

Take Home Assignment 4

Member #1: Arya Bhavesh Shah

Roll No: 25290002

arya.shah@iitgn.ac.in

Question 1 — Conceptual / design (50 marks) - submit as PDF

You are designing a small neuromorphic vision front-end for an always-on camera that should detect a small set of visual events (e.g., “person present”, “no person”, “motion only”). The camera is energy-constrained and should avoid sending raw frames—only event-like/feature information should be transmitted.

1. (10 marks) **Encoding choice:** SNNs use various strategies to encode information into spike trains. The most popular ones are rate encoding and temporal encoding. Compare these spike-encoding strategies for static camera frames: (a) rate-based Poisson encoding (image pixel intensity → spike-rate), and (b) latency/temporal encoding (brighter pixels spike earlier). For each:
 - Describe how encoding is computed.
 - List pros/cons for energy, latency of decision, and robustness to noise.
 - Which encoding would you choose if the system must quickly detect a rare event? Why?
2. (10 marks) **Network architecture:** Propose an SNN architecture (layers, neuron types, connectivity, readout) for the front-end. Be explicit about:
 - Number of layers and their roles.
 - Types of neurons (LIF, adaptive LIF, etc.) and why.
 - Kind of connectivity: sparse random connectivity, or another connectivity motif.
 - How the readout/classifier will be implemented (spiking readout or rate-based classifier after spike accumulation).
3. (10 marks) **Plasticity & adaptation:** Describe what learning rules (eg. STDP) you would deploy and where. Explain:
 - Which rule is used for which synapses and your motivation.
 - How you would adapt the network to new environments after deployment without catastrophic forgetting.
4. (10 marks) **Evaluation & metrics:** Propose an evaluation plan:
 - What datasets or data-collection protocol would you use (synthetic events? real camera data?).
 - Metrics (latency to detection, energy estimate: spikes/sec or average firing rate, accuracy/F1, false alarm rate).
 - Ablation studies you would run (e.g., disable homeostasis, change encoding) and what you expect to observe.

Question 1: Solution

Reading the problem statement, I figure that the goal is to translate pixels into spikes with minimal latency and energy, while maintaining robustness.

Encoding

Spiking neural networks operate on events, not frames. The encoder determines what information is preserved, how fast decisions can be made, and how many spikes (energy) the system spends per inference. This is why encoding matters

Candidate strategies

Rate-based Poisson encoding (reference design)

The idea is to map pixel intensity to a firing rate; spikes are sampled as a Poisson process over a window.

Computation: for each pixel with normalized intensity $x \in [0,1]$, target rate $r = x \cdot f_{\max}$. Per time-step Δt , emit a spike with probability $p = r \cdot \Delta t$.

Pros	Cons
resilient to noise via temporal averaging; simple to implement.	needs many spikes and a long window (high latency, energy hungry).

Latency-based temporal encoding (selected)

The idea is to encode intensity as time-to-first-spike (TTFS). Brighter pixels fire earlier; darker pixels later.

Computation: for pixel intensity x , within $[T_{\min}, T_{\max}]$:

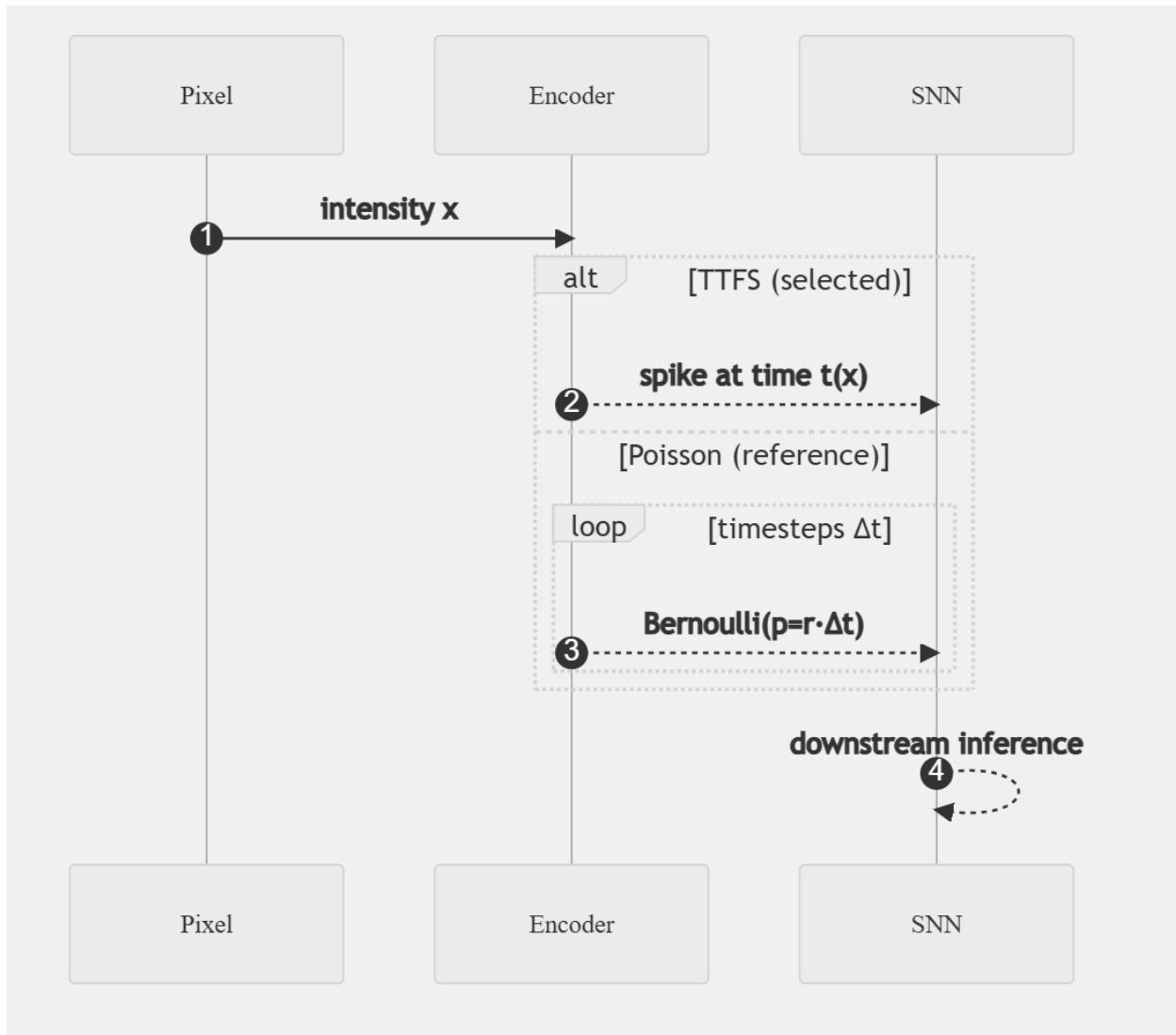
- $t = T_{\max} - x \cdot (T_{\max} - T_{\min})$
- emit a single spike at time t (or none if below threshold).

Pros	Cons
minimal spikes (energy), early decisions (low latency), aligns with event-camera principles.	sensitive to timing noise and jitter, must be mitigated architecturally.

I have selected Latency-based TTFS for end-to-end low-latency, low-energy operation.

Mitigations for fragility: adaptive neurons (ALIF) to stabilize firing, first-spike pooling for invariance, temporal competition to suppress spurious spikes.

The practical choices of parameters such as short windows (20–30 ms) keep latency bounded; sub-ms–1 ms discretization matches plausible hardware tick rates. Contrast thresholds prevent background flicker from generating spurious spikes; optional gamma adjusts perceptual saliency.



Rationale behind my choices

1. The target is rapid event detection with minimal energy. TTFS allows decisions upon the earliest informative spikes, directly minimizing time-to-decision and spikes per inference. Poisson rate codes, by contrast, demand long windows to estimate rates reliably, inflating latency and spike traffic.

2. From the perspective of hardware efficiency, one spike per neuron (typical in TTFS) minimizes synaptic operations and memory reads/writes, dominant contributors to energy on neuromorphic hardware. Rate codes scale SOPs with window length and mean rate.
3. To meet robustness, TTFS is timing-sensitive. We explicitly mitigate this via ALIF adaptation (raising effective thresholds after spikes), first-spike pooling (suppressing late/jittery spikes), and contrast thresholds (avoiding low-SNR inputs). These measures target the primary failure modes of temporal codes without sacrificing their latency advantage.
4. From a future perspective, TTFS harmonizes with event cameras (DVS), which already emit sparse asynchronous events; the encoding stage becomes a pass-through, removing conversion cost and further improving energy efficiency as hardware evolves.

Architecture

While thinking about the architecture, the design goal set by me is pretty clear: extract just enough information to decide among a few events (e.g., person/no-person/motion) with minimal spikes, minimal latency, and without sending raw frames.

High Level Idea

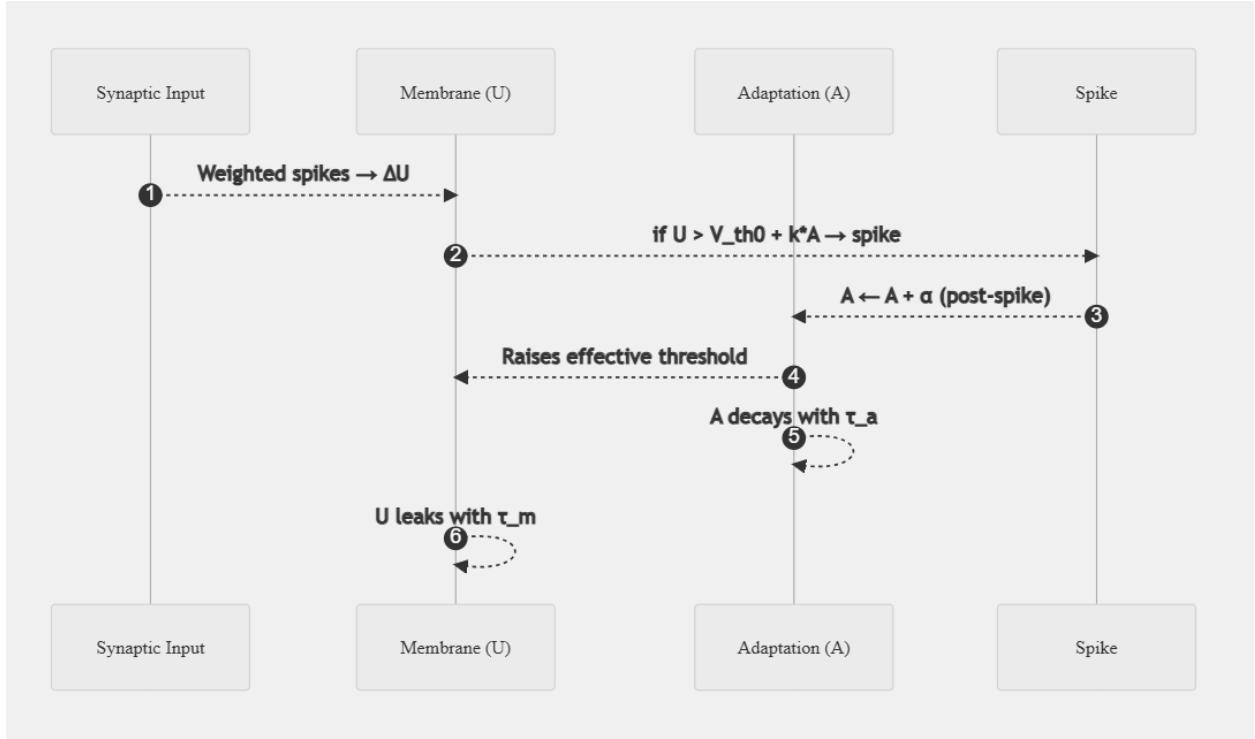
1. Input: low-power camera frames (or future event sensor) mapped to spikes via TTFS (latency code).
2. Core: compact convolutional feature extractor with Adaptive LIF (ALIF) neurons.
3. Pooling: first-spike max pooling for invariance and spike reduction.
4. Classifier: sparse, low-fan-in readout with first-to-spike decision.
5. Uplink: transmit only event-like decisions and highly compressed feature summaries.

Choice of Neuron Model (ALIF)

Adaptive LIF (ALIF) adds a dynamic threshold/adaptation current that increases immediately after a spike and decays with time. The benefit is that it stabilizes firing under timing noise, prevents runaway firing, and encodes short-term history. The following conceptual range of parameters can be set initially:

- Membrane time constant τ_m : 10–30 ms (trade-off temporal smoothing vs. responsiveness).
- Adaptation time constant τ_a : 50–300 ms (controls recovery speed).
- Base threshold V_{th0} : tuned to 0.5–1.0 (normalized units) for single-spike regime.

TFS emphasizes exact timing; naive LIF can over-fire on high-contrast inputs and is brittle to jitter. ALIF's adaptive threshold raises post-spike, suppressing bursty responses and stabilizing timing, which reduces false positives and overall spike counts. The added state (adaptation) is minor in hardware relative to the benefit in stability and robustness.



Layers and Connectivity

L2 Conv ALIF (feature extraction)

- 16 filters, 5×5 kernels, stride 1, valid padding.
- Receptive-field local connectivity → parameter sharing reduces energy and memory.
- Lateral winner-take-all (optional): suppresses redundant spikes within a small neighborhood.

Convolutions enforce locality and weight sharing, substantial savings in parameters and memory traffic versus fully connected layers processing raw pixels. They directly capture edges/corners that are invariant and informative for person/motion cues.

L3 First-Spike Max Pool

- 2×2 non-overlapping windows.
- The earliest spike in the window propagates; others are suppressed.
- Reduces spikes by up to $4 \times$ and adds translation tolerance.

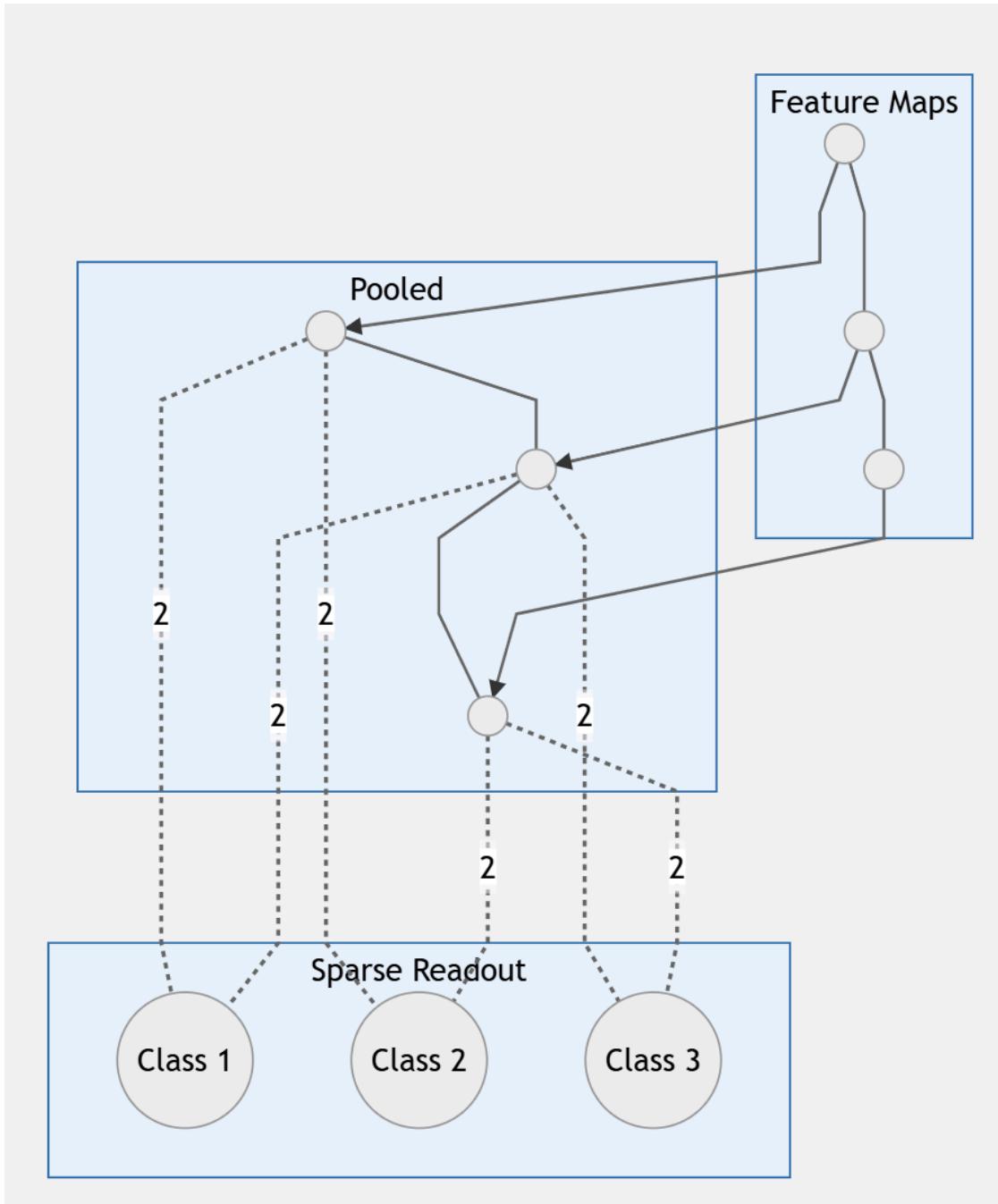
Using the earliest spike as the pooled representative removes redundant activity while preserving the most confident, earliest evidence, simultaneously lowering latency and spike traffic. It is an intuitive temporal analogue of max-pool and has simple hardware implementation.

L4 Sparse FC ALIF (classification)

- 3 output neurons (person/no-person/motion).
- Sparse random connectivity from pooled feature maps with probability $p \approx 0.2$.
- Winner-take-all and first-to-spike readout.

For three classes, dense readout is wasteful. Sparse random projections retain linear separability while reducing multiply-accumulate events, memory fetches, and storage, all scaling with sparsity p . This moves the design toward a Pareto-efficient point on accuracy–energy curves.

Here's the connectivity motif based on the above discussion:



The readout strategy would be as follows:

- First-to-spike at L4 decides the class immediately.

- Tie-breaking: If no spike within a timeout T_{dec} , fall back to counting spikes within $[0, T_{dec}]$ and choose argmax.
- Confidence: Optionally report decision time; earlier spikes indicate higher confidence.

Plasticity

For plasticity, the objective is to enable the system to self-organize features, adapt in the field, and remain stable over long deployments. The core principle being local, on-chip learning. To meet energy and bandwidth constraints, learning must be local, event-driven, and hardware-friendly. We employ Spike-Timing-Dependent Plasticity (STDP) for unsupervised feature discovery, complemented by supervised or semi-supervised readout tuning, and stabilized by neuronal adaptation/homeostasis.

STDP for feature extraction (L2 Conv ALIF)

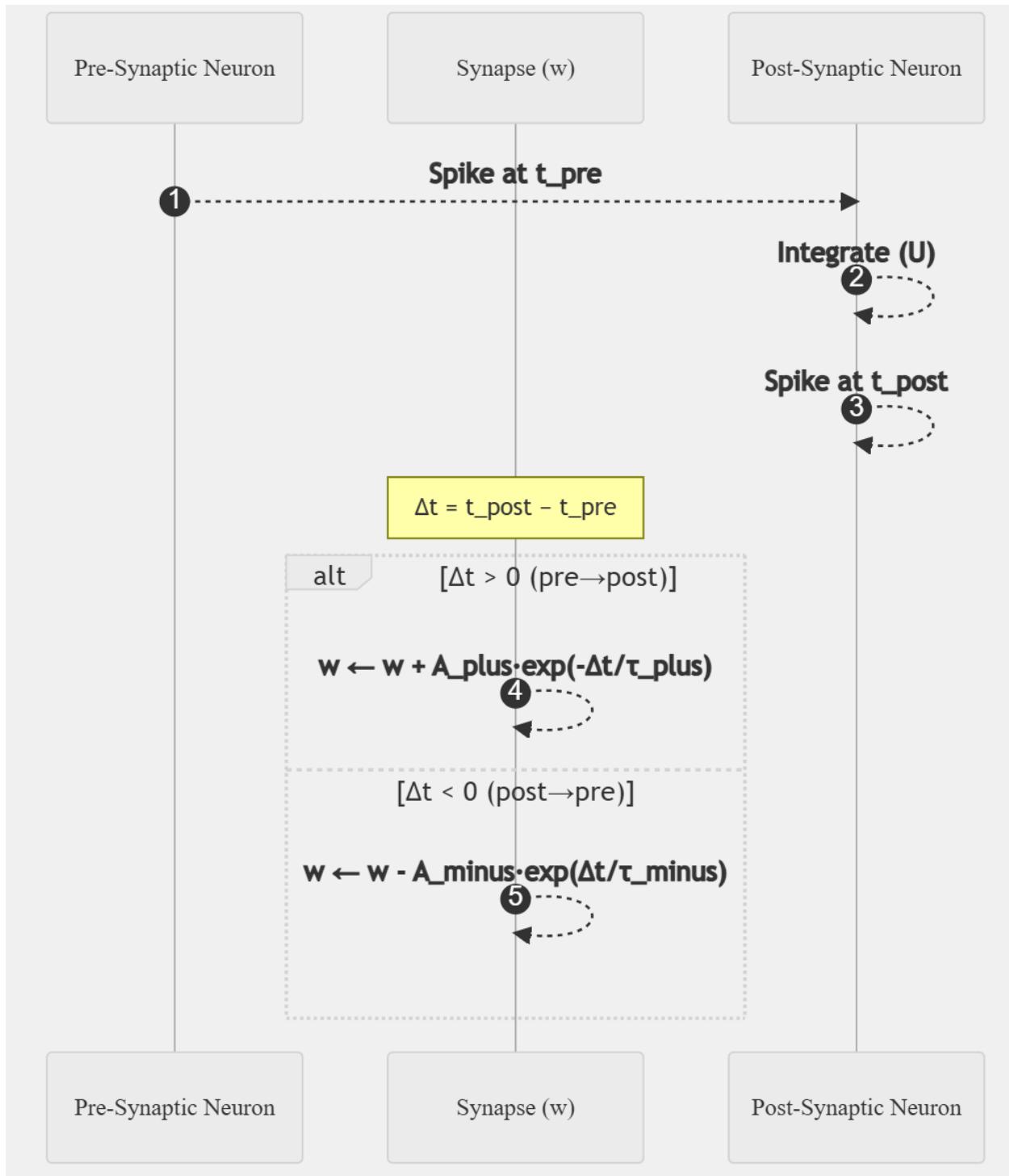
This is at the synapses from input (TTFS-encoded spikes) to convolutional feature neurons. Rule: weight change depends on relative timing ($\Delta t = t_{post} - t_{pre}$):

- If pre spikes just before post ($\Delta t > 0$, small), potentiate (LTP).
- If pre spikes after post ($\Delta t < 0$), depress (LTD).

Effect: neurons become selective to recurring spatiotemporal edge/texture patterns.

Sparsity: combine with lateral inhibition/WTA to encourage diverse, non-redundant filters.

STDP depends only on pre/post spike timing and local traces, no global error signals or backpropagation, making it hardware-friendly and energy-light. It avoids costly memory traffic inherent to label-heavy or off-chip training. Unsupervised STDP at early layers is a well-supported route to emergent oriented edge/corner detectors, providing a compact and robust basis for downstream classification without labels.

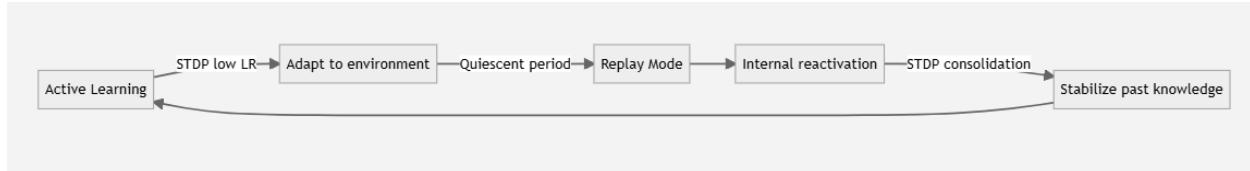


Training Schedule

1. Unsupervised pre-training: stream unlabeled data; enable STDP in L2; WTA enforces sparsity.
2. Freeze L2: after convergence (filters stabilize), lock L2 weights.

3. Supervised readout: train L4 sparse FC to map L2/L3 features to class labels (few-shot or periodic calibration).

There is a problem of catastrophic forgetting when adapting online. In order to mitigate that, the following approach can be taken: interleave on-line adaptation with low-power replay phases which are triggered during quiescent periods (no person/no motion). Internally generate spike patterns that resemble previously learned features/classes (e.g., through a small generative module or sampled activations) and then apply STDP during replay to consolidate old memories.



Evaluation

The purpose is to validate that the design achieves low latency, low energy, and sufficient accuracy for always-on event detection without transmitting raw frames.

Datasets

The datasets can be both real time and synthetic streams as follows:

Real event datasets (for person/motion detection):

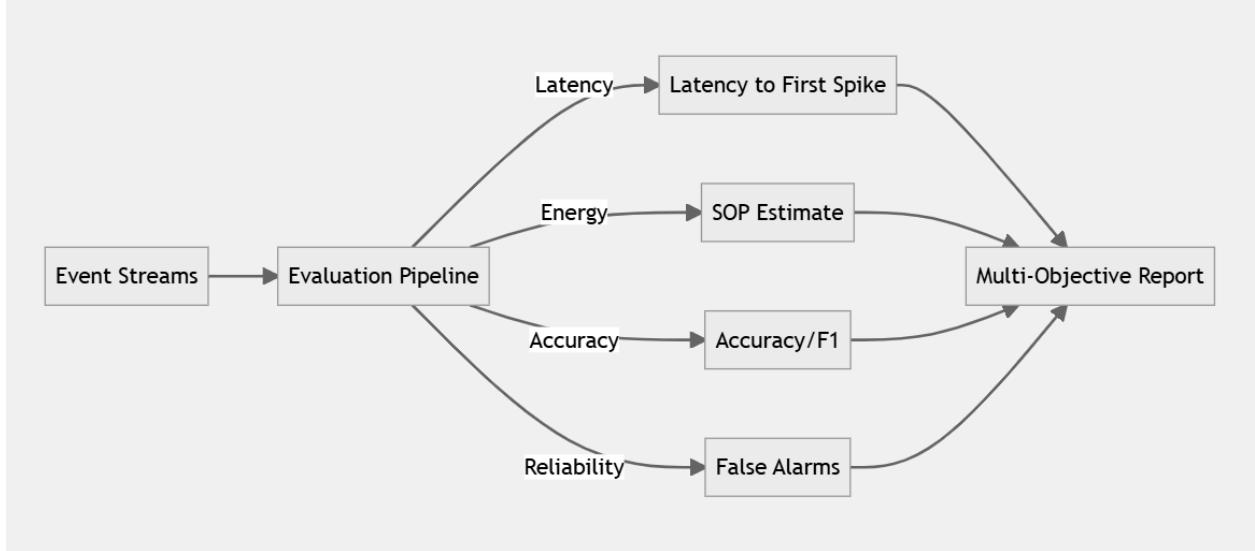
- [PEDRo \(person detection from event cameras\)](#).
- [Prophesee GEN1 \(pedestrians in driving scenes\)](#).
- [DVS128 Gesture](#) (dynamic motion patterns).

Synthetic: TTFS-convert static images (e.g., Caltech-101) to controlled spike trains for stress tests (noise, timing jitter, contrast thresholds).

Real event datasets (PEDRo, GEN1, DVS128) exercise the exact phenomena of interest, pedestrian/person presence and motion, under variable lighting and motion statistics, while synthetic TTFS streams allow controlled stress tests (jitter, contrast) to probe timing sensitivity inherent to TTFS.

Metrics

1. Latency to detection: time from input start to first spike at the correct L4 output.
2. Energy proxy (SOPs): total spike count \times average fan-in/out across layers; estimate synaptic ops and memory accesses per inference.
3. Accuracy / F1: classification metrics, including per-class metrics.
4. False alarm rate: spurious positive decisions per hour/day under idle scenes.
5. Throughput: events per second at fixed power budget.



The principal advantage of TTFS + first-to-spike is reduced decision time. Measuring latency to first correct spike captures whether early evidence truly enables faster decisions than any rate-based baseline.

Experimental Protocol

1. Calibration: set TTFS window $[T_{\min}, T_{\max}]$, neuron thresholds, and pooling parameters to achieve single-spike regime.
2. Unsupervised pre-training: enable STDP in L2 on unlabeled streams; monitor filter emergence and stability.
3. Supervised readout: freeze L2; train sparse L4 readout on small labeled set; evaluate.
4. Latency-energy characterization: sweep TTFS window, τ_m , τ_a , pooling size; record latency and SOPs.
5. Robustness tests: add temporal jitter, contrast noise, and background flicker; measure accuracy and false alarms.
6. Continual learning: alternate online adaptation and replay phases; measure retention vs. plasticity.

On neuromorphic hardware, data movement and synaptic operations dominate energy. Total spike counts combined with fan-in/out estimate SOPs and memory traffic without requiring specific hardware. This lets us compare encoding and architectural variants fairly. In always-on regimes, false positives directly translate to wasted downstream wake-ups and energy. Including false alarm rate ensures the design's efficiency goals remain intact in deployment conditions.

We can think of following ablations for a more robust and detailed experimentation:

- Remove adaptation (ALIF \rightarrow LIF): expect higher firing rates, reduced stability, lower accuracy under noisy timing.
- Swap encoding (TTFS \rightarrow Poisson rate): expect increased latency and SOPs, potential modest noise robustness gains.

- Dense readout (sparse → dense): expect higher energy/params with limited accuracy improvement; possible overfitting.

Each ablation isolates a core design choice: ALIF (stability), TTFS (latency/energy), sparsity (efficiency). Observing predicted degradations validates that benefits come from principled mechanisms rather than incidental hyperparameters.

Reporting

- Pareto plots: accuracy vs latency, accuracy vs SOPs.
- Confusion matrices: per-class performance, false alarms.
- Spike activity summaries: spikes-per-layer per inference.
- Qualitative: raster plots and earliest-decision timelines.

Pareto plots reveal trade-offs among accuracy, latency, and energy, critical when a single scalar score hides whether gains come at unacceptable costs.

Question 2 — Implementation (50 marks) — Brian2-based SNN experiment

Implement a small feedforward SNN in **Brian2** that receives Poisson spike-encoded MNIST images, uses a hidden layer of LIF neurons, and uses a **linear readout** trained on spike counts. Then implement and test **one** synaptic plasticity/tuning tweak (choice below) and report how it changes network behavior and classification performance.

Baseline model

1. **Input encoding:** Convert each grayscale image (28×28) to Poisson spike trains for a fixed encoding interval T_{enc} (e.g., 100 ms).
2. **Network architecture:**
 - Input layer: 28×28 Poisson sources (flattened $\rightarrow N_{in} = 784$).
 - Hidden layer: Number of LIF neurons which you determine. Dense or fractional connectivity.
 - Readout: For each image, record spike counts of hidden neurons. Train a linear classifier (sklearn LogisticRegression or RidgeClassifier) to map spike-count vectors \rightarrow digit label.
3. **Neuron model:** Standard LIF with membrane time constant τ_m , resting potential V_{rest} , reset V_{reset} , threshold V_{th} . Use refractoriness.
4. **Synapses:** You can randomly assign initial weight or can sample them from normal distribution (e.g., $N(0.5, 0.1)$) clipped to positive for excitatory input.
5. **Training readout:** Freeze SNN weights; train readout offline using spike-count features. Report classification accuracy on test set.

Required: Jupyter Notebook with code implementation, plots and accompanying analysis.

Extra credit

You can modify your baseline architectures with these potential tweaks (choose one) and observe behavior of the model. Does it improve classification accuracy or not?

Homeostatic firing-rate regulation: Neurons optimize for homeostasis unlike artificial neurons which fire based on inputs. Try implementing a target firing rate for each hidden neuron.

Change neuron/time constants: Systematically vary τ_m (e.g., 5 ms, 20 ms, 50 ms) and report how spike timing, spike counts, and classification accuracy change. Provide plots of accuracy vs τ_m and example voltage traces.

Sparse connectivity: Use sparse connectivity (probability of connectivity = 0.05). Compare performance and show a few learned/initial filters.

Question 2: Solution

The detailed Jupyter Notebook with code implementations, plots and accompanying analysis is present here:

- Jupyter Notebook:
https://drive.google.com/file/d/1krJzOWwS5H8FH8-yxQQwNUabvKW_3Ny-/view?usp=sharing
- Code Scripts Referred In The Notebook: <https://github.com/aryashah2k/NeuroAI>