ms)
Pi 4B → Pi 4B (native)	71.8 ± 0.4	127 ± 12
M2 → Pi 4B (transfer)	69.4 ± 0.6	143 ± 15
M2 → Pi 4B (fine-tuned)	71.6 ± 0.5	129 ± 11
M2 → M2 (native)	73.2 ± 0.3	67 ± 5
Pi 4B → M2 (transfer)	72.0 ± 0.5	78 ± 8
Pi 4B → M2 (fine-tuned)	73.1 ± 0.4	68 ± 6

7 Discussion

Data Obfuscation and Privacy. StreamSplit transmits intermediate embeddings rather than raw audio primarily to improve *bandwidth efficiency*. We state explicitly that while this avoids sending raw PCM data, it provides *incidental obfuscation only*. Reconstructing original audio from deep embeddings, though computationally non-trivial, remains theoretically possible [40]. Therefore, StreamSplit does not provide formal differential privacy or cryptographic guarantees. It should be viewed strictly as a resource-optimization framework; edge applications requiring rigorous security would need to integrate orthogonal privacy-preserving protocols—such as those addressing broad fairness and data minimization in IoT [41, 42]—or advanced network-level security mechanisms like real-time attack attribution in 6G [43], which we leave to future work.

Hardware Generalization Limits. Our evaluation spans Raspberry Pi 4B to Apple M2, but StreamSplit's 50MB memory footprint precludes deployment on microcontroller-class devices (ESP32, Arduino). Extending to ultra-low-power platforms would require aggressive quantization and fixed-point GMM redesign. The algorithmic contributions remain architecture-agnostic and could transfer to compressed models.

Network Assumptions. StreamSplit assumes bidirectional edge-server connectivity. Under complete disconnection, the system degrades to Edge-Only mode with reduced representation quality. Tolerating extended offline periods through delayed synchronization remains an open challenge.

Multi-Client Scope. While this work focuses on optimizing the single-stream vertical offloading link, the server-side $\mathcal{L}_{SW}$ term (Eq. 13) naturally mitigates bias in multi-client settings. By enforcing that the *aggregated* buffer distribution matches the global uniform prior, StreamSplit prevents high-throughput devices (which contribute more frames) from dominating the embedding space geometry.

8 Related Work

Contrastive Learning on Edge. Standard contrastive methods require large memory banks [13] or batch sizes [12] infeasible on edge devices. Recent lightweight alternatives employ importance sampling [44] or knowledge distillation [45], but still require substantial memory overhead. StreamSplit introduces *distributional memory*—a compact GMM replacing explicit storage—enabling contrastive learning with <35KB overhead through boundary-aware negative synthesis. While recent works like SynCo [46] propose synthetic negatives, they typically mix feature-level negatives from a queue. StreamSplit differs by generating virtual negatives from a parametric GMM distribution, eliminating the need to store a memory bank entirely. We exclude non-contrastive baselines (e.g., BYOL-A [47], SimSiam [48], VICReg [49]) as they rely on batch normalization statistics (undefined for streaming $N = 1$) or momentum encoders (doubling memory), which are incompatible with strict edge constraints.

Energy Efficiency Context. Recent benchmarking of SSL on the edge [50] confirms that standard CL frameworks (e.g., SimCLR with ResNet-18) exhibit prohibitive energy footprints due to memory and compute intensity. StreamSplit situates itself within this landscape by demonstrating that while purely local execution offers the lowest absolute energy (67.4 mJ, Table 2), it suffers from dimensional collapse. StreamSplit incurs a moderate energy premium over these local baselines to achieve server-grade accuracy, while still reducing energy by 52.3% compared to the static offloading approaches often assumed in prior energy profiles.

Split Computing and Split Learning. Neurosurgeon [21] pioneered adaptive edge-cloud partitioning, with extensions to multi-exit architectures [38] and privacy-aided federated settings [19, 51]. However, existing approaches assume fixed partition points, failing to adapt to runtime volatility. StreamSplit introduces *uncertainty-guided control*, dynamically selecting split points based on system state and data difficulty, achieving 3.5× faster adaptation than heuristic methods.

The Limits of Decoupled Approaches. Existing resource-aware architectures reactively split computation based on telemetry [21, 23]. Naively pairing these schedulers with contrastive objectives fails: inevitable network-induced frame drops shatter the temporal embedding manifold. StreamSplit overcomes this via joint co-design. Proactively, our Uncertainty-Guided Splitter aligns offloading with actual sample difficulty (validating our GMM entropy metric, which yields a Pearson correlation of $r = 0.84$ with server utility). Reactively, our Server Refiner's Hybrid Loss integrates Laplacian regularization ($\mathcal{L}_{Lap}$) [25] to enforce a Lipschitz-continuous smoothing prior. This allows the server to gracefully bridge temporal gaps from sparse updates without explicit interpolation, outperforming independent small-batch [22] or scheduling interventions.

Federated Learning. FL enables distributed training with data locality across edge and cyber-physical systems [52, 53, 54], with recent extensions to contrastive settings [37, 55]. However, FL assumes periodic synchronization and homogeneous architectures, conflicting with streaming applications requiring continuous updates on heterogeneous hardware. StreamSplit addresses this through asynchronous server refinement, complementing rather than replacing FL systems. Similarly, while LW-FedSSL [56] reduces resource usage via layer-wise training, StreamSplit focuses on real-time *inference* and continuous learning via adaptive splitting, which is orthogonal to layer-wise federated training strategies.

9 Conclusion

We introduced StreamSplit, a framework enabling streaming contrastive learning on edge devices. By replacing memory-intensive queues with *Distributional Memory* (< 35KB) and optimizing offloading via *Uncertainty-Guided RL*, StreamSplit resolves fundamental resource conflicts. Our evaluation confirms bandwidth reductions of 77.1% and energy savings of 52.3% while maintaining accuracy within 2.2% of server-only baselines. The framework proves robust across heterogeneous hardware (Pi 4B to M2), offering a scalable path for adaptive edge intelligence.

Acknowledgments

In accordance with ACM's Policy on Authorship, the authors disclose the use of generative AI tools to assist in the conceptual drafting of graphical icons used in preliminary versions of this manuscript's figures. All formal technical schematics, system architectures, and data plots presented in this final version were

manually constructed by the authors to ensure technical accuracy and rigor.

A Theoretical Convergence Guarantees

Lemma 1 (Partition Function Bias). The contrastive objective asymptotically maximizes mutual information. However, for a finite batch size N, the estimator of the partition function $Z_\theta(x) = \mathbb{E}_{z\sim p_\theta}[e^{f(x)^\top z}]$ is biased. The deviation in the empirical loss is dominated by:

$$\Delta\mathcal{L} = \left|\log\left(\frac{1}{N}\sum_{i=1}^{N} e^{f(x)^\top z_i}\right) - \log \mathbb{E}_{z\sim p_\theta}[e^{f(x)^\top z}]\right| \tag{15}$$

Definition 4 (Wasserstein-1 Distance). Let $\mathrm{Lip}_1(\mathbb{S}^{d-1})$ be the set of 1-Lipschitz functions on the hypersphere. The Wasserstein-1 distance between distributions p and q is given by the Kantorovich-Rubinstein duality:

$$\mathcal{W}_1(p,q) = \sup_{h\in \mathrm{Lip}_1} \left|\mathbb{E}_{z\sim p}[h(z)] - \mathbb{E}_{z\sim q}[h(z)]\right| \tag{16}$$

Proof of Theorem 3.1 (Small-Batch Robustness).

Setup. Let the contrastive loss for anchor x be:

$$\mathcal{L}(x) = -\log \frac{e^{f(x)^\top z^+}}{\frac{1}{N}\sum_{i=1}^{N} e^{f(x)^\top z_i}} \tag{17}$$

where z^+ is the positive sample and $\{z_i\}_{i=1}^N$ are negatives sampled from p_θ. The population loss replaces the finite sum with the expectation under p_θ.

Step 1: Distributional Mismatch. Define the critic function $h(z) = e^{f(x)^\top z}$. Since embeddings are ℓ_2-normalized ($\|z\| = 1$) and $\|f(x)\| \le 1$, we have $h(z) \in [e^{-1}, e]$, making h bounded with Lipschitz constant $K \le e$.

By the Kantorovich-Rubinstein duality for $\mathcal{W}_1$:

$$\left|\mathbb{E}_{z\sim p_\theta}[h(z)] - \mathbb{E}_{z\sim \mathcal{U}}[h(z)]\right| \le K\cdot \mathcal{W}_1(p_\theta, \mathcal{U}) < K\epsilon \tag{18}$$

Step 2: Finite Sample Error. By Hoeffding's inequality, for N i.i.d. samples from p_θ:

$$\left|\frac{1}{N}\sum_{i=1}^{N} h(z_i) - \mathbb{E}_{p_\theta}[h(z)]\right| \le (e - e^{-1})\sqrt{\frac{\log(2/\delta)}{2N}} \tag{19}$$

with probability at least $1-\delta$.

Step 3: Combined Bound. Since $\log(\cdot)$ is Lipschitz on $[e^{-1}, e]$ with constant e, composing the bounds via triangle inequality yields:

$$|\mathcal{L}_N - \mathcal{L}_\infty| \le \underbrace{e\cdot K\cdot \epsilon}_{C_1\epsilon} + \underbrace{e(e-e^{-1})\sqrt{\frac{\log(2/\delta)}{2}}}_{C_2}\cdot\frac{1}{\sqrt{N}} \tag{20}$$

Thus, minimizing the diversity gap ϵ (via $\mathcal{L}_{SW}$) directly compensates for the bias induced by small N, allowing edge devices with small batches to approximate large-batch server performance. □

Proof of Theorem 3.2 (Temporal Interpolation).

Setup. Let $\mathbf{Z} \in \mathbb{R}^{T\times d}$ be the embedding matrix where row z_t is the embedding at time t. Let $\mathcal{G} = (V, E, W)$ be the temporal adjacency graph with Laplacian $\mathbf{L}$. For a missing frame at t^*, define the reconstruction as the weighted neighbor average:

$$\hat{z}_{t^*} = \frac{1}{d_{t^*}} \sum_{j\in\mathcal{N}_{nbr}(t^*)} W_{t^*j} z_j \tag{21}$$

where $d_{t^*} = \sum_{j\in\mathcal{N}_{nbr}(t^*)} W_{t^*j}$ is the weighted degree and $\mathcal{N}_{nbr}(t^*)$ denotes the set of temporal neighbors.

Step 1: Local Reconstruction Error. The squared reconstruction error is:

$$||z_{t^*} - \hat{z}_{t^*}||^2 = \left\| z_{t^*} - \frac{1}{d_{t^*}} \sum_{j\in\mathcal{N}(t^*)} W_{t^*j} z_j \right\|^2 \tag{22}$$

$$= \left\| \frac{1}{d_{t^*}} \sum_{j\in\mathcal{N}(t^*)} W_{t^*j}(z_{t^*} - z_j) \right\|^2 \tag{23}$$

By Jensen's inequality (convexity of $||\cdot||^2$):

$$||z_{t^*} - \hat{z}_{t^*}||^2 \le \frac{1}{d_{t^*}} \sum_{j\in\mathcal{N}(t^*)} W_{t^*j} ||z_{t^*} - z_j||^2 \tag{24}$$

Step 2: Relating Local to Global Energy. Define the local energy at node t^*:

$$E_{local}(t^*) = \sum_{j\in\mathcal{N}(t^*)} W_{t^*j} ||z_{t^*} - z_j||^2 \tag{25}$$

The total Dirichlet energy is $E_{total} = \frac{1}{2}\sum_t E_{local}(t) = \mathrm{Tr}(\mathbf{Z}^\top\mathbf{L}\mathbf{Z})$, where the factor of $1/2$ accounts for double-counting edges.

For any node t^*, the local energy is bounded by the total:

$$E_{local}(t^*) \le 2\cdot E_{total} = 2\cdot \mathrm{Tr}(\mathbf{Z}^\top\mathbf{L}\mathbf{Z}) \tag{26}$$

Step 3: Spectral Gap Refinement. For connected graphs, the discrete Poincaré inequality states that for any zero-mean signal $\mathbf{f}$:

$$\mathrm{Var}(\mathbf{f}) \le \frac{1}{\lambda_2}\mathbf{f}^\top\mathbf{L}\mathbf{f} \tag{27}$$

Applying this per-coordinate to the centered embeddings $\tilde{\mathbf{Z}} = \mathbf{Z} - \bar{z}$ (where $\bar{z}$ is the mean embedding), we obtain that the average local deviation is controlled by the spectral gap. For the worst-case node:

$$\max_{t^*} E_{local}(t^*) \le \frac{2}{\lambda_2}\cdot \mathrm{Tr}(\mathbf{Z}^\top\mathbf{L}\mathbf{Z}) \tag{28}$$

Step 4: Final Bound. Combining Steps 1-3 and using $\mathcal{D}_{aff} = \frac{1}{|E|}\mathrm{Tr}(\mathbf{Z}^\top\mathbf{L}\mathbf{Z}) \le \alpha$:

$$||z_{t^*} - \hat{z}_{t^*}||^2 \le \frac{1}{d_{t^*}} E_{local}(t^*) \tag{29}$$

$$\le \frac{1}{d_{t^*}}\cdot\frac{2}{\lambda_2}\cdot|E|\cdot\alpha \tag{30}$$

$$= \frac{2\alpha\cdot|E|}{\lambda_2\cdot d_{t^*}} \tag{31}$$

For unweighted graphs with uniform degree, $d_{t^*} = |\mathcal{N}(t^*)|$, yielding the stated bound. □

Remark. This bound formalizes the intuition of manifold regularization: a high spectral gap λ_2 (ensured by dense temporal connectivity) combined with low Dirichlet energy α (enforced by $\mathcal{L}_{Lap}$) guarantees bounded interpolation error, providing robustness to frame drops in volatile edge-cloud execution.

B Control Plane MDP Definition

This section provides the formal specification of the Markov Decision Process (MDP) utilized by the Uncertainty-Guided Adaptive Splitter. As introduced in Section 4.2.2, the StreamSplit Control Plane models the dynamic edge-cloud partitioning problem as an RL environment. Table 8 details the exact components of this formulation, including the observable state space (incorporating our zero-cost uncertainty metric), the discrete action space dictating the network split point, and the composite reward function designed to balance representation quality against strict latency and energy constraints.

Table 8: Control Plane MDP Definition. The RL agent optimizes a joint objective of task performance, latency constraints, and energy efficiency.

Component	Description / Definition
State s_t	Vector $[U_t, R_{cpu}, B_{net}] \in \mathbb{R}^3$ U_t: Embedding Uncertainty (Entropy, Eq. 11) R_{cpu}: CPU Utilization (%) B_{net}: Est. Uplink Bandwidth (Mbps, EMA)
Action a_t	Split Layer $k \in \{0, 1, \ldots, L\}$ $k = 0$: Full Offload; $k = L$: Full On-Device
Reward r_t	$r_t = \alpha \cdot \mathcal{A}_{task} - \beta \cdot \frac{\text{Lat}_t}{T_{max}} - \eta \cdot \frac{E_t}{E_{budget}}$ (Eq. 12) Balances Accuracy ($\mathcal{A}$), Latency (Lat), Energy (E)

References

[1] N.A. Angel, D. Ravindran, P.M.D.R. Vincent, K. Srinivasan, and Y.-C. Hu. 2022. Recent advances in evolving computing paradigms: cloud, edge, and fog technologies. *Sensors*, 22, 1, 196.

[2] X. Li, Y. Chen, and Z. Wang. 2025. Edge intelligence through in-sensor and near-sensor computing for the artificial intelligence of things. *npj Unconventional Computing*, 1, 40.

[3] Minh K Quan, Mayuri Wijayasundara, Sujeeva Setunge, and Pubudu N Pathirana. 2025. Quantum-enhanced transformers for robust acoustic scene classification in iot environments. In *2025 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 295–299.

[4] Minh K Quan, Mayuri Wijayasundara, Sujeeva Setunge, and Pubudu N Pathirana. 2025. Quantum-inspired genetic algorithm for robust source separation in smart city acoustics. In *2025 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 1966–1971.

[5] Yusong Wu, Ke Chen, Tianyu Zhang, Yuchen Hui, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. 2023. Large-scale contrastive language-audio pretraining with feature fusion and keyword-to-caption augmentation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE.

[6] Aaqib Saeed, David Grangier, and Neil Zeghidour. 2021. Contrastive learning of general-purpose audio representations. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 3875–3879.

[7] Haider Al-Tahan and Yalda Mohsenzadeh. 2021. CLAR: contrastive learning of auditory representations. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 2530–2538.

[8] H.A. Alharbi, M. Aldossary, J. Almutairi, and I.A. Elgendy. 2023. Energy-aware and secure task offloading for multi-tier edge-cloud computing systems. *Sensors*, 23, 6, 3254.

[9] P. Zhang, M. Li, and Q. Wang. 2025. Edge computing in big data: challenges and benefits. *International Journal of Data Science and Analytics*. In press.

[10] A. Sakip, R. Yersainov, M. Atashikova, T. Rakhimzhan, D.M. Bui, and E.N. Huh. 2023. Lightweight energy-efficient offloading framework for mobile edge/cloud computing. In *2023 17th International Conference on Ubiquitous Information Management and Communication (IMCOM)*. IEEE, 1–8.

[11] L. Lin, X. Liao, H. Jin, and P. Li. 2019. Computation offloading toward edge computing. *Proceedings of the IEEE*, 107, 8, 1584–1607.

[12] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning* (PMLR). Vol. 119, 1597–1607.

[13] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 9729–9738.

[14] Jean-Bastien Grill et al. 2020. Bootstrap your own latent: a new approach to self-supervised learning. In *Advances in Neural Information Processing Systems*. Vol. 33, 21271–21284.

[15] K. Akherfi, M. Gerndt, and H. Harroud. 2018. Mobile cloud computing for computation offloading: issues and challenges. *Applied Computing and Informatics*, 14, 1, 1–16.

[16] M.Y. Akhlaqi and Z.B.M. Hanapi. 2023. Task offloading paradigm in mobile edge computing—current issues, adopted approaches, and future directions. *Journal of Network and Computer Applications*, 212, 103568.

[17] G.I. Arcas, T. Cioara, I. Anghel, D. Lazea, and A. Hangan. 2024. Edge offloading in smart grid. *Smart Cities*, 7, 1, 680–711.

[18] S. Zhou, W. Jadoon, and I.A. Khan. 2023. Computing offloading strategy in mobile edge computing environment: a comparison between adopted frameworks, challenges, and future directions. *Electronics*, 12, 11, 2452.

[19] C. Thapa, P.C.M. Arachchige, S. Camtepe, and L. Sun. 2022. Combined federated and split learning in edge computing for ubiquitous intelligence in internet of things: state-of-the-art and future directions. *Sensors*, 22, 16, 6355.

[20] J. He, D. Zhang, Y. Zhou, and S. Wang. 2023. A split-federated learning and edge-cloud based efficient and privacy-preserving large-scale item recommendation model. *Journal of Cloud Computing*, 12, 47.

[21] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: collaborative intelligence between the cloud and mobile edge. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 615–629.

[22] Hao Fang, Dong Ouyang, Hong Zhang, Shutao Wang, Xing Liu, Xiaohui Xie, Jie Li, and Kui Ren. 2022. Enable deep learning on mobile devices: methods, systems, and applications. *ACM Computing Surveys*, 55, 5, 1–38.

[23] Y. Huang, H. Zhang, X. Shao, X. Li, and H. Ji. 2023. Roofsplit: an edge computing framework with heterogeneous nodes collaboration considering optimal cnn model splitting. *Future Generation Computer Systems*, 140, 79–90.

[24] Soheil Kolouri, Kimia Nadjahi, Umut Simsekli, Roland Badeau, and Gustavo K. Rohde. 2019. Generalized sliced wasserstein distances. In *Advances in Neural Information Processing Systems*. Vol. 32, 261–272.

[25] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7, 2399–2434.

[26] Tongzhou Wang and Phillip Isola. 2020. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning (ICML)*. PMLR, 9929–9939.

[27] L. Kong, G. Chen, and K. Wang. 2024. Cost optimization in edge computing: a survey. *Artificial Intelligence Review*, 57, 285.

[28] Z. Wang et al. 2022. Online continual learning with contrastive vision transformer. In *European Conference on Computer Vision (ECCV)*. Springer, 645–662.

[29] Yunrui Sun, Gang Hu, Yinglei Teng, and Dunbo Cai. 2024. Split federated learning over heterogeneous edge devices: algorithm and optimization. *arXiv preprint arXiv:2411.13907*.

[30] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A kernel two-sample test. *Journal of Machine Learning Research*, 13, 723–773.

[31] Zoubin Ghahramani and Hagai Attias. 2000. Online variational bayesian learning. In *Advances in Neural Information Processing Systems (NIPS)*. Vol. 13. MIT Press, 520–526.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[33] Adam Paszke et al. 2019. Pytorch: an imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 8024–8035.

[34] Mark Borgerding. 2003. KissFFT: a mixed-radix fast fourier transform library. https://github.com/mborgerding/kissfft. Accessed: 2024. (2003).

[35] Francis Y. Yan, Jesber Ma, Greg D. Hill, Deepak Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for internet congestion-control research. In *USENIX Annual Technical Conference (ATC)*, 731–743.

[36] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. 2017. Audio set: an ontology and human-labeled dataset for audio events. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 776–780.

[37] Fengda Zhang et al. 2023. Federated unsupervised representation learning. In *Frontiers of Information Technology & Electronic Engineering*. Vol. 24, 1181–1193.

[38] Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D. Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 1–15.

[39] Adel Noureddine. 2021. PowerJoular: monitor power consumption of software and hardware. https://github.com/joular/powerjoular. Accessed: 2024. (2021).

[40] Nicholas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sez, Florian Tramèr, Borja Balle, Daphne Ippolito, and Eric Wallace. 2023. Extracting training data from diffusion models. In *USENIX Security Symposium*, 5253–5270.

[41] Sina Shaham, Arash Hajisafi, Minh K Quan, Dinh C Nguyen, Bhaskar Krishnamachari, Charith Peris, Gabriel Ghinita, Cyrus Shahabi, and Pubudu N Pathirana. 2025. Privacy and fairness in machine learning: a survey. *IEEE Transactions on Artificial Intelligence*, 6, 7, 1706–1726.

[42] Minh K Quan, Dinh C Nguyen, Van-Dinh Nguyen, Mayuri Wijayasundara, Sujeeva Setunge, and Pubudu N Pathirana. 2024. Toward privacy-preserving waste classification in the internet of things. *IEEE Internet of Things Journal*, 11, 14, 24814–24830.

[43] Minh K Quan and Pubudu N Pathirana. [n. d.] Domain-adapted granger causality for real-time cross-slice attack attribution in 6g networks. In *NeurIPS 2025 Workshop on CauScien: Uncovering Causality in Science*.

[44] Weiming Zhuang, Xin Gan, Yonggang Wen, and Shuai Zhang. 2022. Collaborative unsupervised visual representation learning from decentralized data. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 4912–4921.

[45] Zhi Wang, Xuan Luo, Ruijie Wang, and Xin Li. 2023. EdgeCL: edge-cloud collaborative learning for on-device ai. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 289–301.

[46] Nikolaos Giakoumoglou and Tania Stathaki. 2024. Synco: synthetic hard negatives for contrastive visual representation learning. *arXiv preprint arXiv:2410.02401*.

[47] Daisuke Niizumi et al. 2021. Byol for audio: self-supervised learning for general-purpose audio representation. In *IJCNN*.

[48] Xinlei Chen and Kaiming He. 2021. Exploring simple siamese representation learning. In *CVPR*.

[49] Adrien Bardes et al. 2022. Vicreg: variance-invariance-covariance regularization for self-supervised learning. In *ICLR*.

[50] Fernanda Famá, Roberto Pereira, Charalampos Kalalas, Paolo Dini, Lorena Qendro, Fahim Kawsar, and Mohammad Malekzadeh. 2025. Contrastive self-supervised learning at the edge: an energy perspective. *arXiv preprint arXiv:2510.08374*.

[51] Minh K Quan, Dinh C Nguyen, Van-Dinh Nguyen, Mayuri Wijayasundara, Sujeeva Setunge, and Pubudu N Pathirana. 2023. Hiersfl: local differential privacy-aided split federated learning in mobile edge computing. In *2023 IEEE Virtual Conference on Communications (VCC)*. IEEE, 103–108.

[52] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 1273–1282.

[53] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14, 1–2, 1–210.

[54] Minh K Quan, Pubudu N Pathirana, Mayuri Wijayasundara, Sujeeva Setunge, Dinh C Nguyen, Christopher G Brinton, David J Love, and H Vincent Poor. 2025. Federated learning for cyber physical systems: a comprehensive survey. *IEEE Communications Surveys & Tutorials*.

[55] Ekdeep Singh Lubana, Chi Ian Tang, Fahim Kawsar, Robert P. Dick, and Akhil Mathur. 2022. Orchestra: unsupervised federated learning via globally consistent clustering. In *International Conference on Machine Learning (ICML)*, 14461–14484.

[56] Ye Lin Tun, Chu Myaet Thwal, Huy Q Le, Minh NH Nguyen, and Choong Seon Hong. 2024. Lw-fedssl: resource-efficient layer-wise federated self-supervised learning. *arXiv preprint arXiv:2401.11647*.