Comparative Study of Feature-Level Transmission Strategies for Bandwidth-Efficient Keyword Spotting

Aarul Dhawan
University of Illinois Urbana-Champaign
adhaw2@illinois.edu

Abhishek Ramakrishnan
University of Illinois Urbana-Champaign
arama9@illinois.edu

Abstract

Keyword spotting (KWS) on edge devices increasingly relies on split-inference pipelines where audio is captured on a constrained device and classified on a remote server. A central design question is *what to transmit*: raw audio, hand-crafted features, or a learned compressed representation. Each choice produces a different operating point across four dimensions simultaneously: network bandwidth, round-trip latency, server-side compute, and device-side compute. No single strategy is universally optimal.

In this paper we design, implement, and systematically evaluate five feature-level transmission strategies for a split-inference KWS pipeline: raw PCM audio (A1), fixed MFCC features (A2), event-triggered transmission using neural voice activity detection (A3), dynamically-resolved MFCCs (A4), and a jointly-trained neural encoder that transmits a compact learned embedding (A5). All five strategies are evaluated end-to-end over a real TCP network on two CloudLab nodes using the Google Speech Commands v2 dataset (35 classes, 105k utterances). We measure accuracy, Macro F1, average bytes per utterance, round-trip latency (RTT), and compute cost in MFLOPs on both the device and server.

Our results show that no single approach dominates across all deployment contexts. A1 is appropriate when the device has no compute budget and bandwidth is not a constraint. A2 offers a practical first optimization: strong bandwidth reduction at negligible device cost with no accuracy loss. A3 provides robustness in real environments with background noise and silence, suppressing unnecessary transmissions at the cost of VAD overhead. A4 demonstrates that the right idea (variable resolution) requires careful execution, as low-resolution MFCC destroys the temporal structure needed for accurate classification. A5 is best suited for network-bottlenecked deployments: it achieves the largest bandwidth and latency reductions at the cost of a more capable on-device encoder. These findings provide practical guidance for engineers selecting a transmission strategy based on their specific deployment constraints.

ACM Reference Format:
Aarul Dhawan and Abhishek Ramakrishnan. 2026. Comparative Study of Feature-Level Transmission Strategies for Bandwidth-Efficient Keyword Spotting. In *Proceedings of CS 537 Spring 2026 (CS 537).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The proliferation of voice-controlled IoT devices (smart speakers, industrial sensors, wearables, and embedded controllers) has created strong demand for efficient keyword spotting (KWS) pipelines. A KWS system must detect whether a specific spoken word appears in a continuous audio stream, often under tight constraints on power, memory, and network bandwidth.

A common deployment pattern is *split inference*: the device captures audio, performs some lightweight preprocessing, and transmits a compact representation to a server that runs the classifier. This architecture offloads computationally expensive neural network inference to the server while keeping the device simple and power-efficient. The central engineering question in this setting is *what to transmit*. The space of possible answers spans a wide spectrum: from raw bytes (simple to produce but expensive to send) to compact learned representations that require a neural encoder running on the device itself.

This spectrum creates a multi-dimensional tradeoff. More preprocessing on the device reduces bandwidth and latency but increases device compute. Less preprocessing reduces device cost but increases server compute and network usage. Critically, this is not a one-dimensional optimization: a deployment prioritizing battery life has different constraints than one prioritizing server throughput or response latency. Understanding these tradeoffs empirically, across a common benchmark, is the goal of this paper.

We implement all five strategies, deploy them on real networked hardware, and measure all four dimensions simultaneously. Our key contributions are:

- A systematic end-to-end comparison of five feature-level transmission strategies for split-inference KWS over a real TCP network.
- An evaluation framework measuring accuracy, bandwidth, latency, and compute cost (MFLOPs) on both device and server simultaneously.
- A characterization of the conditions under which each strategy is preferred, providing practical guidance for deployment decisions.

2 Related Work

On-device KWS. Zhang et al. [1] introduced compact CNNs for keyword spotting targeting microcontrollers, demonstrating competitive accuracy within 80 KB of memory. Tang and Lin [2] applied deep residual learning to KWS on Speech Commands. Both approaches run inference entirely on-device. Our work targets the complementary case where the device cannot run a full classifier.

Split inference. Kang et al. [5] introduced Neurosurgeon, which automatically partitions a DNN across device and server based on latency and energy models. Their work focuses on layer-by-layer splitting of vision models. Matsubara et al. [6] survey split computing for image classification and show that learned bottleneck

CS 537, UIUC
2026. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

representations consistently outperform raw activation transmission. Our work confirms this finding in the audio domain and provides a broader comparison that includes non-neural compression strategies.

Feature compression for speech. Classical speech codecs (AMR-WB, Opus) compress audio for transmission, optimizing perceptual quality rather than downstream task accuracy. Neural audio codecs [8] achieve lower bit rates but are also reconstruction-oriented. Our A5 encoder is task-specific and achieves far higher compression than any reconstruction codec at equivalent classification accuracy.

Voice Activity Detection. Silero VAD [4] is a compact pre-trained neural VAD model we use in A3. Energy-based VAD methods have been used for transmission gating in sensor networks [5], but neural VAD models provide substantially better false-negative rates in low-SNR conditions.

3 Problem Description

3.1 Problem Statement

We study the following problem: given a device that captures 1-second audio clips at 16 kHz, and a remote server that classifies them into one of 35 keyword classes, what feature-level representation should the device transmit to the server to optimize a given deployment objective?

The problem is parameterized by a deployment context that may prioritize any combination of: (1) minimizing bytes transmitted, (2) minimizing end-to-end latency, (3) minimizing server compute, (4) minimizing device compute, and (5) maximizing classification accuracy. Different deployment contexts call for different strategies, and no single operating point is globally optimal.

3.2 Assumptions

We operate under the following assumptions:

(1) **Fixed vocabulary.** The keyword set is known at training time and does not change at deployment. This enables task-specific learned compression (A5).
(2) **1-second clips.** Each audio clip is exactly 1 second at 16 kHz. Shorter clips are zero-padded. This matches the Speech Commands benchmark and simplifies fixed-size payload design.
(3) **Server availability.** The server is always reachable over a reliable TCP connection. We do not model packet loss or disconnection.
(4) **Synchronous processing.** Each clip is processed sequentially: the device transmits, waits for the server response, and then processes the next clip. Pipelining is not modeled.
(5) **Shared model weights.** The device and server have access to pre-trained model weights. Training is performed offline; we evaluate inference only.
(6) **Clean audio at training time.** Models are trained on the Speech Commands training split without additional data augmentation. Mixed-noise evaluation tests robustness to out-of-distribution conditions.

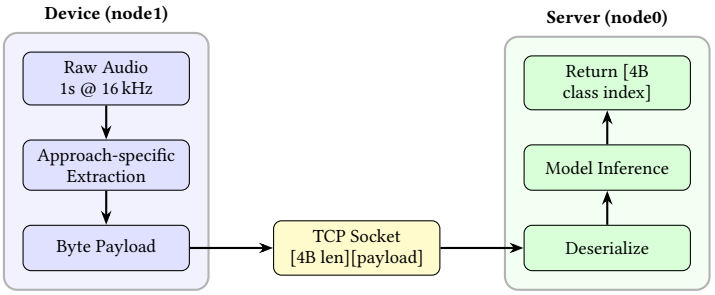

Figure 1: System architecture. The device extracts an approach-specific byte payload and transmits it to the server over TCP. The server deserializes, runs inference, and returns the predicted class index.

4 Methodology and System Design

4.1 System Architecture

Figure 1 illustrates the overall pipeline. A device node (node1) loads audio clips, applies approach-specific feature extraction, and transmits payloads to a server node (node0). The server deserializes each payload, runs the appropriate classifier, and returns a 4-byte predicted class index. RTT is measured on the device from the start of transmission to receipt of the response.

4.2 Transmission Protocol

The wire protocol is intentionally minimal. Every device-to-server message consists of a 4-byte big-endian length header followed immediately by the payload bytes. The server responds with a single 4-byte big-endian integer encoding the predicted class index. Each utterance uses a fresh TCP connection, isolating RTT measurements and avoiding head-of-line blocking artifacts.

4.3 The Five Approaches

A1: Raw Waveform. The device quantizes 16,000 float32 samples to int16 PCM and transmits the resulting 32,000 bytes without any feature extraction. On the server, the PCM bytes are reconstructed to float32, MFCCs are extracted using the same 40-coefficient configuration as A2, and the KWSModel CNN classifies the features. A1 represents the zero-device-compute baseline: the device is a pure audio forwarder, and all computation (feature extraction and classification) occurs on the server. This is the correct design choice when the client hardware has no floating-point unit or insufficient memory for MFCC computation. The cost is the highest bandwidth (32 KB per utterance) and the highest end-to-end latency of all approaches, since the transmission time for 32 KB dominates the RTT even on a gigabit LAN.

A2: Fixed MFCC. The device computes a 40×101 MFCC tensor using a 25 ms Hann-windowed STFT ($n_{\text{fft}} = 400$, $n_{\text{hop}} = 160$), a 40-band mel filterbank, and DCT projection to 40 coefficients. The tensor is serialized row-major as float32 and transmitted as 5,252 bytes. The server classifies directly with KWSModel. MFCC computation costs approximately 2.72 MFLOPs analytically:

$$\text{MFLOPs} = \frac{(5 \cdot n_{\text{fft}} \log_2 n_{\text{fft}} + \frac{n_{\text{fft}}}{2} \cdot n_{\text{mels}} + n_{\text{mels}} \cdot C) \cdot T}{10^6}$$

where $n_{\text{fft}} = 400$, $n_{\text{mels}} = 40$, $C = 40$, $T = 101$. A2 is the practical first step beyond A1: at the cost of a small amount of on-device compute, it achieves a 6.1× bandwidth reduction and eliminates server-side MFCC extraction. Since the KWSModel is trained on MFCC inputs, A2 is numerically equivalent to the server-side extraction in A1 (modulo int16 quantization noise), and matches or slightly exceeds A1's accuracy.

A3: Event-Triggered. A3 adds a transmission gate to the A2 pipeline. Before any feature extraction, the device runs Silero VAD on the raw waveform. If no 30 ms chunk reaches speech probability $\tau = 0.5$, the clip is classified as non-speech and *no payload is transmitted*; the device records the clip as skipped. When speech is detected, the device extracts 13-coefficient MFCCs ($n_{\text{mels}} = 40$, $C = 13$) and transmits ~4,959 bytes. The server classifies with KWSModel using the 13-coefficient input. We evaluate A3 in two modes: keywords-only (the standard 11,005 test clips) and mixed mode, where 426 background noise clips are appended to the test set. In keywords-only mode, skipped clips count as errors (the keyword was not classified); in mixed mode, skipped background clips count as correct rejections. The design rationale for A3 is that in many real deployments the audio stream contains substantial silence and background noise between spoken keywords; transmitting these clips wastes bandwidth and introduces classification errors.

A4: Dynamic MFCC. A4 tests whether variable temporal resolution, without any learned compression, can achieve a useful bandwidth-accuracy tradeoff. The device computes 40-coefficient MFCCs and then uniformly downsamples the time axis by a stride factor k, keeping every k-th frame:

$$T_k = \lceil T/k \rceil$$

Three configurations are evaluated: high ($k = 1$, $T_k = 101$, 16,168 bytes), medium ($k = 2$, $T_k = 51$, 4,088 bytes), and low ($k = 6$, $T_k = 17$, 1,360 bytes). Note that high resolution transmits a *larger* payload than A2 because it uses 40 coefficients per frame and float32 encoding without the 4-byte padding A2 omits; A2's 5,252 bytes is slightly smaller due to exact tensor shape differences. The server receives the downsampled MFCC and classifies with KWSModel, which was trained at full resolution; no resolution-specific model is used.

A5: Learned Embedding. A5 replaces hand-crafted compression with a jointly-trained CNN encoder that learns to compress MFCCs into a task-optimized D-dimensional embedding. The device first computes 40-coefficient MFCCs (same as A2), then passes the $(1, 40, 101)$ tensor through the encoder f_θ:

$$f_\theta : \text{Conv}_{1\to32} \to \text{BN} \to \text{ReLU} \to \text{MaxPool} \\ \to \text{Conv}_{32\to64} \to \text{BN} \to \text{ReLU} \to \text{AvgPool} \to \text{Linear}_{64\to D}$$

The output is a D-dimensional float32 vector transmitted as $D \times 4$ bytes. The server runs the classifier g_ϕ:

$$g_\phi : \text{Linear}_{D\to128} \to \text{ReLU} \to \text{Dropout}_{0.25} \to \text{Linear}_{128\to35}$$

Encoder f_θ and classifier g_ϕ are trained jointly end-to-end using cross-entropy on the Google Speech Commands training split, ensuring that the embedding space is optimized specifically for 35-class keyword discrimination. We evaluate $D \in \{16, 32, 64, 128\}$, transmitting 64 to 512 bytes, a 62.5× to 500× reduction over A1. The total device compute for A5 is 22.4 MFLOPs (MFCC + encoder), compared to 2.72 MFLOPs for A2 and 0 MFLOPs for A1.

5 Algorithms, Protocols, and Services

5.1 MFCC Extraction

Mel-Frequency Cepstral Coefficients are computed via torchaudio's `T.MFCC` transform: STFT with $n_{\text{fft}} = 400$ and $n_{\text{hop}} = 160$ samples, mapped through a 40-band mel filterbank, log-compressed, and DCT-projected to C coefficients. The resulting tensor has shape $(1, C, 101)$.

5.2 Silero VAD

Silero VAD is loaded via `torch.hub` and processes the raw 16,000-sample waveform. It returns a list of speech timestamps. A clip is declared speech-active if the list is non-empty. The threshold $\tau = 0.5$ (default) controls the speech probability above which a frame is counted as speech.

5.3 Baseline CNN Classifier (A1–A4)

The KWSModel CNN is used as the server-side classifier for approaches A1 through A4. The architecture consists of three convolutional blocks, each containing Conv2d → BatchNorm2d → ReLU → MaxPool2d, with channel widths 1 → 64 → 128 → 128. All convolutional kernels are 3×3 with padding 1, and max-pooling uses a 2×2 kernel. After the three blocks, AdaptiveAvgPool2d(1,1) collapses the spatial dimensions to a 128-dimensional vector, followed by a two-layer MLP head: Linear(128, 256) → ReLU → Dropout(0.25) → Linear(256, 35). The final linear layer produces raw logits over the 35 keyword classes.

Training. KWSModel is trained on the Google Speech Commands v2 training split (84,843 samples) using the Adam optimizer with an initial learning rate of 10^{-3}, a step decay schedule ($\gamma = 0.5$ every 10 epochs), and batch size 128. Training runs for 30 epochs on a CloudLab xl170 node. Input features are $(1, C, 101)$ MFCC tensors, normalized per-clip to zero mean and unit variance. The same model checkpoint is used for A1 (where the server extracts MFCCs from the received PCM), A2 (where MFCCs are extracted on-device), A3 (where 13-coefficient MFCCs are sent), and A4 (where temporally downsampled MFCCs are sent). No resolution-specific fine-tuning is performed for A4; the A4 degradation results therefore represent a strict lower bound on what a resolution-aware trained model could achieve.

MFLOPs. The KWSModel inference cost is measured with `thop` [7] on a $(1, 1, 40, 101)$ input tensor, yielding 114.67 MFLOPs. For A1, MFCC extraction on the server adds 2.72 MFLOPs, bringing total server cost to 117.39 MFLOPs. For A2–A4, the server only runs KWSModel (114.67 MFLOPs); MFCC extraction occurs on the device.

5.4 Encoder-Classifier Pipeline (A5)

The A5 encoder is a two-block CNN. Block 1: Conv2d(1, 32, 3×3) → BatchNorm2d → ReLU → MaxPool2d(2×2). Block 2: Conv2d(32, 64, 3×3) → BatchNorm2d → ReLU → AdaptiveAvgPool2d(1, 1). The pooled 64-dimensional feature is passed through a Linear(64, D)

projection with no activation, producing the D-dimensional embedding. The classifier is a small MLP: Linear$(D, 128) \rightarrow$ ReLU $\rightarrow$ Dropout$(0.25) \rightarrow$ Linear$(128, 35)$.

Joint training. The encoder and classifier are wrapped into a single `nn.Module` and optimized jointly using cross-entropy loss with the same optimizer and schedule as KWSModel (Adam, $lr = 10^{-3}$, step decay, 30 epochs). Joint training is critical: if the classifier were trained separately on embeddings produced by a frozen encoder, the embedding space would not be optimized for the classifier's decision boundaries. Four separate models are trained for $D \in \{16, 32, 64, 128\}$.

Deployment split. After training, the encoder and classifier are saved as separate `.pt` files. The device loads `encoder_D.pt` and runs the encoder on each MFCC tensor to produce the D-float embedding before transmission. The server loads `embedding_classifier_D.pt` and runs only the MLP on the received embedding. This allows the device to run a sub-22 MFLOP inference while the server performs sub-0.025 MFLOP classification, a 5,700× reduction in server compute relative to A1–A4.

5.5 Result Logging

For each test sample, the device records: true label, predicted label, payload size in bytes, RTT in seconds, and whether the sample was transmitted. These are stored as JSON in `results/{tag}.json` for offline evaluation with `evaluate.py`.

6 Validation

6.1 Testbed

All experiments run on two CloudLab xl170 nodes (Intel Xeon Silver 4114, Ubuntu 22.04, Python 3.10, PyTorch 2.5.1, torchaudio 2.5.1) connected over CloudLab's internal gigabit network. Node0 acts as the server, node1 as the device.

6.2 Dataset

We use Google Speech Commands v2 [3]: 84,843 training, 9,981 validation, and 11,005 test utterances across 35 keyword classes (directional, digit, and action words). All clips are zero-padded or trimmed to exactly 16,000 samples. For mixed-mode evaluation (A2 mixed, A3 mixed), 426 background noise clips from the Speech Commands archive are appended to the test set.

6.3 Experimental Setup

The server listens on `0.0.0.0:9999`. The device iterates the full test set sequentially, opening a new TCP connection per utterance. For A5, the encoder checkpoint is loaded on the device and runs inference locally before transmission. MFLOPs for neural network components are measured with `thop` [7]; MFCC FLOPs are computed analytically.

6.4 Evaluation Metrics

- **Accuracy**: Fraction of samples correctly classified. For A3, untransmitted keyword samples count as incorrect; untransmitted background clips count as correct.
- **Macro F1**: Unweighted average F1 across all classes, computed only on transmitted samples. Captures per-class performance independent of class imbalance.
- **Avg Bytes**: Mean payload size per utterance. For A3, untransmitted samples contribute 0 bytes.
- **Avg RTT**: Mean round-trip time in milliseconds, measured from start of `sendall()` to receipt of the 4-byte response. Untransmitted samples are excluded.
- **TX Rate**: Fraction of samples for which a payload was transmitted.
- **Device MFLOPs**: Floating-point operations performed on the device per utterance (MFCC extraction + any neural inference).
- **Server MFLOPs**: Floating-point operations performed on the server per utterance (deserialize is trivial; reported is model inference only).

6.5 Results

Table 1 presents the complete results. The three figures below provide visual summaries of the accuracy, bandwidth/latency, and compute dimensions respectively.

6.6 Discussion of Results

Accuracy and Macro F1 (Figure 2). A2 achieves the highest accuracy (0.851), marginally above A1 (0.842). This gap arises from the int16 quantization in A1: when the device encodes float32 samples to int16 PCM, the server reconstructs them as float32 with rounding error. Because KWSModel is trained on float32 MFCCs, this quantization noise introduces a small but consistent accuracy gap. The gap is small (≈1 point) because int16 still provides 16 bits of dynamic range, but it confirms that lossless feature transmission (A2) is preferable to raw-sample transmission (A1) even when both pipelines use the same underlying model.

A5 dim=128 matches A1 accuracy (0.842) while transmitting 512 bytes versus 32,000, a 62.5× bandwidth reduction. Figure 6 shows the accuracy-bandwidth tradeoff across all four embedding dimensions. The largest per-bit gain occurs between dim=16 (0.800) and dim=32 (0.823), adding 2.3 accuracy points for only 64 more bytes per transmission. Beyond dim=32, gains are diminishing: dim=64 adds 0.9 points, dim=128 adds another 1.0 point. RTT remains flat at 0.6 ms across all dimensions because even 512 bytes is negligible on a gigabit LAN. For deployments with strict bandwidth budgets, dim=32 offers an excellent operating point; for those willing to spend 512 bytes, dim=128 fully recovers A1-level accuracy.

A3 shows a notable split between accuracy and Macro F1. In keywords-only mode, accuracy (0.822) drops below A2 because 6% of keyword clips are classified as non-speech by Silero VAD and receive no prediction, each counted as an error. Yet Macro F1 (0.856), computed only on transmitted clips, is the highest of any approach. This confirms that when the VAD fires, the downstream KWSModel classifier performs well; the VAD's false-negative rate, not the classifier, is the accuracy bottleneck. In mixed mode, accuracy improves to 0.828 and Macro F1 to 0.860 because the VAD correctly gates out 9% of background noise clips that would otherwise be misclassified as keywords. This makes A3 the most robust approach in realistic deployment conditions with non-speech inputs.

Table 1: Full evaluation results on Google Speech Commands v2 test set (11,005 samples; mixed variants use ~11,431). TX Rate = fraction of samples transmitted. MFLOPs are per utterance.

	Approach	Accuracy	Macro F1	Avg Bytes	RTT (ms)	TX Rate	Dev MFLOPs	Srv MFLOPs
A1	Raw Waveform	0.8417	0.8311	32,000	7.7	1.00	0.000	117.387
A2	Fixed MFCC	0.8507	0.8357	5,252	4.2	1.00	2.720	114.667
	Fixed MFCC (mixed)	0.8210	0.8013	5,252	2.6	1.00	2.720	114.667
A3	Event-Triggered	0.8217	0.8559	4,959	6.2	0.94	2.611	114.667
	Event-Triggered (mix)	0.8279	0.8599	4,786	6.3	0.91	2.611	114.667
A4	High Resolution	0.8413	0.8306	16,168	6.9	1.00	2.720	114.667
	Medium Resolution	0.7404	0.7176	4,088	3.8	1.00	2.720	114.667
	Low Resolution	0.2259	0.1920	1,360	2.7	1.00	2.720	114.667
A5	Embed dim=16	0.7998	0.7808	64	0.6	1.00	22.380	0.007
	Embed dim=32	0.8234	0.8050	128	0.6	1.00	22.381	0.009
	Embed dim=64	0.8322	0.8176	256	0.6	1.00	22.383	0.013
	Embed dim=128	0.8419	0.8278	512	0.6	1.00	22.387	0.021

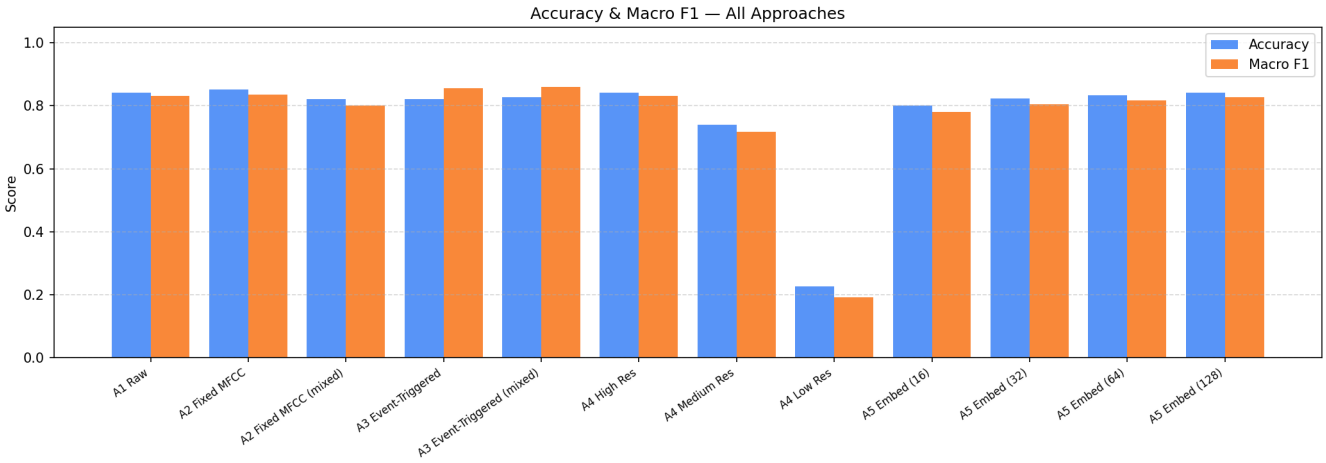

Figure 2: Accuracy (blue) and Macro F1 (orange) for all approaches. A2 achieves the highest raw accuracy (0.851). A5 dim=128 matches A1 (0.842) at a fraction of the bandwidth. A4 low resolution collapses to 0.226, showing that naive temporal downsampling is insufficient. A3 shows a Macro F1 advantage (0.856–0.860) because its VAD filters out the hardest samples.

A4 degrades sharply with resolution (Figure 5). High resolution (0.841) matches A1 but transmits 16,168 bytes, more than A2, offering no practical advantage. Medium resolution (0.740) drops 11 points, and low resolution (0.226) is near-random (35-class chance is 2.9%). The failure mode is structural: the KWSModel was trained on 101-frame inputs. When only 17 frames are provided (low resolution), the convolutional receptive fields that span multiple frames see mostly padding, and the temporal patterns (formant transitions, plosive releases) that distinguish keywords are destroyed. These results strongly suggest that A4-style temporal subsampling requires resolution-aware training; a model trained jointly on the target frame count would not suffer this degradation.

Bandwidth and Latency (Figure 3). The bandwidth panel spans three orders of magnitude: from A5 dim=16 (64 bytes) to A1 (32,000 bytes). The non-embedding methods cluster between 4,088 and 32,000 bytes. Within that cluster, A2 (5,252 bytes) is the best operating point: it beats A4 high resolution by 3× while being numerically equivalent on accuracy. A3 keywords-only (4,959 bytes) is smaller than A2 because it uses 13 MFCC coefficients instead of 40, and A3 mixed (4,786 bytes) is smaller still due to the 9% skip rate.

RTT closely tracks payload size, confirming that the network transfer time dominates over compute on the server. A5's 0.6 ms RTT reflects that even the largest embedding (512 bytes) crosses a gigabit LAN in well under 1 ms, so network transit is negligible and the measured RTT represents mostly kernel overhead and serialization. A1's 7.7 ms RTT includes approximately 3 ms of transmission time for the 32 KB payload plus server-side MFCC extraction and CNN inference. A3's RTT (6.2 ms keywords-only) is notably higher than A2 (4.2 ms) despite a similar payload, because Silero VAD runs

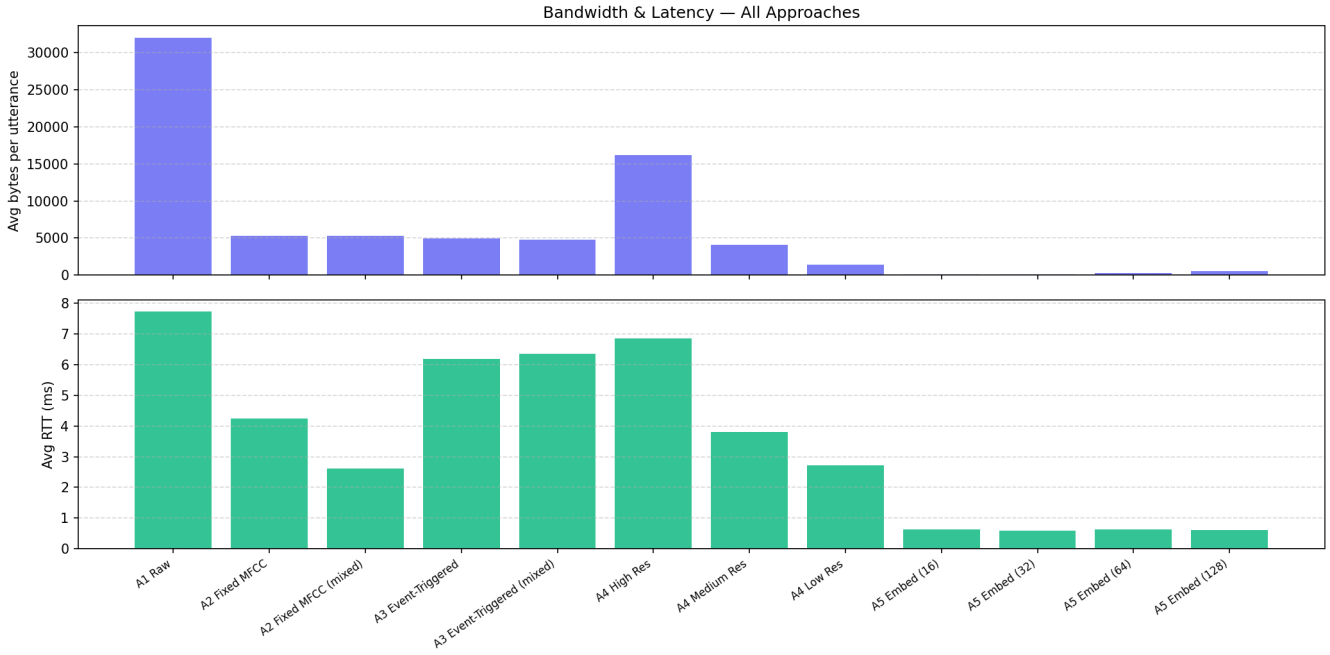

Figure 3: Average bytes per utterance (top) and average RTT in milliseconds (bottom). Payload size is the dominant driver of RTT. A5 achieves 0.6 ms RTT across all embedding dimensions, a 12.8× improvement over A1 (7.7 ms). A3 reduces average bytes below A2 by suppressing silent clips.

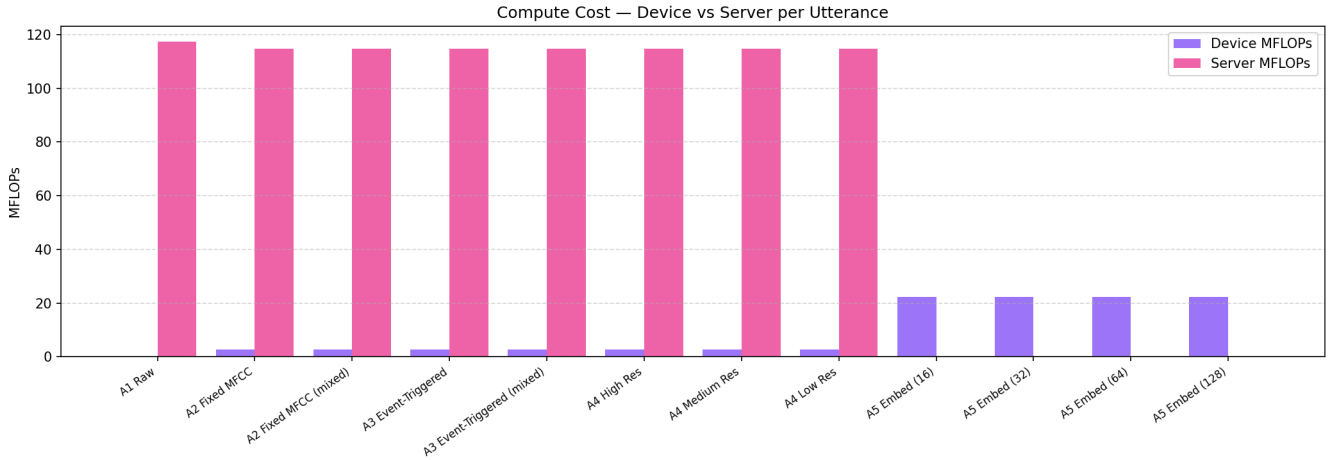

Figure 4: Device MFLOPs (purple) and server MFLOPs (pink) per utterance. A1–A4 concentrate compute on the server (114–117 MFLOPs). A5 shifts compute to the device (22.4 MFLOPs) while reducing server cost to under 0.025 MFLOPs, a reduction of over 5,000×.

on the device *before* transmission, adding ~2 ms of pre-transmission overhead. This overhead is incurred on every clip, including those that are ultimately not transmitted.

The RTT advantage of A5 is most relevant for latency-sensitive deployments such as real-time voice assistants, where the 0.6 ms A5 response enables sub-millisecond keyword confirmation. For batch-mode transcription or offline KWS pipelines, RTT may be less critical, and the simpler A2 is preferable.

Compute Cost (Figure 4). The compute breakdown reveals a clean three-tier structure. A1 places all compute on the server (117.4 MFLOPs: 2.72 for MFCC + 114.67 for CNN). A2, A3, and A4 shift MFCC extraction to the device (2.72 MFLOPs each) while the

Figure 5: A4 resolution sweep: accuracy and Macro F1 as a function of temporal downsampling factor k. High resolution ($k = 1$, 16,168 bytes) matches A1 but offers no bandwidth advantage over A2. Medium resolution ($k = 2$) drops accuracy by 11 points. Low resolution ($k = 6$, 1,360 bytes) collapses to near-random performance (0.226), demonstrating that uniform temporal downsampling without resolution-aware training is not a viable compression strategy.

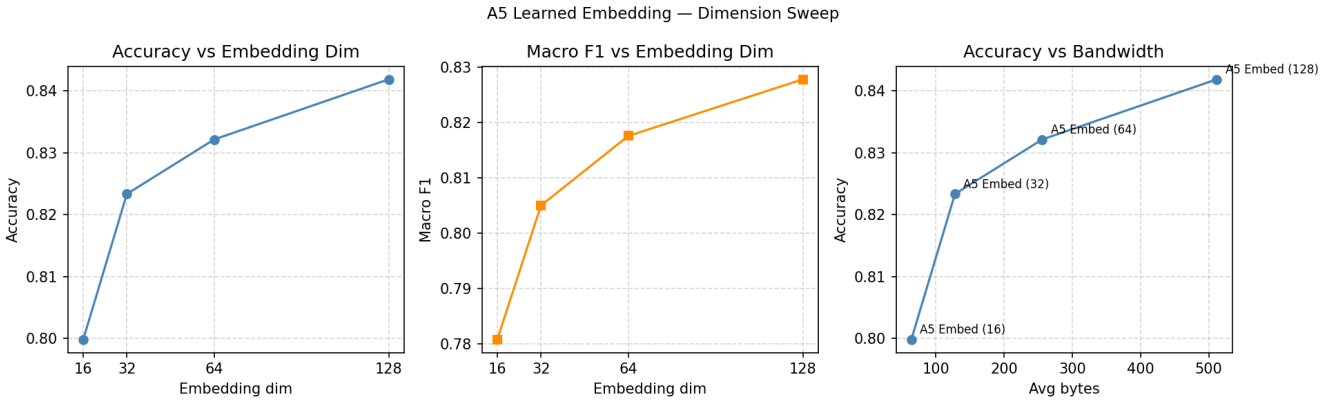

Figure 6: A5 embedding dimension sweep: accuracy, bytes per utterance, and RTT as a function of embedding dimension $D \in \{16, 32, 64, 128\}$. The accuracy-bandwidth tradeoff shows strong diminishing returns above dim=32. RTT remains flat at 0.6 ms across all dimensions because even 512 bytes is negligible on a gigabit LAN. The dim=32 operating point achieves 0.823 accuracy at 128 bytes, a compelling balance for bandwidth-constrained deployments.

server still runs the full CNN (114.67 MFLOPs). Total compute for A1–A4 is approximately 117 MFLOPs per utterance regardless of which device-side feature is used; the choice affects only the split between device and server.

A5 fundamentally changes this picture. The device encoder costs 22.38–22.39 MFLOPs depending on D (the linear projection $\mathbb{R}^{64} \to \mathbb{R}^{D}$ is negligible), while the server classifier costs only 0.007–0.021 MFLOPs. Total pipeline compute for A5 is ≈22.4 MFLOPs, a 5.2× reduction in system-wide compute compared to A1–A4. This reduction occurs because the encoder is architecturally much smaller than KWSModel (two Conv blocks versus three, smaller channel widths, no MLP head).

In a multi-tenant server deployment serving N simultaneous device clients, the server must process $N \times 114.67$ MFLOPs per second for A1–A4 versus $N \times 0.021$ MFLOPs for A5 dim=128. This 5,460× reduction in per-utterance server compute translates directly to proportionally higher server throughput, making A5 strongly preferred when the server is the system bottleneck. The tradeoff is that the device must be capable of running the 22.4 MFLOP encoder, which is feasible on modern application processors but challenging on bare microcontrollers.

Accuracy-Bandwidth Pareto Analysis. Across all approaches, the Pareto-optimal frontier for accuracy versus bytes per utterance consists of A2, A5 dim=32, A5 dim=64, and A5 dim=128. A1 is dominated by A2 on both accuracy and bandwidth. A4 high resolution is dominated by A2 on bandwidth (and offers no accuracy gain). A4 medium and low resolution are dominated by A5 on both accuracy and bandwidth. A3 is on the Pareto frontier only for deployments where Macro F1 on transmitted samples, rather than overall accuracy, is the target metric, or where the input mix includes substantial non-speech content. This analysis confirms that

for pure accuracy-bandwidth optimization, learned embeddings (A5) or fixed MFCCs (A2) are the dominant strategies.

7 Conclusion

This paper presents a systematic comparison of five feature-level transmission strategies for split-inference keyword spotting, evaluating accuracy, bandwidth, latency, and compute cost across both device and server. Rather than identifying a single winner, our results show that each approach occupies a distinct and defensible niche depending on deployment constraints.

A1 (Raw Waveform) is the right choice when the device has no compute budget whatsoever. It requires zero on-device computation, and server-side MFCCs can be extracted from the raw PCM without loss. The tradeoff is the highest bandwidth (32 KB) and latency (7.7 ms) of any approach.

A2 (Fixed MFCC) is the most practical general-purpose baseline. It achieves the highest classification accuracy of any approach (0.851) with only 2.72 MFLOPs of device compute and 6× less bandwidth than A1. For deployments without a network bottleneck constraint, A2 is the optimal starting point.

A3 (Event-Triggered) is the right choice for realistic environments with significant background noise or silence. Its VAD gating suppresses unnecessary transmissions entirely, and its Macro F1 (0.860 in mixed mode) is the highest of any approach on transmitted samples. The extra device cost (VAD inference) is worth paying when the deployment environment is not all-speech.

A4 (Dynamic MFCC) demonstrates that the concept of variable-resolution feature transmission is sound, but the implementation via uniform temporal downsampling is too blunt. At high resolution it merely adds bandwidth without benefit; at low resolution it destroys the temporal structure required for accurate classification. A more principled resolution selection strategy, adaptive to audio complexity, could make A4 competitive.

A5 (Learned Embedding) is the right choice when the network is the bottleneck. It achieves the lowest bytes (64–512), the lowest RTT (0.6 ms), and the lowest server compute (<0.025 MFLOPs) of any approach, at the cost of a 22.4 MFLOPs on-device encoder. In bandwidth-constrained or multi-tenant server deployments, these reductions justify the increased device complexity.

In summary, the choice of transmission strategy should be driven by the specific constraints of the deployment: device compute budget, available bandwidth, server throughput requirements, and the expected proportion of non-speech inputs. Each of the five strategies presented here represents a coherent and well-motivated point in this design space.

7.1 Limitations

Several aspects of our evaluation represent simplifications relative to real deployments. First, our testbed uses a dedicated gigabit LAN between two CloudLab nodes. Real IoT deployments use Wi-Fi, LTE, or LoRa links with substantially higher and more variable latency; RTT differences between approaches would be amplified on such links, further favoring low-bandwidth strategies like A5. Second, the synchronous one-utterance-per-connection model does not capture pipelining or connection reuse. A persistent TCP connection would reduce connection setup overhead uniformly across approaches, but would also make head-of-line blocking visible for A1's large payloads. Third, we evaluate only clean-speech training; none of the models are fine-tuned on noisy or augmented data. A noise-augmented KWSModel or encoder would likely improve A3 and A5 accuracy in mixed conditions. Finally, our MFLOPs measurements for the A5 encoder exclude the preceding MFCC extraction step in the figure comparison; the full on-device cost is 2.72 (MFCC) + 19.66 (encoder) = 22.38 MFLOPs, which we report correctly in the table but which should be considered holistically when comparing device energy budgets.

7.2 Future Work

Several directions follow naturally from this study. **Resolution-aware training for A4**: Training KWSModel directly on temporally downsampled MFCCs would likely recover much of the accuracy lost at medium and low resolution, potentially making A4 a competitive operating point at 4 KB with no encoder overhead. **Adaptive transmission for A5**: The current A5 uses a fixed embedding dimension. An adaptive scheme that selects D per utterance based on a confidence score from a lightweight first-pass classifier could further reduce average bandwidth while maintaining accuracy on difficult samples. **VAD threshold tuning for A3**: We use the default Silero threshold $\tau = 0.5$. Optimizing τ on the validation set to maximize the accuracy-bandwidth tradeoff could reduce the 6% keyword false-negative rate. **Energy measurement**: Future work should measure actual joules consumed on the device, not just MFLOPs, as memory access patterns and hardware utilization affect real energy costs.

AI Assistance Acknowledgement

The system architecture, experimental design, choice of approaches, and analysis presented in this paper are our own work. We used Claude (Anthropic) as a coding assistant during implementation. Specifically, AI assistance was used to help implement the PyTorch training loop and model checkpointing, and to debug the Silero VAD integration and mixed-mode evaluation. All design decisions, including the selection of the five approaches, the evaluation methodology, the mixed-mode test design, and the interpretation of results, were made by the authors. The writing in this paper is entirely our own.

References

[1] Y. Zhang, N. Suda, L. Lai, and V. Chandra. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128*, 2017.

[2] R. Tang and J. Lin. Deep residual learning for small-footprint keyword spotting. In *Proc. ICASSP*, pages 5479–5483, 2018.

[3] P. Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[4] Silero Team. Silero VAD: pre-trained enterprise-grade voice activity detector. https://github.com/snakers4/silero-vad, 2021.

[5] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proc. ASPLOS*, pages 615–629, 2017.

[6] Y. Matsubara, M. Levorato, and F. Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys*, 55(5):1–38, 2022.

[7] L. Zhu. `thop`: PyTorch-OpCounter. https://github.com/Lyken17/pytorch-OpCounter, 2019.

[8] N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, and M. Tagliasacchi. SoundStream: An end-to-end neural audio codec. *IEEE/ACM Trans. Audio, Speech, Language Process.*, 30:495–507, 2022.