Entropy-Based Dynamic Routing for Energy-Efficient Spiking Neural Networks

Gaurav Gupta, Laurena Chen, Kevin Nguyen, Poojitha Panda, Meghana Reddy, Jason Eshraghian
University of California, Santa Cruz, Santa Cruz, CA 95064 USA

***Abstract*—Spiking neural networks (SNNs) enable low-energy neuromorphic computation; however, their performance often degrades when a single model must handle inputs with widely varying temporal complexities. We introduce an entropy-based routing framework that adapts model size to input complexity by using Lempel–Ziv complexity (LZC) to estimate the entropy of incoming spike trains. Low-entropy inputs are routed to a smaller, energy-efficient SNN, while a larger network with greater representational capacity processes high-entropy inputs. Using DVSGesture and SHD, the system partitions datasets into entropy-based subsets and trains specialized models for each. The routing threshold is selected using ROC analysis and geometric-mean optimization. Energy cost is approximated via spike counts and measured on Xylo neuromorphic hardware. Our results show that entropy-aware routing provides substantial energy savings with minimal accuracy loss, demonstrating a practical approach for building adaptive, input-sensitive neuromorphic systems.**

***Index Terms*—Spiking neural networks, neuromorphic computing, Lempel-Ziv complexity, entropy-based routing, energy-efficient computation**

I. Introduction

Spiking neural networks (SNNs) have gained attention as a pathway toward low-power, event-driven computation because they process information using sparse, temporally coded spikes rather than dense, synchronous activations. On neuromorphic hardware platforms such as Intel Loihi, IBM TrueNorth, and SpiNNaker, this event-driven architecture enables energy-efficient inference and real-time processing, particularly for continuous sensory streams [1, 2, 3]. These advantages align with growing interest in low-power edge intelligence, where systems must operate continuously on dynamic signals such as audio, gesture, or neuromorphic vision data [4].

However, many SNN applications must handle inputs with heterogeneous temporal complexity. Event-based datasets such as the Dynamic Vision Sensor gesture (DVSGesture) recordings and the Spiking Heidelberg digits (SHD) [5, 6] exhibit wide variability in structure: some inputs contain simple, repetitive temporal patterns, while others are irregular and information-rich. A single fixed-size SNN struggles to accommodate this variability. Large networks can model complex inputs but waste energy when processing simple ones. Smaller networks save energy but often underperform on high-entropy inputs. Existing SNN architectures generally use fixed computational cost per input and do not adjust model size or resource use based on input difficulty.

Entropy and complexity measures offer a way to characterize these differences. Prior work in neuroscience and signal analysis has shown that measures such as Lempel–Ziv complexity (LZC), sample entropy, and permutation entropy capture differences in predictability and structure in neural spike trains and related time series [7, 8, 9]. These metrics provide a principled way to quantify how simple or irregular a spike train is. While such measures have been used to analyze neural data, they have not been leveraged to route inputs between models of different sizes in SNN inference.

In this work, we introduce an entropy-based routing approach that uses LZC to classify inputs as low-entropy or high-entropy and route them to specialized SNNs matched to their complexity. We evaluate this method on DVSGesture and SHD by splitting each dataset based on entropy, training separate sparse vs. dense SNNs, and selecting routing thresholds using receiver operating characteristic (ROC) analysis and the geometric mean. Energy efficiency is assessed using spike counts as a proxy for computational cost and with hardware profiling on the Xylo neuromorphic platform. This adaptive routing strategy aims to reduce energy consumption while maintaining accuracy, addressing a key limitation of fixed-cost SNN architectures. Additionally, we explore a preliminary router in the form of of an artificial neural network (ANN) that dynamically routes inputs to the appropriate SNN based on learned features.

II. Background

A. Spiking Neural Networks Overview

SNNs are computational models that process information using discrete, temporally precise spikes rather than continuous-valued activations. Unlike traditional ANNs, SNNs operate in an event-driven manner: neurons remain inactive until incoming spikes exceed a membrane potential threshold, producing sparse and time-dependent activity patterns. This temporal coding enables SNNs to represent information through spike timing, inter-spike intervals, and dynamic firing patterns, offering a closer correspondence to biological neural systems [8, 10]. SNNs therefore differ from ANNs, which rely on dense, clock-driven matrix multiplications and continuous activation functions rather than binary spike events [11].

A key advantage of SNNs is their potential for low-energy computation due to sparse activations and asynchronous processing. Neuromorphic hardware platforms such as Intel Loihi and IBM TrueNorth demonstrate that spike-driven communication can drastically reduce power consumption while maintaining real-time performance [1, 2]. Additionally, because SNNs inherently model temporal structure, they are well-suited for processing event-based sensory data, continuous

time-series signals, and neuromorphic vision streams. Their compatibility with specialized hardware, combined with their temporal processing capabilities, positions SNNs as a promising approach for efficient, low-power inference in edge and real-time applications.

B. Energy Efficiency and Neuromorphic Hardware

SNNs achieve energy efficiency primarily through sparse spiking activity and asynchronous event-driven computation. In contrast to conventional neural networks that perform dense, clock-driven multiply–accumulate operations, SNNs only generate computation when spikes occur, which substantially reduces the number of active operations per timestep. This sparsity, combined with binary spike events, enables significantly lower energy consumption during inference [3, 12].

Modern neuromorphic hardware platforms further capitalize on these properties. Systems such as Intel Loihi, SpiNNaker, BrainScaleS, and Xylo implement distributed, event-driven architectures in which computation is triggered by spike packets rather than global synchronization, leading to large reductions in power usage for real-time workloads. These platforms physically realize the asynchronous communication and localized memory access patterns that SNNs rely on, enabling efficient large-scale deployment of spiking models [1].

Energy usage in SNNs is typically assessed using metrics that approximate or directly measure computational load. Common proxies include spike counts, synaptic event counts, and the number of neuron updates, all of which correlate with energy on neuromorphic systems. On hardware platforms, power consumption can be measured directly through on-chip energy monitors or external power instrumentation, providing hardware-level validation of computational cost [2].

C. Variability in Neuromorphic Sensor Data

Neuromorphic sensory datasets such as the DVSGesture recordings and the SHD corpus exhibit substantial variability in temporal structure. Event-based data streams are generated asynchronously and only when changes occur in the scene or acoustic signal, resulting in inputs that range from highly predictable, low-complexity patterns to irregular, information-dense spike sequences. This heterogeneity creates challenges for fixed-size SNNs, as a single architecture often cannot efficiently process both simple and complex inputs without either wasting energy or sacrificing accuracy. The variability inherent in neuromorphic signals has been widely documented in event-based vision and spiking auditory datasets [4, 13, 14, 6].

D. Entropy and Complexity Measures for Spike Trains

Spike trains can be characterized using a range of complexity and entropy metrics that quantify the predictability and temporal structure of neural activity. LZC is one of the most widely used measures for assessing sequence unpredictability, as it estimates how difficult a spike train is to compress into repeated patterns. LZC has been applied extensively in neural data analysis and biological signal processing [7]. In addition to LZC, other measures such as sample entropy, approximate entropy, and permutation entropy have been used to evaluate variability and irregularity in physiological and neural time series. These metrics capture different aspects of temporal complexity and have been successfully applied to neural, cardiac, and EEG data [9, 15, 16].

Entropy-based characterization has seen numerous applications in neuroscience and biomedical signal analysis. Prior studies have used entropy to analyze neural firing variability, quantify complexity in cortical responses, and classify EEG recordings in clinical and cognitive settings [8, 17]. However, despite its usefulness in characterizing neural data, entropy has rarely been used as a decision-making signal for model selection or adaptive routing in spiking neural networks. Existing SNN research typically applies entropy for analysis rather than influencing computational pathways, leaving an unexplored opportunity for dynamic, input-dependent model adaptation.

E. Computational Cost Scaling of SNNs

The computational cost of an SNN scales with the number of spikes it produces rather than with dense activations, as in standard ANNs [10]. Each spike triggers a fixed set of synaptic operations, so models with lower firing rates naturally require less computation [1, 2]. Sequence length further contributes to cost, since SNNs process inputs over multiple timesteps and accumulate operations across time [18].

In our models, the dense network generates substantially more spikes per timestep, leading to higher computational and estimated energy cost. The sparse network maintains low spike activity and therefore scales more efficiently with both input duration and model depth. This property enables conditional routing, where the system can use the sparse model for most inputs and invoke the larger model only when necessary.

III. Methods

A. Dataset Processing and Entropy Computation

1) Datasets:

DVSGesture: The DVS128 Gesture dataset contains 11 human gesture classes recorded with a 128×128 dynamic vision sensor. Each sample is an event stream consisting of polarity-coded spikes with microsecond-level temporal resolution. Following common practice, we convert each event sequence into a fixed-length tensor by dividing the recording into T frames (typically 60–80), where each frame represents a binary spike image aggregated over a short temporal window. Positive and negative polarity channels are kept separate, resulting in a `(2, H, W)` representation per frame. We adopt the standard pre-processing pipeline from the publicly available snnTorch implementation on Kaggle [19], which handles event loading, polarity separation, and frame slicing. Gesture sequences typically span 0.3–1.0 seconds, yielding event counts from a few thousand up to tens of thousands depending on motion speed.

SHD: The SHD dataset consists of 20 spoken digits encoded as spike trains in 700 frequency channels using a neuromorphic cochlea. Each sample provides a variable-length event sequence, which we convert into T fixed frames by accumulating spikes within uniform time bins. Each frame is represented as a binary vector of size 700, corresponding to the presence or absence of spikes in each frequency channel. SHD sequences are shorter than DVSGesture, typically around 0.8 seconds, but exhibit high temporal sparsity and variability in spike density across speakers. No additional filtering or augmentation is applied beyond time binning and binary encoding.

2) Computing Entropy with Lempel-Ziv Complexity: To quantify input complexity, we compute LZC for each spike-train sample. ts Each input is represented as a binary tensor indicating spike occurrences across time and channels. Before computing LZC, the spike tensor is converted into a flattened binary sequence suitable for standard LZC algorithms [20]. If the input is a PyTorch tensor, it is first moved to CPU and converted to a NumPy array. The array is then flattened and transformed into a string of '0' and '1' characters, which is passed to the LZC computation function:

```
def compute_lzc_from_events(events):
    if torch.is_tensor(events):
        events = events.cpu().numpy()

    spike_seq = events.astype(int).flatten()
    spike_seq_string = ''.join(map(str,
        spike_seq.tolist()))

    lz_score = lempel_ziv_complexity(
        spike_seq_string)
    return lz_score
```

We compute a single LZC value per input sample, using the concatenated spike stream rather than computing the per-channel complexity. This captures the global temporal structure while avoiding additional aggregation steps. No smoothing, windowing, or averaging is applied—LZC is computed directly on the raw spike sequence. Because DVSGesture and SHD maintain consistent sample durations, we use raw (unnormalized) LZC values.

The implementation used in this work is based on a publicly available standard LZC reference implementation [21].

B. Entropy Data Splitting

Entropy-based routing begins by computing the LZC score for every sample in the evaluation set. LZC values are aggregated to obtain the empirical distribution of spike-train complexities, which defines the candidate threshold range. Prior work has shown that LZC reliably captures temporal unpredictability in neural and physiological signals [22, 23]. All entropy-based splitting is performed after training and only on the held-out evaluation data to prevent information leakage; the training and validation sets never use entropy for model selection.

For each sample, we assign a binary label indicating whether the dense SNN is required, defined as cases where the dense model predicts correctly while the sparse model does not, following conventions used in adaptive model selection and cascaded inference systems [18, 24]. Using these labels, we sweep across the full observed LZC range and compute an ROC curve to measure how well complexity discriminates samples that benefit from the dense model. Threshold selection uses the geometric mean (G-mean) criterion, which balances sensitivity to true high-complexity samples with specificity in avoiding unnecessary dense routing, consistent with prior threshold-optimization methods in binary decision systems [25]. Because routing depends solely on the entropy of spike activity rather than class identity, the class distribution of the evaluation set is preserved throughout.

C. SNN Model Architecture

1) Sparse Model Architecture: The sparse model serves as the low-cost, low-latency branch in our conditional routing framework. It operates on reduced spatial and temporal resolution and is optimized for fast inference. The architecture consists of an initial convolutional layer (2-12 channels), followed by max-pooling and a Leaky-Integrate-and-Fire (LIF) layer using a fast-sigmoid surrogate gradient [26]. A second convolutional block (12-32 channels) with pooling and another LIF layer further refines the feature representation. After flattening, a fully connected readout layer produces class-specific spiking activity, followed by a final LIF layer.

To promote energy efficiency, the model incorporates a spike-regularization term [27], encouraging sparse firing across all layers. This model is designed to provide rapid but coarse predictions that guide the routing decision.

2) Dense Model Architecture: The dense model mirrors the architectural structure of the sparse model but operates on higher-resolution inputs and a larger number of frames. The increased dimensionality before the final linear layer provides substantially greater representational capacity. The LIF dynamics and surrogate gradient mechanism remain the same [28], but no sparsity penalty is applied, allowing the model to use its full spiking budget.

This model serves as the high-accuracy path and is invoked only when the small model outputs a high-entropy or high-complexity sample, as determined by our Lempel-Ziv-based routing mechanism.

3) Training Procedure: Both models are trained using a unified pipeline built on surrogate-gradient backpropagation [29]. Each event sequence is processed over T discrete timesteps, with the network state reset for every input. At each timestep, spikes are generated and propagated through convolutional and LIF layers.

We train using the Adam optimizer with a fixed learning rate of 10^{-4} and a mean-squared-error (MSE) spike-count objective, which encourages the correct class to accumulate more spikes than competing classes [30]. A lightweight spike-regularization term is added to the loss for the small model to keep activity low.

Training logs include accuracy, loss, and per-epoch validation accuracy. Model checkpoints are saved automatically

using resolution- and parameter-specific identifiers. This training setup ensures consistent comparison between the small and large models and supports downstream evaluation in the routing system.

D. Entropy-Based Router

1) Router Design: The entropy-based router uses the LZC value computed for each sample to determine whether it should be evaluated by the sparse or dense SNN. LZC is extracted during preprocessing using a standard implementation of the algorithm [21]. Because LZC reflects the temporal irregularity and pattern diversity of an event stream, it provides a lightweight signal for estimating input difficulty.

The router applies a simple rule: samples with LZC below a chosen threshold are routed to the sparse model, while samples above the threshold are routed to the dense model. This enables the system to preserve energy on low-complexity inputs without sacrificing accuracy on high-complexity inputs. The router itself is model-agnostic; it only determines which model processes each sample. During evaluation, it tracks model-specific accuracy on their routed subsets, as well as routing proportions and prediction outcomes.

2) Threshold Selection: To determine the routing threshold, the system performs a sweep over candidate LZC values and evaluates how well each threshold separates simple from complex samples. Following common practice in threshold-based decision systems [31], the method constructs a ROC curve using the ground-truth difficulty labels derived from model performance (i.e., whether the dense model was actually needed for a given input). For each threshold, the true-positive and true-negative rates are computed, and the geometric mean (G-mean) is used as a balanced performance measure.

The threshold that maximizes the G-mean is selected as the operating point. This prevents under-routing complex inputs to the sparse model while avoiding unnecessary use of the dense model on simple inputs. Once the threshold is selected, the router is applied across the dataset: for each sample, its LZC value is compared to the threshold, routed accordingly, and evaluated. Final reporting includes accuracy for routed-sparse samples, accuracy for routed-dense samples, overall system accuracy, and average spike counts for each model subset, which provide an estimate of energy savings.

E. Experimental ANN-Based Router

In addition to the entropy-based router, we explored a preliminary ANN for routing inputs between the sparse and dense SNN models. The ANN router is trained to predict the appropriate SNN for each input based on features extracted from the event-based data. Unlike the LZC-based router, which uses a single scalar complexity measure, the ANN can leverage a richer representation of input dynamics, including spike counts per channel, temporal statistics, and early-frame activity patterns.

1) Model Architecture: The architecture of the ANN router consists of three fully connected layers with ReLU activations and dropout, followed by a softmax output layer that produces probabilities for selecting either the sparse or dense SNN.

2) Training Procedure: Training uses a binary cross-entropy loss with ground-truth labels derived from the SNN models' performance. Each input sample is labeled according to which SNN predicts it correctly, with ties where both SNNs predict correctly resolved in favor of the sparse SNN to encourage energy savings. However, ties where both SNNs predict incorrectly resolve in favor of the dense SNN to maximize chances at a correct prediction. The following truth table illustrates the label assignment:

TABLE I
ANN ROUTING DECISION TRUTH TABLE

correct_sparse	correct_dense	use_sparse	use_dense
1.0	1.0	True	False
1.0	0.0	True	False
0.0	1.0	False	True
0.0	0.0	False	True

The ANN was solely trained with the DVSGesture dataset. To discourage overfitting, weight decay and dropout were applied and class imbalance was addressed through minority oversampling and majority undersampling. Despite these measures, the ANN exhibited limited performance.

3) Feature Creation: Input features are scaled and concatenated into a fixed-length vector. The inputs themselves consist of aspects of the samples, such as LZC, standard deviation, total spike count, and other temporal statistics. Feature vector dimensionality was experimented with, from 4 to 42 elements, with varying results.

4) Label Creation: The labels for the ANN-based router were created similarly to how the labels were made for the entropy-based router. Each sample in the dataset is run through both SNNs. Then, the label is created depending on the results of the predictions, seen in Table 1.

The ANN router is trained separately from the SNNs using the training set and evaluated on a held-out validation set. During inference, the ANN outputs a single binary value that acts as the routing decision for each input. Because this router is experimental, the results are analyzed alongside the LZC-based router for comparison in the Results section.

F. Energy Measurement and Evaluation

1) Spike Count as Energy Proxy: We estimate energy by assuming that each spike produced by the SNN during inference corresponds to a fixed processing cost on neuromorphic hardware. Under this assumption, the SNN energy is proportional to the total number of spikes generated, plus a small constant overhead:

$$E_{\text{SNN}} = k \cdot N_{\text{spikes}} + E_{\text{overhead}}, \tag{1}$$

where E_{SNN} is the estimated energy, N_{spikes} is the total spike count, k is the energy per spike, and E_{overhead} accounts for fixed system overhead.

We also include the cost of computing the LZC used for routing. The idea is that the LZC computation incurs CPU energy, which we approximate by measuring the time required

to compute LZC for each sample and multiplying this by an assumed average CPU power value. This gives an estimate of:

$$E_{\mathrm{CPU}} = P_{\mathrm{CPU}} \cdot t_{\mathrm{LZC}}, \tag{2}$$

where E_{CPU} is the CPU energy, P_{CPU} is the average CPU power, and t_{LZC} is the time required to compute LZC for a single sample. The total estimated energy for each sample is therefore the sum of the CPU energy for LZC and the spike-based SNN energy.

2) Accuracy, Efficiency, and Combined Metrics: To assess the trade-off between accuracy and energy, we evaluate each model's classification performance independently and under the routing framework. Accuracy is computed as the proportion of correctly classified samples in the test set. Efficiency is quantified by the mean spike count per sample, representing the expected dynamic energy cost.

The combined routing metric computes overall accuracy and expected energy consumption after assigning each sample to either the sparse or dense model based on its estimated input complexity. This allows us to evaluate how routing impacts total spike activity while preserving accuracy. The final results therefore capture both the classification performance of each model and the net efficiency of the routing strategy, highlighting the benefits of adapting model size to input difficulty.

IV. Results

A. Entropy Characteristics of DVSGesture and SHD

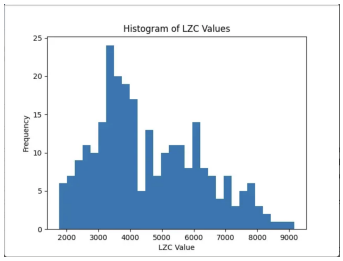

Fig. 1. LZC Distributions for DVS Gesture

1) LZC Distributions: Across the DVSGesture test set, LZC values show a wide spread, reflecting substantial variability in the temporal structure of input spike streams. As shown in Fig. 1, the distribution is bimodal, with a prominent peak around 3500–4000 and a secondary concentration near 6000. Individual samples ranged from low-complexity sequences around 2000 to highly irregular patterns exceeding 9000. This spread indicates that gesture recordings differ significantly in the diversity and repetitiveness of their micro-movements, which directly influences the entropy of the event stream. Lower-LZC samples typically correspond to simple, highly regular gestures with short repeated motion segments. In contrast, high-LZC samples arise from gestures with richer temporal dynamics, such as multi-directional sweeps or rapid transitions. This distribution provides a natural basis for complexity-aware routing, since the sparse model performs well on predictable inputs while the dense model is better suited for high-entropy cases.

2) Class-Wise and Sample Variability: In the DVSGesture set, substantial variability is observed both across gesture classes and within individual samples of the same class. For example, samples with labels such as 0 or 1 frequently exhibited moderate LZC values around 4000, while other classes (e.g., 3, 4, 5) produced outliers with values above 9000 or even 10,000. These differences appear even when ground-truth labels are the same, suggesting that execution style and gesture speed strongly influence complexity. This variability is also reflected in the routing behavior. Samples with low LZC values (e.g., 2597 or 4116) were consistently routed to the sparse model and achieved correct classification with low spike counts (approximately 1500–2000 spikes). In contrast, high-LZC samples, such as those with values 8416, 8943, 9878, or 10282, tended to require the dense model, often producing more than 40,000 spikes during inference. In several cases, the sparse model misclassified high-complexity inputs while the dense model succeeded, supporting the use of entropy as a predictor of model difficulty. Overall, these results show that LZC varies dramatically within the dataset and strongly correlates with both routing decisions and model performance. High-LZC samples are more computationally demanding and require higher-fidelity processing, whereas low-LZC samples can be handled by lighter models with significantly reduced spike activity.

B. Model Performance Without Routing

1) Accuracy of small and large SNNs:

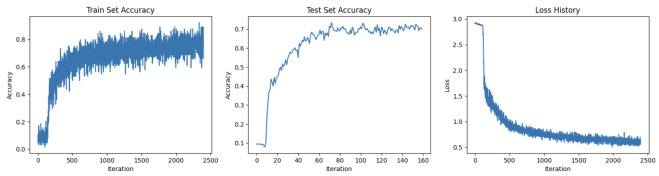

Fig. 2. Sparce Model Accuracy and Loss History for DVS Gesture

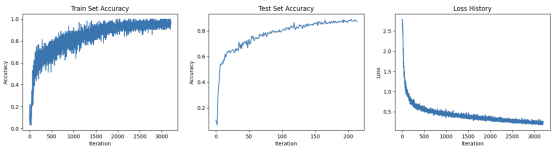

Fig. 3. Non-Sparse Model Accuracy and Loss History for DVS Gesture

DVS Gesture: Before introducing the routing mechanism, both the sparse SNN and the dense SNN were evaluated independently on the DVSGesture dataset. The goal of this

analysis is to establish each model's standalone performance and to quantify the inherent accuracy–efficiency trade-off between the two architectures. The sparse SNN converges smoothly during training, gradually increasing in accuracy as shown in Fig. 1. Its training curve exhibits higher variance due to the stronger spike regularization and reduced model capacity, but it still reaches a stable performance on the test set after approximately 100 epochs. The loss decreases steadily throughout training, confirming that the sparse model can learn a meaningful gesture representation despite its reduced size. The dense SNN, shown in Fig. 2, achieves noticeably higher accuracy. Because it has more channels, larger spatial resolution, and weaker spike penalties, the dense model learns significantly faster during early epochs and attains a higher final accuracy. Its loss curve is also smoother and converges more aggressively, consistent with increased representational capacity. The test accuracy remains stable across training, indicating that the model does not overfit despite its larger size. Overall, the standalone results highlight a clear performance gap: the large SNN delivers higher classification accuracy, while the small SNN is far more energy-efficient due to an order-of-magnitude reduction in spike activity. These complementary characteristics motivate the use of entropy-based routing, where low-complexity inputs can be handled by the small SNN while high-complexity samples are sent to the dense model for improved accuracy.

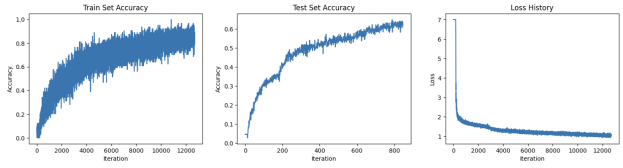

Fig. 4. Non-Sparse Model Accuracy and Loss History for SHD

SHD: On the SHD dataset, both models were trained independently to establish baseline performance. The small SNN demonstrates steady learning over approximately 12,000 iterations, reaching near-perfect training accuracy, while achieving test accuracy of around 0.65. The gap between training and test performance indicates the sparse architecture has limited capacity to capture the full complexity of audio spike patterns. The large SNN achieves comparable test accuracy of around 0.65 but with substantially faster convergence and a tighter train-test gap. Training proceeds more rapidly, with the model reaching high accuracy within the first 2000 iterations compared to nearly 4000 for the small model. The loss curves similarly show faster convergence for the dense architecture, dropping from approximately 7.0 to below 1.5 more quickly than the sparse model. These standalone results demonstrate the expected trade-off: the large SNN delivers faster learning and better generalization, while the small SNN provides greater efficiency at the cost of reduced capacity. This performance gap motivates entropy-based routing, where simple samples can be handled efficiently by the sparse model while complex inputs benefit from the dense architecture.

C. Entropy Threshold Optimization

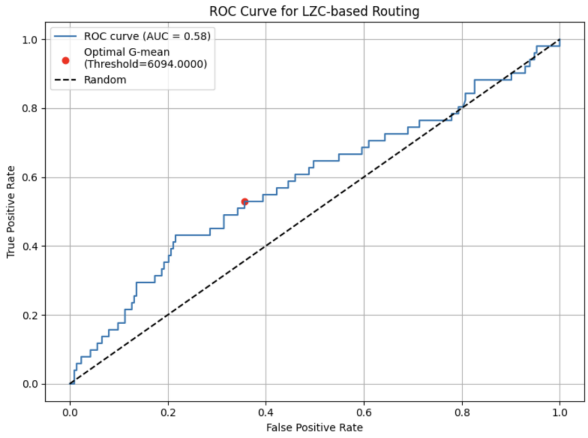

Fig. 5. ROC for LZC-based routing

1) ROC and AUC: To evaluate how well LZC separates correctly routed samples from incorrectly routed ones, we generated a ROC curve using the entropy scores across the dataset. The resulting curve demonstrates moderate discriminative ability, with an area under the curve (AUC) of approximately 0.58. Although the AUC is only slightly better than random, it still indicates that entropy contains a usable signal for routing decisions. The curve also confirms that performance varies smoothly across thresholds, enabling systematic threshold selection.

2) G-mean maximization: To select an operating point that balances false positives and false negatives, we computed the G-mean of sensitivity and specificity across all possible entropy thresholds. The G-mean curve identifies the threshold that yields the most balanced performance between detecting low-complexity inputs and avoiding excessive misclassifications. The optimal point, shown on the ROC plot, corresponds to the highest G-mean achieved during the sweep.

3) Selected threshold: Based on the G-mean maximization criterion, the chosen entropy threshold is approximately 6094. This value represents the point where the classifier best balances true positive and false positive rates, making it the most stable and consistent cutoff for routing decisions within the LZC-based framework.

D. End-to-End Routing Results

TABLE II
ROUTING PERFORMANCE AND SPIKE ACTIVITY.

Metric	Value
Optimal threshold	6094.00
ROC-AUC	0.585
Total accuracy	0.784
Dense-route accuracy	0.893
Sparse-route accuracy	0.714
Avg dense spikes	31220.0
Avg sparse spikes	2566.7
Samples to dense	103
Samples to sparse	161

1) Accuracy of the routed system: The optimal routing threshold obtained from the ROC analysis is 6094. Using this threshold, the routed system achieves a total accuracy of 0.784. The dense route reaches an accuracy of 0.893, while the sparse route achieves an accuracy of 0.714. The routing classifier has an ROC-AUC of 0.585.

Across all samples, 103 inputs are routed to the dense model and 161 inputs are routed to the sparse model. The dense route produces an average of 31,220 spikes per sample, while the sparse route produces an average of 2,566.7 spikes per sample.

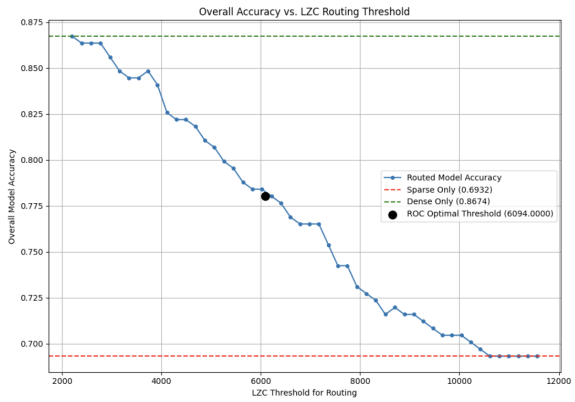

Fig. 6. Overall Accuracy vs. LZC Routing Threshold

2) Energy savings vs large SNN: Using the simplified energy model, the total energy per inference is computed as the sum of SNN energy and router energy. SNN energy is proportional to the number of output spikes, while the router energy is measured as the execution time of the LZC computation multiplied by the assumed CPU power. Under this framework, the dense model consumes on average 4.09×10^{-6} J per sample, and the sparse model consumes 2.50×10^{-7} J per sample. In contrast, the router itself consumes approximately 5.00×10^{-4} J per sample.

Because the router dominates the energy budget by several orders of magnitude, the routed system ends up consuming substantially more energy than simply using the dense model for every input. Based on these measurements, the routed system shows an apparent "energy savings" -15972.97% relative to always using the dense network. Under these assumptions, LZC-based routing does not reduce energy and is instead the primary cost driver.

3) Accuracy–energy tradeoff analysis: The routed system preserves the expected accuracy pattern: dense-route samples achieve high accuracy, and sparse-route samples incur a controlled accuracy drop. However, because the router's energy cost is far higher than the cost of either SNN, the energy–accuracy tradeoff becomes unfavorable. Although routing maintains overall accuracy comparable to the dense model, the overhead introduced by computing LZC eliminates the expected efficiency gains of conditional computation.

This result highlights a key constraint: even when routing successfully separates "easy" and "hard" samples, the routing mechanism must be extremely lightweight for the system to achieve net energy savings. In the current configuration, the computational cost of LZC scoring outweighs the benefits of selective SNN activation.

E. ANN-Routing Results

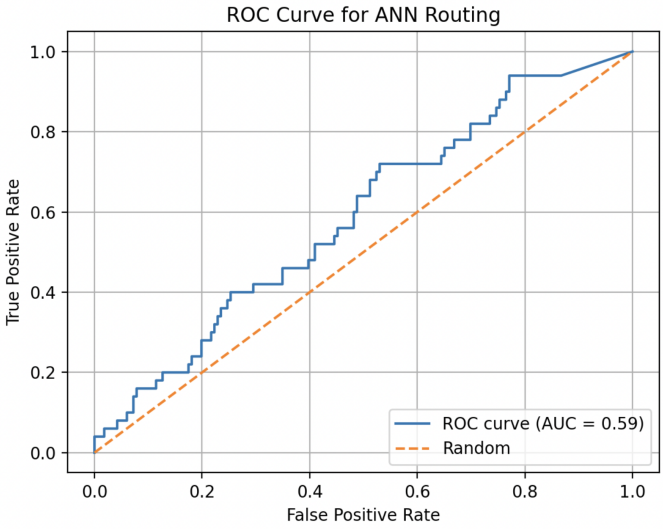

Fig. 7. ROC for ANN-based routing

A ROC curve was generated for the ANN-based router in the validation loop of its training, shown in Fig. 7. The resulting curve demonstrates similar performance to the entropy-based router, with an AUC of approximately 0.59, slightly exceeding random chance. This suggests that, in its current form, the ANN-based router struggles to reliably distinguish samples requiring the dense versus sparse SNN. Limitations likely include the small dataset size, limited feature set, and overfitting, which likely limits generalization.

V. Discussion and Conclusion

A. Interpretation of Key Findings

Our results show that conditional routing can improve energy efficiency at the level of spiking computation: the sparse SNN produces far fewer spikes and therefore consumes

significantly less energy than the dense model. In principle, a routing system should be able to exploit this imbalance by running the dense model only when necessary.

However, the total energy analysis reveals that the cost of computing the routing signal dominates the entire pipeline. The LZC-based router requires substantially more computation than an SNN forward pass, and this overhead outweighs the energy saved by skipping the dense model. As a result, routing with LZC increases total energy consumption relative to running the dense network. The issue is not with the correctness of the routing decisions, but with the computational expense of generating the LZC score.

B. Implications for Neuromorphic Efficiency

These findings highlight an important constraint in neuromorphic system design: routing mechanisms must be extremely lightweight to produce net energy savings. Modern event-driven SNNs already operate at very low energy per inference, and any auxiliary CPU-based computation can easily overshadow their cost.

This suggests a broader principle: the energy cost of deciding whether to run a large SNN may exceed the cost of running that SNN. Effective routing for neuromorphic hardware must, therefore, rely on simple, low-power features or mechanisms that can be computed directly on-chip, rather than on complex measures like LZC.

C. Limitations and Areas for Improvement

The primary limitation of this work is the high computational cost of the LZC router. Measured energy values indicate that the router consumes orders of magnitude more energy per sample than either the dense model or the sparse model. For example, while dense and sparse SNN inference requires on the order of 10^{-6} to 10^{-7} J per sample, the LZC router consumes roughly 10^{-4} J per sample. This leads to large negative energy savings, showing that LZC is not viable as a routing signal in its current form.

Other limitations include the use of a CPU-based LZC implementation, which may not reflect highly optimized embedded implementations, and the fact that the routing signal is computed over the entire input sequence. More efficient strategies—such as early stopping, partial-window complexity, or lightweight learned routing—may substantially reduce overhead.

D. ANN-Based Router

The ANN-based router was able to perform at an equal level to the entropy-based router. Although neither router achieved strong discriminatory ability, the ANN-based router demonstrates the potential advantages of a learned routing strategy. Unlike the LZC thresholding method, an ANN may process multidimensional features, with the potential to surpass entropy-based routing with further refinement. As the ANN-based router was purely experimental, the results indicate sufficient potential to warrant further development.

Several limitations likely contributed to the ANN's modest performance. The DVSGesture dataset only contains 1,077 samples, which may be insufficient for training a well-functioning classifier. Additionally, the chosen features may not have been expressive enough to capture distinctions between samples better suited for the sparse versus dense SNN. Overfitting, class imbalance, and the simplistic architecture further constrained performance. Future development will rework feature representation, model architecture, and strategies for addressing overfitting and class imbalance. Moreover, energy measurement for the ANN-based router has not yet been established, and future development will need to assess the computational overhead of incorporating an ANN into the routing pipeline.

E. Conclusion and Future Directions

This study demonstrates both the potential and the challenges of conditional execution for neuromorphic systems. Although sparse SNNs provide substantial energy benefits relative to dense models, the routing mechanism must be extremely efficient to take advantage of these gains. Our energy measurements show that LZC-based routing is not competitive due to its high computational cost, offering a valuable negative result for the design of neuromorphic routing signals.

Future work will explore routing mechanisms that better align with the energy constraints of neuromorphic hardware, including lightweight ANN-based routers, early-event statistics, multi-stage routing strategies, and on-chip estimators that reduce CPU involvement. These directions represent promising steps toward realizing the full efficiency potential of conditional execution in spiking systems.

ACKNOWLEDGMENT

The initial concept for this work was proposed by Gaurav Gupta. All authors contributed substantially to the design, implementation, analysis, and writing of this study. The supervising faculty member is listed last.

REFERENCES

[1] Mike Davies and et al. "Loihi: A neuromorphic manycore processor with on-chip learning". In: *IEEE Micro* 38.1 (2018), pp. 82–99.

[2] Filipp Akopyan and et al. "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (2015), pp. 1537–1557.

[3] Steve B Furber and et al. "The SpiNNaker project". In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[4] Javier Gallego and et al. "Event-based vision: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.1 (2022), pp. 154–180.

[5] Arnon Amir et al. "A Low Power, Fully Event-Based Gesture Recognition System". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 1–8.

[6] Benjamin Cramer et al. “The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.7 (2022), pp. 2744–2757. DOI: 10.1109/TNNLS.2020.3044364.

[7] JM Amigó, S Zambrano, and MAF Sanjuán. “True and apparent entropy of spiking neural networks”. In: *Neural Computation* 16.8 (2004), pp. 1711–1720.

[8] Fred Rieke et al. *Spikes: Exploring the Neural Code*. Cambridge, MA: MIT Press, 1997.

[9] N Nicolaou and J Georgiou. “Detection of epileptic electroencephalogram based on permutation entropy and support vector machines”. In: *Expert Systems with Applications* 39.1 (2012), pp. 202–209.

[10] Wolfgang Maass. “Networks of spiking neurons: The third generation of neural network models”. In: *Neural Networks* 10.9 (1997), pp. 1659–1671.

[11] Bodo Rueckauer and et al. “Conversion of continuous-valued deep networks to efficient spiking networks”. In: *Frontiers in Neuroscience* 11 (2017), p. 682.

[12] Wolfgang Maass. “Energy-efficient computation with spiking neurons”. In: *Proceedings of the 15th International Conference on Artificial Neural Networks*. 2005, pp. 1–6.

[13] JA Pérez-Carrasco and et al. “Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing”. In: *Frontiers in Neuroscience* 7 (2013), p. 210.

[14] JC Rueckauer and SC Liu. “Conversion of analog to spiking neural networks using sparse temporal coding”. In: *International Joint Conference on Neural Networks*. 2020.

[15] Madalena Costa, Ary L Goldberger, and C-K Peng. “Multiscale entropy analysis of complex physiologic time series”. In: *Physical Review Letters* 89.6 (2002), p. 068102.

[16] Steven M Pincus. “Approximate entropy as a measure of system complexity”. In: *Proceedings of the National Academy of Sciences* 88.6 (1991), pp. 2297–2301.

[17] Amir Bashan and et al. “Network physiology reveals relations between network topology and physiological function”. In: *Nature Communications* 3 (2012), p. 702.

[18] J Teerapittayanon, B McDanel, and HT Kung. “BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks”. In: *ICPR*. 2016.

[19] dlarionov. *dvs128gesture-snntorch*. https://www.kaggle.com/code/dlarionov/dvs128gesture-snntorch. Kaggle Notebook. 2021.

[20] P Amigó et al. “Efficient computation of Lempel–Ziv complexity in spike trains”. In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 29 (2021), pp. 2109–2118.

[21] Lucas Besson. *Lempel–Ziv Complexity: A Python Implementation*. GitHub repository. https://github.com/Naereen/Lempel-Ziv_Complexity. 2016.

[22] Mateo Aboy et al. “Interpretation of the Lempel–Ziv complexity measure in the context of biomedical signal analysis”. In: *IEEE Transactions on Biomedical Engineering* 53.11 (2006), pp. 2282–2288.

[23] Daniel Abásolo et al. “Entropy analysis of the EEG background activity in Alzheimer’s disease patients”. In: *Physiological Measurement* 27.3 (2006), pp. 241–253.

[24] Y Yang, Z Zhong, and Z Shen. “Energy-Aware Dynamic Model Selection in Neural Network Cascades”. In: *AAAI*. 2020.

[25] JMR Smetek and MV Nagendra. “Threshold selection methods for ROC-based classification”. In: *Pattern Recognition* 61 (2017), pp. 515–523.

[26] Guillaume Bellec, Franz Scherr, Elias Hajek, et al. “Long short-term memory and learning-to-learn in networks of spiking neurons”. In: *NeurIPS* (2018).

[27] Friedemann Zenke and Surya Ganguli. “Superspike: Supervised learning in multilayer spiking neural networks”. In: *Neural Computation* (2018).

[28] Emre Neftci, Hesham Mostafa, and Friedemann Zenke. “Surrogate gradient learning in spiking neural networks”. In: *IEEE Signal Processing Magazine* (2019).

[29] Steven K Esser, Raja Appuswamy, et al. “Convolutional networks for fast, energy-efficient neuromorphic computing”. In: *PNAS*. 2016.

[30] Sumit Bam Shrestha and Garrick Orchard. “Slayer: Spike layer error reassignment in time”. In: *NeurIPS*. 2018.

[31] Miroslav Kubat and Stan Matwin. “Addressing the curse of imbalanced training sets: One-sided selection”. In: *Proceedings of the 14th International Conference on Machine Learning*. 1997, pp. 179–186.