Adaptive Residual Quantization for Memory-Efficient Temporal Action Segmentation

Gerard Donahue[1,2†] **Guven Gergerli**[1,3†] **Ayush Gupta**[1,4†]
Reza Ghoddoosian[1] **Enna Sachdeva**[1] **Faizan Siddiqui**[1]
[1]Honda Research Institute USA [2]Northeastern University
[3]Purdue University [4]Johns Hopkins University
donahue.g@northeastern.edu, faizan_siddiqui@honda-ri.com

Abstract

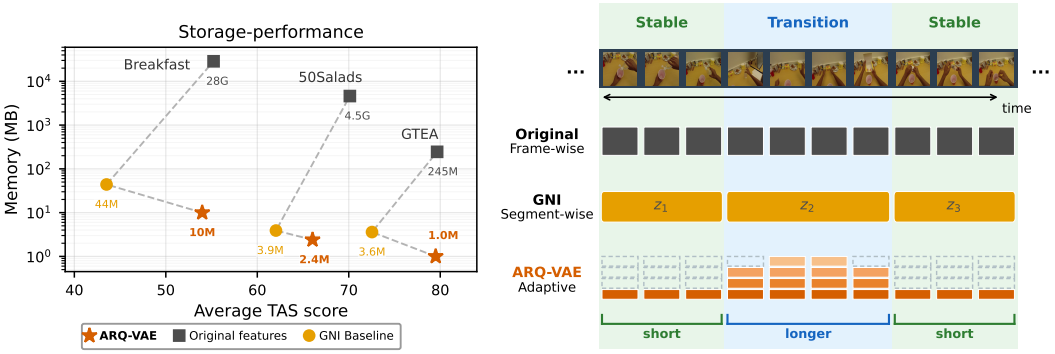
Temporal action segmentation (TAS) is commonly trained on large banks of pre-extracted frame features, but storing and replaying those features becomes a practical bottleneck for long-form video, especially in incremental TAS where past-task data must remain available to mitigate catastrophic forgetting. Recent TAS replay and condensation methods reduce this burden, but they rely on dense frame-wise annotations and allocate storage through fixed segment-level structures, limiting their flexibility for long procedural videos whose frame complexity varies substantially over time. We introduce Adaptive Residual-Quantized Variational AutoEncoder (ARQ-VAE), a compact and label-free memory representation for TAS features that learns a discrete residual quantization of frame-level video features and then adaptively truncates the residual chain on a per-frame basis, retaining only the residual prefix that best reconstructs each frame feature. This yields a practical compact feature memory that does not depend on frame-wise labels and can therefore support multiple TAS settings with the same stored representation, while a lightweight fine-tuning stage further improves reconstruction quality for the truncated codes that are actually retained. Across standard TAS benchmarks, ARQ-VAE achieves a substantially improved storage–performance trade-off over prior replay and condensation baselines. On Breakfast, our final representation reduces training-set storage from 28 GB for the original I3D feature bank to 10 MB while retaining strong downstream segmentation performance; with ASFormer, it achieves 71.5 Edit and 51.2 F1@50 in the supervised setting. We further show that the same label-free memory remains effective for incremental and unsupervised TAS. We position ARQ-VAE as a practical memory representation for TAS and a promising direction for broader dataset condensation applications.

1 Introduction

Temporal action segmentation (TAS) seeks to assign an action label to every frame in a long untrimmed video, recovering both the temporal extent and ordering of actions over time [8, 41, 14, 23]. In practice, modern TAS systems are rarely trained directly on raw RGB video. Instead, they typically rely on pre-extracted frame features from pretrained video backbones (e.g., I3D [3], DINOv3 [32], VideoMAE [34]). These features provide rich visual-temporal representations and make TAS training more practical, but they also introduce a substantial storage burden: long procedural videos must be stored, loaded, and processed densely at the frame level [3, 8, 41]. On benchmarks such as Breakfast and 50Salads, where videos are long and training is frame-wise, feature storage is therefore not merely an implementation detail but a practical bottleneck [19, 33].

[†]Work done during an internship at Honda Research Institute USA.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Memory-performance trade-off across datasets.

(b) Adaptive memory of ARQ-VAE.

Figure 1: Storage efficiency across TAS settings. ARQ-VAE achieves a better storage–performance trade-off while growing memory more slowly than feature replay and GNI.

This bottleneck is particularly sharp in incremental temporal action segmentation (iTAS), where tasks arrive sequentially and models must learn new actions without catastrophically forgetting previously learned ones [6, 18, 17, 10]. Replay-based methods are a natural solution because they recall past task data while learning from new task data [6, 13, 28, 22, 12, 37]. However, replay in TAS is expensive: past videos are long, frame-level features are dense, and memory usage grows with each new task. Thus, a compact feature memory is central to whether replay remains practical at all.

Recent TAS-specific methods have begun to address this challenge. TCA [6] reconstructs past features with a conditional generative model trained using action labels and temporal coherence supervision, while GNI [7] stores optimized latent representations that are decoded through a similar label-conditioned generator. Although these methods show that compact replay is effective for TAS, their memory design has two important limitations. First, they require dense frame-wise supervision, restricting them to costly annotation-rich settings. Second, they store coarse action-segment-level latent exemplars, imposing a rigid allocation of memory across time. This is a poor fit for long procedural videos: frame complexity is highly non-uniform, action transitions are often harder than stable interiors, and a useful memory should spend more capacity where the feature sequence is harder to preserve [19, 33, 47, 31].

We propose a compact feature memory for TAS based on residual quantization (RQ). RQ represents each frame feature as a sequence of discrete residual codes whose cumulative contribution reconstructs the original continuous feature [35, 21]. This gives each frame a chain of discrete residual codes that can be stored as a sequence of integer indices. RQ is attractive for TAS because it provides explicit, compact, supervision-agnostic storage. However, standard RQ is fixed-rate: every frame stores the same number of codes, regardless of each frame's reconstruction difficulty.

Our key idea is to make residual quantization adaptive at the frame level. After training a standard RQ variational autoencoder (RQ-VAE) [21], we decode prefixes of each frame's residual code chain and store an adaptive prefix based on reconstruction behavior. Unlike fixed-rate or segment-level methods that impose a rigid storage allocation across time, **Adaptive Residual-Quantized Variational AutoEncoder (ARQ-VAE)** treats reconstruction difficulty as frame-dependent. Fig. 1b compares our adaptive approach with fixed-rate approaches that operate at the frame and segment levels. ARQ-VAE retains longer residual prefixes for hard-to-reconstruct frames, such as action transitions or rare visual configurations, and shorter prefixes for redundant frames from stable action interiors. This produces a compact, label-free feature memory whose capacity adapts to learned per-frame reconstruction complexity. We further introduce a lightweight fine-tuning stage that aligns the decoder and active codebooks with the truncated representations actually stored.

Empirically, as shown in Fig. 1a, ARQ-VAE improves the storage–performance trade-off across supervised, incremental, and unsupervised TAS settings. On the Breakfast dataset [19], ARQ-VAE compresses the original 28 GB I3D feature bank to 10 MB while maintaining strong downstream segmentation performance with standard TAS backbones. In replay-based iTAS, ARQ-VAE also improves over prior compact baselines, and the same representation generalizes to unsupervised TAS.

In summary, this paper makes three core contributions: First, we introduce a practical compact memory representation for TAS based on the label-free residual quantization of frame features. Second, we propose ARQ-VAE, which adaptively allocates residual depth on a per-frame basis and improves the storage–performance trade-off relative to fixed-rate quantization. Third, we demonstrate that this representation is effective across supervised, incremental, and unsupervised TAS. Together, these results position adaptive residual quantization as a capable and versatile component for memory-efficient temporal action segmentation.

2 Related Work

Temporal action segmentation. Temporal action segmentation (TAS) has been studied under supervised, weakly supervised, unsupervised, and incremental settings. Representative supervised models include MS-TCN, ASRF, ASFormer, and FACT [8, 14, 41, 23]; weaker supervision has been explored through timestamps and activity-level labels [30, 5], and unsupervised segmentation through learned representations and clustering [20]. Standard benchmarks include the Breakfast, 50Salads, and GTEA datasets [19, 33, 9]. More recently, incremental TAS has begun to study sequential task learning under replay constraints. TCA introduces label-conditioned replay for incremental action segmentation [6], and GNI studies the storage–performance trade-off through latent dataset condensation [7]. This incremental setting makes memory efficiency especially important, motivating TAS condensation methods that reduce storage while preserving downstream segmentation performance.

Dataset condensation and feature-space replay. Dataset distillation and condensation methods reduce memory cost by learning a small synthetic set that substitutes for the original training data [38, 44, 40]. A more closely related line of work compresses the training signal in representation space rather than replaying raw inputs, for example, by storing feature descriptors [13], replaying intermediate activations [28], synthesizing classifier-side features [22], or quantizing hidden representations for compressed replay [12, 37]. Recent work has extended these ideas to long-video settings through latent replay with foundation models and compressed code replay [27, 25]. Our setting differs from both synthetic-set condensation and small-buffer replay: we retain the full training set and reduce memory by replacing each continuous feature vector with a compact discrete code.

Vector quantization for feature compression. Vector quantization [26] is a natural mechanism for compressing continuous representations into compact discrete codes. Learned discrete-representation models such as VQ-VAE [35] and RQ-VAE [21] show that high-dimensional features can be encoded as short code sequences while retaining useful structure. In audio coding, VRVQ [4] varies the residual depth per frame by training an allocator jointly with the codec; ARQ-VAE instead selects each frame's depth post hoc from a standard trained RQ-VAE. Recent work has expanded this framework with alternative quantization schemes and visual tokenizers, including finite scalar quantization, lookup-free quantization, joint image-video tokenization, and binary spherical quantization [26, 42, 36, 45], as well as improved methods for learning and updating codebooks [46, 29]. Classical compact-coding methods such as product, additive, and composite quantization are also related because they target efficient coding of continuous vectors for storage and retrieval [15, 2, 43]. In continual learning, quantization has additionally been used for prototype augmentation and prompt selection [24, 16]. In contrast, we use residual quantization specifically as a storage mechanism for fixed pre-extracted video features.

3 Method

3.1 Problem Setup

Compact Feature Memory for Temporal Action Segmentation. Temporal action segmentation (TAS) maps a sequence of pre-extracted frame features $X = \{x_t\}_{t=1}^{T}$, where $x_t \in \mathbb{R}^D$, to a framewise label sequence $Y = \{y_t\}_{t=1}^{T}$ with $y_t \in \{1, \dots, C\}$, where C is the number of action classes. We consider the standard feature-based TAS setting used by modern segmentation models, where training and inference operate on dense frame features rather than raw RGB video [8, 41, 23]. Let the TAS training set be $\mathcal{D} = \{X_i\}_{i=1}^{N}$, where each training video i is a sequence of T_i frame features $X_i = \{x_t\}_{t=1}^{T_i}$.

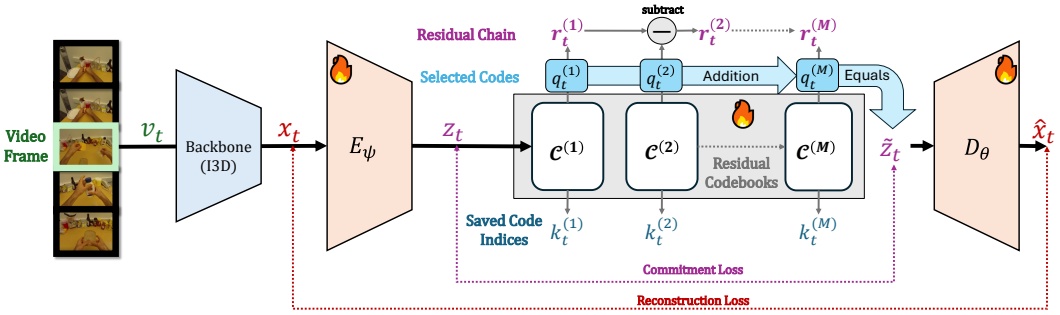


Figure 2: **Residual-Quantized Variational AutoEncoder (RQ-VAE) for compact TAS feature memory.** A pretrained video backbone extracts a frame feature x_t from video frame v_t. The encoder E_ψ maps x_t into latent space, where residual quantization selects a sequence of discrete indices $k_t^{(m)}$ and code vectors $q_t^{(m)}$ from codebooks $\mathcal{C}^{(1)}, \ldots, \mathcal{C}^{(M)}$. These stage-wise codes are summed to form the quantized latent $\tilde{z}_t = \sum_{m=1}^{M} q_t^{(m)}$, which is decoded by D_θ to reconstruct $\hat{x}_t$. The stored representation is the sequence of discrete stage indices.

Our goal is to replace the original feature training set with a compact stored representation $\hat{\mathcal{D}}$ whose storage cost is significantly smaller than that of $\mathcal{D}$ while simultaneously preserving the information content from $\mathcal{D}$ needed for downstream TAS training. A TAS model is then trained on reconstructed features from $\hat{\mathcal{D}}$ and evaluated on real test features at inference time. The downstream TAS architecture and loss remain unchanged; only the stored training bank is replaced.

This formulation is motivated most notably by incremental TAS, where prior-task features must remain available for replay [6]. At the same time, the memory itself should not depend on framewise labels. Since TAS is also studied under weakly supervised and unsupervised regimes [30, 5, 39], we seek a *label-free* feature memory that can be reused across training settings rather than one tied to costly frame-wise TAS supervision [7].

3.2 Residual Quantization for Compact TAS Feature Memory

We first review the mechanics of residual-quantized variational autoencoders (RQ-VAE). The core idea of Residual Vector Quantization (RVQ) is to compress high-dimensional continuous vectors by representing them as a cumulative sum of discrete codes. We visualize RQ-VAE in Fig. 2.

Given a dense frame feature x_t extracted from a video backbone (e.g., I3D [3] or DINOv3 [32]), an encoder E_ψ maps it to a continuous latent vector $z_t = E_\psi(x_t) \in \mathbb{R}^d$. To quantize this vector, RVQ applies a sequence of M refinement steps, which we refer to as *stages*. Each stage $m \in \{1, \ldots, M\}$ has a learnable *codebook* $\mathcal{C}^{(m)} \in \mathbb{R}^{K \times d}$, consisting of K embedding vectors $e_k^{(m)} \in \mathbb{R}^d$. Here, $e_k^{(m)}$ denotes the kth code vector of stage m.

The RVQ process operates iteratively. The first stage ($m = 1$) is a coarse discrete approximation of the original latent vector z_t. Each subsequent stage then quantizes the error (or "residual") left over by the previous stage. Specifically, starting with the initial target $r_t^{(1)} = z_t$, each stage m selects the nearest codebook entry to the current residual and computes the new residual for the next stage:

$$k_t^{(m)} = \underset{k \in \{1,\ldots,K\}}{\arg\min} \left\| r_t^{(m)} - e_k^{(m)} \right\|_2^2, \qquad q_t^{(m)} = e_{k_t^{(m)}}^{(m)}, \qquad r_t^{(m+1)} = r_t^{(m)} - q_t^{(m)}.$$

The final quantized latent $\tilde{z}_t = \tilde{z}_t^{(M)}$ is the cumulative sum of these selected codes, $\tilde{z}_t = \sum_{m=1}^{M} q_t^{(m)}$, which a decoder D_θ then maps back to the reconstructed feature $\hat{x}_t = D_\theta(\tilde{z}_t)$. While utilizing a single stage ($M = 1$) reduces this architecture to a standard VQ-VAE [35], the multi-stage residual form provides the higher representational capacity necessary to capture the complex visual-temporal dynamics of long-video features [21].

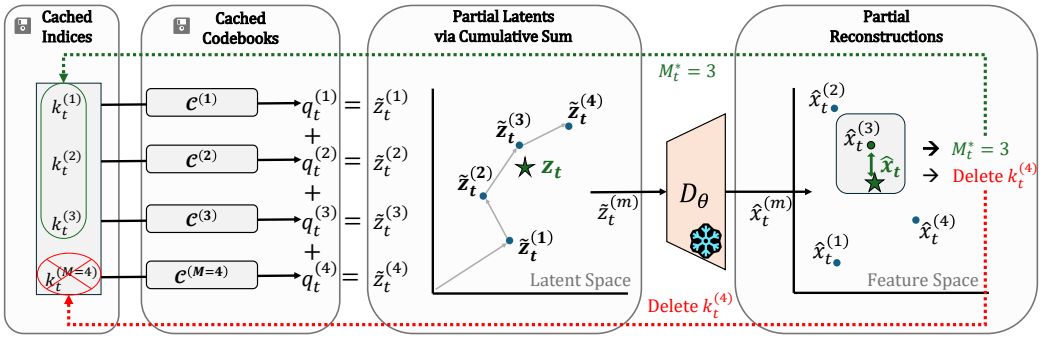


Figure 3: **Adaptive prefix selection in ARQ-VAE.** After training a fixed-depth RQ-VAE, we decode every prefix of the residual chain. For frame t, the partial latents $\tilde{z}_t^{(m)}$ yield partial reconstructions $\hat{x}_t^{(m)}$. ARQ-VAE retains the prefix length m_t^* with the lowest reconstruction error (shown in green) and discards the remaining later-stage indices (shown in red). The resulting memory therefore stores a variable number of stage indices per frame.

As the first step of our pipeline, we train this RQ-VAE at its full, fixed depth M using the standard straight-through estimator [35, 21]:

$$\mathcal{L}_{\text{RQ}} = \underbrace{\sum_t \|x_t - \hat{x}_t\|_2^2}_{\text{Reconstruction}} + \underbrace{\sum_t \sum_{m=1}^{M} \left(\left\| \text{sg}[r_t^{(m)}] - q_t^{(m)} \right\|_2^2 + \beta \left\| r_t^{(m)} - \text{sg}[q_t^{(m)}] \right\|_2^2 \right)}_{\text{Multi-Stage Commitment}}, \tag{1}$$

where $\text{sg}[\cdot]$ denotes the stop-gradient operation. Intuitively, the multi-stage commitment loss keeps each selected code vector $q_t^{(m)}$ close to the residual $r_t^{(m)}$ it is meant to represent, while also encouraging the encoder residuals to stay close to the selected discrete codes. The stop-gradient operator separates these two updates: the first term updates the selected codes toward the residuals, whereas the second term updates the encoder output to commit to the chosen codes. This separation stabilizes training and helps avoid index collapse, where only a small subset of codebook entries are repeatedly selected [21, 11]. This corresponds directly to the Commitment Loss depicted in Fig. 2.

For TAS, RQ-VAE provides a compact, label-free memory by storing only M discrete indices $k_t^{(m)}$ for each frame. However, the standard formulation is strictly *fixed-rate*: every frame utilizes all M stages, regardless of how much capacity individual frames require in practice. We therefore use full-depth RQ-VAE as an initial step and convert it into an adaptive-rate memory described next.

3.3 Adaptive Residual-Quantized Variational AutoEncoder

To overcome the limitations of fixed-rate residual quantization, we convert the standard RQ-VAE into an adaptive-rate feature memory by selecting, for each frame, the residual-code prefix that minimizes reconstruction error. As illustrated in Fig. 3, after training the full-depth RQ-VAE, we evaluate every possible *prefix* of the residual chain for a given frame. We define a prefix of length m as the contiguous sequence of residual codes from the first stage up to stage m. For a prefix of length m, the *partial latent* and *partial reconstruction* are defined as

$$\tilde{z}_t^{(m)} = \sum_{i=1}^{m} q_t^{(i)}, \quad \hat{x}_t^{(m)} = D_\theta(\tilde{z}_t^{(m)}).$$

We then select the optimal retained depth for each frame, m_t^*, as the prefix length whose partial reconstruction minimizes the L2 error with respect to the original frame feature x_t:

$$m_t^* = \underset{m \in \{1,\ldots,M\}}{\arg\min} \left\| x_t - \hat{x}_t^{(m)} \right\|_2^2. \tag{2}$$

Consequently, the stored memory for frame t becomes the truncated index sequence $(k_t^{(1)}, \ldots, k_t^{(m_t^*)})$ rather than the full length-M chain of indices $(k_t^{(1)}, \ldots, k_t^{(M)})$.

Table 1: **Temporal Action Segmentation results** on three TAS benchmark datasets.

	GTEA [9]				50Salads				Breakfast			
	Acc	Edit	F1@{10,25,50}	Storage	Acc	Edit	F1@{10,25,50}	Storage	Acc	Edit	F1@{10,25,50}	Storage
MS-TCN [8]												
Original	75.8	81.2	86.4 / 83.5 / 71.4	245 MB	79.3	65.3	73.2 / 70.9 / 61.9	4.5 GB	66.5	62.8	55.7 / 51.0 / 40.1	28 GB
Mean	71.2	73.3	77.1 / 73.7 / 59.4	7.2 MB	69.0	42.7	50.0 / 46.1 / 37.4	7.8 MB	47.6	31.8	27.8 / 23.3 / 15.6	96 MB
Coreset	66.7	66.1	72.4 / 68.9 / 53.2	7.2 MB	61.7	43.3	49.9 / 46.3 / 35.4	7.8 MB	49.7	36.8	32.3 / 27.5 / 19.3	96 MB
TCA [6]	60.9	54.1	59.2 / 55.3 / 39.3	–	56.4	33.6	39.8 / 35.8 / 25.9	–	34.2	20.7	17.9 / 13.8 / 8.4	–
Encoded	70.4	65.5	72.2 / 68.8 / 52.1	3.6 MB	69.0	43.6	50.6 / 46.0 / 37.4	3.9 MB	37.9	49.8	40.0 / 32.8 / 19.4	44 MB
VQ-VAE	61.5	75.1	72.0 / 66.2 / 52.5	0.1 MB	70.4	56.0	59.3 / 56.4 / 45.6	0.6 MB	49.8	48.6	40.3 / 34.0 / 22.6	4 MB
GNI [7]	**75.2**	71.9	78.3 / 74.6 / 62.7	3.6 MB	74.4	59.5	65.1 / 61.0 / 50.2	3.9 MB	55.5	45.6	46.7 / 41.1 / 28.7	44 MB
RQ-VAE	73.5	80.6	84.6 / 82.2 / 69.1	1.1 MB	72.2	58.6	64.6 / 61.5 / 52.3	4.1 MB	58.6	57.4	51.8 / 45.5 / 33.6	22 MB
ARQ-VAE	73.5	81.0	84.1 / 81.0 / 68.4	1.0 MB	73.9	60.6	64.7 / 61.6 / 52.4	2.4 MB	61.8	59.4	53.8 / 47.9 / 35.8	10 MB
ARQ-VAE*	75.1	**83.1**	**86.5 / 83.0 / 69.8**	1.0 MB	**75.5**	**63.8**	**69.2 / 66.0 / 55.7**	2.4 MB	**64.6**	**60.9**	**55.9 / 50.3 / 38.2**	10 MB
ASFormer [41]												
Original	79.5	86.2	89.4 / 88.5 / 80.0	245 MB	86.5	78.7	85.4 / 83.0 / 77.0	4.5 GB	73.4	75.6	76.7 / 71.3 / 59.0	28 GB
Mean	72.2	76.9	82.1 / 79.7 / 65.1	7.2 MB	71.6	49.8	56.6 / 52.5 / 43.4	7.8 MB	52.2	43.2	43.5 / 38.3 / 26.7	96 MB
Coreset	71.0	75.4	81.0 / 78.1 / 62.9	7.2 MB	69.4	46.8	56.6 / 52.9 / 39.6	7.8 MB	52.0	48.3	48.3 / 42.4 / 29.7	96 MB
TCA [6]	62.2	57.8	63.0 / 57.4 / 39.9	–	66.8	44.0	52.2 / 47.3 / 36.6	–	36.6	28.2	26.3 / 22.1 / 14.3	–
Encoded	69.2	70.2	73.3 / 67.3 / 49.8	3.6 MB	71.2	45.4	55.0 / 50.4 / 40.2	3.9 MB	37.6	53.6	50.7 / 41.3 / 24.0	44 MB
VQ-VAE	61.3	76.4	72.8 / 68.4 / 54.9	0.1 MB	71.0	64.2	69.3 / 65.6 / 54.9	0.6 MB	—	—	— / — / —	4 MB
GNI [7]	77.9	82.7	86.4 / 84.5 / 70.4	3.6 MB	**81.2**	68.9	77.0 / 73.8 / 64.7	3.9 MB	59.8	48.8	54.1 / 47.7 / 34.1	44 MB
RQ-VAE	76.9	83.5	86.2 / 85.2 / 75.3	1.1 MB	77.9	69.6	76.2 / 73.2 / 63.6	4.1 MB	**68.9**	71.4	**72.4 / 65.9 / 52.9**	22 MB
ARQ-VAE	77.5	82.9	87.0 / 85.9 / 73.2	1.0 MB	79.2	72.2	77.3 / 74.5 / 65.6	2.4 MB	67.3	70.5	70.9 / 64.4 / 50.5	10 MB
ARQ-VAE*	**78.7**	**85.3**	**88.1 / 86.4 / 76.6**	1.0 MB	79.1	**72.4**	**78.8 / 75.2 / 65.7**	2.4 MB	68.3	**71.5**	71.7 / 65.6 / 51.2	10 MB

Crucially, this selection rule is entirely data-driven and feature-based: it relies strictly on reconstruction error, requiring no action labels, segment boundaries, or external heuristics. Frames whose features are easily reconstructed by shallow prefixes naturally retain fewer stages. Conversely, frames containing complex visual dynamics that continue to benefit from further refinement retain deeper prefixes (see Section 3.3.2 and Fig. 4 for a qualitative analysis demonstrating how stable action interiors are compressed more aggressively than complex action transitions). By allocating storage at the frame level, ARQ-VAE treats feature complexity as highly dynamic, ensuring storage capacity is spent precisely where it empirically improves feature fidelity.

3.3.1 ARQ-VAE Fine-Tuning

Because prefix selection evaluates truncations using a decoder optimized for full-depth reconstruction, there is a mismatch between the initial training objective and the final stored memory. To resolve this, we introduce a short fine-tuning stage immediately after prefix selection.

After the optimal prefix lengths for each frame $\{m_t^*\}$ are computed, we freeze their corresponding per-frame indices and discard the encoder entirely. We then fine-tune only the decoder D_θ and the codebooks to directly minimize the reconstruction error of the truncated sequences:

$$\mathcal{L}_{\text{ft}} = \sum_t \left\| x_t - \hat{x}_t^{(m_t^*)} \right\|_2^2. \tag{3}$$

This practical refinement explicitly adapts the network to the precise variable-length representations that will be replayed during downstream TAS training. We use an asterisk (ARQ-VAE*) to refer to the fine-tuned version of ARQ-VAE.

3.3.2 Downstream Feature Reconstruction and Storage

Once ARQ-VAE* memory is constructed, the original dense feature bank $\mathcal{D}$ is discarded. The compact representation consists strictly of variable-length index sequences $\{(k_t^{(1)}, \ldots, k_t^{(m_t^*)})\}_{t=1}^{T_i}$ for all training frames, the stage-wise codebooks, and the fine-tuned decoder D_θ. Additionally, we prune the codebooks by discarding any code vectors that are never referenced by the training data, ensuring no storage capacity is wasted on codes never utilized by frames across the dataset.

During downstream TAS training, the network queries this memory rather than reading from a disk-bound feature bank. For a given frame t, the reconstructed feature $\hat{x}_t$ is generated on the fly by retrieving the active code vectors corresponding to its stored indices and passing their cumulative sum through D_θ. Because the encoder is discarded and the representation relies solely on discrete indices, the overall storage footprint is defined explicitly by the bit-payload of the retained indices ($m_t^* \log_2 K$ bits per frame) and the cost of the pruned floating-point codebook vectors. We present the

mathematical breakdown of ARQ-VAE storage accounting in Appendix B. Following prior work [7], reported storage excludes the float32 decoder, a fixed 2.369 MB for I3D features regardless of dataset size (Appendix C gives complete footprints and standard compression baselines).

4 Experiments

We evaluate ARQ-VAE across supervised [8, 41], incremental [6], and unsupervised [39] TAS, and analyze its behavior through ablations over K and M. We report results for ARQ-VAE after adaptive prefix selection (3.3) and ARQ-VAE* after fine-tuning (3.3.1).

4.1 Experimental Setup

Datasets. We utilize four common TAS benchmark datasets: GTEA [9], 50Salads [33], YouTube Instructional (YTI) [1], and Breakfast [19]. GTEA is a small egocentric dataset with 28 videos and 11 action classes. 50Salads contains 50 long videos with 17 action classes and approximately 20 action instances per video on average. YTI contains instructional videos across 5 activities and is used in the 5-task iTAS setting. Breakfast is the largest benchmark, with 1,712 videos recorded in 18 kitchens and 48 action classes, and is used in the 5- and 10-task iTAS settings. As a result, the storage footprint varies substantially across datasets: GTEA is the smallest, 50Salads and YTI are intermediate, and Breakfast is the largest.

Baselines. We compare against TAS replay and condensation baselines from TCA [6] and GNI [7]. **Original** trains on the full pre-extracted I3D [3] feature bank. **Mean** and **Coreset** are segment-level baselines: Mean stores the average feature for each ground-truth action segment, while Coreset stores the real frame feature closest to that segment mean; both repeat the stored feature to the original segment length. **TCA** generates replay features with a label-conditioned temporally coherent VAE, **Encoded** stores mean TCA encoder latents per segment, and **GNI** optimizes segment-level latents by network inversion and decodes them with the TCA generator. For incremental TAS, we also report **Finetune**, which trains on current-task data without replay, and **Exemplar**, which stores one real frame feature per previous-task segment and repeats it for replay [6]. For unsupervised TAS, we compare reconstructed training features against original I3D features using ASOT [39]. See Appendix D for more details regarding each baseline and its implementation.

Evaluation. Following prior work [8, 41], we report frame-wise Accuracy, segment-wise Edit score, and segment-wise F1@{10,25,50} for supervised TAS in Table 1 and incremental TAS in Table 2. The main supervised TAS results are averaged over the standard dataset splits using K-fold cross-validation, while incremental TAS results are averaged over task-wise performance to account for class imbalance and reported across three unique task sequences. For unsupervised TAS in Table 3, we report Mean over Frames (MoF), F1@50, and mean intersection over union (mIoU). We also evaluate feature reconstruction quality using Mean Absolute Error (MAE) and Mean Squared Error (MSE).

Implementation details. Our RQ-VAE follows the lightweight TCA/GNI encoder–decoder design [6, 7]: two-layer MLP encoder and decoder with latent dimension $d = 256$. Residual quantization uses $M = 8$ stage-specific codebooks $\{\mathcal{C}^{(m)}\}_{m=1}^{M}$, each in $\mathbb{R}^{K\times d}$ where $K = 128$. We train the full-depth RQ-VAE with Eq. (1) for 7.5K iterations using learning rate $1e^{-3}$, then apply prefix selection to store a truncated residual prefix per frame. ARQ-VAE* fine-tunes for another 7.5K iterations with fixed prefixes, updating only the active codebooks and decoder parameters θ.[1]

For supervised and incremental TAS, we use MS-TCN [8] and ASFormer [41]. MS-TCN is trained for 50 epochs with learning rate $5e^{-4}$, and ASFormer for 120 epochs with learning rate $1e^{-4}$. For unsupervised TAS, we use ASOT [39] with reconstructed ARQ-VAE features and follow its original setup. For supervised TAS, ARQ-VAE is trained once on the full training feature bank for each split; for incremental TAS, it is trained separately on each arriving task subset so that only compact previous-task memories are retained for replay.

[1] Main experiments used NVIDIA RTX A6000 GPUs with 48 GB memory and CUDA 12.5. The computational-cost study on 50Salads used a V100 GPU (Appendix M). Additional implementation details are in Section E.

Table 2: **Main iTAS results** on Breakfast and YouTube Instructional averaged over 3 task sequences.

# Tasks	Method	MS-TCN Acc	MS-TCN Edit	MS-TCN F1@{10,25,50}			ASFormer Acc	ASFormer Edit	ASFormer F1@{10,25,50}		
		Breakfast [19]									
10	Original	50.6	48.8	51.3	46.8	37.1	48.1	45.2	45.9	42.4	34.2
	Finetune	7.4	7.2	7.5	7.0	5.4	9.9	9.8	10.3	9.4	7.5
	Exemplar	16.1	13.3	13.8	12.5	9.5	12.4	11.2	11.7	10.7	8.5
	TCA [6]	29.4	25.9	26.3	23.5	17.7	34.2	32.4	33.1	30.1	23.4
	GNI [7]	38.2	31.6	32.8	29.7	22.8	46.7	41.1	41.7	38.3	30.8
	RQ-VAE	41.6	46.1	45.8	40.0	30.8	40.3	44.4	45.8	40.7	31.8
	ARQ-VAE	49.5	51.0	**52.1**	**46.8**	34.8	53.6	**56.1**	**57.8**	**51.9**	42.4
	ARQ-VAE*	**51.0**	**51.6**	51.8	46.1	**35.0**	**53.9**	55.7	56.9	52.4	**42.9**
5	Original	55.8	58.0	59.9	54.5	43.5	60.1	59.7	63.3	56.4	45.0
	Finetune	15.4	15.8	16.6	15.8	12.7	15.7	16.1	16.9	15.8	13.2
	Exemplar	32.5	28.9	30.8	28.5	22.9	29.5	27.5	28.7	26.7	22.0
	TCA [6]	54.5	49.4	51.1	46.9	37.7	57.2	56.8	58.3	54.0	43.6
	GNI [7]	36.5	44.5	44.6	40.4	30.9	43.0	45.3	47.0	41.7	31.1
	RQ-VAE	50.5	52.8	54.6	49.1	38.3	55.1	59.3	60.8	55.9	48.1
	ARQ-VAE	57.3	57.7	58.8	53.7	**42.9**	59.6	57.3	59.9	56.1	46.4
	ARQ-VAE*	**59.0**	**58.0**	**59.8**	**54.2**	42.5	**64.5**	**62.9**	**64.9**	**60.1**	**51.8**
		YouTube Instructional [1]									
5	Original	61.6	42.7	37.2	31.5	18.7	69.0	52.9	48.7	42.7	27.6
	Finetune	13.6	2.8	3.6	2.7	0.6	13.9	11.5	11.1	9.8	6.3
	Exemplar	30.8	19.7	19.8	16.0	9.3	22.1	18.9	17.7	15.3	10.0
	TCA [6]	30.2	25.0	21.9	18.5	11.1	25.2	20.9	20.1	17.5	11.4
	GNI [7]	50.3	33.2	29.5	25.2	12.8	53.4	36.9	31.7	25.5	12.6
	RQ-VAE	57.4	39.0	34.5	29.1	16.1	63.7	46.1	42.6	37.1	21.5
	ARQ-VAE	63.0	44.9	40.8	34.2	19.3	**68.3**	**52.8**	47.6	40.1	24.2
	ARQ-VAE*	**64.6**	**48.2**	**42.1**	**35.2**	**20.4**	68.2	51.8	**47.8**	**42.1**	**25.5**

Figure 4: Reconstruction MAE heatmap for an example video, showing frame-wise prefix lengths m_t^* with black boxes.

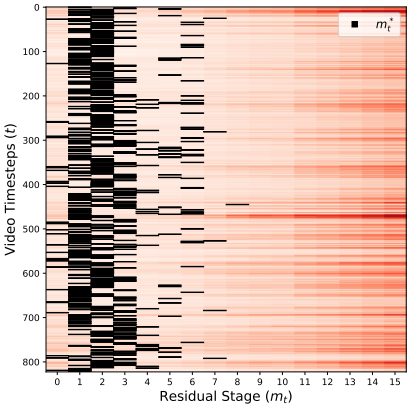


Table 3: **Unsupervised action segmentation** with ASOT [39]. Gray rows use original I3D features; other rows replace I3D with reconstructed training features.

Features	GTEA MoF	GTEA F1	GTEA mIoU	GTEA Size	50 Salads MoF	50 Salads F1	50 Salads mIoU	50 Salads Size	Breakfast MoF	Breakfast F1	Breakfast mIoU	Breakfast Size
I3D	56.9	52.7	36.1	245 MB	65.2	51.2	32.8	4.5 GB	60.0	58.7	40.0	28 GB
RQ-VAE	58.1	51.9	36.8	1.3 MB	60.4	43.6	27.3	4.9 MB	**65.2**	**55.0**	**36.5**	25 MB
ARQ-VAE	**61.0**	55.2	38.3	1.1 MB	59.1	43.4	26.1	2.7 MB	64.1	53.3	34.7	11 MB
ARQ-VAE*	59.0	**55.9**	**40.6**	1.1 MB	**61.9**	**45.1**	**28.0**	2.7 MB	64.6	54.5	36.1	11 MB

Table 4: **Summary reconstruction error** and storage averaged across various (K, M) settings.

Dataset	MSE ↓ RQ-VAE	MSE ↓ ARQ-VAE	MSE ↓ ARQ-VAE*	Storage (MB) ↓ RQ-VAE	Storage (MB) ↓ ARQ-VAE
GTEA	0.104	0.026 (-75.0%)	**0.022** (-78.8%)	0.989	0.451 (-54.4%)
50Salads	0.076	0.024 (-68.4%)	**0.022** (-70.3%)	4.512	1.471 (-67.4%)
Breakfast	0.042	0.025 (-41.5%)	**0.023** (-45.2%)	21.901	8.392 (-61.7%)

4.2 Experimental Results

4.2.1 Supervised Temporal Action Segmentation

Table 1 evaluates supervised temporal action segmentation on GTEA [9], 50Salads [33], and Breakfast [19] using MS-TCN [8] and ASFormer [41]. We ask whether the proposed discrete memory can preserve segmentation quality while reducing storage relative to the original feature bank and prior compact baselines. Across datasets and backbones, RQ-VAE provides a strong compressed representation, while ARQ-VAE* achieves a strong overall storage–performance trade-off.

The clearest gains appear on Breakfast, the largest and most storage-intensive benchmark. Compared with GNI, ARQ-VAE* improves ASFormer from 48.8 to 71.5 Edit and from 34.1 to 51.2 F1@50, while reducing storage from 44 MB to 10 MB. With MS-TCN, it improves Edit from 45.6 to 60.9 and F1@50 from 28.7 to 38.2 under the same storage reduction. These gains suggest that adaptive frame-level allocation is especially beneficial when long videos contain substantial redundancy alongside temporally important frames that require higher reconstruction fidelity. GTEA and 50Salads show the same overall pattern: ARQ-VAE* closes more of the gap to the original feature bank than prior compact methods, particularly on Edit and F1, indicating better preservation of temporal structure and segment boundaries.

4.2.2 Incremental Temporal Action Segmentation

Table 2 evaluates replay-based incremental temporal action segmentation under the TCA protocol [6]. Breakfast and YouTube Instructional are partitioned by activity and learned sequentially, with

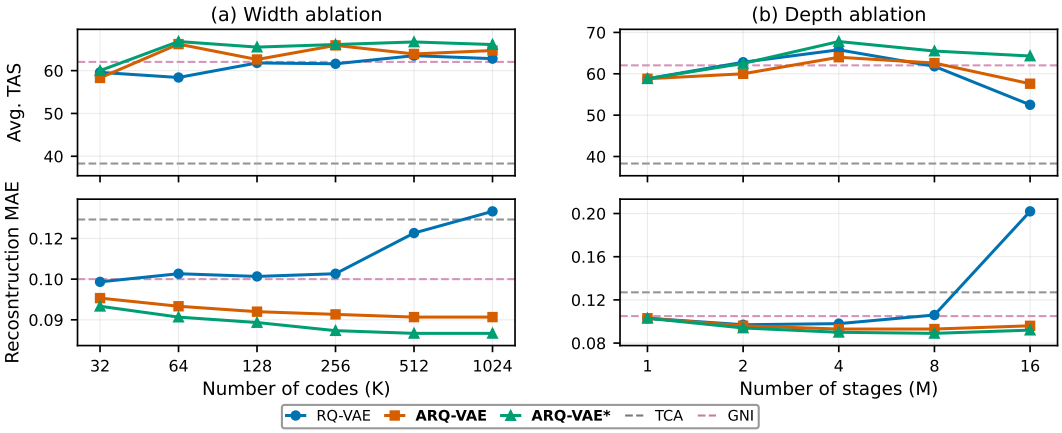


Figure 5: Hyperparameter ablations on the number of codes (K) and residual stages (M). **(a)** Ablation varying K. **(b)** Ablation varying M.

Breakfast evaluated in 10-task and 5-task settings and YouTube Instructional in a 5-task setting. Results are averaged over seen tasks and three unique task sequences. In this setting, the condensed representation must not only reduce memory cost but also preserve enough information to support replay and mitigate catastrophic forgetting.

Again, the strongest gains appear in the hardest settings, especially on Breakfast. In the 10-task regime, ARQ-VAE and ARQ-VAE* substantially outperform GNI for both backbones: with MS-TCN, accuracy rises from 38.2 to 49.5 and 51.0, while with ASFormer it rises from 46.7 to 53.6 and 53.9. The segmental metrics follow the same trend. The 5-task Breakfast and 5-task YouTube Instructional results tell a similar story, with our method consistently improving Edit and F1 over prior replay baselines. Together with Table 1, these results show that adaptive residual quantization is effective both as a compact training representation and as a substantially stronger replay mechanism for iTAS.

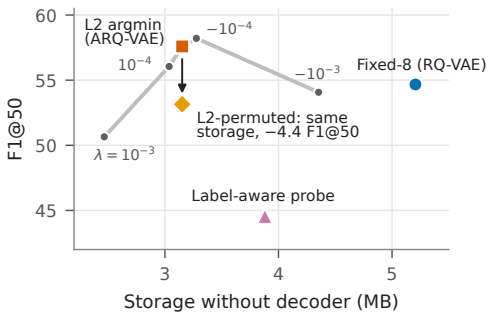
4.2.3 Unsupervised Temporal Action Segmentation

Table 3 evaluates ASOT [39] when original I3D training features are replaced with reconstructed features. Here, I3D serves as the uncompressed reference, so the goal is to preserve segmentation quality while substantially reducing storage. On GTEA, ARQ-VAE variants match or improve the I3D metrics while reducing storage from 245 MB to 1.1 MB. On 50Salads and Breakfast, the reconstructed features remain reasonably close to I3D while compressing 4.5 GB to 2.7 MB and 28 GB to 11–25 MB, respectively. Overall, these results show that ARQ-VAE remains useful beyond supervised or replay-specific TAS.

4.2.4 Ablation Studies and Additional Analysis

Width and depth ablation. We analyze the source of ARQ-VAE's improved storage–performance trade-off through the summary reconstruction and storage results in Table 4, the frame-wise prefix visualization in Fig. 4, and the width/depth sweeps in Fig. 5. Table 4 shows that adaptive truncation reduces both reconstruction error and storage across $K \in \{32, 64, 128, 256, 512, 1024\}$ and $M \in \{2, 4, 8, 16\}$ (full width/depth sweeps in Appendices F–H). This robustness comes from the prefix selection shown in Fig. 4: redundant frames often stop after only a few residual stages, while harder frames retain deeper prefixes. Unlike fixed-depth RQ-VAE, which forces every frame to use the same residual budget even when later stages add little benefit or degrade reconstruction, ARQ-VAE trims unhelpful later stages on a per-frame basis. The sweeps in Fig. 5 further show that this makes ARQ-VAE less brittle to the choice of codebook width K and residual depth M. ARQ-VAE* performs best overall, indicating that adaptive truncation and fine-tuning together produce a more reliable compact feature memory.

Reconstruction error as a selection criterion. ARQ-VAE selects prefix lengths by minimizing feature-space reconstruction error (Eq. (2)), whereas segmentation also depends on action boundaries and temporal structure. To test whether this choice matters, we reuse the same RQ-VAE so that only the retained prefix lengths differ (Fig. 6; Appendix L). Unlike the cross-validated results in Table 1,



Criterion	MB	Acc	Edit	F1@10	F1@25	F1@50
● Fixed-8	5.21	69.84	59.87	67.88	63.78	54.67
■ L2 argmin	**3.15**	**74.92**	**62.01**	**69.20**	**65.18**	**57.59**
◆ L2-permuted	**3.15**	70.50	58.98	64.41	60.81	53.15
▲ Label-aware probe	3.88	68.45	51.11	57.52	54.10	44.57

Figure 6: **Prefix-selection criteria** on a single 50Salads split with MS-TCN. **Left:** F1@50 vs. storage without decoder; the gray curve sweeps λ (λ=0 is L2 argmin). **Right:** TAS metrics per criterion; MB is storage without decoder. Full results in Tables 9 and 10.

these diagnostics use a single 50Salads train/test split, so their absolute values are not directly comparable. Besides Fixed-8 (RQ-VAE) and L2 argmin (ARQ-VAE), we evaluate **L2-permuted**, which shuffles the L2-selected depths across frames, preserving the depth histogram and storage while breaking the link between frames and depths, and a supervised **label-aware probe**, which keeps the prefix with the lowest correct-class cross-entropy under a linear frame-wise classifier trained on TAS labels. At identical storage, shuffling lowers F1@50 from 57.59 to 53.15, below even Fixed-8: frame-specific depth allocation matters beyond the total number of retained codes. The probe uses more storage yet reaches only 44.57 F1@50, so this label-aware probe does not improve selection in this experiment.

Explicit rate–distortion control. L2 argmin does not penalize storage. Selecting $m_t(\lambda) = \arg\min_m [D_t(m) + \lambda m]$, where $D_t(m)$ is the MSE of the length-m prefix, makes this trade-off explicit without retraining and recovers L2 argmin at λ=0. As the gray curve in Fig. 6 shows, $\lambda > 0$ reduces storage at the expected cost in segmentation performance, a mild bias toward longer prefixes (λ=-10^{-4}) slightly improves F1@50 to 58.21, and a strong bias (λ=-10^{-3}) lowers it to 54.07, so more stages are not monotonically better. Because RQ-VAE is trained for full-depth reconstruction and its latent residuals pass through a nonlinear decoder, later stages can move an already accurate reconstruction away from its target, even though training shows no sign of codebook collapse. These results support L2 as an effective, if not universal, label-free selection criterion in this setting.

Additional analyses in the appendix include prefix-length histograms (I), empirical frame difficulty (J), qualitative alignment of residual codes with action segments (K), ARQ-VAE performance on contemporary DINOv3 features (N), and computational cost (M). Notably, prefix selection adds 16.47 s to RQ-VAE preparation time (102.91 s) on the **50Salads** dataset. For 50 MS-TCN epochs, training with on-the-fly decoding took 212.30 s versus 259.43 s for FP32; while the initial ARQ-VAE run took 331.68 s including one-time preparation and training.

5 Conclusion

We presented ARQ-VAE, a compact, label-free feature memory for temporal action segmentation that keeps only the residual prefix that best reconstructs each frame. This allocation improves the storage–performance trade-off over prior replay and condensation methods across supervised, incremental, and unsupervised temporal action segmentation, compressing the 28 GB Breakfast I3D feature bank to 10 MB, and carries over to DINOv3 features and the FACT [23] architecture (Appendices N and O) at a small one-time cost.

Limitations and future work. ARQ-VAE compresses pre-extracted features, therefore its effectiveness is bounded by the features of the underlying video backbone. Additionally, prefix selection in ARQ-VAE relies on a label-free reconstruction proxy. A worthy future direction is to investigate the potential of a prefix selection specific to temporal action segmentation, which could better specialize the adapted codebooks for the TAS setting. Finally, although the mechanism is task-agnostic, our evidence is specific to TAS; extending adaptive residual memories to other frozen feature caches, such as the latents used to train video diffusion models, is a promising direction for future work.

References

[1] Jean-Baptiste Alayrac, Piotr Bojanowski, Nishant Agrawal, Josef Sivic, Ivan Laptev, and Simon Lacoste-Julien. Unsupervised learning from narrated instruction videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4575–4583, 2016.

[2] Artem Babenko and Victor Lempitsky. Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 931–938, 2014. URL `https://openaccess.thecvf.com/content_cvpr_2014/html/Babenko_Additive_Quantization_for_2014_CVPR_paper.html`.

[3] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[4] Yunkee Chae, Woosung Choi, Yuhta Takida, Junghyun Koo, Yukara Ikemiya, Zhi Zhong, Kin Wai Cheuk, Marco A Martínez-Ramírez, Kyogu Lee, Wei-Hsiang Liao, et al. Variable bitrate residual vector quantization for audio coding. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2025.

[5] Guodong Ding and Angela Yao. Temporal action segmentation with high-level complex activity labels. *IEEE Transactions on Multimedia*, 2022. doi: 10.1109/TMM.2022.3231099.

[6] Guodong Ding, Hans Golong, and Angela Yao. Coherent temporal synthesis for incremental action segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 28485–28494, 2024.

[7] Guodong Ding, Rongyu Chen, and Angela Yao. Condensing action segmentation datasets via generative network inversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17733–17742, 2025.

[8] Yazan Abu Farha and Jurgen Gall. Ms-tcn: Multi-stage temporal convolutional network for action segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3575–3584, 2019.

[9] Alireza Fathi, Xiaofeng Ren, and James M Rehg. Learning to recognize objects in egocentric activities. In *CVPR 2011*, pages 3281–3288. IEEE, 2011.

[10] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[11] Robert Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.

[12] Tyler L. Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In *Computer Vision – ECCV 2020*, volume 12353 of *Lecture Notes in Computer Science*, pages 466–483, 2020. doi: 10.1007/978-3-030-58598-3_28.

[13] Ahmet Iscen, Jeffrey Zhang, Svetlana Lazebnik, and Cordelia Schmid. Memory-efficient incremental learning through feature adaptation. In *Computer Vision – ECCV 2020*, volume 12361 of *Lecture Notes in Computer Science*, pages 699–715, 2020. doi: 10.1007/978-3-030-58517-4_41.

[14] Yuchi Ishikawa, Seito Kasai, Yoshimitsu Aoki, and Hirokatsu Kataoka. Alleviating oversegmentation errors by detecting action boundaries. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2322–2331, 2021.

[15] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. doi: 10.1109/TPAMI.2010.57.

[16] Li Jiao, Qiuxia Lai, Yu Li, and Qiang Xu. Vector quantization prompting for continual learning. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1072. URL `https://proceedings.neurips.cc/paper_files/paper/2024/hash/3baf4eeffad860ca9c54aeab632716b4-Abstract-Conference.html`.

[17] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[18] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[19] Hilde Kuehne, Ali Arslan, and Thomas Serre. The language of actions: Recovering the syntax and semantics of goal-directed human activities. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 780–787, 2014.

[20] Sateesh Kumar, Sanjay Haresh, Awais Ahmed, Andrey Konin, M. Zeeshan Zia, and Quoc-Huy Tran. Unsupervised action segmentation by joint representation learning and online clustering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20174–20185, 2022.

[21] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11523–11532, 2022.

[22] Xialei Liu, Chenshen Wu, Mikel Menta, Luis Herranz, Bogdan Raducanu, Andrew D. Bagdanov, Shangling Jui, and Joost van de Weijer. Generative feature replay for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 915–924, 2020. doi: 10.1109/CVPRW50498.2020.00121.

[23] Zijia Lu and Ehsan Elhamifar. Fact: Frame-action cross-attention temporal modeling for efficient action segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18175–18185, 2024.

[24] Tamasha Malepathirana, Damith Senanayake, and Saman Halgamuge. NAPA-VQ: Neighborhood-aware prototype augmentation with vector quantization for continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11640–11650, 2023. URL `https://openaccess.thecvf.com/content/ICCV2023/html/Malepathirana_NAPA-VQ_Neighborhood-Aware_Prototype_Augmentation_with_Vector_Quantization_for_Continual_Learning_ICCV_2023_paper.html`.

[25] Shivani Mall and João F. Henriques. Cram: Large-scale video continual learning with bootstrapped compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15045–15055, 2025.

[26] Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: VQ-VAE made simple. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=8ishA3LxN8`.

[27] Oleksiy Ostapenko, Timothée Lesort, Pau Rodríguez, Md Rifat Arefin, Arthur Douillard, Irina Rish, and Laurent Charlin. Continual learning with foundation models: An empirical study of latent replay. In *Proceedings of The 1st Conference on Lifelong Learning Agents*, volume 199 of *Proceedings of Machine Learning Research*, pages 60–91, 2022.

[28] Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10203–10209, 2020. doi: 10.1109/IROS45743.2020.9341460.

[29] Peijia Qin and Jianguo Zhang. Learning to quantize for training vector-quantized networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025. URL `https://proceedings.mlr.press/v267/qin25j.html`.

[30] Rahul Rahaman, Dipika Singhania, Alexandre H. Thiery, and Angela Yao. A generalized and robust framework for timestamp supervision in temporal action segmentation. In *Computer Vision – ECCV 2022*, volume 13664 of *Lecture Notes in Computer Science*, pages 279–296, 2022. doi: 10.1007/978-3-031-19772-7_17.

[31] Yuhan Shen and Ehsan Elhamifar. Progress-aware online action segmentation for egocentric procedural task videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18186–18197, 2024.

[32] Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.

[33] Sebastian Stein and Stephen J McKenna. Combining embedded accelerometers with computer vision for recognizing food preparation activities. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 729–738, 2013.

[34] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training. *Advances in neural information processing systems*, 35:10078–10093, 2022.

[35] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[36] Junke Wang, Yi Jiang, Zehuan Yuan, Binyue Peng, Zuxuan Wu, and Yu-Gang Jiang. Omnitokenizer: A joint image-video tokenizer for visual generation. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL `https://proceedings.neurips.cc/paper_files/paper/2024/hash/31994923f58ae5b2d661b300bd439107-Abstract-Conference.html`.

[37] Kai Wang, Joost van de Weijer, and Luis Herranz. Acae-remind for online continual learning with compressed feature replay. *Pattern Recognition Letters*, 150:122–129, 2021. doi: 10.1016/j.patrec.2021.06.025.

[38] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

[39] Ming Xu and Stephen Gould. Temporally consistent unbalanced optimal transport for unsupervised action segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14618–14627, 2024.

[40] Enneng Yang, Li Shen, Zhenyi Wang, Tongliang Liu, and Guibing Guo. An efficient dataset condensation plugin and its application to continual learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 67625–67642, 2023.

[41] Fangqiu Yi, Hongyu Wen, and Tingting Jiang. Asformer: Transformer for action segmentation. *arXiv preprint arXiv:2110.08568*, 2021.

[42] Lijun Yu, José Lezama, Nitesh B. Gundavarapu, Luca Versari, Kihyuk Sohn, David Minnen, Yong Cheng, Vighnesh Birodkar, Agrim Gupta, Xiuye Gu, Alexander G. Hauptmann, Boqing Gong, Ming-Hsuan Yang, Irfan Essa, David A. Ross, and Lu Jiang. Language model beats diffusion – tokenizer is key to visual generation. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=gzqrANCF4g`.

[43] Ting Zhang, Chao Du, and Jingdong Wang. Composite quantization for approximate nearest neighbor search. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 838–846, 2014. URL `https://proceedings.mlr.press/v32/zhangd14.html`.

[44] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=mSAKhLYLSsl`.

[45] Yue Zhao, Yuanjun Xiong, and Philipp Krähenbühl. Image and video tokenization with binary spherical quantization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=yGnsH3gQ6U.

[46] Chuanxia Zheng and Andrea Vedaldi. Online clustered codebook. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023. doi: 10.1109/ICCV51070.2023.02084. URL https://openaccess.thecvf.com/content/ICCV2023/html/Zheng_Online_Clustered_Codebook_ICCV_2023_paper.html.

[47] Qing Zhong, Guodong Ding, and Angela Yao. Onlinetas: An online baseline for temporal action segmentation. *Advances in Neural Information Processing Systems*, 37:58984–59005, 2024.

A Broader Impacts

ARQ-VAE can make long-horizon video understanding more practical by reducing the cost of storing and replaying dense frame features, especially in incremental and continual learning settings with limited memory. This may lower barriers for long-form video understanding, robotics, and embodied AI systems in resource-constrained environments. Although we believe our work is unlikely to have direct negative societal impacts, we acknowledge the importance of considering potential misuse scenarios, such as the exploitation of continually adapting systems in unethical surveillance or misinformation campaigns.

B Storage Accounting

A central advantage of our representation is that its storage cost is explicit. After compression, the memory consists of stored residual indices and the active code vectors that remain after pruning unused entries. Let L denote the total number of stored frames, and let m_t be the number of retained indices for frame t. The index payload is

$$S_{\text{idx}} = \left(\sum_{t=1}^{L} m_t \right) \log_2 K \quad \text{bits},$$

where $m_t = M$ for standard RQ-VAE and $m_t = m_t^*$ for ARQ-VAE. Since we use power-of-two codebook sizes, i.e., $K = 2^x$, each index requires exactly $x = \log_2 K$ bits.

Let A_m be the number of active code vectors remaining in stage m after pruning. Under float32 storage, the codebook cost is

$$S_{\text{code}} = 32d \sum_{m=1}^{M} A_m \quad \text{bits},$$

where d is the latent dimension. The total storage is therefore

$$S_{\text{total}} = S_{\text{idx}} + S_{\text{code}}. \tag{4}$$

We report storage after pruning inactive codes. We do not include decoder parameters in the main storage tables (per prior work [7]), and we do not store a separate prefix-length token, since each frame is already represented by a truncated list of residual indices.

C Comparison with Standard Feature-Compression Methods

We compare ARQ-VAE against commonly used feature-compression baselines: FP16 downcasting, deterministic per-channel INT8 quantization, lossless compression (`np.savez_compressed`), and PCA with d retained components. INT8 calibration is performed once during preprocessing; the evaluated I3D feature bank is nonnegative, with an observed range of approximately $[0, 5.18]$. Unlike the main tables, storage here is the complete footprint, including all method-specific parameters: PCA projections, INT8 metadata, and the decoder for RQ-VAE and ARQ-VAE. Results use MS-TCN on the same single 50Salads train/test split as Appendix L, evaluated on the original, uncompressed test features. In this experiment, ARQ-VAE requires 3.152 MB under the paper's payload convention and 5.521 MB once the 2.369 MB decoder is included, compared with 3777.877 MB for the original FP32 training bank, so the compression remains substantial under either accounting convention.

As shown in Table 5 and Fig. 7, FP16, INT8, and lossless compression preserve feature quality but provide only modest storage reductions of $1.13\times$ to $4\times$. PCA reaches the same high-compression regime as our method, yet loses substantially more downstream utility: at roughly 7.5 MB, Fixed-8 RQ-VAE achieves 54.67 F1@50 versus 37.67 for PCA with d=4. ARQ-VAE improves this trade-off further, reducing complete storage to 5.521 MB while reaching 57.59 F1@50. ARQ-VAE is therefore most advantageous in the high-compression regime, where conventional precision reduction remains comparatively large and aggressive dimensionality reduction loses substantial downstream TAS utility.

Table 5: **Comparison with standard feature compression** on 50Salads with MS-TCN. Storage is the complete footprint, including all method-specific parameters (e.g., the decoder for RQ-VAE and ARQ-VAE).

Method	Complete storage (MB)	Compression	MSE ↓	MAE ↓	Acc	Edit	F1@{10,25,50}
Original FP32	3777.877	1.00×	0	0	81.61	70.79	76.39 / 73.61 / 64.81
FP16	1888.947	2.00×	≈0	0.00004	81.43	73.34	77.86 / 75.52 / 67.13
INT8	944.487	4.00×	0.00002	0.00312	80.30	63.42	69.62 / 66.67 / 58.23
Lossless	3331.176	1.13×	0	0	81.61	70.79	76.39 / 73.61 / 64.81
PCA, d=2	3.725	1014.18×	0.02948	0.11284	46.92	39.90	37.83 / 33.48 / 25.22
PCA, d=4	7.431	508.41×	0.02732	0.10837	67.58	44.69	49.71 / 46.21 / 37.67
Fixed-8 RQ-VAE	7.574	498.77×	0.02605	0.10664	69.84	59.87	67.88 / 63.78 / 54.67
ARQ-VAE (L2 argmin)	5.521	684.30×	0.02068	0.09296	74.92	62.01	69.20 / 65.18 / 57.59

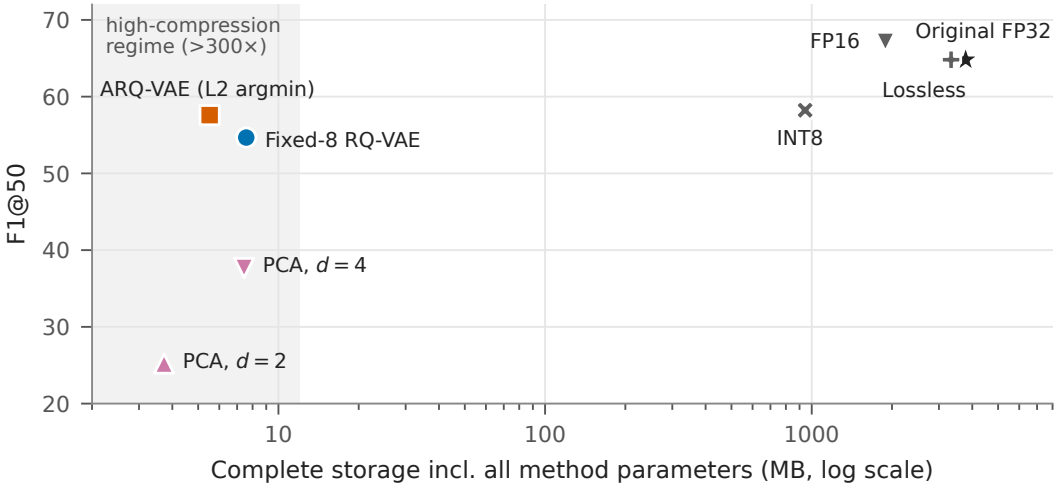


Figure 7: **Storage–performance trade-off against standard feature compression** on 50Salads with MS-TCN. Storage is the complete footprint, including all method-specific parameters, on a log scale. In the high-compression regime, ARQ-VAE retains far more downstream TAS performance than PCA at comparable storage.

D Baseline Details

This section gives additional implementation details for the baselines used in the supervised, incremental, and unsupervised TAS experiments. We follow the baseline definitions from TCA [6] and GNI [7] whenever applicable.

Original features. The **Original** baseline trains the downstream TAS model directly on the full pre-extracted feature bank. For a training video with frame features $X = \{x_t\}_{t=1}^{T}$, the model receives the original dense sequence X and the corresponding frame-wise labels. No feature compression, condensation, replay synthesis, or reconstruction is applied. In incremental TAS, **Original** replay stores and reuses the original feature sequences from previous tasks, and therefore serves as an uncompressed replay reference under the same task protocol.

Mean. The **Mean** baseline compresses each ground-truth action segment into a single average feature vector. Let segment s_n contain frame indices $\mathcal{I}_n = \{t_n, \dots, t_n + \ell_n - 1\}$ with length ℓ_n. Mean stores

$$\bar{x}_n = \frac{1}{\ell_n} \sum_{t \in \mathcal{I}_n} x_t.$$

During TAS training or replay, this stored vector is repeated ℓ_n times to restore the original temporal length of the segment. Thus, the reconstructed segment is static:

$$\hat{x}_t = \bar{x}_n \quad \text{for all } t \in \mathcal{I}_n.$$

This baseline primarily compresses the temporal dimension, reducing a video from frame-level storage to one feature vector per action segment.

Coreset. The **Coreset** baseline uses the same segment-level compression ratio as Mean, but stores a real frame feature instead of the averaged feature. For each segment s_n, we first compute the segment mean $\bar{x}_n$ as above and then select the frame feature nearest to it:

$$x_n^{\text{core}} = x_{t_n^*}, \qquad t_n^* = \arg\min_{t \in \mathcal{I}_n} \|x_t - \bar{x}_n\|_2^2.$$

During training or replay, x_n^{core} is repeated for all ℓ_n frames in the segment. The difference from Mean is therefore the stored representative: Mean stores the arithmetic average, while Coreset stores the closest actual frame feature.

TCA. **TCA** [6] is a label-conditioned generative replay baseline for incremental TAS. It trains a VAE-style generator over frame features conditioned on the action label a and a temporal coherence variable c, where c encodes the relative position of a frame inside an action segment. For replay, a segment is generated by sampling a latent code z, fixing it across the segment, varying c from the start to the end of the segment, and decoding features as

$$\hat{x}_i = D_\theta(z, a, c_i), \qquad i = 1, \ldots, \ell.$$

Generated segments are concatenated according to sampled action sequences and durations to form replay videos. In the TCA baseline, the latent codes are sampled from a normal prior during generation rather than stored as optimized per-segment representations.

Encoded. **Encoded** is the non-inversion comparison used in GNI [7]. It uses the trained TCA encoder to map each frame in a segment to a latent distribution and stores the average encoded latent for that segment. If the encoder produces latent means μ_t for frames $t \in \mathcal{I}_n$, Encoded stores

$$z_n^{\text{enc}} = \frac{1}{\ell_n} \sum_{t \in \mathcal{I}_n} \mu_t.$$

At reconstruction time, this latent is decoded with the TCA decoder using the segment action label and coherence variable. Unlike GNI, Encoded does not optimize the latent code by network inversion; it simply averages the encoder outputs.

GNI. **GNI** [7] builds on the TCA generator but replaces averaged encoded latents with optimized segment latents. After training the TCA generator, GNI freezes the decoder and optimizes a latent code for each segment so that the decoded segment features match the original segment features:

$$z_n^* = \arg\min_z \sum_{i=1}^{\ell_n} \|D_\theta(z, a_n, c_i) - x_{t_n+i-1}\|_2^2.$$

The optimized latent codes are then stored as the condensed dataset and decoded during downstream TAS training or replay. Thus, GNI performs segment-level condensation in the latent space of a label-conditioned generator.

Finetune. **Finetune** is used only in the incremental TAS setting. At each task step, the TAS model is trained on the current task data only, using the original current-task features and labels. No previous-task data, replay features, stored exemplars, or generated samples are used. This baseline measures catastrophic forgetting when the model adapts to new tasks without any memory mechanism.

Exemplar. **Exemplar** is the direct-replay baseline used in TCA [6]. For each previous-task action segment, it stores one representative real frame feature. The stored frame is then repeated across the segment duration to form a static replay segment, analogous to Coreset-style temporal inflation:

$$\hat{x}_t = x_n^{\text{ex}} \quad \text{for all } t \in \mathcal{I}_n.$$

This baseline stores actual features rather than generated or quantized features, but it loses temporal variation within each action segment because replay segments are static after repetition.

VQ-VAE. **VQ-VAE** is our single-stage vector-quantized autoencoder baseline. It uses one discrete codebook to quantize each frame feature into a single code index and reconstructs the feature through the decoder. This is equivalent to using one quantization stage, i.e., $M = 1$, and therefore tests whether a single discrete code per frame is sufficient for compact TAS feature memory.

RQ-VAE. **RQ-VAE** is the fixed-rate residual quantization baseline. Each frame feature is encoded into a sequence of M residual code indices, and every frame stores all M indices. Unlike ARQ-VAE, RQ-VAE does not adapt the number of retained residual stages per frame. It therefore isolates the benefit of residual quantization itself from the benefit of adaptive prefix truncation.

ASOT setting for unsupervised TAS. For unsupervised TAS, we use ASOT [39] as the downstream segmentation method. The **Original** unsupervised baseline trains ASOT with the original I3D feature bank. Our RQ-VAE, ARQ-VAE, and ARQ-VAE* rows replace only the training feature bank with reconstructed features, while keeping the ASOT procedure unchanged. This evaluates whether the compressed representation remains useful without relying on frame-level action labels during downstream training.

E Additional Implementation Details

Unless otherwise stated, we use $K{=}128$ codes per stage and $M{=}8$ residual stages for all main ARQ-VAE experiments. The reconstruction summary in Table 4 averages over a broader grid of (K, M) settings, but we use this single default configuration for downstream TAS experiments; empirically, performance is not highly sensitive to the exact choice of K and M after adaptive prefix selection. We train RQ-VAE and ARQ-VAE with Adam using a batch size of 256, learning rate 10^{-3}, and commitment weight $\beta{=}0.25$. I3D features are used directly without additional normalization. Mini-batches for ARQ-VAE training are sampled uniformly over all training frames.

For downstream TAS training, we store the adaptive code representation and reconstruct features on the fly. The dataloader loads the retained indices for each frame, retrieves the corresponding active codebook entries, and applies the decoder during collation to produce reconstructed training features. TAS backbones are still trained video-wise, following the standard MS-TCN and ASFormer setup, rather than with uniformly sampled frame batches. For storage accounting, we count the retained discrete indices and the pruned active codebooks. The encoder is discarded, unused codebook entries are removed, and prefix lengths are not stored separately because each frame is represented directly by its retained variable-length index sequence. Supervised TAS results use full cross-validation over the official dataset splits: four splits for Breakfast and GTEA, and five splits for 50Salads. For incremental TAS, we use a single split for Breakfast and YouTube Instructional and report averages over three unique task sequences. For supervised and unsupervised experiments, ARQ-VAE is trained at the dataset-split level on the full training feature bank. For incremental TAS, ARQ-VAE is instead trained on each task subset sequentially, matching the iTAS protocol: after a task is compressed, only its compact ARQ-VAE memory is retained for replay in later tasks.

F Ablation on Number of Codes

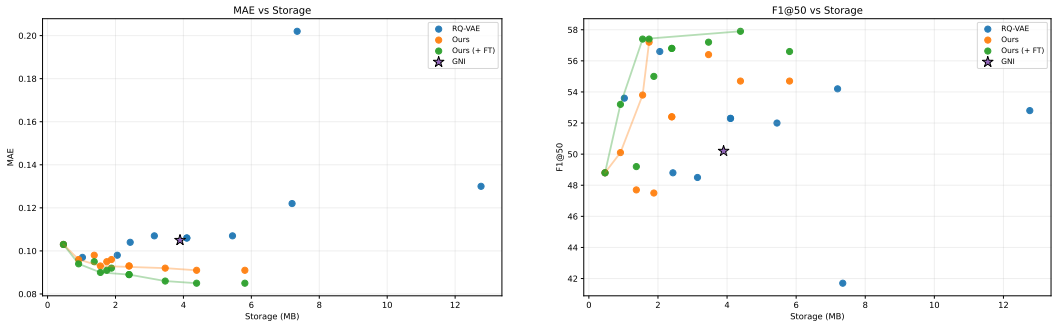
This ablation studies the effect of codebook width K while keeping the number of residual stages fixed at $M{=}8$. The goal is to evaluate how the size of each stage-level codebook affects the storage–performance tradeoff. As shown in Table 6, increasing K generally improves reconstruction capacity but also increases storage, and fixed-depth RQ-VAE does not always translate larger codebooks into better TAS performance. ARQ-VAE variants are more robust across widths because they retain only the useful residual prefixes, with ARQ-VAE* giving the strongest overall balance between task performance, reconstruction quality, and compactness.

G Ablation on Number of Residual Stages

This ablation studies the effect of residual depth M while keeping the codebook width fixed at $K{=}128$. The goal is to test whether adding more residual stages consistently improves reconstruction and downstream TAS performance, or whether fixed-depth quantization becomes brittle when too many stages are forced onto every frame. As shown in Table 7, fixed-depth RQ-VAE is sensitive to M: moderate depths work best, while very large depth can hurt both reconstruction and task performance. ARQ-VAE is more stable because it can discard unnecessary later stages on a per-frame basis, and ARQ-VAE* performs best overall after adapting the decoder and active codebooks to the retained prefixes.

Table 6: Width ablation with fixed depth $M=8$. Higher is better for task metrics; lower is better for reconstruction metrics. Average and standard deviation are computed across the six tested width settings for each method.

Method	K	Task metrics						Reconstruction metrics		
		Acc	Edit	F1@10	F1@25	F1@50	AVG	MAE	Rel L2	Storage (MB)
RQ-VAE	32	70.5	56.2	63.2	59.1	48.8	59.6	0.104	0.512	**2.433**
	64	68.3	55.4	61.7	58.0	48.5	58.4	0.107	0.516	3.144
	128	72.2	58.6	64.6	61.5	52.3	61.8	**0.106**	**0.510**	4.100
	256	70.5	58.6	64.8	62.0	52.0	61.6	0.107	**0.510**	5.445
	512	**73.3**	60.4	**66.5**	**63.0**	**54.2**	**63.5**	0.122	0.570	7.199
	1024	70.6	**61.3**	66.4	**63.0**	52.8	62.8	0.130	0.610	12.763
	Average	70.9	58.4	64.5	61.1	51.4	61.3	0.113	0.538	5.847
	Std. Dev.	1.56	2.09	1.69	1.91	2.09	1.77	0.010	0.039	3.460
ARQ-VAE	32	72.0	55.1	60.4	56.1	47.7	58.3	0.098	0.485	**1.374**
	64	74.3	**63.5**	**69.7**	**66.5**	**57.2**	**66.2**	0.095	0.469	1.743
	128	73.9	60.6	64.7	61.6	52.4	62.6	0.093	0.458	2.400
	256	**75.0**	62.6	69.0	66.4	56.4	65.9	0.092	0.449	3.464
	512	74.7	60.2	66.7	63.3	54.7	63.9	**0.091**	**0.444**	4.388
	1024	75.0	62.1	67.9	63.7	54.7	64.7	**0.091**	0.445	5.809
	Average	74.2	60.7	66.4	62.9	53.9	63.6	0.093	0.458	3.196
	Std. Dev.	1.04	2.74	3.13	3.51	3.14	2.66	0.002	0.015	1.548
ARQ-VAE*	32	74.3	57.4	61.5	57.7	49.2	60.0	0.095	0.464	1.374
	64	74.2	**65.2**	**70.4**	**67.0**	57.4	**66.8**	0.091	0.444	1.743
	128	75.3	61.7	68.6	65.3	56.8	65.5	0.089	0.432	2.400
	256	76.0	62.5	69.3	65.6	57.2	66.1	0.086	0.420	3.464
	512	**76.4**	63.6	69.7	65.7	**57.9**	66.7	**0.085**	**0.413**	4.388
	1024	74.7	64.1	69.3	65.8	56.6	66.1	**0.085**	0.414	5.809
	Average	75.1	62.4	68.1	64.5	55.9	65.2	0.089	0.431	3.196
	Std. Dev.	0.83	2.51	3.01	3.09	3.00	2.37	0.004	0.018	1.548



(a) Reconstruction error (MAE) versus storage.

(b) TAS performance (F1@50) versus storage.

Figure 8: Storage–performance plots for the width and depth ablations, using reconstruction mean absolute error and downstream temporal action segmentation performance.

H Storage–Performance Plots for Width/Depth Ablations

I Prefix Length Histograms

Fig. 9 shows the distribution of selected ARQ-VAE prefix lengths across the three TAS benchmark datasets. The selected depths are spread across the full range of residual stages rather than collapsing to a single stage, indicating that frames vary substantially in how much residual capacity they need. Many frames are reconstructed with short prefixes, reflecting redundancy in stable action regions, while a nontrivial fraction retain deeper prefixes, corresponding to frames that benefit from additional refinement. This supports the central motivation for adaptive residual quantization: a fixed residual budget is unnecessarily rigid for long procedural videos, where different frames require different amounts of memory.

Table 7: Depth ablation with fixed width $K=128$. Higher is better for task metrics; lower is better for reconstruction metrics. Average and standard deviation are computed across the five tested depth settings for each method.

Method	M	TAS metrics						Reconstruction metrics		
		Acc	Edit	F1@10	F1@25	F1@50	AVG	MAE	Rel L2	Storage (MB)
RQ-VAE	1	69.6	56.7	60.9	57.8	48.8	58.8	0.103	0.506	**0.467**
	2	**73.9**	60.1	65.1	61.5	53.6	62.8	**0.097**	0.478	1.027
	4	73.2	**63.5**	**69.8**	**65.7**	**56.6**	**65.8**	0.098	**0.476**	2.053
	8	72.2	58.6	64.6	61.5	52.3	61.8	0.106	0.510	4.100
	16	63.5	50.7	55.3	51.5	41.7	52.5	0.202	0.926	7.349
	Average	70.5	57.9	63.1	59.6	50.6	60.3	0.121	0.579	2.999
	Std. Dev.	3.82	4.24	4.84	4.84	5.08	4.54	0.041	0.174	2.503
ARQ-VAE	1	69.6	56.7	60.9	57.8	48.8	58.8	0.103	0.506	**0.467**
	2	73.0	56.9	61.6	58.2	50.1	60.0	0.096	0.470	0.914
	4	**73.9**	**62.1**	**66.8**	**63.2**	**53.8**	**64.0**	**0.093**	**0.456**	1.556
	8	73.9	60.6	64.7	61.6	52.4	62.6	**0.093**	0.458	2.400
	16	70.1	55.4	59.8	55.4	47.5	57.6	0.096	0.478	1.881
	Average	72.1	58.3	62.8	59.2	50.5	60.6	0.096	0.474	1.444
	Std. Dev.	1.88	2.58	2.57	2.81	2.31	2.42	0.004	0.018	0.686
ARQ-VAE*	1	69.6	56.7	60.9	57.8	48.8	58.8	0.103	0.506	0.467
	2	73.6	59.5	64.7	61.5	53.2	62.5	0.094	0.458	0.914
	4	**75.4**	**66.8**	**71.3**	**68.3**	**57.4**	**67.8**	0.090	0.437	1.556
	8	75.3	61.7	68.6	65.3	56.8	65.5	**0.089**	**0.432**	2.400
	16	73.9	61.7	67.2	63.9	55.0	64.3	0.092	0.452	1.881
	Average	73.6	61.3	66.5	63.4	54.2	63.8	0.094	0.457	1.444
	Std. Dev.	2.08	3.28	3.48	3.46	3.10	3.04	0.005	0.026	0.686

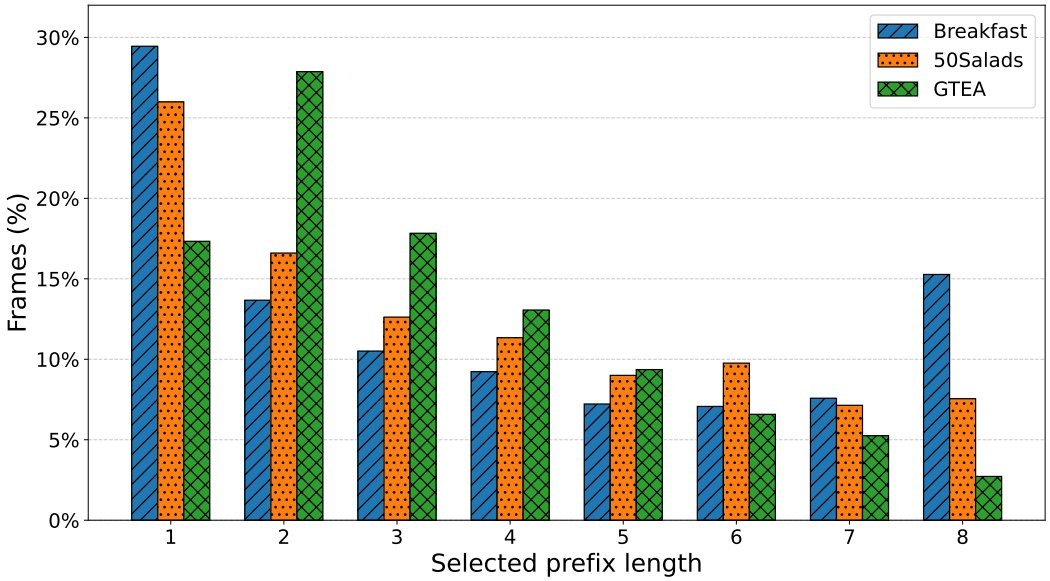


Figure 9: **Distribution of selected prefix lengths** across GTEA, 50Salads, and Breakfast.

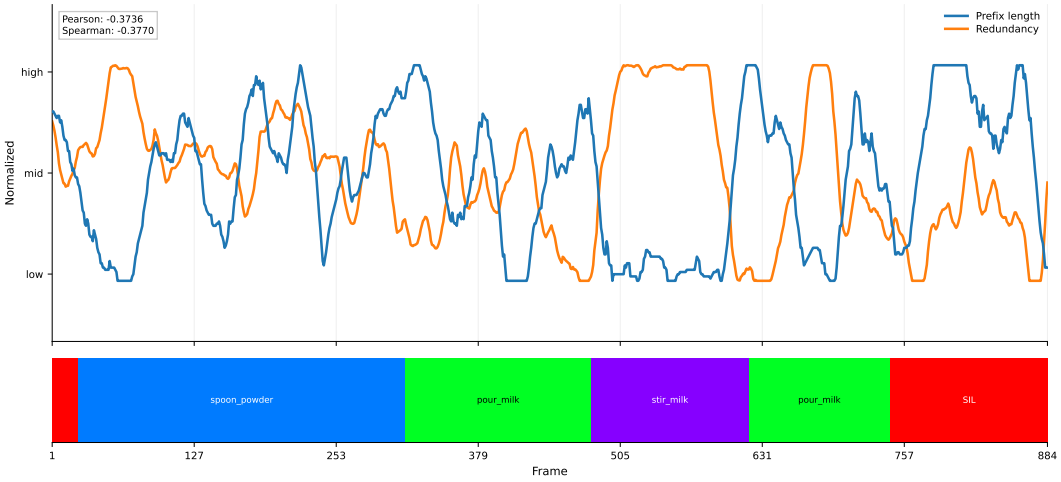
J Prefix Length Tracks Reconstruction Difficulty

To validate the hypothesis that *hard* reconstruction tasks require deeper residual refinement while *easy* and temporally redundant reconstruction tasks can be represented with shorter prefixes, we analyze the frame-wise prefix lengths selected by ARQ-VAE after prefix selection. Recall that ARQ-VAE stores, for each training frame, the residual prefix that achieves the best reconstruction among all decoded prefixes. Under this view, the selected prefix length $m_t^\star$ serves as an operational measure of frame difficulty: frames that are already well reconstructed by shallow prefixes are assigned shorter codes, while frames that continue to benefit from additional residual refinement retain deeper prefixes.

Table 8: **Frame-wise prefix length is inversely correlated with feature redundancy.** *Pred.* denotes local-context predictability and *Var.* denotes local temporal variance. Negative correlation means that more redundant frames tend to use shorter prefixes, and larger redundancy scores indicate greater temporal redundancy.

Dataset	Metric	Mean P↓	Mean S↓	% neg. vids↑	Pooled P↓
50Salads	Pred.	-0.2057	-0.1966	100.0	-0.2163
	Var.	-0.1828	-0.1847	97.5	-0.1951
GTEA	Pred.	-0.2856	-0.2675	100.0	-0.2999
	Var.	-0.2688	-0.2524	100.0	-0.2860
Breakfast	Pred.	-0.2135	-0.2031	90.9	-0.2587
	Var.	-0.2046	-0.1913	88.0	-0.2511

Figure 10: **Example video-level relationship between prefix length and feature redundancy.** Top: normalized trajectories of the selected prefix length and the feature redundancy score over time, with Pearson and Spearman correlations shown in the inset. Bottom: ground-truth action segmentation for the same video. Regions with sustained actions such as `spoon_powder`, `pour_milk`, and `stir_milk` tend to exhibit higher redundancy and shorter selected prefixes, while lower-redundancy portions of the sequence coincide with longer prefixes. This example illustrates the inverse relationship quantified in Table 8: ARQ-VAE allocates more residual depth to temporally harder, less redundant frames and fewer stages to stable action interiors.



Protocol. This analysis is performed on the *training* feature bank, since ARQ-VAE compresses the stored training set and does not reconstruct test features. For this analysis, we globally normalize the pre-extracted frame features using the training-set mean and standard deviation. We then compute two frame-wise redundancy measures. **Pred.** measures *local-context predictability*: a frame is more redundant if it is well explained by the average of its temporal neighbors. **Var.** measures *local temporal variance*: a frame is more redundant if its feature vector lies in a temporally stable window with low variation. To capture how prefix length varies with redundancy over time, we compute Pearson and Spearman correlations between the raw prefix length $m_t^\star$ and each redundancy score within each video. For clarity, any moving-average prefix curve is used only for visualization and is *not* used in the reported statistics. We report mean per-video Pearson and Spearman correlations (Mean P and Mean S), the fraction of videos with negative correlation (% neg. videos), and pooled Pearson correlation across all frames (Pooled P) in Table 8.

Results. Table 8 shows a consistent inverse relationship between redundancy and selected prefix length across all three datasets. The mean per-video Pearson and Spearman correlations are negative in every case. The sign is also highly stable across videos: for both metrics, the fraction of videos

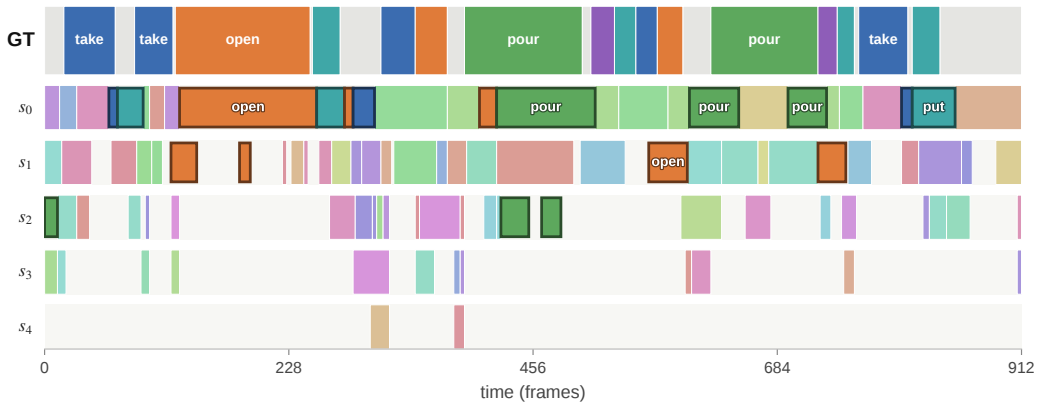


Figure 11: **Residual code reuse on a GTEA example.** The top row shows ground-truth action segments, while lower rows show frequently reused code indices at residual stages $s_0, \ldots, s_4$. Colored intervals indicate retained codes, and greyed-out regions indicate later-stage codes removed by ARQ-VAE prefix truncation. Reused codes often align with repeated actions such as *take*, *open*, and *pour*, suggesting a weak correspondence between discrete codes and recurring procedural structure.

with negative correlation is at least 88.0% on Breakfast and reaches 100% on GTEA. The pooled frame-wise correlations are likewise uniformly negative. Thus, regardless of whether redundancy is measured by temporal predictability or local feature stability, frames that are easier to explain in context tend to terminate after fewer residual stages in ARQ-VAE.

Interpretation. These results provide direct evidence for the intended ARQ-VAE behavior. Prefix length is not assigned using labels, boundaries, or any external notion of frame difficulty; it emerges purely from the reconstruction-based prefix-selection rule. The negative correlations in Table 8, together with the qualitative timeline visualization in Fig. 10, show that ARQ-VAE allocates longer prefixes to rapidly changing or harder-to-reconstruct regions and shorter prefixes to stable, redundant regions. This supports the central narrative of the method: adaptive residual quantization spends capacity where the feature sequence is harder to preserve.

K Qualitative Analysis of Reused Residual Codes

Figs. 11 and 12 provide a qualitative view of how residual codes are reused over time. ARQ-VAE is trained only to reconstruct pre-extracted features, without action labels, but residual quantization still learns a shared discrete vocabulary. In procedural videos, repeated actions often have similar visual-temporal structure, so the same code indices can reappear within semantically related segments.

Each figure shows ground-truth actions on top and reused code indices across residual stages below. The alignment is not one-to-one: a code is not an action label, and an action may use multiple codes depending on context, viewpoint, and temporal phase. Instead, the plots show a weaker pattern: early-stage codes often form coherent intervals that overlap with actions such as *take*, *open*, *pour*, *put*, and *fold*. This suggests that the discrete memory captures recurring feature patterns rather than simply memorizing frame positions.

The figures also visualize adaptive prefix selection. Colored blocks are retained codes, while greyed-out later stages are removed by ARQ-VAE. Early stages tend to preserve coarse action-level structure, whereas later stages are more fragmented and are kept mainly for frames needing extra reconstruction detail. This supports the intuition behind ARQ-VAE: common procedural patterns can reuse compact codes, while harder frames receive deeper residual prefixes only when needed.

L Prefix-Selection Criterion and Rate–Distortion Control

This section expands on Section 4.2.4. All criteria use the same pretrained $M{=}8$, $K{=}128$ RQ-VAE; none retrains the base codec. These diagnostic experiments use a single standard train/test partition

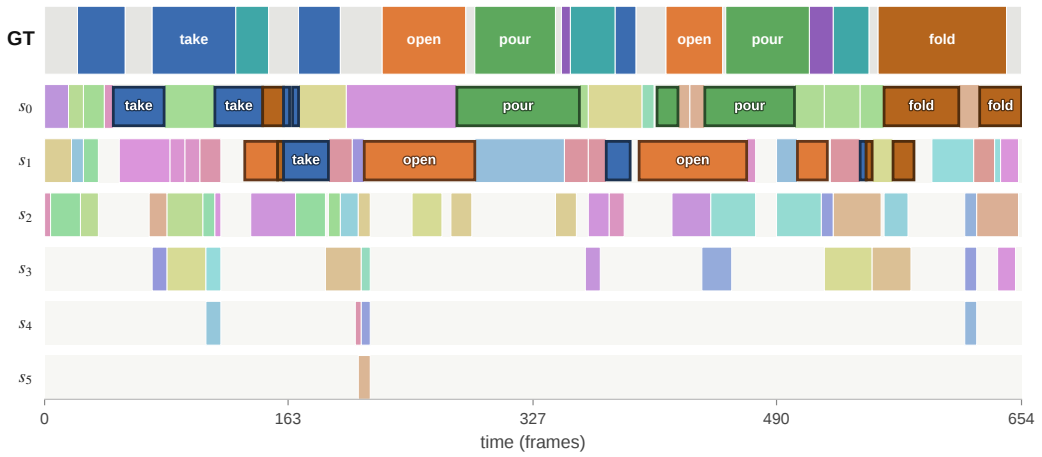


Figure 12: **Residual code reuse on a second GTEA example.** As in Fig. 11, early residual stages contain longer reused code intervals that overlap with major action segments, while later stages are more fragmented and are often discarded by adaptive prefix selection. Repeated codes appear across multiple instances of actions such as *take*, *open*, *pour*, and *fold*.

of 50Salads with the 50-epoch MS-TCN protocol, and TAS models are evaluated on the original, uncompressed test features. Storage follows the paper's convention (indices and active codebooks, decoder excluded), while Complete additionally includes the 2.369-MB decoder. Fig. 6 in the main text summarizes all criteria.

Selection criteria. **Fixed-8** retains all eight stages and is equal to RQ-VAE. **L2 argmin** selects the reconstruction-optimal prefix independently for each frame, Eq. (2), and is equal to ARQ-VAE. **L2-permuted** randomly reassigns the L2-selected depths across frames, preserving the exact depth histogram and storage of ARQ-VAE while removing the frame/depth association. The **label-aware probe** trains a linear frame classifier on the original training features and TAS labels, then selects the prefix with the minimum correct-class cross-entropy; it tests whether local label discriminability provides a better framewise signal than reconstruction error. We use a framewise probe because a standard TAS model such as MS-TCN is temporal: a forward pass requires the full frame sequence, so exactly optimizing per-frame prefix lengths for a segmentation loss could require up to 8^T sequence configurations and is computationally infeasible. The probe is trained on all labeled training frames, including frames near action transitions, but it does not explicitly model temporal ordering, boundaries, or segment-level metrics.

Table 9: **Prefix-selection criteria** on 50Salads with MS-TCN. Storage excludes the decoder; Complete includes it.

Criterion	Mean depth	Storage (MB)	Complete (MB)	MSE ↓	MAE ↓	Acc	Edit	F1@{10,25,50}
Fixed-8 (RQ-VAE)	8.00	5.206	7.574	0.02605	0.10664	69.84	59.87	67.88 / 63.78 / 54.67
L2 argmin (ARQ-VAE)	3.55	**3.152**	**5.521**	**0.02068**	**0.09296**	**74.92**	**62.01**	**69.20 / 65.18 / 57.59**
L2-permuted	3.55	**3.152**	**5.521**	0.02380	0.09900	70.50	58.98	64.41 / 60.81 / 53.15
Label-aware probe	5.13	3.880	6.249	0.02420	0.10087	68.45	51.11	57.52 / 54.10 / 44.57

The cleanest controlled comparison in Table 9 is L2 argmin versus L2-permuted: both use the same depth histogram over all frames and 3.152 MB of storage, yet preserving the reconstruction-derived frame/depth association improves F1@50 from 53.15 to 57.59 and lowers MSE from 0.02380 to 0.02068. The gain is therefore not explained by the average number of retained codes alone; assigning deeper prefixes to the right frames matters. The label-aware probe uses TAS supervision and a task-discriminative loss, but it retains deeper prefixes (5.13 on average) and does not improve selection in this experiment. We do not claim that feature-space L2 is a universal segmentation objective; these results support it as a useful label-free criterion that yields effective frame-specific allocation.

Explicit rate–distortion control. L2 argmin minimizes distortion without an explicit storage penalty. We therefore also select prefixes with the rate–distortion objective

$$m_t(\lambda) = \underset{m \in \{1,\dots,M\}}{\arg\min} \big[D_t(m) + \lambda m \big], \tag{5}$$

where $D_t(m)$ is the reconstruction MSE of the length-m prefix. Since every stage stores one index from a $K{=}128$ codebook, m is proportional to the index rate. All points reuse the same RQ-VAE and cached prefix errors, so no allocator or additional codec training is introduced. Positive λ penalizes rate, $\lambda{=}0$ recovers L2 argmin, and negative values are included as high-rate diagnostics that favor additional stages.

Table 10: **Rate–distortion prefix selection**, Eq. (5), on 50Salads with MS-TCN. Storage excludes the decoder; Complete includes it.

Criterion	Mean depth	Storage (MB)	Complete (MB)	MSE ↓	Acc	Edit	F1@50
Fixed-8 (RQ-VAE)	8.00	5.206	7.574	0.02605	69.84	59.87	54.67
$\lambda{=}10^{-3}$ (lower rate)	2.06	**2.466**	**4.835**	0.02126	68.54	53.75	50.65
$\lambda{=}10^{-4}$ (mild rate penalty)	3.30	3.037	5.406	0.02069	72.64	57.87	56.05
$\lambda{=}0$ (L2 argmin, ARQ-VAE)	3.55	3.152	5.521	**0.02068**	74.92	**62.01**	57.59
$\lambda{=}-10^{-4}$ (mild high-rate bias)	3.82	3.277	5.646	0.02069	**75.28**	60.57	**58.21**
$\lambda{=}-10^{-3}$ (strong high-rate bias)	6.15	4.353	6.722	0.02191	74.67	56.75	54.07

As shown in Table 10, positive λ provides explicit lower-rate control, reducing mean depth and storage with the expected performance trade-off. A mild high-rate bias at $\lambda{=}-10^{-4}$ increases mean depth only from 3.55 to 3.82 and complete storage from 5.521 to 5.646 MB, while marginally improving F1@50 from 57.59 to 58.21. Thus, when the storage budget permits, $\lambda < 0$ can select a slightly higher-rate operating point with improved downstream performance. A stronger bias at $\lambda{=}-10^{-3}$ instead reduces F1@50 to 54.07, showing that retaining more stages is not monotonically beneficial. The λ term therefore provides controllable operating points for different storage budgets.

Why can a shorter prefix reconstruct better? A standard RQ-VAE is optimized for full-depth reconstruction, but pointwise monotonic improvement is not guaranteed once residual corrections learned in latent space are passed through a nonlinear decoder. This is especially relevant for heterogeneous long-video features spanning many scenes, actions, and levels of visual complexity: many frames lie in visually stable action interiors, while others capture transitions, viewpoint changes, or complex motion, and for frames already represented well by a short prefix, later stages may add unnecessary detail in feature space. This behavior does not stem from unstable optimization: in our training, reconstruction loss decreases stably while codebook perplexity increases steadily, with no indication of codebook collapse. Nor is ARQ-VAE merely deleting globally harmful late stages. The redundancy analysis in Appendix J finds systematically shorter prefixes for temporally redundant frames and longer prefixes for harder-to-reconstruct regions, and the L2-permuted control above shows that *which* frames receive deeper prefixes matters. Consistent with this explanation, ARQ-VAE lowers MSE from 0.02605 to 0.02068 and improves F1@50 from 54.67 to 57.59 relative to Fixed-8 RQ-VAE.

Selecting prefixes before fine-tuning. Fine-tuning the decoder and active codebooks in ARQ-VAE* can change which prefix minimizes reconstruction error. ARQ-VAE* deliberately fine-tunes with the selected indices and prefix lengths held fixed, directly adapting the model to the compact representation that will actually be stored. This preserves the stored payload and avoids an alternating procedure in which the selection, and therefore the storage, changes after every update.

M Computational Cost

All computational-cost results in this section use the **40-video I3D training feature bank of 50Salads**, with the $M{=}8$, $K{=}128$ codec and a 50-epoch downstream MS-TCN schedule. These timings describe a single training run. We report codec-related compute separately from downstream MS-TCN optimization. Our protocol decodes compressed features on the fly, so each epoch reconstructs each video batch from its stored codes. When sufficient transient memory is available, the compact bank may alternatively be decoded once before TAS optimization, trading memory residency for

reduced repeated decoding; all timings below use on-the-fly reconstruction. All timings were measured on a single NVIDIA Tesla V100-SXM2 GPU with 32 GB of memory, using PyTorch 2.6.0 with CUDA 11.8 and Python 3.10.18; absolute times will vary across hardware.

Table 11: **Codec-related computational cost** on 50Salads. Codec preparation is a one-time offline cost; decoding reconstructs the complete 40-video training bank once per epoch, and the 50-epoch columns follow the MS-TCN schedule.

Representation	Codec prep (s)	Decode / epoch (s)	Decode, 50 epochs (s)	Codec total (s)
RQ-VAE	**102.91**	3.78	189.00	291.91
ARQ-VAE	119.38	**3.43**	**171.50**	**290.88**
ARQ-VAE*	333.49	3.54	177.00	510.49

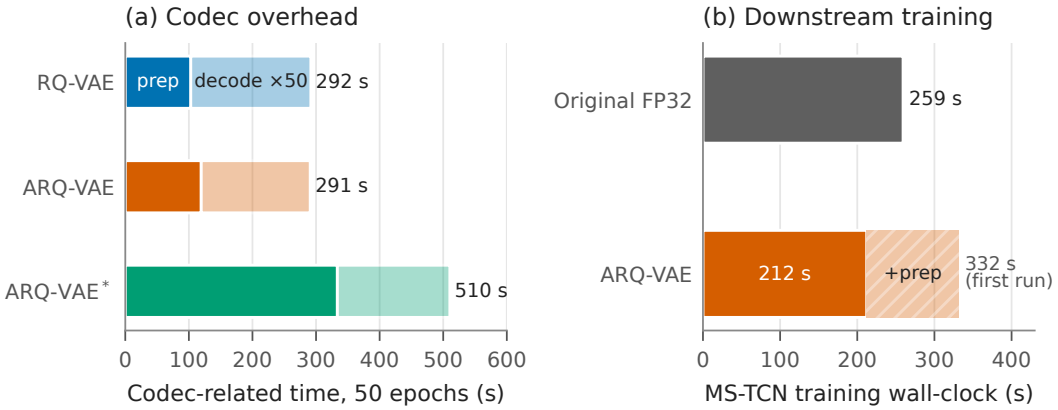


Figure 13: **Computational cost** on 50Salads. **(a)** Codec-related time over the 50-epoch MS-TCN schedule: one-time preparation (solid) plus on-the-fly decoding (light). **(b)** End-to-end MS-TCN training wall-clock, including feature access and on-the-fly reconstruction; the hatched segment adds ARQ-VAE's one-time preparation, paid only on the first run.

For RQ-VAE, codec preparation in Table 11 includes codec training and code extraction. ARQ-VAE adds prefix reconstruction, error evaluation, and selection, an incremental cost of only 16.47 s. ARQ-VAE* additionally fine-tunes the decoder and active codebooks, increasing preparation time from 119.38 s to 333.49 s (about $2.8\times$) while leaving decoding time nearly unchanged. ARQ-VAE reconstructs the complete bank in 3.43 s per epoch, slightly faster than fixed-depth RQ-VAE because it decodes fewer residual stages on average. The codec totals in Table 11 include preparation and repeated decoding, but exclude MS-TCN optimization; decoding is already included in the end-to-end training times and must not be added again. As shown in Fig. 13, under the same 50-epoch schedule, end-to-end MS-TCN fitting took 212.30 s with ARQ-VAE versus 259.43 s with the original FP32 bank, so decoding did not increase the observed downstream training wall-clock in this run. Including the one-time 119.38 s preparation, the first ARQ-VAE run costs 331.68 s; subsequent runs on the same compact bank reuse this preparation.

N Generalization to Alternative Pre-extracted Features

In Table 12, we evaluate whether ARQ-VAE generalizes beyond the I3D feature space by repeating the 50Salads experiments with DINOv3 [32] frame-wise features. The goal of this analysis is not to optimize a new backbone-specific pipeline, but to test whether adaptive residual quantization remains effective when the input feature distribution changes. The results show the same overall pattern observed with I3D: RQ-VAE provides a compact fixed-rate representation, ARQ-VAE reduces storage by truncating unnecessary residual stages, and ARQ-VAE* further improves reconstruction quality after fine-tuning on the retained prefixes. This suggests that the benefit of adaptive residual depth is not specific to I3D, but reflects a more general property of long-video feature banks: many frames can be represented with shallow residual prefixes, while harder frames benefit from additional codes.

Table 12: Results on DINOv3 frame features under the same condensation pipeline. Lower is better for reconstruction metrics and storage; higher is better for TAS metrics. The results show that the proposed method is not specific to I3D features and extends effectively to a different pretrained feature space.

Method	Reconstruction			TAS metrics				
	MSE ↓	MAE ↓	Storage ↓	Acc ↑	Edit ↑	F1@10 ↑	F1@25 ↑	F1@50 ↑
Original	–	–	4.5 GB	**70.4**	**67.9**	**69.6**	**66.8**	**54.3**
RQ-VAE	0.034	0.143	4.1 MB	60.3	53.9	59.1	55.8	41.9
ARQ-VAE	0.017	0.099	**1.8** MB	65.1	56.6	57.9	53.2	41.1
ARQ-VAE*	**0.014**	**0.092**	**1.8** MB	67.2	61.6	63.8	57.9	45.2

O Results with a Recent TAS Backbone (FACT)

ARQ-VAE modifies only the stored training feature bank; the downstream TAS architecture, its inference-time input, and its training objective are unchanged, so it is compatible in principle with any feature-based TAS model. To verify this with a more recent architecture, we integrate FACT [23] into our codebase and run the complete 5-split 50Salads cross-validation for four training banks. Every row uses the same splits, seeds, codec checkpoints, and code pools as our MS-TCN experiments, so the segmentation backbone is the only change; each compressed row trains FACT on the reconstructed feature bank and evaluates it on the original test features.

Table 13: **FACT [23] on 50Salads**, averaged over the five standard splits. Original trains on the full I3D feature bank; the other rows train on reconstructed features.

Method	Acc	Edit	F1@10	F1@25	F1@50
Original	85.10	79.29	84.77	83.08	77.61
RQ-VAE	77.65	70.99	76.30	74.16	66.28
ARQ-VAE	79.13	73.02	79.06	76.30	66.52
ARQ-VAE*	**80.23**	**73.38**	**79.26**	**77.19**	**67.28**

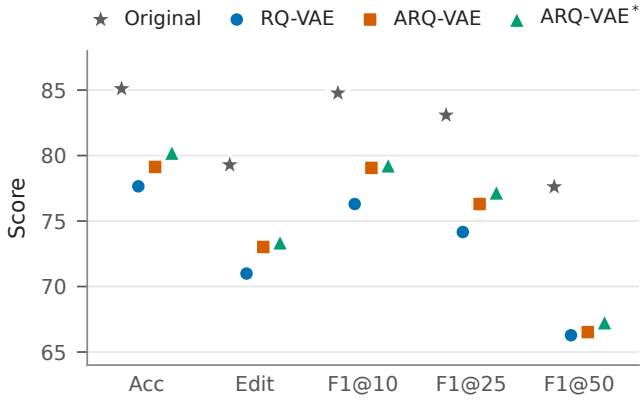


Figure 14: **FACT on 50Salads** (5-split average). The ordering RQ-VAE < ARQ-VAE < ARQ-VAE* observed with MS-TCN and ASFormer is unchanged.

As shown in Table 13 and Fig. 14, ARQ-VAE* is the best of the three stored representations on all five metrics, and it outperforms RQ-VAE in all five splits on Acc, F1@10, F1@25, and F1@50 (+2.58 Acc and +2.96 F1@10 on average). The ordering RQ-VAE < ARQ-VAE < ARQ-VAE* is unchanged from MS-TCN and ASFormer, and swapping MS-TCN for FACT required no change to the method. Moreover, FACT trained on the 2.4 MB ARQ-VAE* memory exceeds MS-TCN trained on the full 4.5 GB feature bank (Table 1) on all five metrics, showing that a stronger architecture and the compact feature bank are complementary.