SpikeCLIP-SNN

Berk Yilmaz

Table of Contents

Research Problem

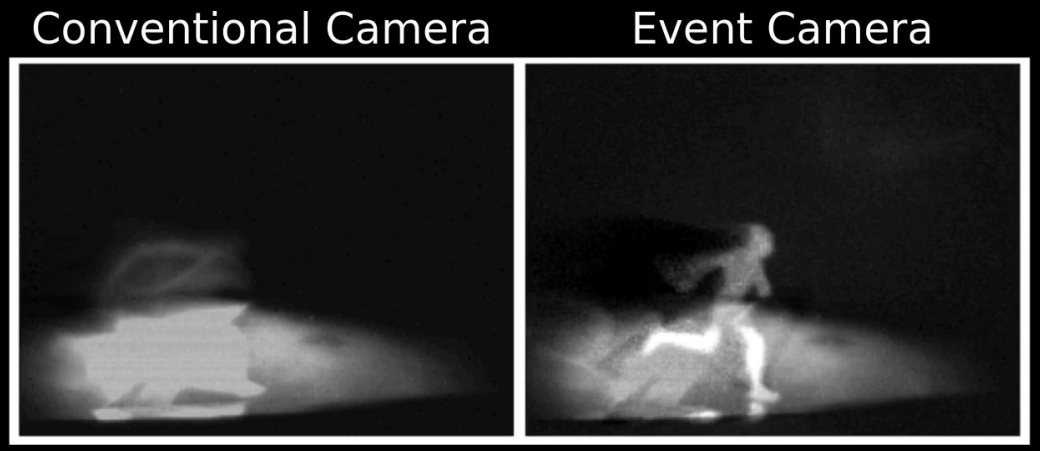


Fig 1. Event camera and conventional camera

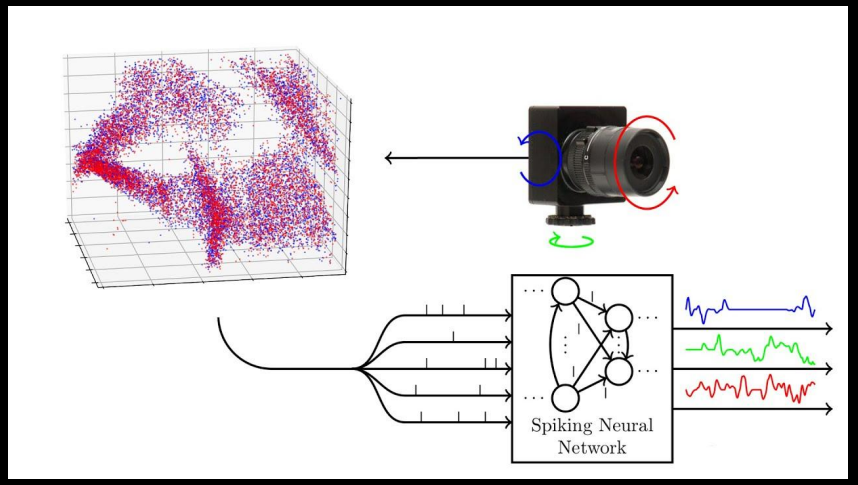


Fig 2. Event camera and conventional camera

Spike cameras, as described by Huang et al. in 2023 are neuromorphic sensors inspired by the human retina

Each pixel continuously monitors incoming light intensity

When the accumulated photons cross a threshold a *binary signal* is emitted

Then the accumulator resets and starts again

This happens independently for every single pixel

Challenge

How do we reconstruct an image from a stream of spikes?

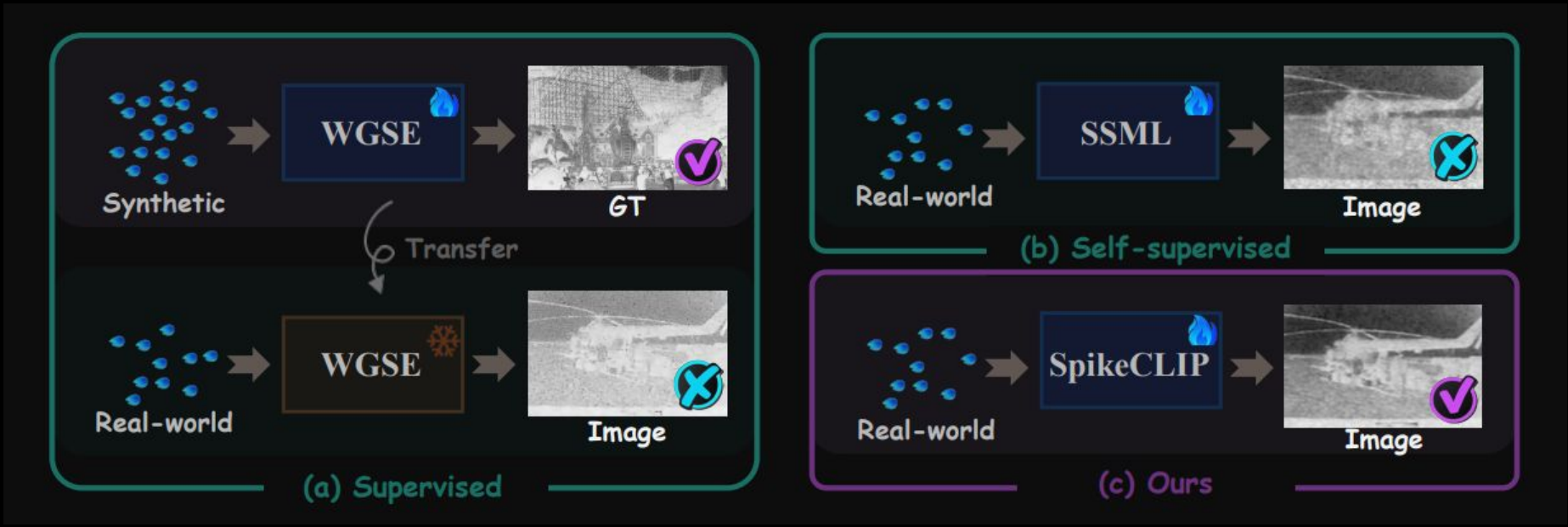


Fig 3. Reconstruction of Image

Humans can’t see spikes

We need to convert this binary spike stream back into an image we can understand

Previous Approaches

Spike-based image reconstruction methods can be roughly categorized into three types:

Model-based methods

Build mathematical relationships between spike streams and images based on the camera's sampling mechanism and biological principles

TFP/TFI *Zhu et al.2019*	-	Virtual exposure + inter-spike interval analysis
Synaptic Plasticity *Zheng et al.2021*	-	Short-term synaptic plasticity + temporal regularity

supervised methods, ***and***

The deep learning approach: let neural networks learn the mapping from paired data

SpikeFormer *She & Qing2022*	-	Hierarchical transformer, multi-scale features
Recurrent Restoration *Zhu et al.2023*	-	Recurrent architecture for temporal modeling
Fluctuation Suppression *Zhao et al.2024*	-	Spike representation module for noise reduction

Previous Limitations

Model-based Methods Limitations

- Too many hand-tuned parameters
- Works in lab conditions only
- Fails to generalize to real-world
- Cannot handle noise or varying lighting

Supervised Methods Limitations

- Requires paired training data
- Trained on *SYNTHETIC* data
- Real world ≠ Synthetic world
- Domain Gap

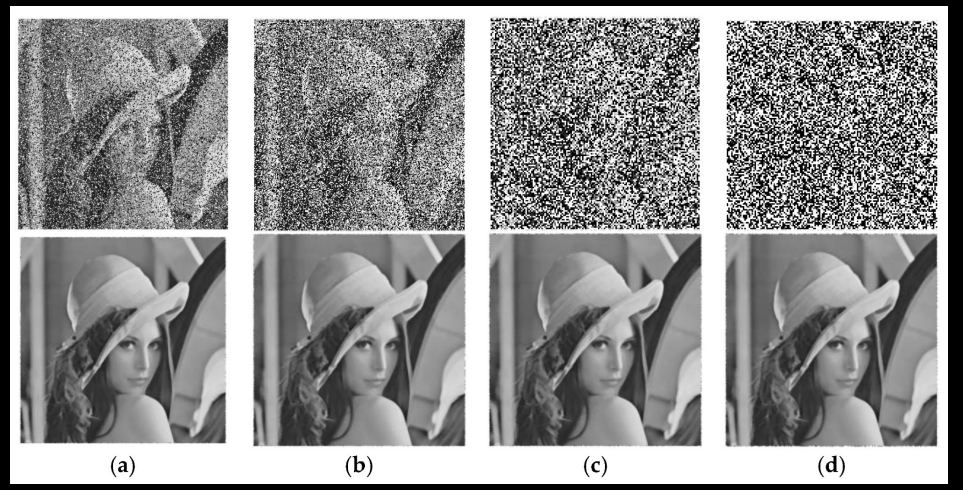


Fig 3. Rethinking High-speed Image Reconstruction Framework with Spike Camera

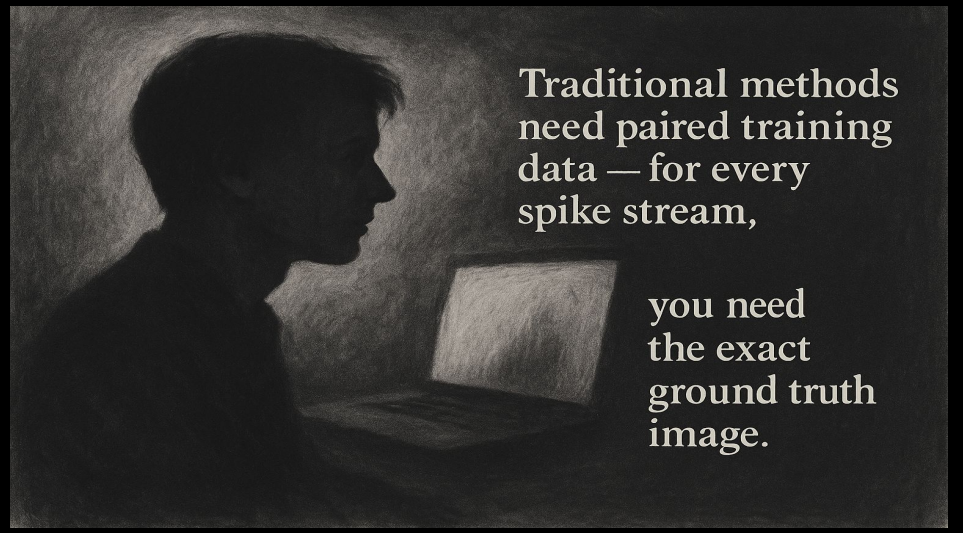


Fig 4. Image

Attempt 1, Group 3 - Self-Supervised Methods

Self-Supervised Methods

The attempted fix: train directly on real data *without ground truth*

Overcome domain gap by training on real-world data. Use pseudo-labels instead of ground truth

SSML *Chen et al.2022* - Distillation learning with blind-spot denoising
SJRE *Chen, Yu & Huang2023* - Joint optimization with optical flow constraints

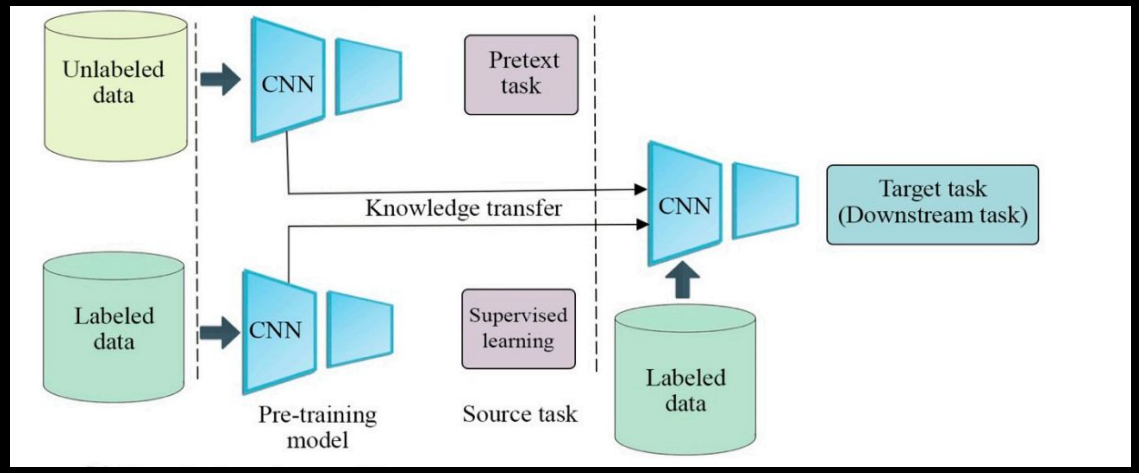


Fig 5. Self-supervised learning framework

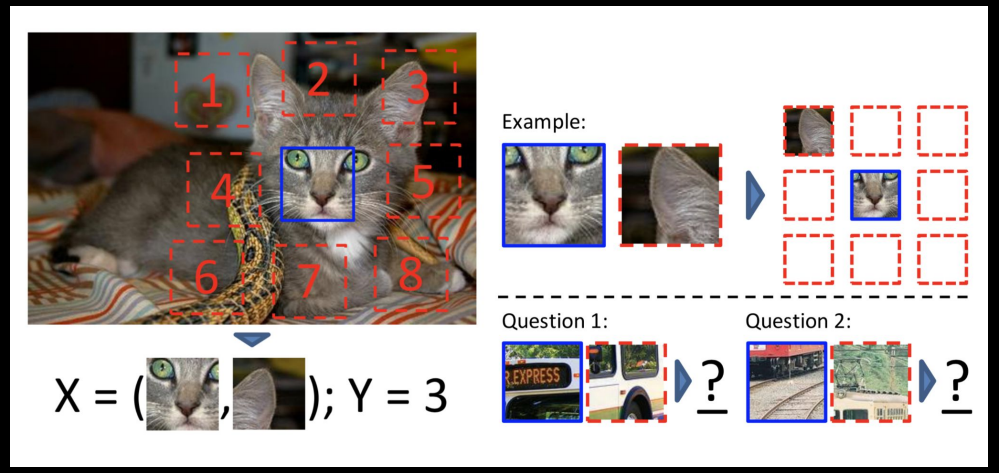


Fig 6. Self-supervised learning example

Previous Limitations

Self-supervised Learning Limitations

- Both methods depend on TFI for pseudo-labels
- TFI fails in low-light conditions

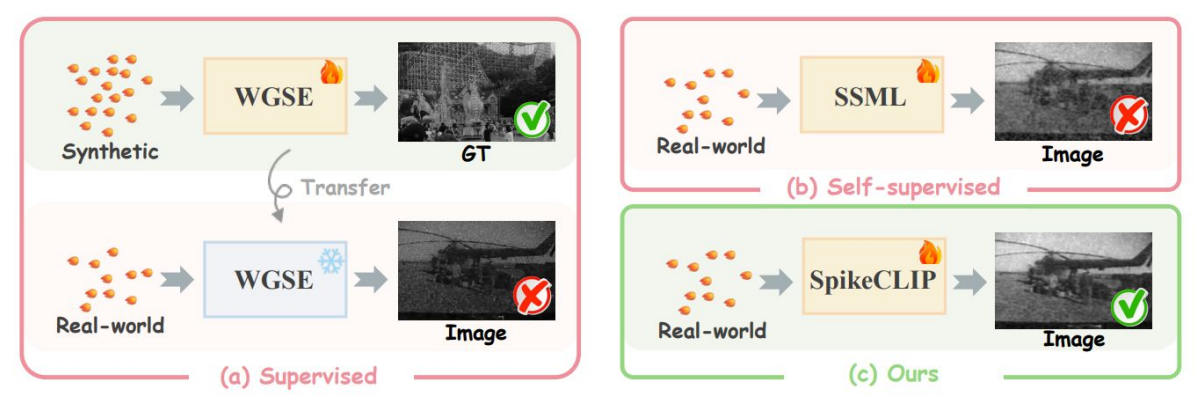


Fig 7. Self-supervised learning vs Paper's improvements

Fig 8. TFSTP vs TFI vs SSML vs Paper's implementation

Motivation: Bridging the Gap

This motivates us to explore an orthogonal direction using vision-language models for reconstruction supervision. CLIP (Radford et al. 2021) has been widely applied in various downstream recognition tasks

Existing Approaches	Limitations
Model-based (TFP, TFI)	Limited to simple degradations
Supervised	Synthetic-to-real domain gap
Self-Supervised	Lacks semantic guidance

Can vision-language priors from CLIP provide semantic supervision for spike reconstruction?

CLIP in Low-level Vision

CLIP's vision-language priors have recently been applied to image restoration tasks, enabling unsupervised/self-supervised training

Recent Applications	
Deblurring	Blur map estimation from dual-pixel pairs (Yang et al. 2024)
Low-light Enhancement	CLIP-LIT uses learnable prompts to guide toward "bright" features (Liang et al. 2023)
Denoising	Generalizable denoising via CLIP transfer (Cheng et al. 2024)
Event Reconstruction	Class prompts enable label-free reconstruction (Cho et al. 2023)

No paired ground truth needed

SpikeCLIP: Three-Stage Framework

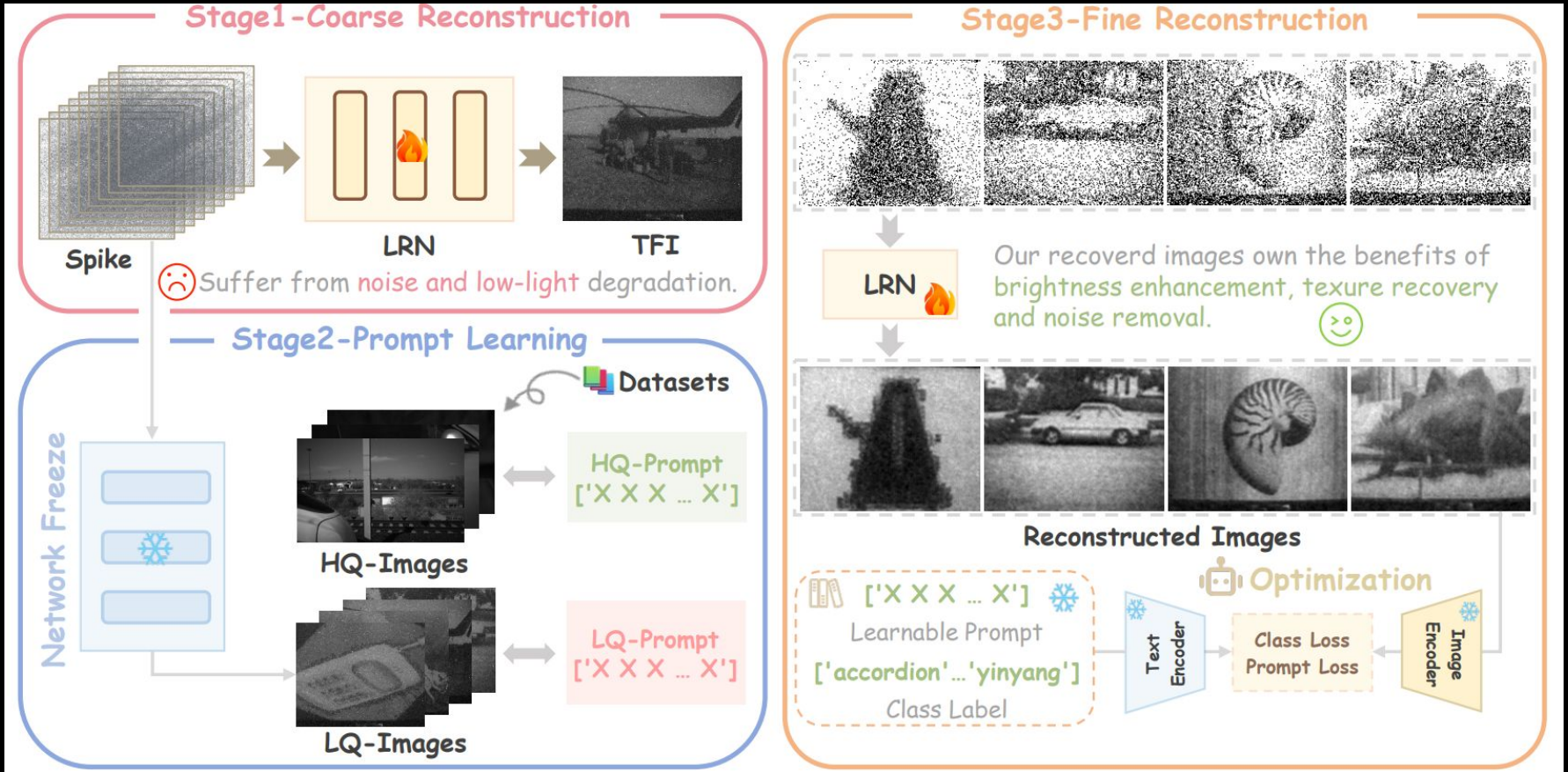


Fig 9. Three-Stage Framework

Fundamental Difference
Sensor & Data

U-CALTECH

Aspect	Detail
Sensor	Real spike camera (not event camera)
Data Type	Continuous binary spike streams
Samples	~5,000 spike recordings
Split	0-4999 train, remaining test
Conditions	Low-light / challenging lighting
Ground Truth	No paired Images

U-CIFAR

Aspect	Detail
Sensor	Spike camera
Conditions	Low-light
Ground Truth	No paired Images

N-CALTECH101

Aspect	Detail
Sensor	Event Camera
Data Type	(x, y, t, polarity) events
Samples	8,709
Split	80/20
Conditions	Normal-light
Ground Truth	CALTECH101 (not used)

Data Pipeline: N-Caltech101

Event camera outputs the pixels that change brightness

Each event = (x, y, timestamp, polarity)
x, y : pixel position
timestamp: (ms)
polarity: brightness increase (+1/-1)

N -Caltech101 Binary File Format

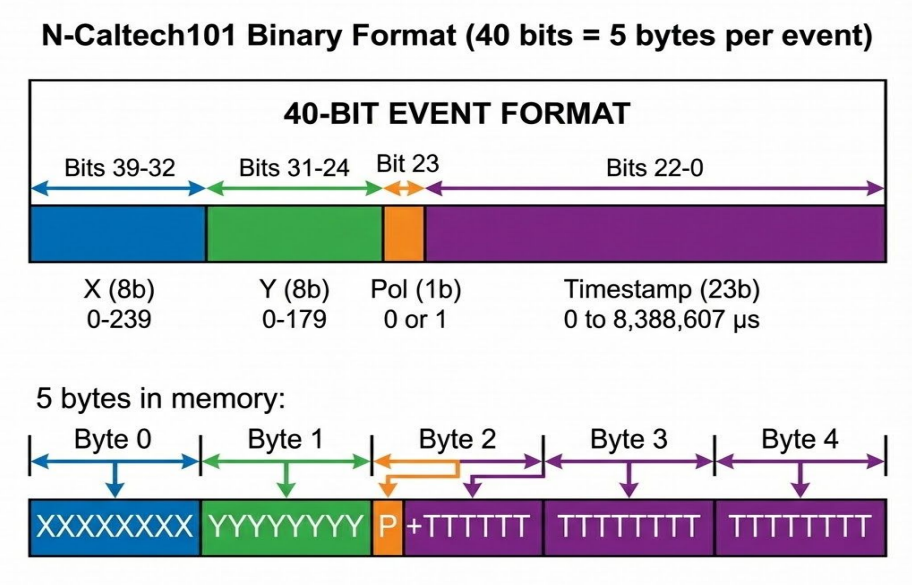


Fig 10. Bitwise representation of N-Caltech101

The challenge is that;

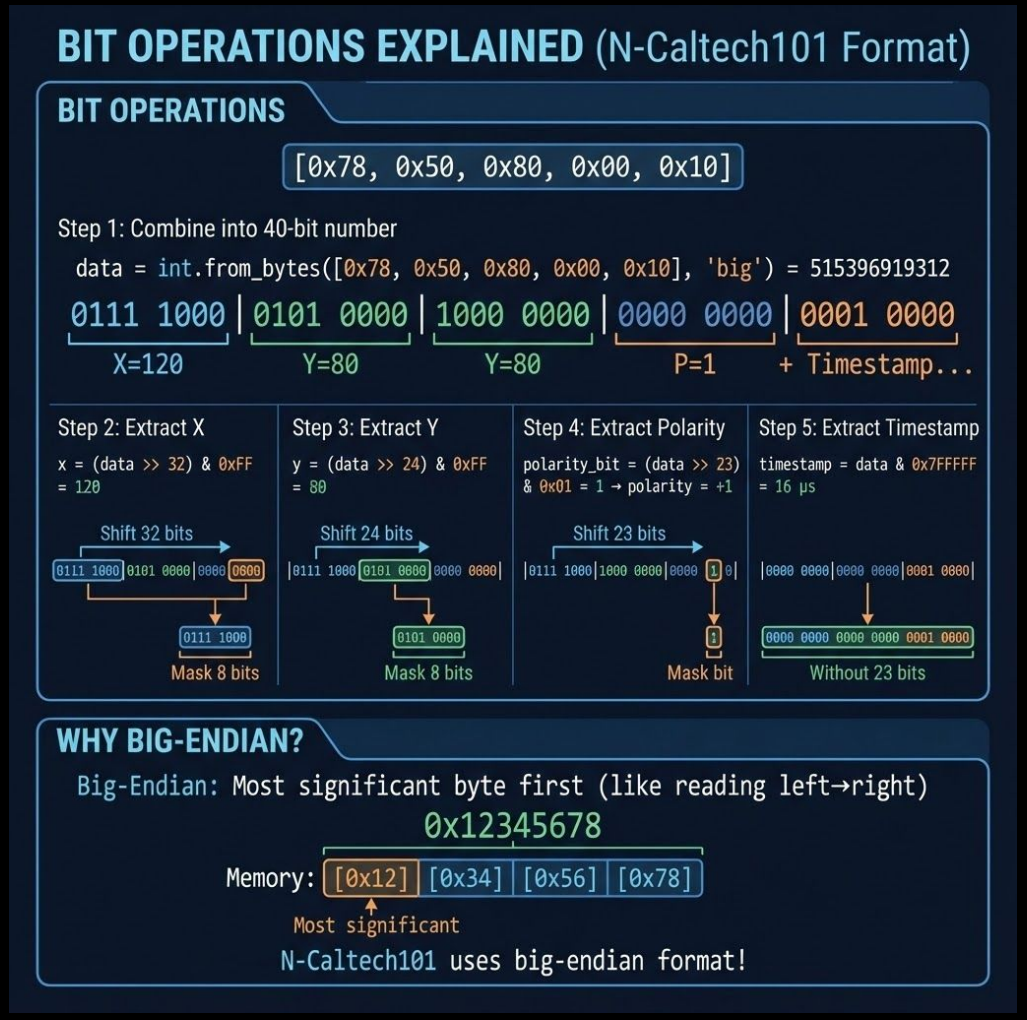
N -Caltech101 stores events in a **compact binary format** 40 bits, or 5 bytes per event

Looking at the diagram on left:

- **Bits 39-32** (blue): X coordinate, 8 bits, range 0-239
- **Bits 31-24** (green): Y coordinate, 8 bits, range 0-179
- **Bit 23** (orange): Polarity, just 1 bit, 0 or 1
- **Bits 22-0** (purple): Timestamp, 23 bits, up to ~8 million microseconds

Data Pipeline: N-Caltech101

Bitwise Operations



spikeclip_snn/utils/event_utils

load_env() function

```
# Parse each event
for i in range(num events):
    #Get the corresponding 5 bytes for event i
    #Calcualting the offset so i can use in window slice
    offset = i * event size
    event_bytes = raw_data[offset:offset + event_size]

    # We have 5 bytes for this event
    # Combine the bytes into single chunk 40 bytes
    data = int.from_bytes(event_bytes, byteorder="big")

    # Extracting the fields
    # Bit operations
    # Shift all bits to RIGHT by 32 poistions
    # But even though we get >32 bits there could
    # be potenatial mixes to ensure this
    # A mask of 8 bits introduced

    # data >> 24 = 0b0111100001010000
    #    & 0xFF
    #    0111100001010000  (shifted result)
    #    0000000011111111  (mask)
    #  ——————————————————  &
    #    0000000001010000  (only Y)
    x = (data >> 32) & 0xFF
    y = (data >> 24) & 0xFF
    # bit 23, masking with 0x01
    polarity bit = (data >> 23) & 0x01
    # No shift to timestamp, only masking
    timestamp = data & 0x7FFFFF
    # Convert polarity to binary
    polarity = 1 if polarity bit == 1 else -1
    # Add event [x, y, timestamp, [polarity]
    events.append([x,y,timestamp,polarity])
# Convert the event into NumPy array
events = np.array(events, dtype = dtype)
```

Fig 11. Bit operations explained

Data Pipeline : Voxel Grid

Neural Networks need; fixed input size, dense tensors, spatial structure (width x height), temporal information is preserved

Raw events have; variable counts, sparse, unstructured list format, timestamps scattered irregularly

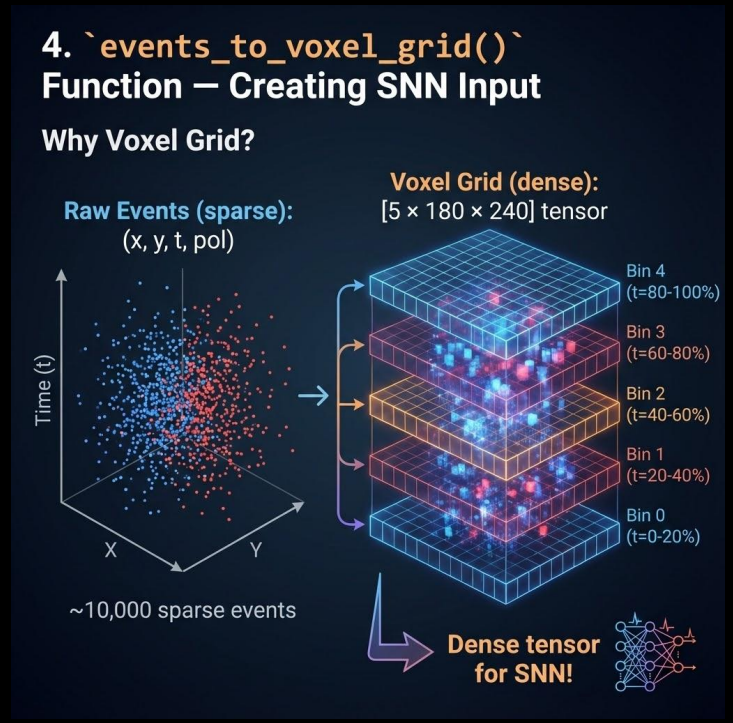


Fig 12. Why voxel grid?

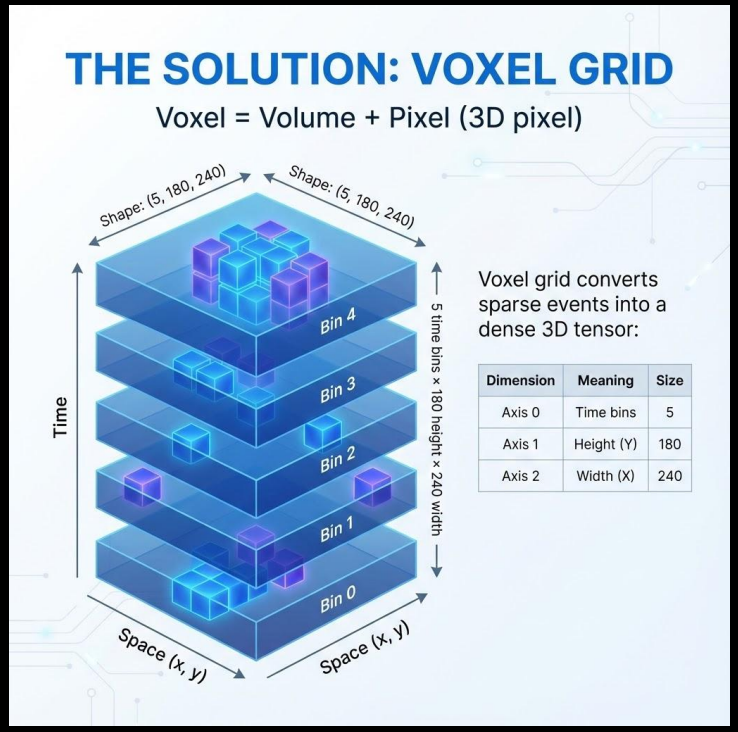


Fig 13. Voxel Grid as a solution

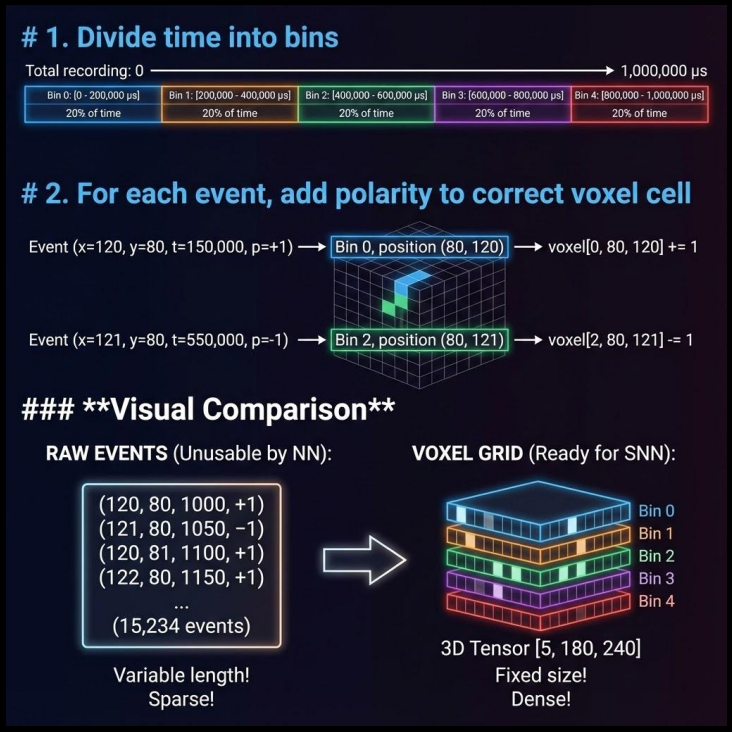


Fig 14. Construction of the voxel grid

Data Pipeline : Dataloader

The Problem: Training One Sample at a Time is Slow

The Solution: DataLoader Batches Multiple Samples

PyTorch's DataLoader automatically groups multiple samples together into a single tensor that can be processed in parallel

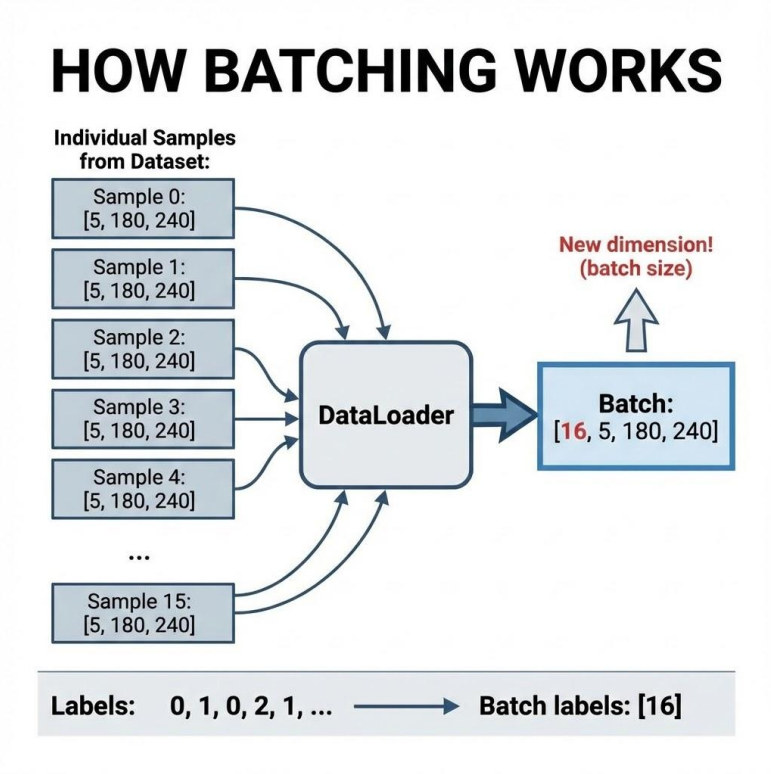


Fig 15. How batching works

SpikeCLIP: Coarse Reconstruction

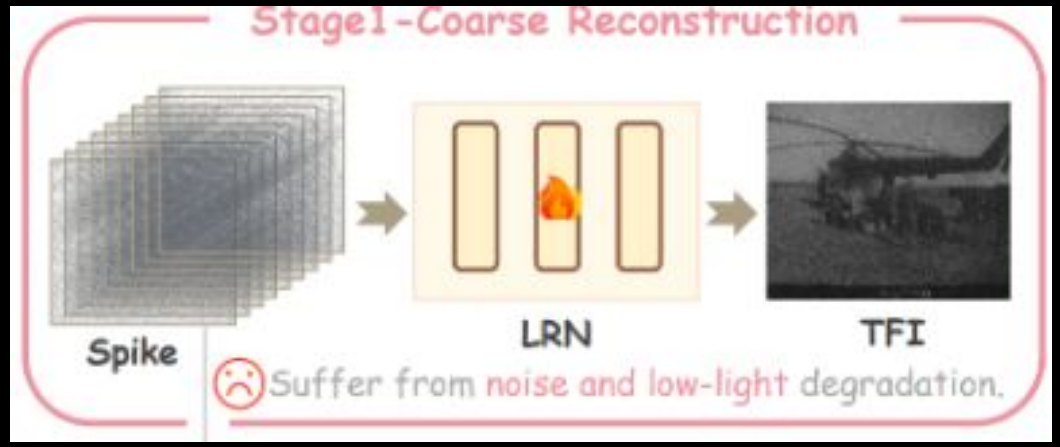


Fig 16. Paper's Stage-1 implementation

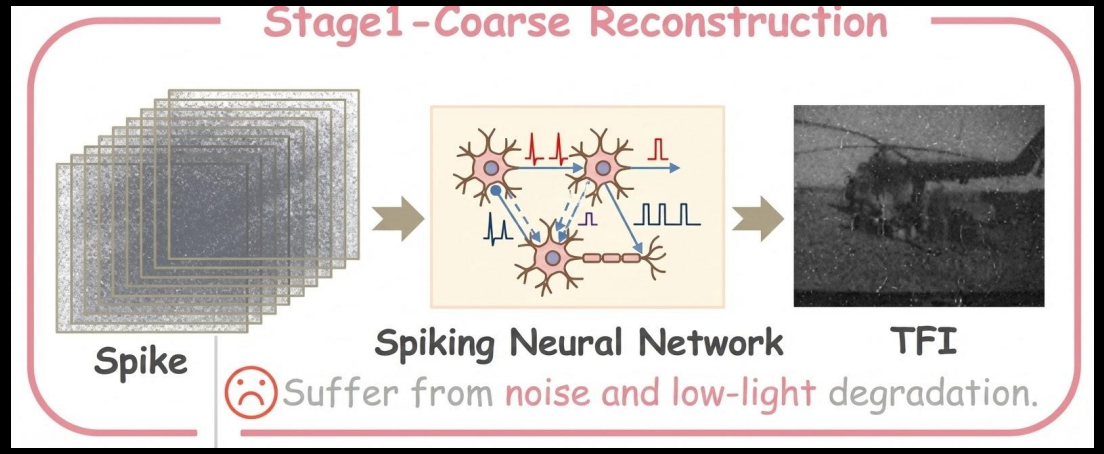


Fig 17. New Stage-1 implementation with Spiking Neural Network architecture

SNN Architecture

- Event cameras produce **temporal, spike-like data**
- Regular CNNs process everything in one pass and ignore this temporal structure
- SNNs naturally process data over time using spiking neurons

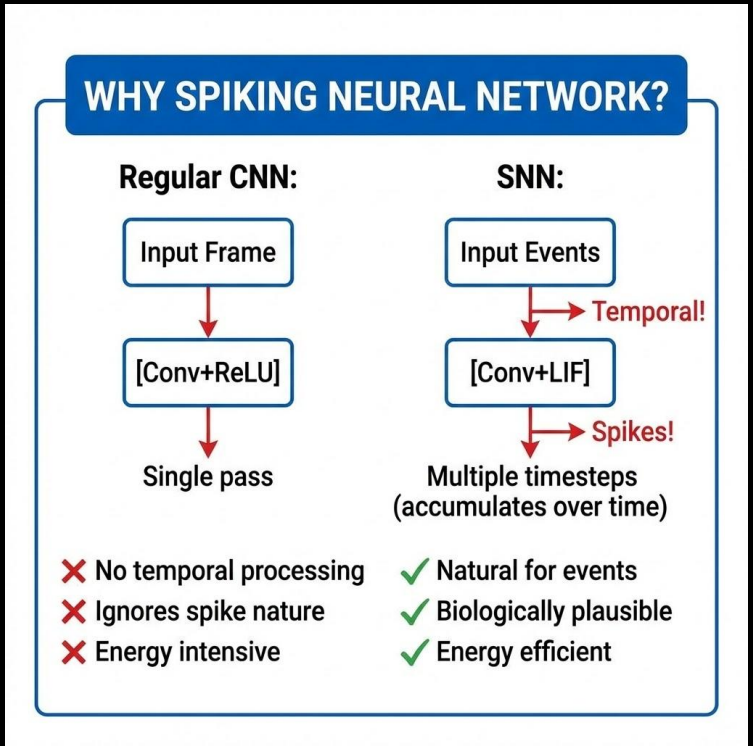


Fig 18. Why spiking neural network?

SNN Architecture Overview

Our architecture has two parts:

Encoder uses spiking neurons (LIF) to process the temporal voxel data over 50 timesteps

Decoder uses standard convolutions to reconstruct the final image. The encoder compresses the input while the decoder upsamples back to original resolution

The LIF neuron is the core of our SNN. It **accumulates** input over time into membrane potential

The membrane **leaks** with decay rate β = 0.95. Then potential exceeds threshold, it **fires a spike** and resets

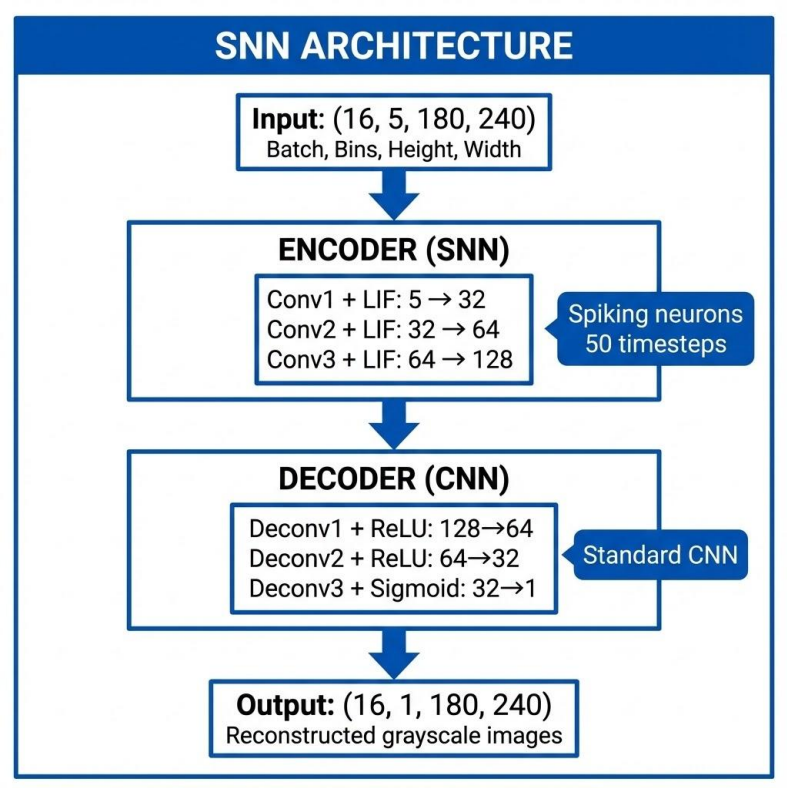


Fig 19. SNN architecture

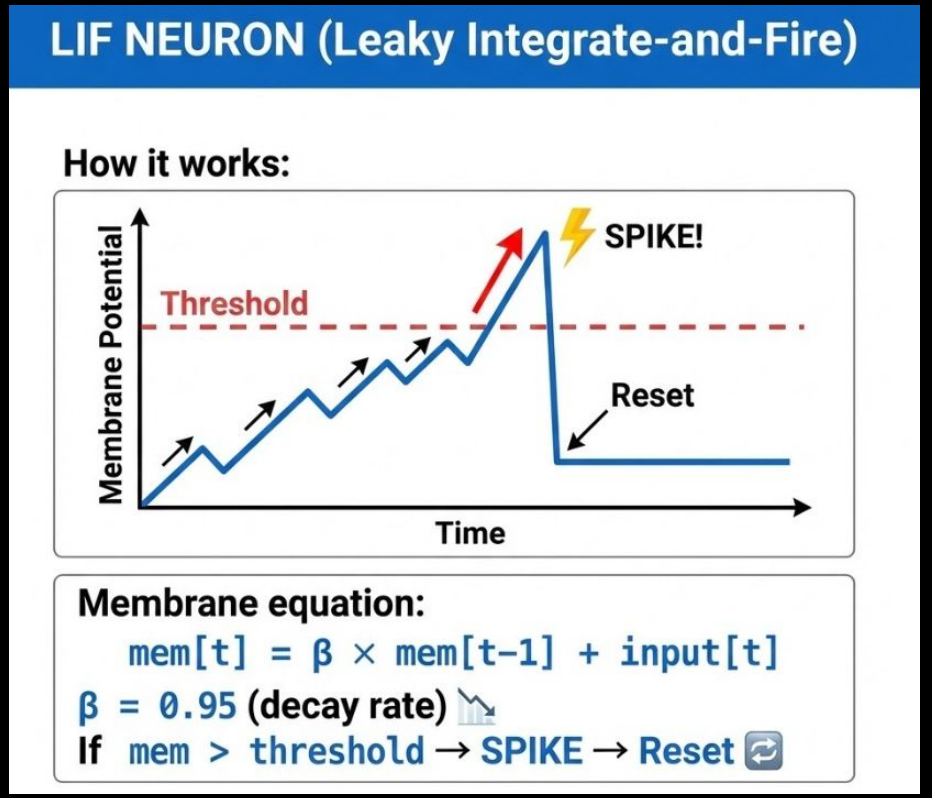


Fig 20. LIF Neuron

SNN Architecture Overview

We process the input over 50 timesteps. At each step, spikes propagate through the network. At the end, we average all spike outputs this gives us stable features that capture the temporal dynamics of the event data

The SNN outputs reconstructed grayscale images of shape (16, 1, 180, 240). Before feeding to CLIP, we resize to 224×224, convert to RGB by repeating the channel 3 times, and normalize with CLIP's expected statistics. Now it's ready for CLIP encoding

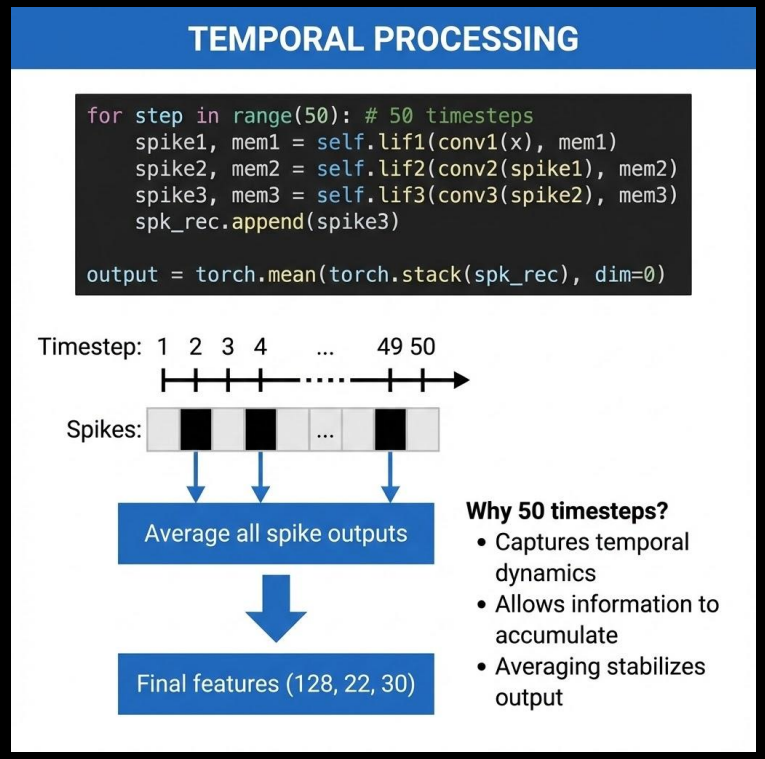


Fig 21. Temporal Processing

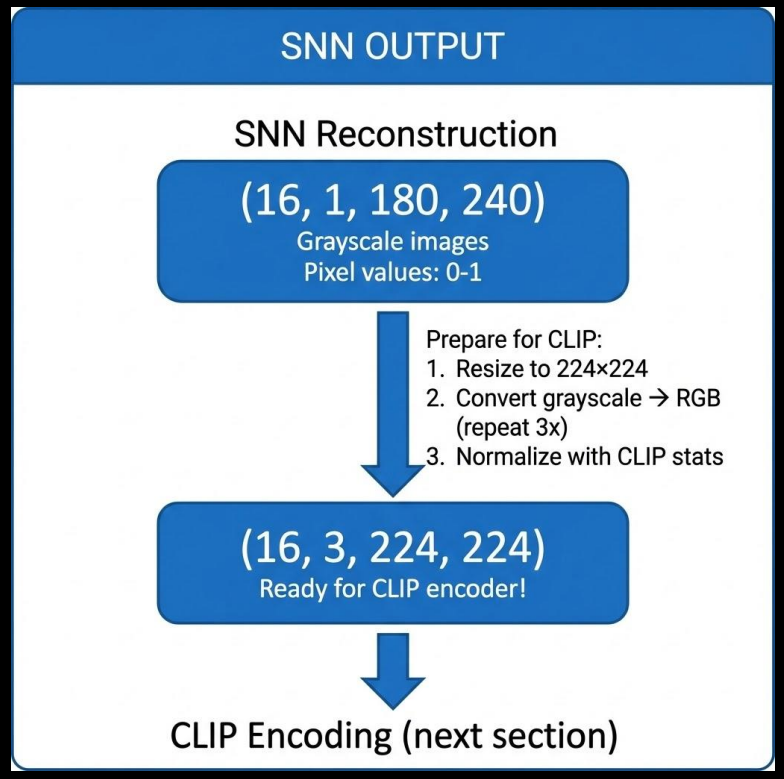


Fig 22. SNN Output

SNN Architecture: Key Design Choices

1 - LIF Neurons: β = 0.95 (Membrane Decay)

β value	Test Results	Use Case
0.50	Fast decay, short memory	Fast-changing signals
0.80	Medium decay	General use
0.95	Slow decay, long memory	Event data
0.99	Very slow decay	Risk of saturation

β = 0.95 because event data is sparse, the neuron needs time to accumulate enough input before firing. A lower β would cause it to forget too quickly.

2 - Surrogate Gradient: fast_sigmoid(slope=50)

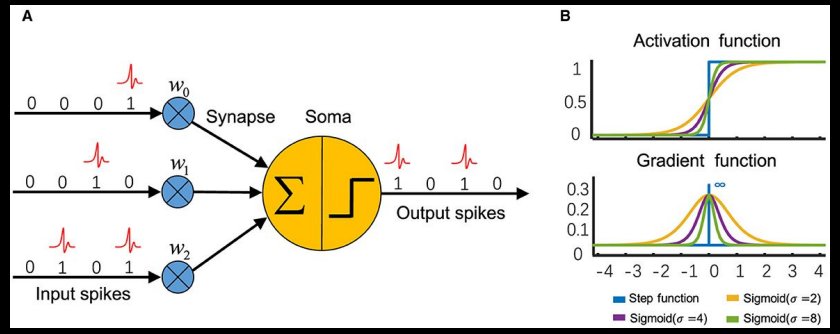


Fig 23. Surrogate gradient

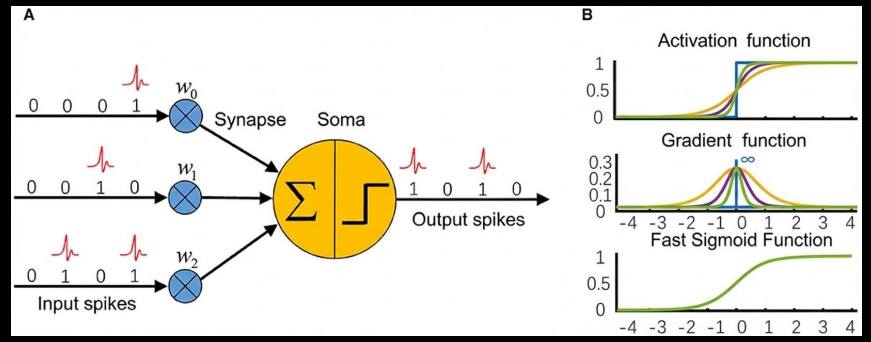


Fig 24. Surrogate gradient / Fast Sigmoid Function

SNN Architecture: Key Design Choices

3 - Temporal Averaging

If we only look at one timestep, the output is noisy and unreliable. By averaging over 50 timesteps, we get a firing rate between 0 and

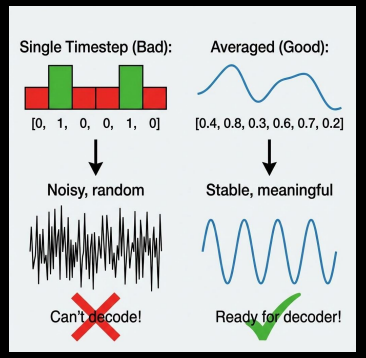


Fig 25. Temporal Averaging example

4- Bilinear Interpolation

Our encoder uses **stride-2 convolutions** which halve the size at each layer. Due to **integer rounding**, the decoder doesn't perfectly reconstruct 180×240 where we might get 176×240 instead
Bilinear interpolation smoothly resizes to the exact output size without blocky artifacts

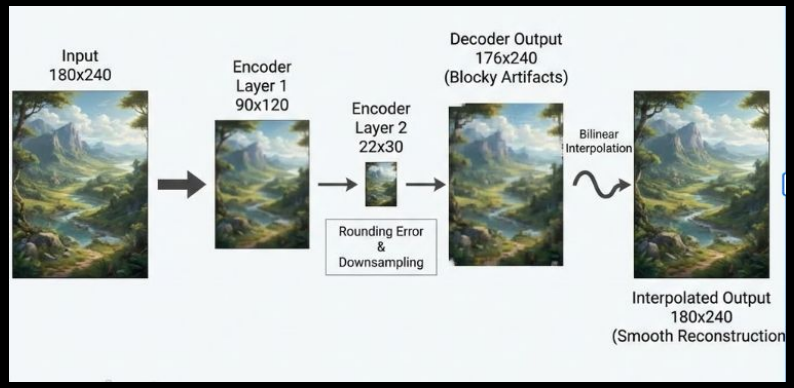


Fig 26. Bilinear Interpolation example

Loss Function: InfoNCE Loss

Instead of using MSE loss which compares pixels, we use **InfoNCE loss** a contrastive learning approach.
Make our reconstructed images **semantically similar** to their class descriptions

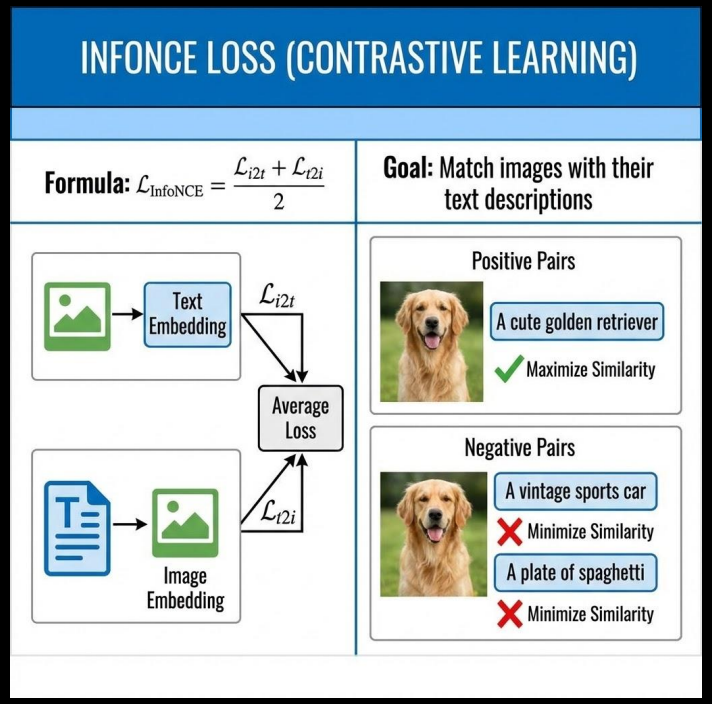
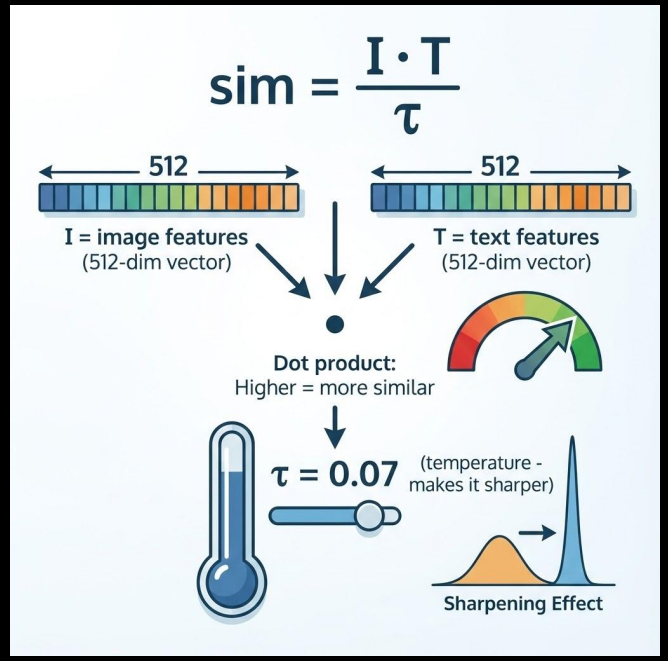


Fig 27. InfoNCE Loss

InfoNCE has two parts:
Image-to-Text *asks 'does this airplane image match the text airplane?' and*
Text-to-Image *asks the reverse. We average both for balanced learning*

We compute **cosine similarity** between image and text features. Both are 512-dimensional vectors from CLIP



I = image features (512-dim vector)
T = text features (512-dim vector)
τ = 0.07 (temperature - makes it sharper)

Fig 28. Cosine Similarity

Loss Function: InfoNCE Loss

Similarity matrix between all images and all texts in the batch.
The **diagonal** shows matching pairs, this should be high
The **off-diagonal** shows non-matching pairs, this should be low

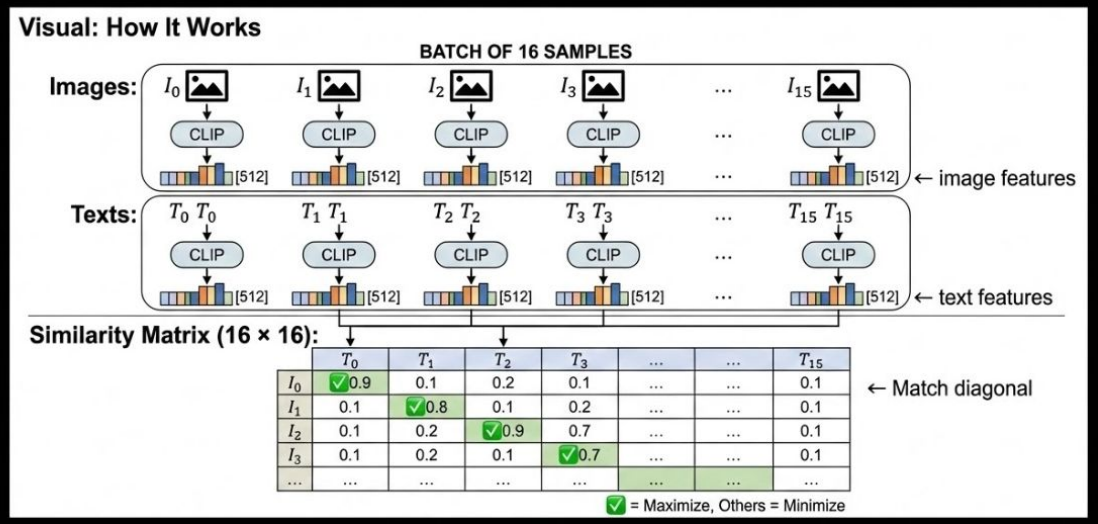


Fig 29. InfoNCE Loss Function

For each class, random text templates are used. This diversity helps the model generalize better, It learns the concept 'airplane', not just one specific phrase

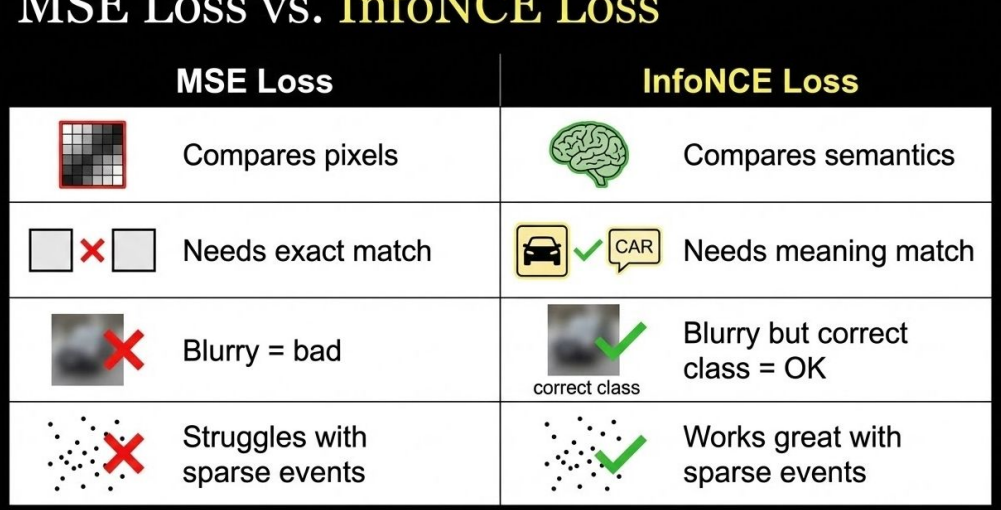


Fig 30. MSE Loss vs InfoNCE Loss

```
templates = [
"a photo of a {class}",
"a cropped photo of the {class}",
"a clear photo of a {class}",
"an image of a {class}"
]
```

SpikeCLIP: Hyperparameter Tuning

Parameter	Tried	Final
Learning Rate	0.001	3e-4
Beta (LIF decay)	0.9	0.95
Temperature	0.1	0.07
Gradient Clipping	no max_norm	max_norm = 1

SpikeCLIP: Stage 1 Pipeline

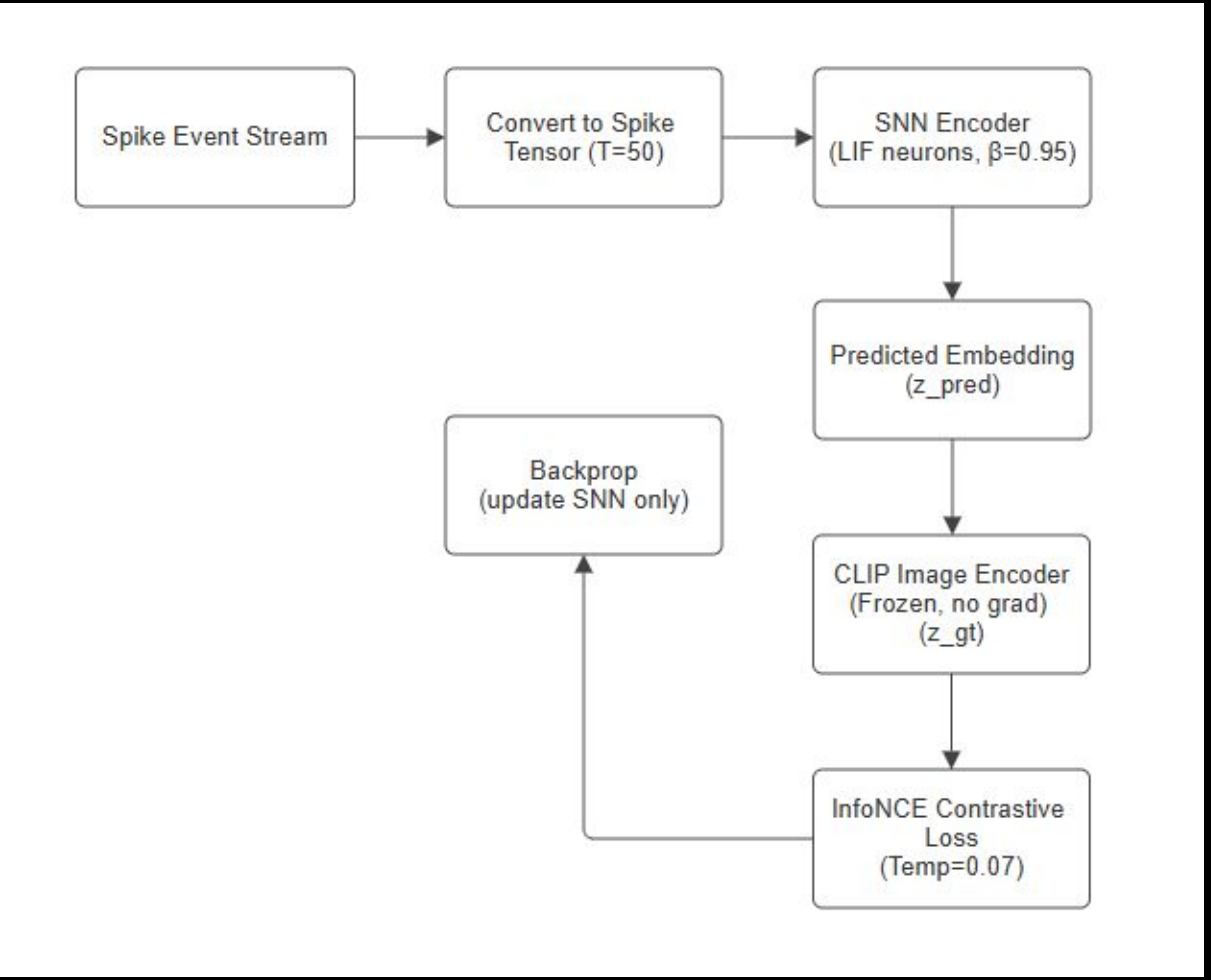


Figure 31. Stage 1 Pipeline

SpikeCLIP: Stage 1 Results

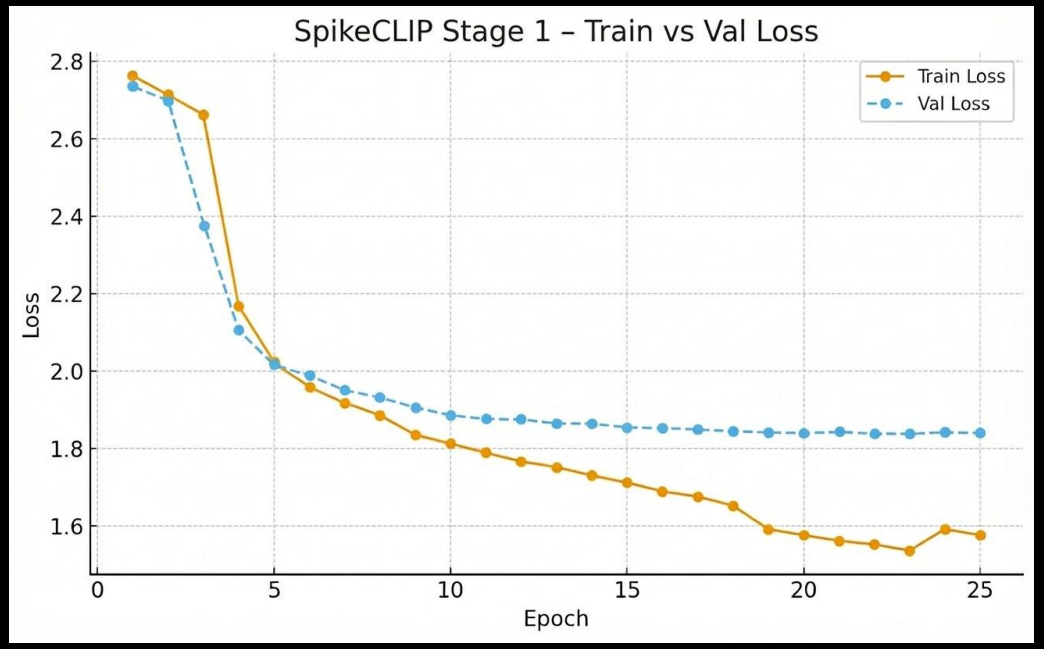


Fig 32. SpikeCLIP Stage 1 - Train vs Validation Loss

*In our setup, the network is not trained to reconstruct pixels directly
We are doing;

- Temporal spike encoding -> feature reconstruction
- Loss = semantic CLIP loss + regularization, not strict pixel-level MSE

Model learns semantic similarity and global feature. Because of this PSNR value is not expected to improve in this stage

Learned quickly at the beginning
Epoch 1 → 5:
Train loss: **2.76 → ~2.02**
Val loss: **2.73 → ~2.02**
*PSNR: around 10 dB

After epoch ~5, slow-improvement / plateau regime
Train loss keeps decreasing almost monotonically down to ~**1.55–1.6**
Val loss decreases much more slowly and then hovers around **1.84 ± ~0.01** from roughly epoch 15 onward
*Val PSNR stays almost flat around 9.8–10.0 dB, with tiny oscillations.

*The model was trained on single GPU 4080 Super
16GB, GDDR6X Memory, BW 256 bit
10240 cores, 320 TMUS, 112 ROPS

Training concluded in 32 hours

SpikeCLIP: Stage 1 Output

Fig 33. Stage 1 SNN Output - 1

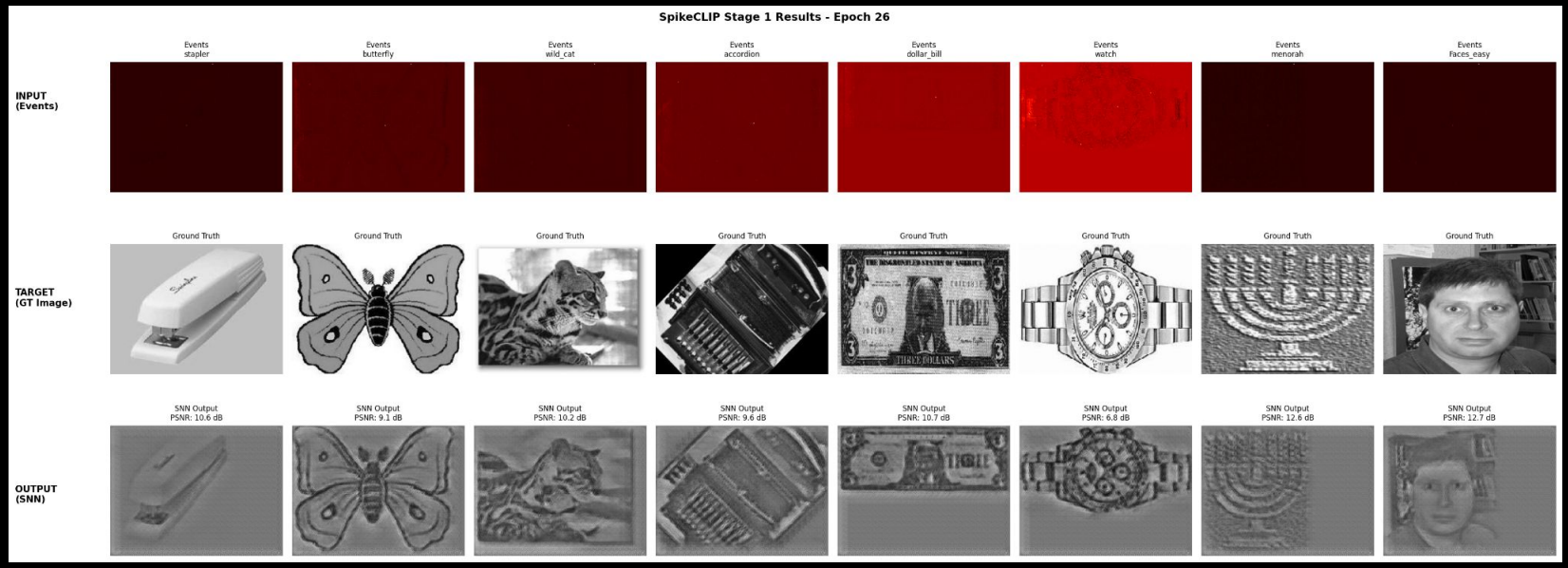


Fig 34. Stage 1 SNN Output - 2

SpikeCLIP: Stage 1 Output

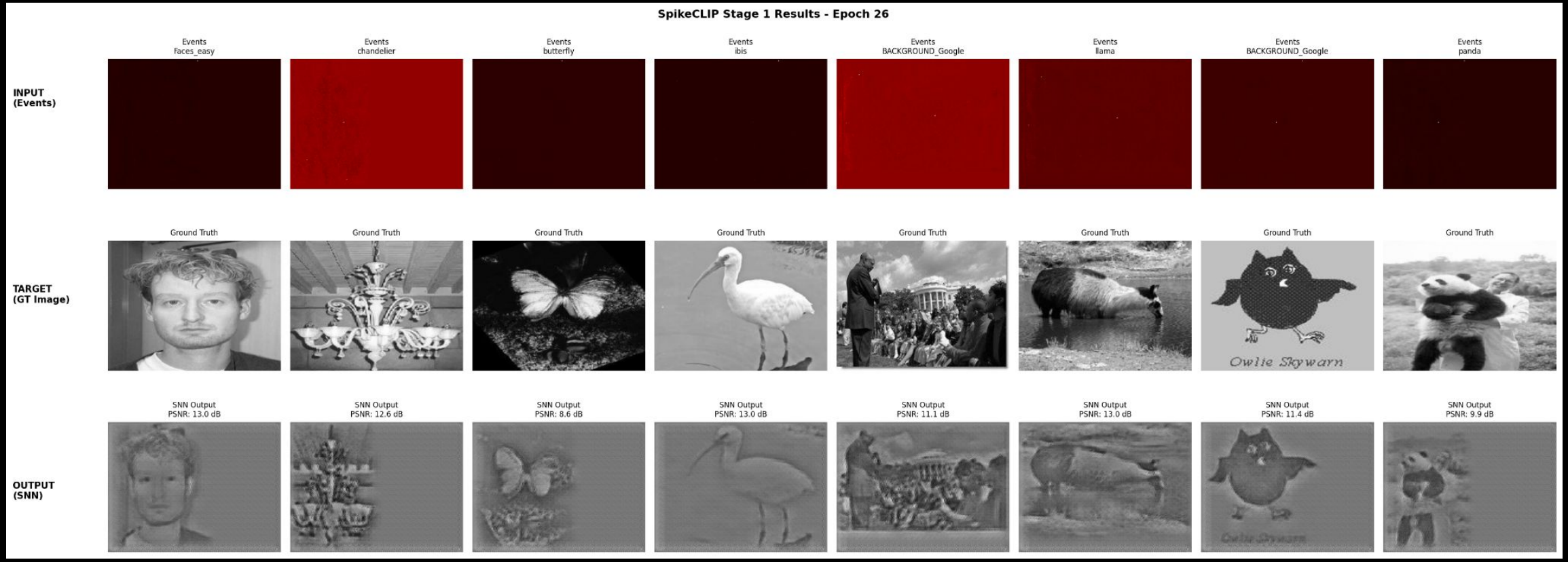


Fig 35. Stage 1 SNN Output - 3

SpikeCLIP: Stage 1 Output

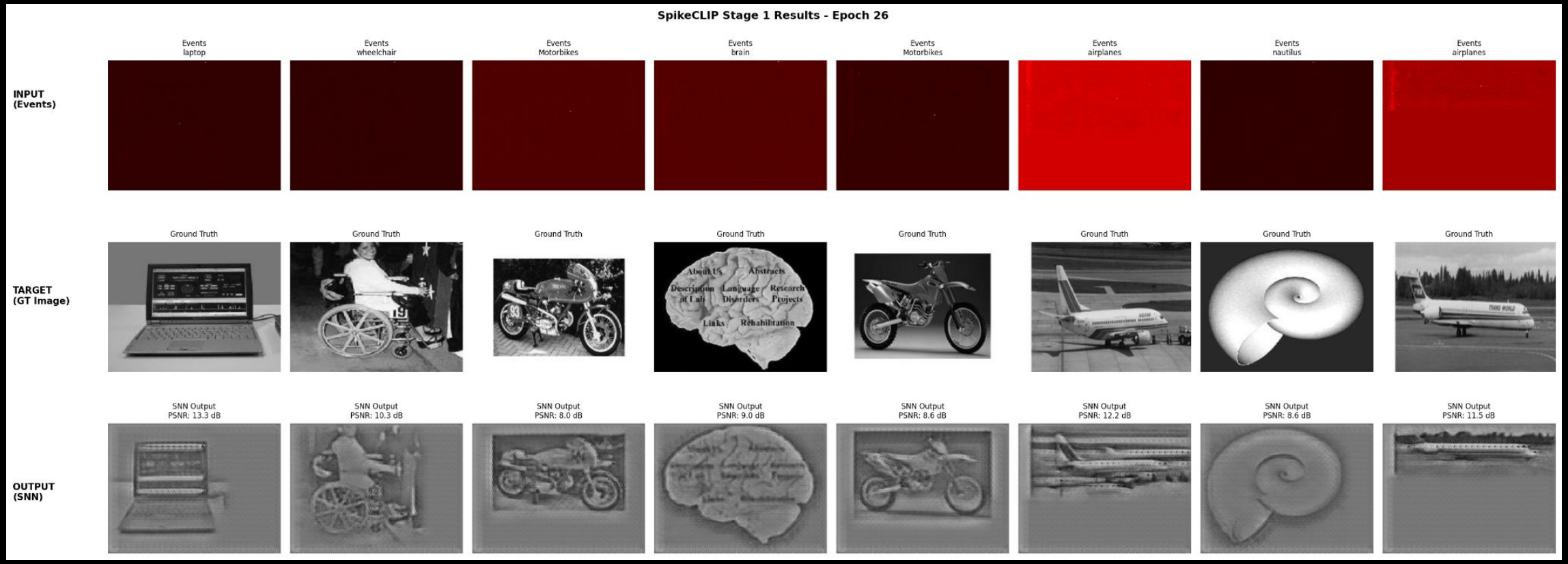


Fig 36. Stage 1 SNN Output - 4

SpikeCLIP: Stage 1 Output

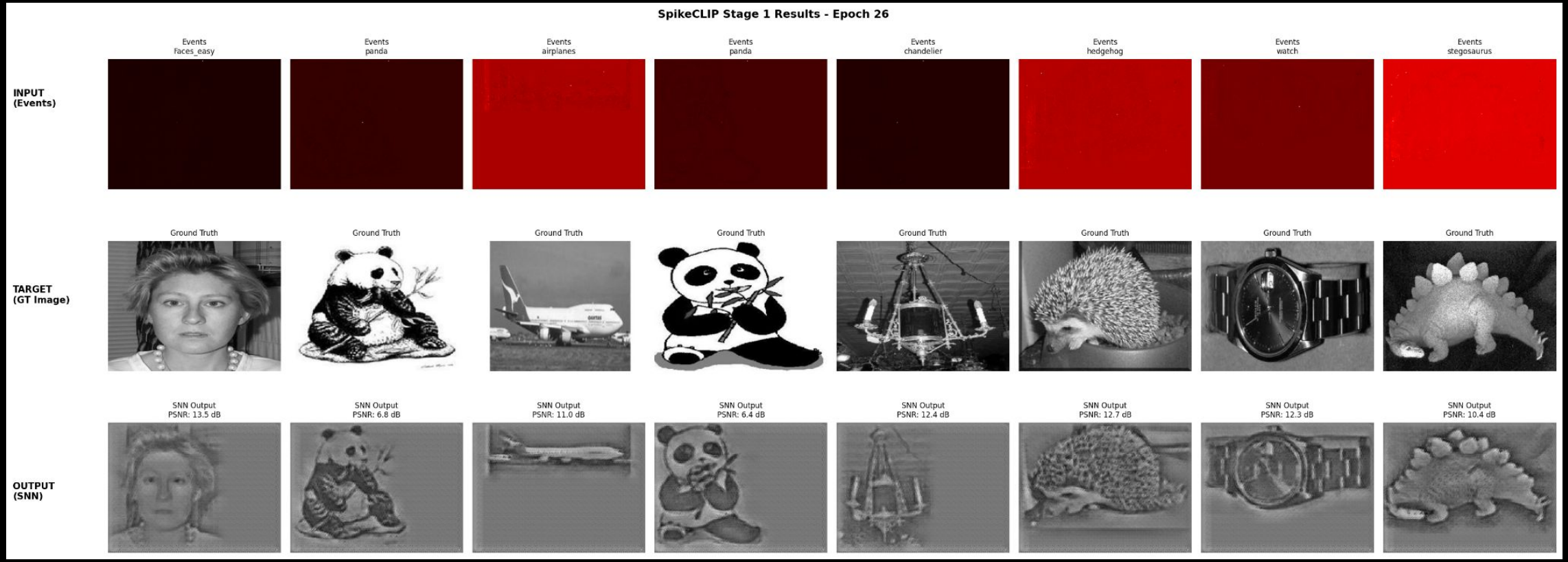


Fig 37. Stage 1 SNN Output - 5

SpikeCLIP: What was tried? But didn't work

MSE Loss Alone (Initial Approach)

- First tried pure MSE loss with paired N-Caltech101 + Caltech101 images
- This violates the SpikeCLIP paper's core philosophy (weak supervision without paired data)

Solution : Switched to pure InfoNCE loss with CLIP

Loss Stuck at ~2.77-2.08 (Flat Training)

- Early training showed loss stuck without improvement

Solution : Steeper surrogate gradient

Small Batch Size Bottleneck

- Contrastive learning needs more negatives for meaningful gradients
- Batch=8 means log(8)=2.08, matching stuck loss

Solution : Increased batch size to 32 = log(32) = 3.47, gives 31 negatives

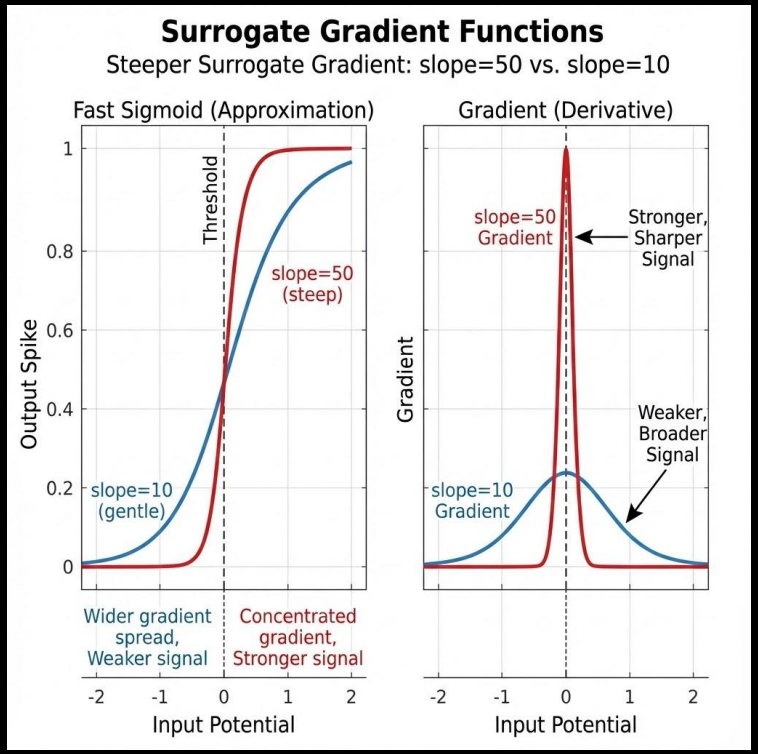


Fig 38. Surrogate Gradient Functions side by side

Stage 2 : Prompt Learning

Learn prompts that distinguish High Quality vs Low Quality

Learn two prompts (HQ and LQ) that distinguish high-quality from low-quality images. These prompts guide Stage 3 to improve quality.
Paper approach:

- Train a PromptCLIP model to classify HQ vs LQ images
- Learn context tokens that capture quality differences
- Use these prompts in Stage 3 to push images toward HQ

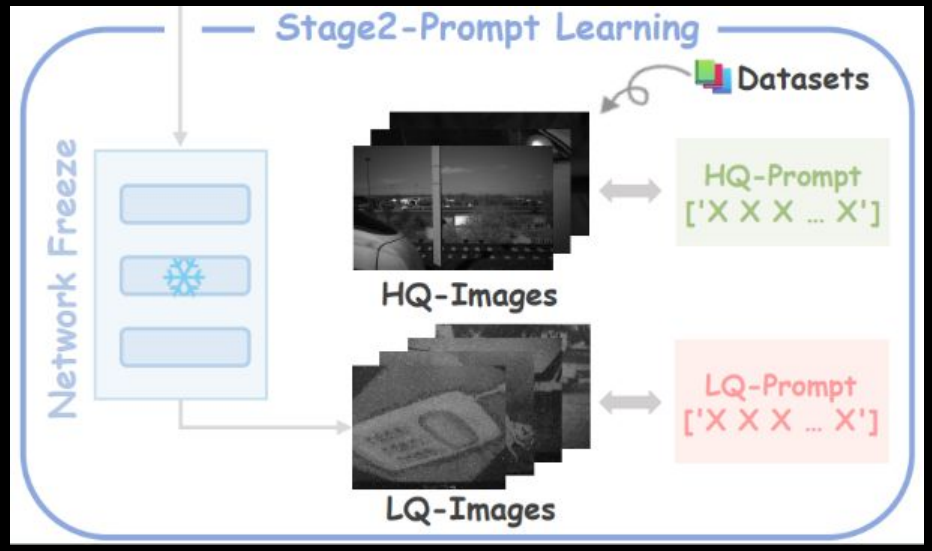


Fig 39. Paper's Stage 2 implementation

Stage 2 : Initial Approach

What We Built

Architecture:

- PromptCLIP model (*CoOp-style)
- 4 learnable context tokens: [V1] [V2] [V3] [V4]
- Two prompts: "a photo of [V1] [V2] [V3] [V4] high quality image" and "a photo of [V1] [V2] [V3] [V4] low quality image"
- Binary classifier: HQ (label 1) vs LQ (label 0)

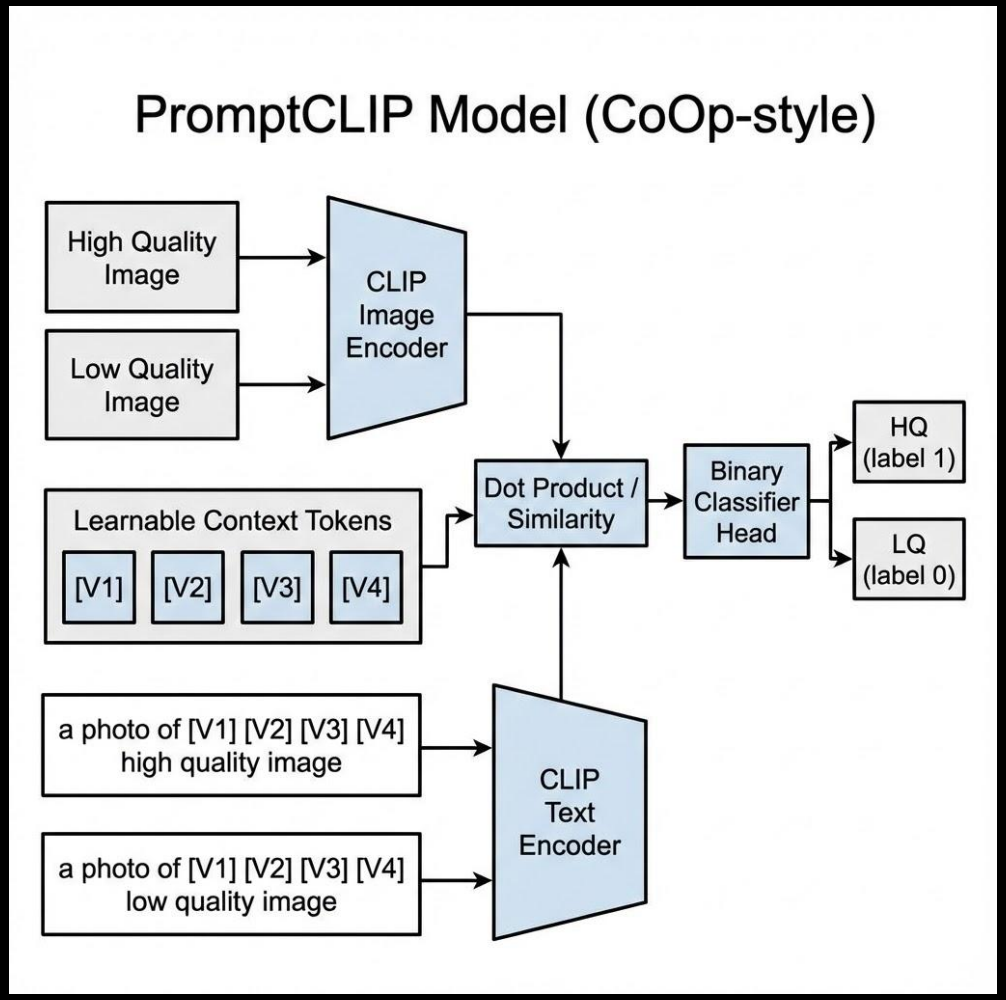


Fig 40. PromptCLIP Model (CoOp-style)

Stage 2 : Initial Approach

What We Built

Dataset Creation:

Used frozen Stage 1 SNN to generate LQ images

- Used ground truth images as HQ
- Created pairs: (LQ_image, label=0) and (HQ_image, label=1)

This is where initial approach becomes a stealth bug

If we set HQ = Caltech101 dataset;
Question becomes : Is this a REAL PHOTO or a NEURAL NETWORK OUTPUT?

However, we want;
What makes an image HIGH QUALITY vs LOW QUALITY

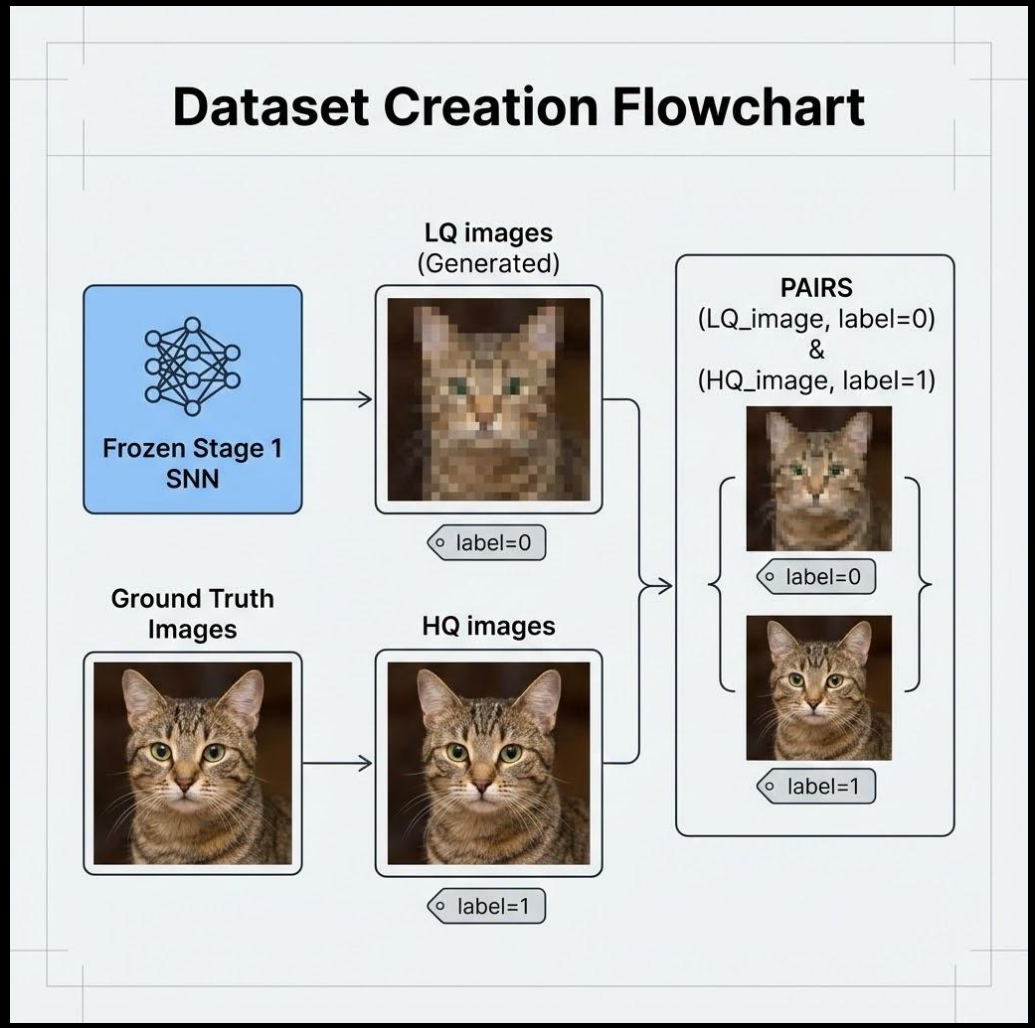


Fig 41. Dataset Creation Flowchart

Stage 2 : Initial Approach

What We Built

Dataset Creation:

Used frozen Stage 1 SNN to generate LQ images

- Used ground truth images as HQ
- Created pairs: (LQ_image, label=0) and (HQ_image, label=1)

This is where initial approach becomes a stealth bug

If we set HQ = Caltech101 dataset;
Question becomes : Is this a REAL PHOTO or a NEURAL NETWORK OUTPUT?

However, we want;
What makes an image HIGH QUALITY vs LOW QUALITY

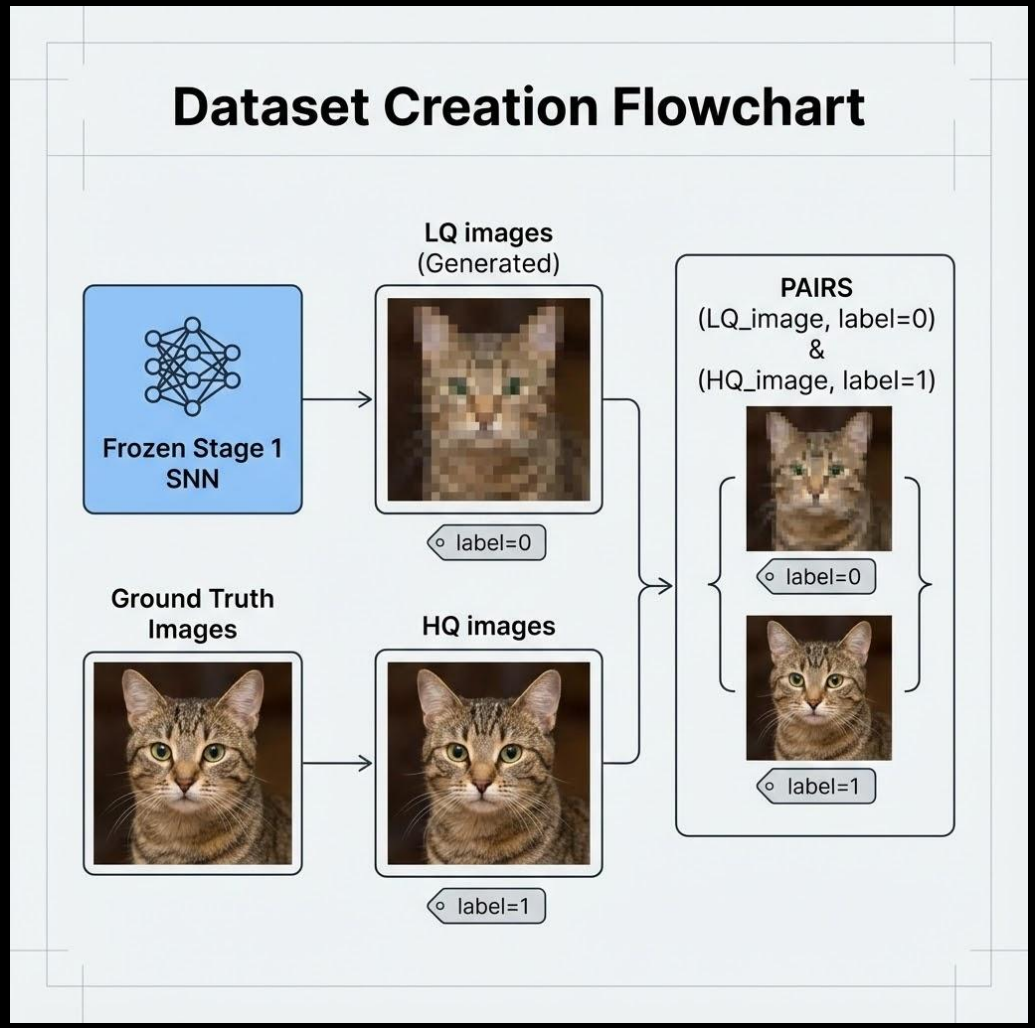


Fig 41. Dataset Creation Flowchart

Stage 2 : Initial Approach

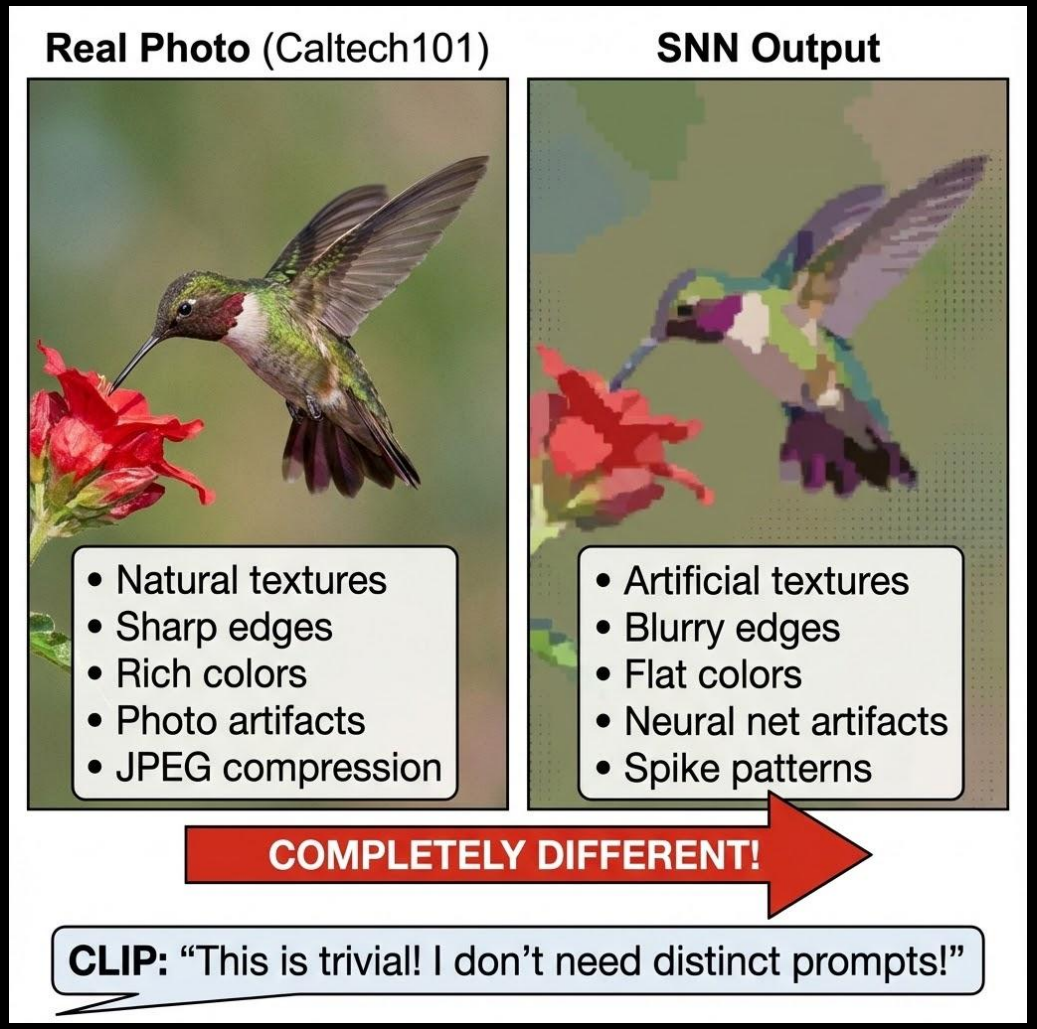


Fig 42. Real photo vs SNN output

SNN Output

- In stage 2 we set HQ = Caltech101 Images
- No error was thrown
- <1% label difference

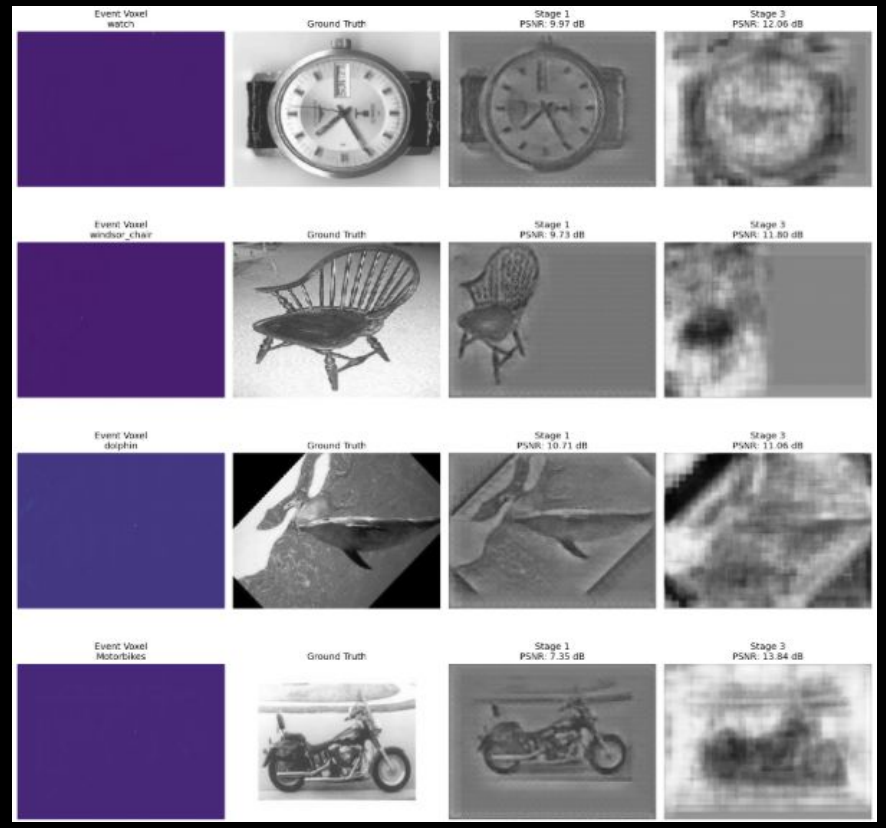


Fig 43. Output after Stage 3, where in Stage 2 MSE loss is used

Stage 2 : HQ/LQ Change

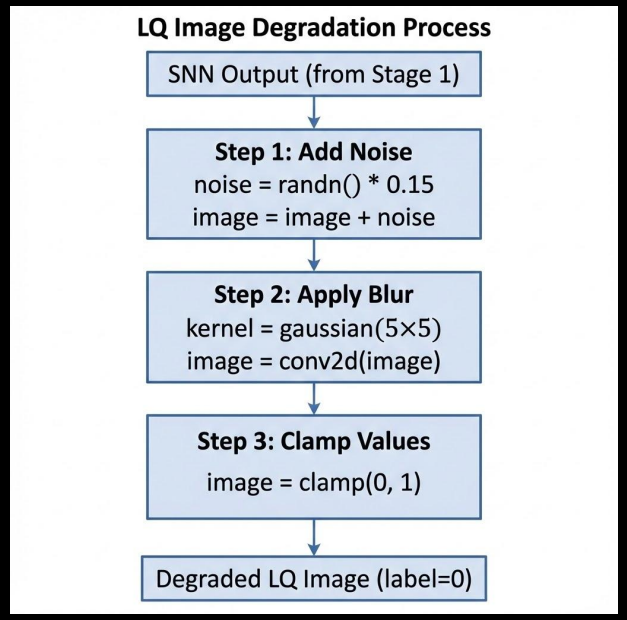


Fig 44. LQ Image Degradation Process

Gaussian noise:
- Standard deviation: 0.15
- Adds random noise
- Makes images noisier

Gaussian blur:
- Kernel size: 5x5
- Sigma: 1.0
- Makes images blurrier

Combined degradation:
- Apply both noise and blur to LQ images
- Creates a clear visual difference from HQ

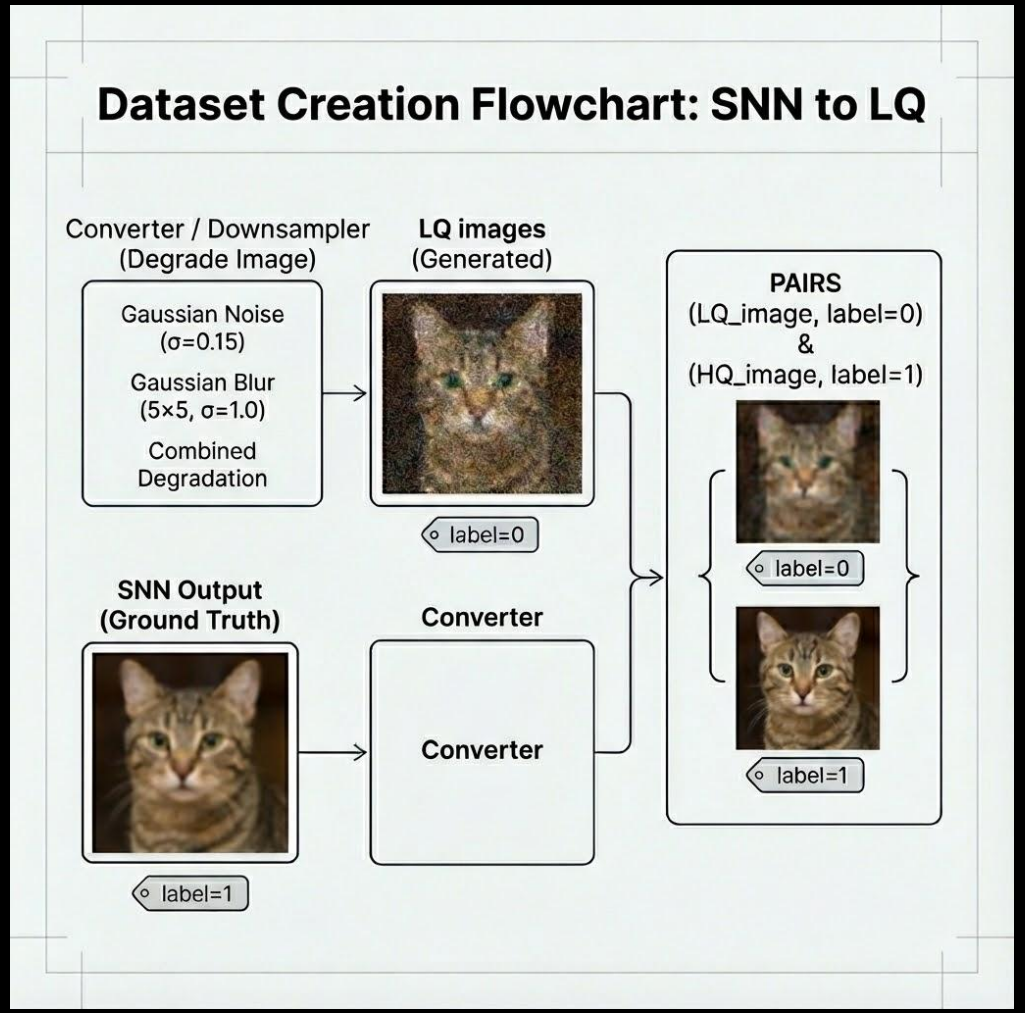


Fig 45. Dataset Creation Flowchart: SNN to LQ

Stage 2 : Hyperparameter Adjustments

	Initial	Final	Reason	Effect
Learning Rate	0.001	0.002	Need stronger updates to learn distinct prompts	Faster convergence, more exploration
Epochs	50	5	Overfits after epoc 5	%100 validation accuracy >= %7 prompt difference (<1% while using HQ=NC) 8% prompt difference
Weight Decay	None	1e-4	Regularization to prevent overfitting	More generalizable prompts
Label Smoothing	0.1	0.2	Prevents overconfidence	Softer prompts, better generalization

Stage 2 : Training Results

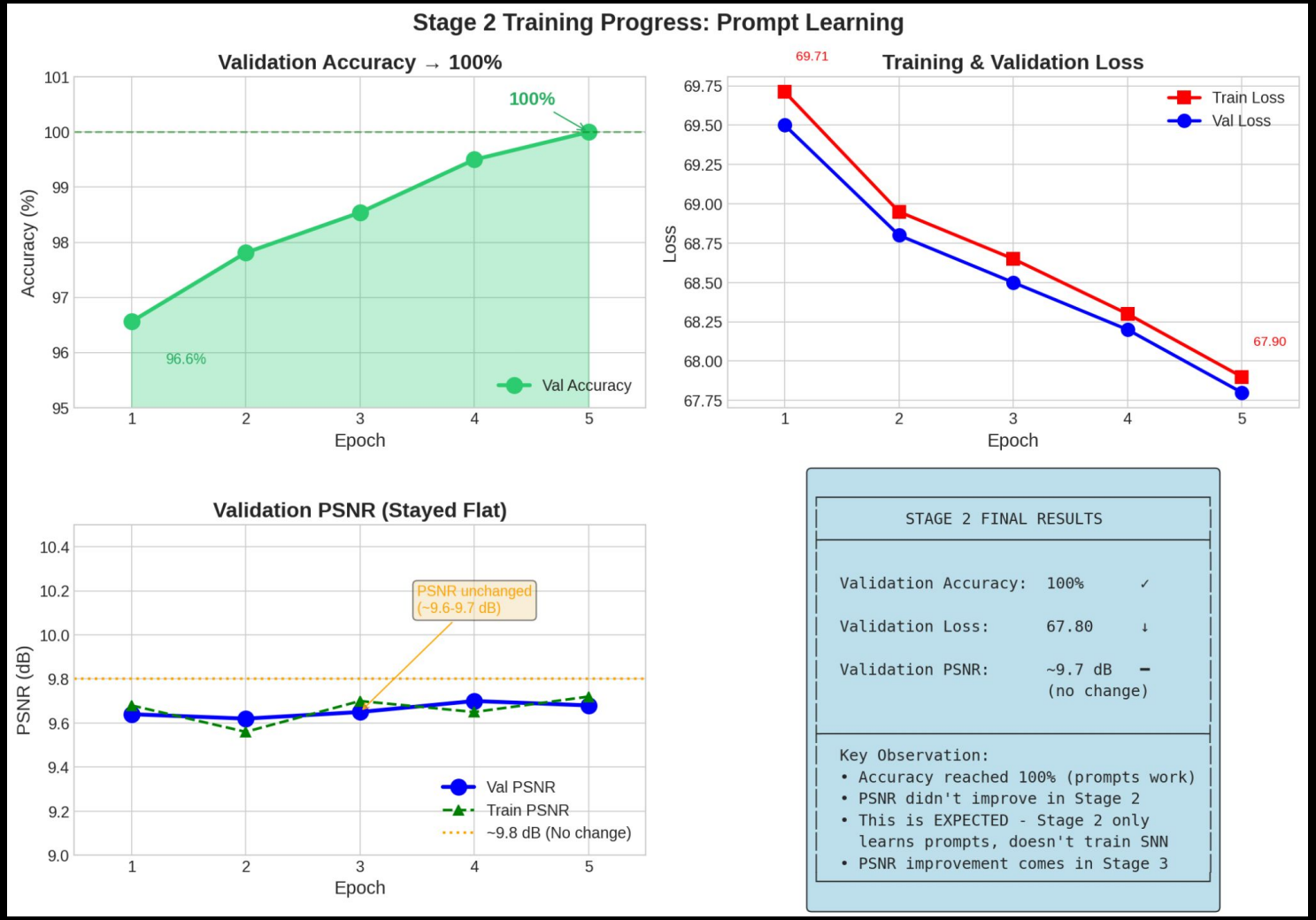


Fig 46. Stage 2 Training Results

Stage 2 : The Complete Stage 2 Pipeline

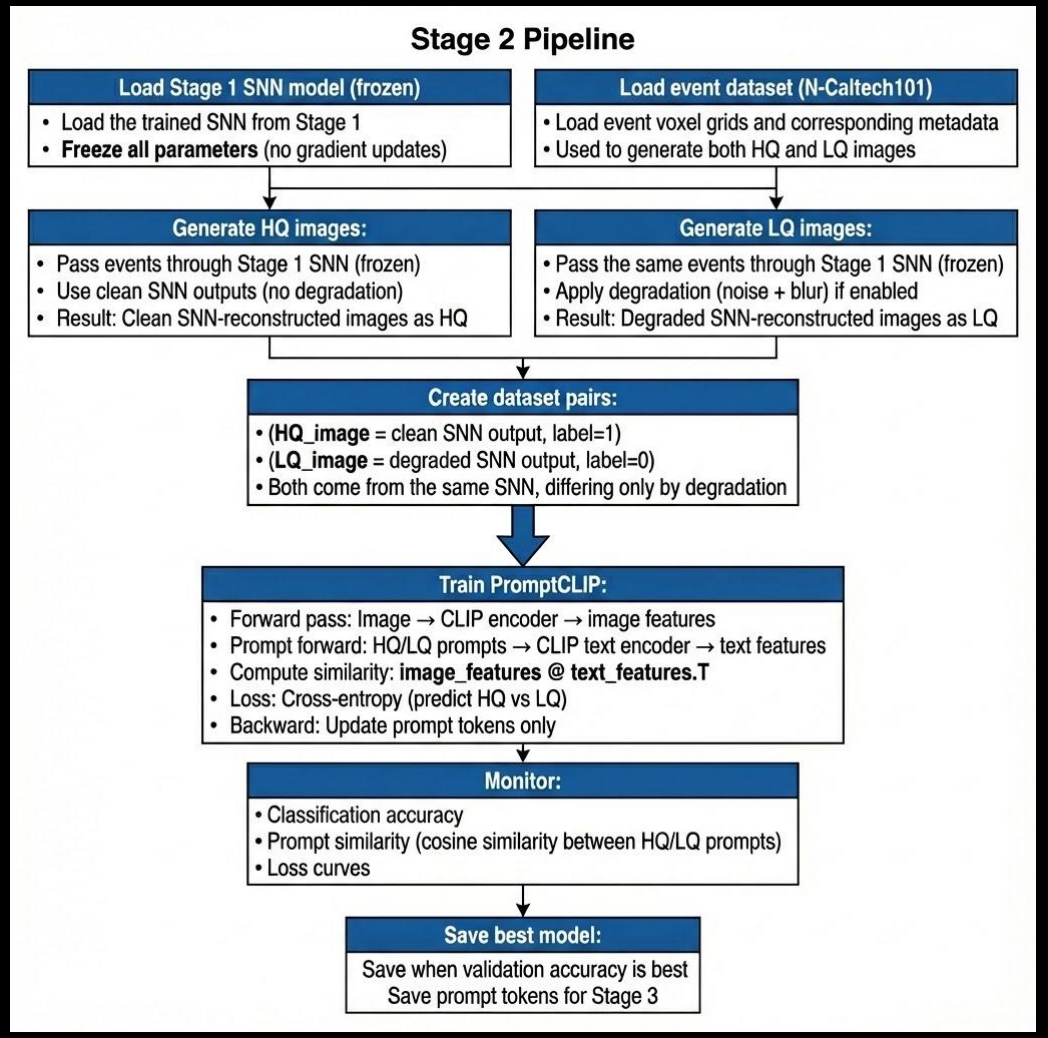


Fig 47. Complete Stage 2 Pipeline

Stage 2 : Key Insights

Classification accuracy alone isn't enough
Solution : Monitor prompt similarity

Quality gap matter:
- Similar quality gap -> similar prompts
- Large quality gap -> distinct prompts

Solution : Degradation helps create a clear gap

Hyperparameters affect prompt learning
- Higher learning rate -> More exploration
- Smaller epochs -> Better convergence
- Weight decay -> Better generalization

Python caching can cause issues:
Solution : Check __pycache__ when code changes don't take effect
importlib.reload() if needed

Stage 3 : Quality-Guided SNN Fine-Tuning

Fine-tune the Stage 1 SNN using the Stage 2 prompts to improve reconstruction quality

Concept:

- Stage 1: SNN learns basic reconstruction (semantic alignment)
- Stage 2: Learns HQ/LQ prompts to distinguish quality
- Stage 3: Uses these prompts to guide the SNN toward higher quality

Paper approach:

- Combine two losses:
  - semantic alignment (L_class) and quality guidance (L_prompt)
- Push SNN outputs toward HQ prompts and away from LQ prompts
- Maintain semantic correctness while improving quality

Stage 3 : Quality-Guided SNN Fine-Tuning

Architecture Components

1. SNN Model
   a. Loaded from Stage 1 checkpoint
   b. All parameters trainable (258,273)
   c. Processes event voxels -> reconstructed images
2. CLIP Model
   a. ViT-B/32 encoder
   b. All parameters frozen
   c. Extracts images features for loss computation
3. Prompt model (frozen)
   a. Loaded from Stage 2 checkpoint
   b. All parameters frozen
   c. Provides HQ/LQ text features
4. Loss Function
   a. Combined loss
   b. Lambda

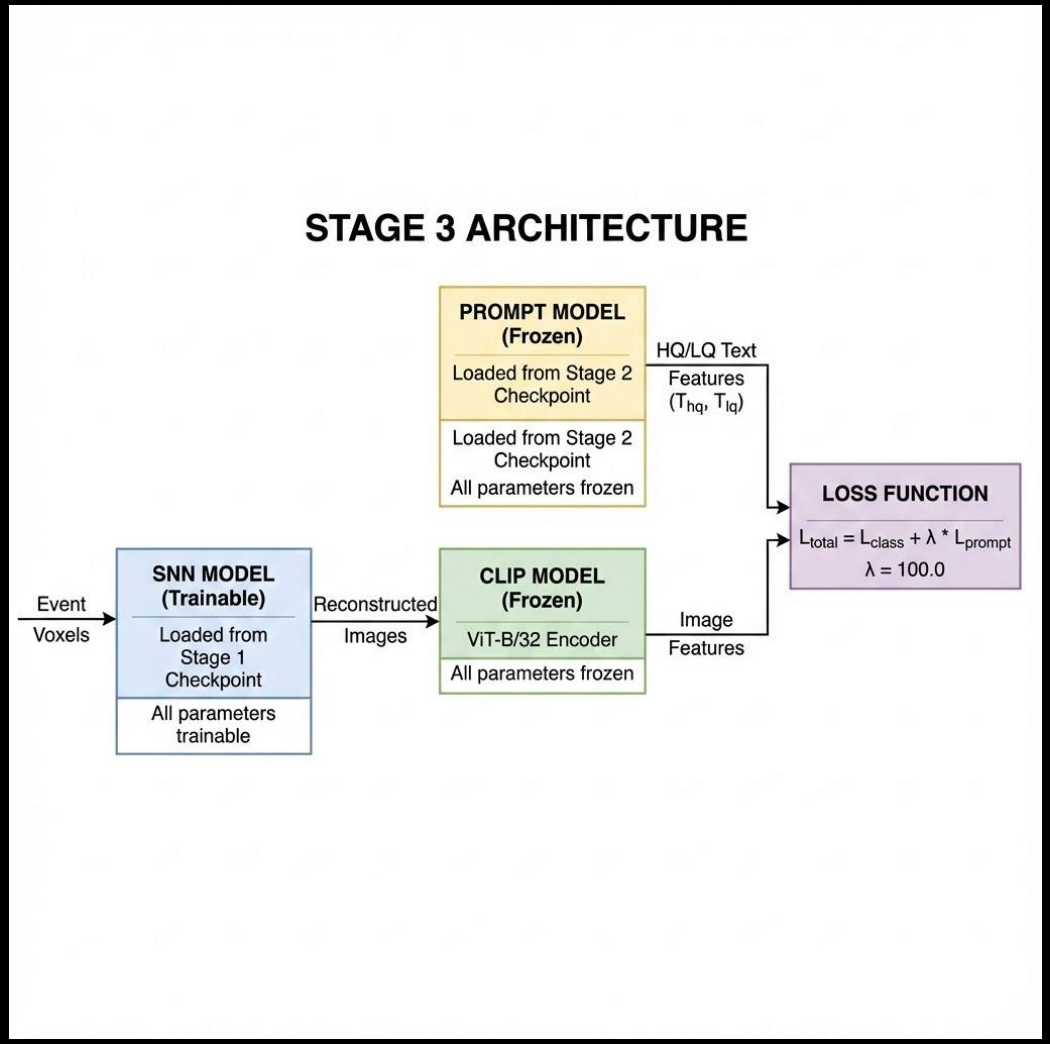


Fig 48. Stage 3 Architecture

Stage 3 : Model Loading and Setup

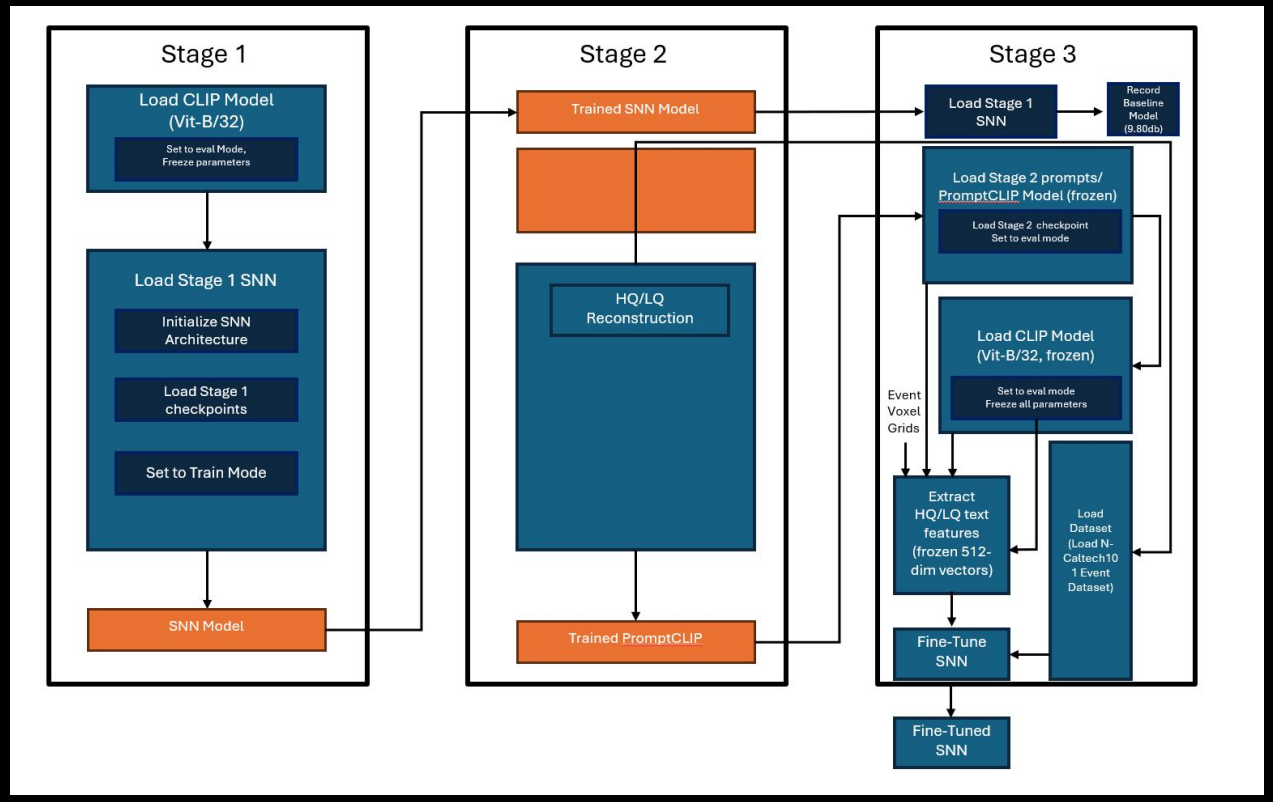


Fig 49. Model Loading and Setup

1. Load CLIP Model
   a. Load ViT-B/32
   b. Set to eval mode
   c. Freeze all parameters

2. Load Stage 1 SNN
   a. Load Stage 1 checkpoint
   b. Set to train mode (will-be fine tuned)
   c. Record baseline PSNR (9.80db)

3. Load Stage 2 Prompts
   a. Load PromptCLIP model
   b. Load Stage 2 checkpoint
   c. Set to eval mode
   d. Freeze all parameters
   e. Extract HQ/LQ text features
      i. HQ_text_features (512-dim vector)
      ii. LQ_text_features (512-dim vector)

4. Load Dataset
   a. Load N-Caltech 101 event dataset
   b. Get class names for class text generation
   c. Split into train/val (80/20)

Stage 3 : Training Loop (Per Epoch)

For each batch:

1. Forward pass through SNN
   a. Input: Event voxels (B, 5 , H, W)
   b. Output: Reconstructed images (B, 1, H, W)
   c. Uses 50 timestamps for temporal integration

2. Prepare images for CLIP
   a. Resize to 224x224 (CLIP input size)
   b. Convert grayscale to RGB (repeat channel 3x)
   c. Normalize using CLIP's mean/std
      i. Mean [0.4815, 0.4578, 0.4082]
      ii. Std [0.2686, 0.2613, 0.2758]

3. Extract CLIP image features
   a. Pass through CLIP image encoder
   b. Get features (B, 512)
   c. L2 normalize features
   d. no torch.no_grad(), gradients must flow back to SNN

4. Generate class text features
   a. For each label in batch, create text " a photo of a {class_name} "
   b. Tokenize texts
   c. Pass through CLIP text encoder (frozen)
   d. Get class text features (B, 512)
   e. L2 normalize features
   f. use torch.no_grad() (text encoder is frozen)

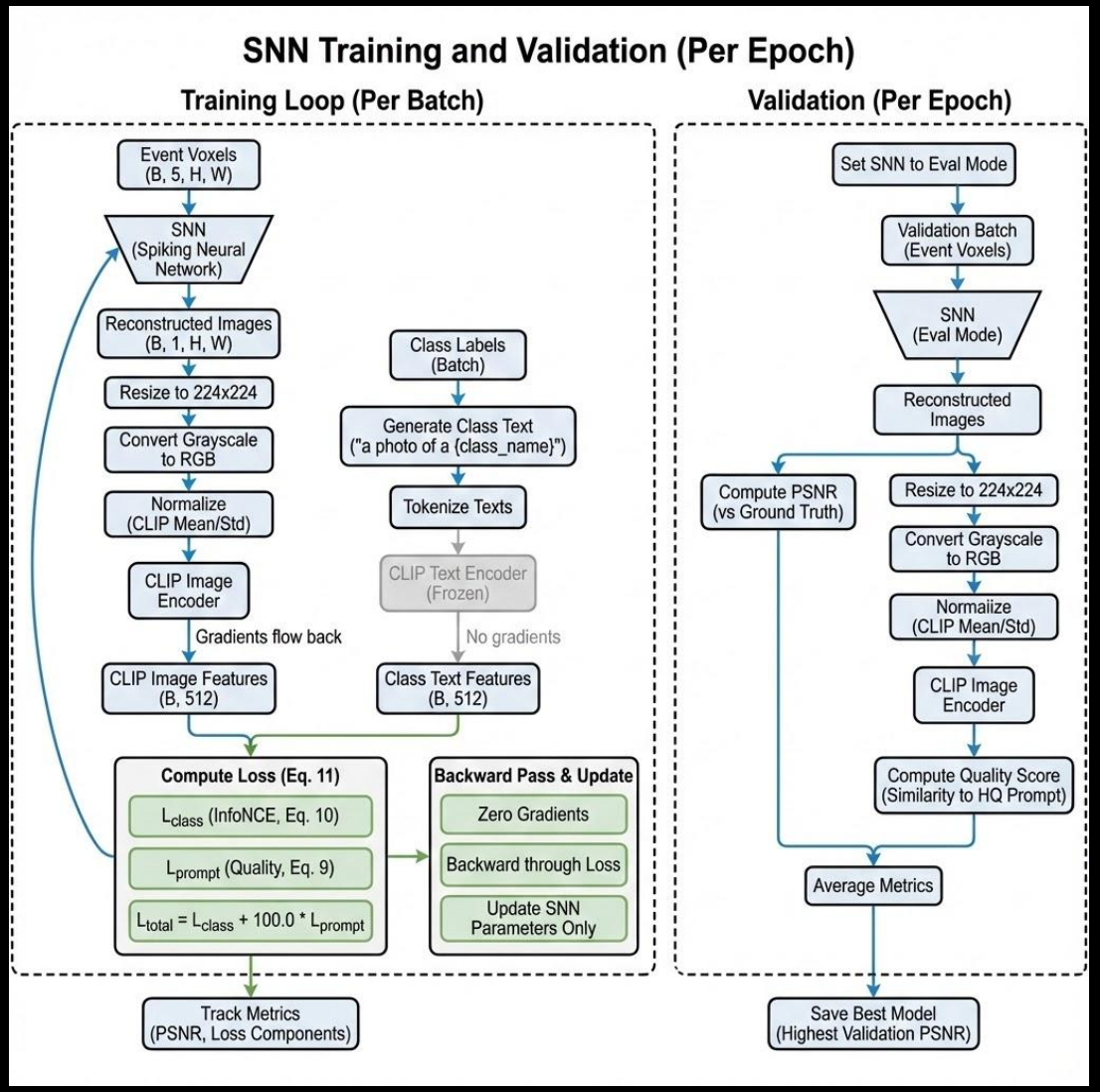


Fig 50. SNN Training and Validation (Per epoch)

Stage 3 : Training Loop (Per Epoch)

For each batch:

5. Compute Loss (paper Eq. 11)
   a. L_class: InfoNCE loss semantic alignment (Eq. 10)
   b. L_prompt: Quality guidance loss (Eq. 9)
   c. L_total = L_class + 100.0 * L_prompt

6. Backward Pass
   a. Backward through loss
   b. Gradient flow; Loss -> CLIP image features -> SNN output → SNN parameters
   c. Update SNN parameters only

7. Track Metrics
   a. Compute PSNR (SSN output vs ground truth)
   b. Track loss components (class, prompts, total)

Validation (Per epoch)
after every epoch
   a. Set SNN to eval mode
   b. For each validation batch
      i. Forward through SNN (no gradients)
      ii. Compute PSNR
      iii. Extract CLIP features
      iv. Compute quality score
   c. Average metrics
   d. Save best model (highest validation PSNR)

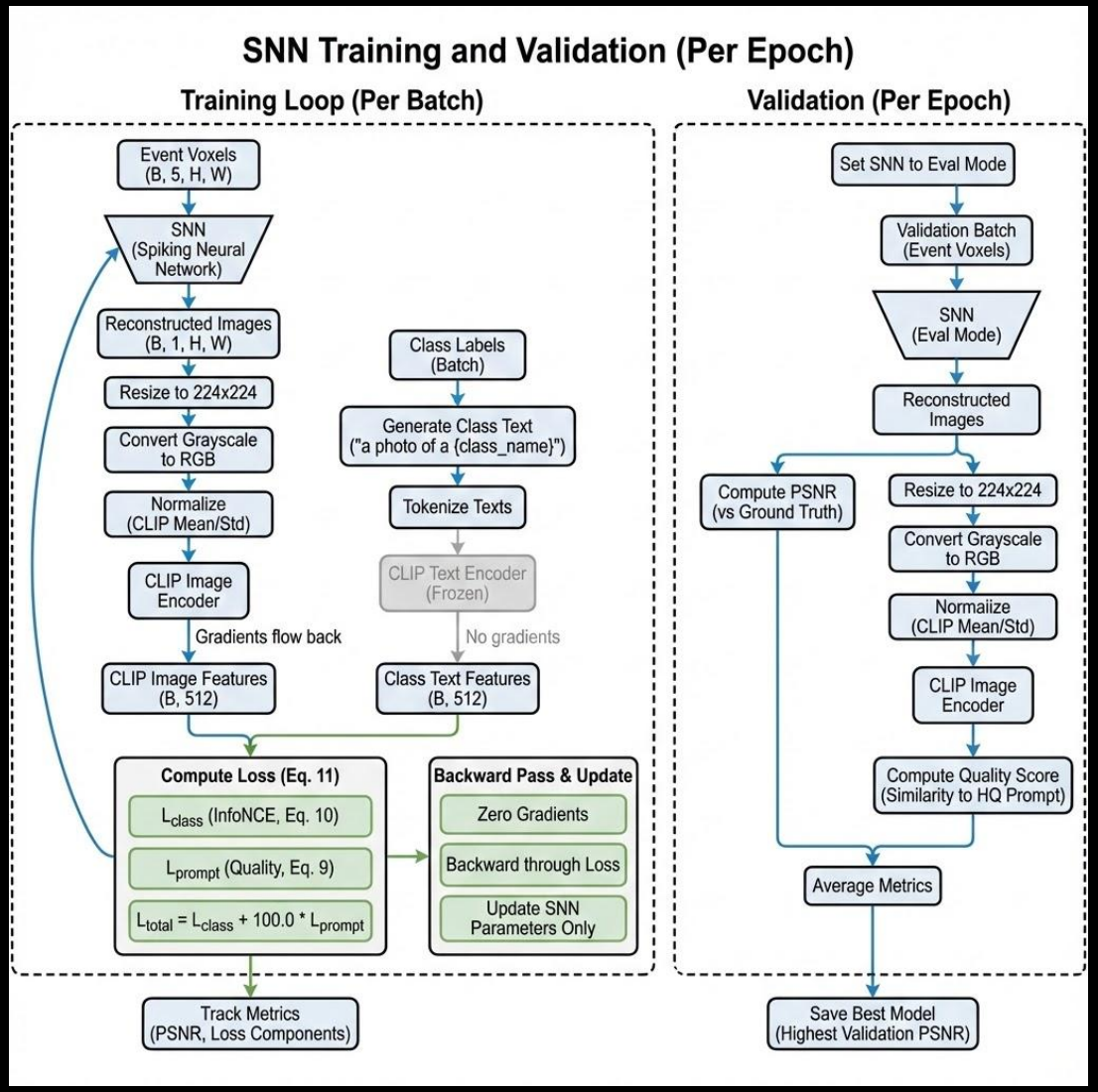


Fig 50. SNN Training and Validation (Per epoch)

Stage 3: Loss Functions In Depth

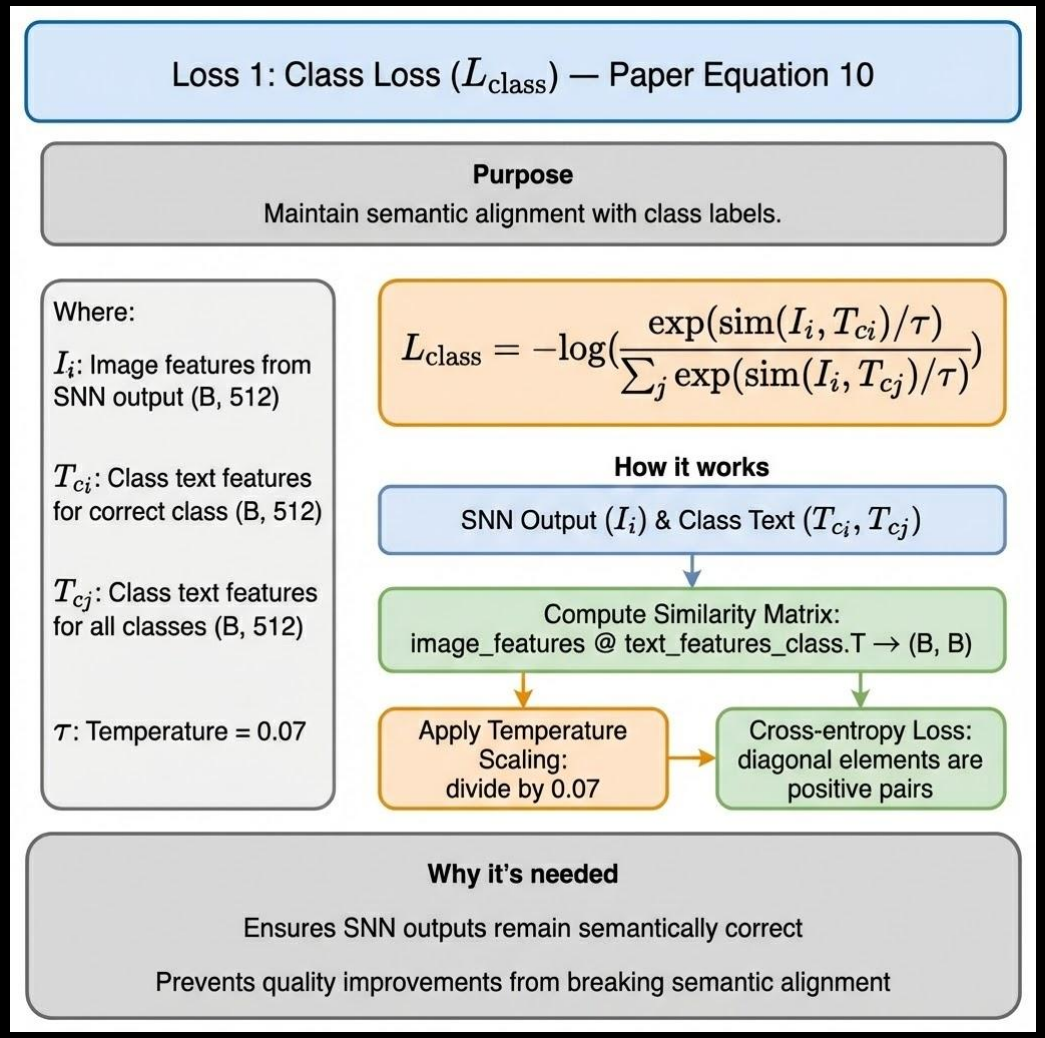


Fig 51. Class Loss

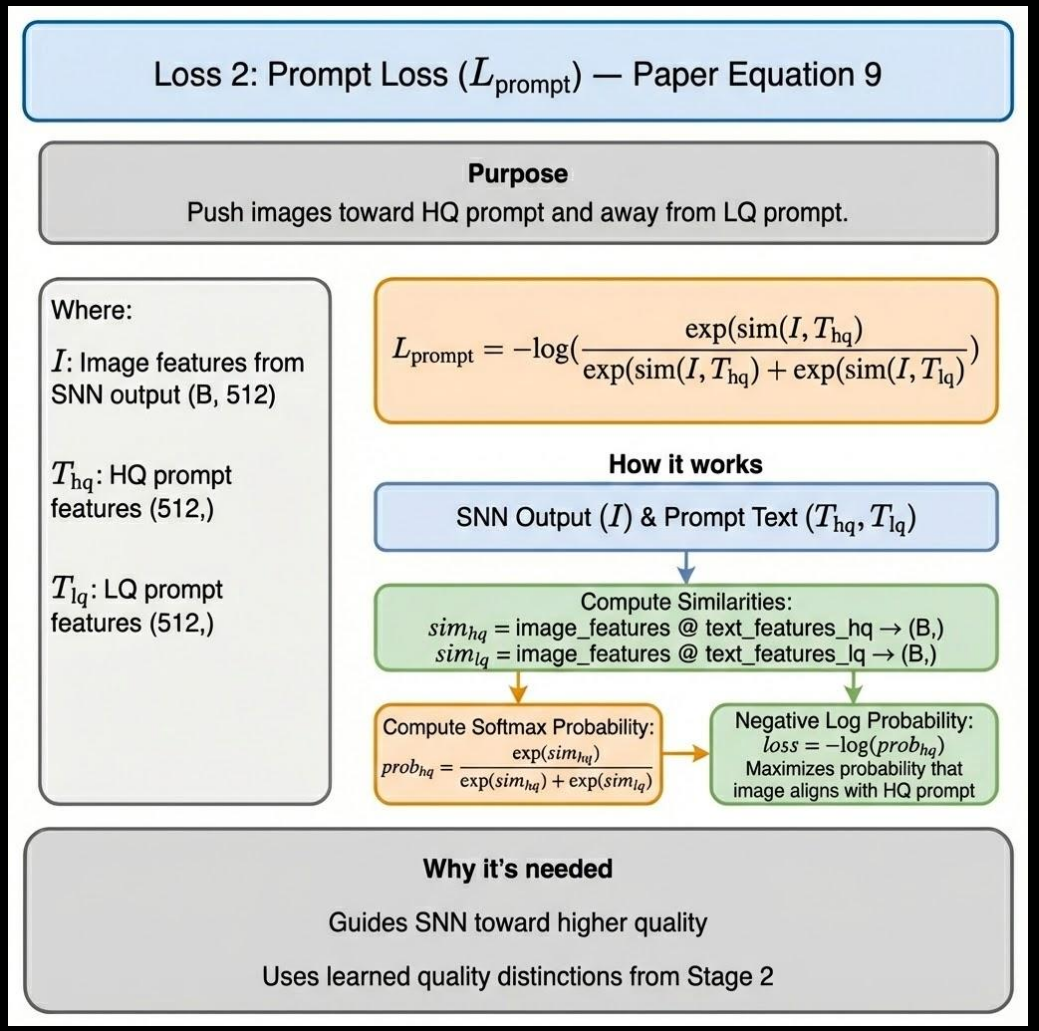


Fig 52. Prompt Loss

Stage 3: Loss Functions In Depth

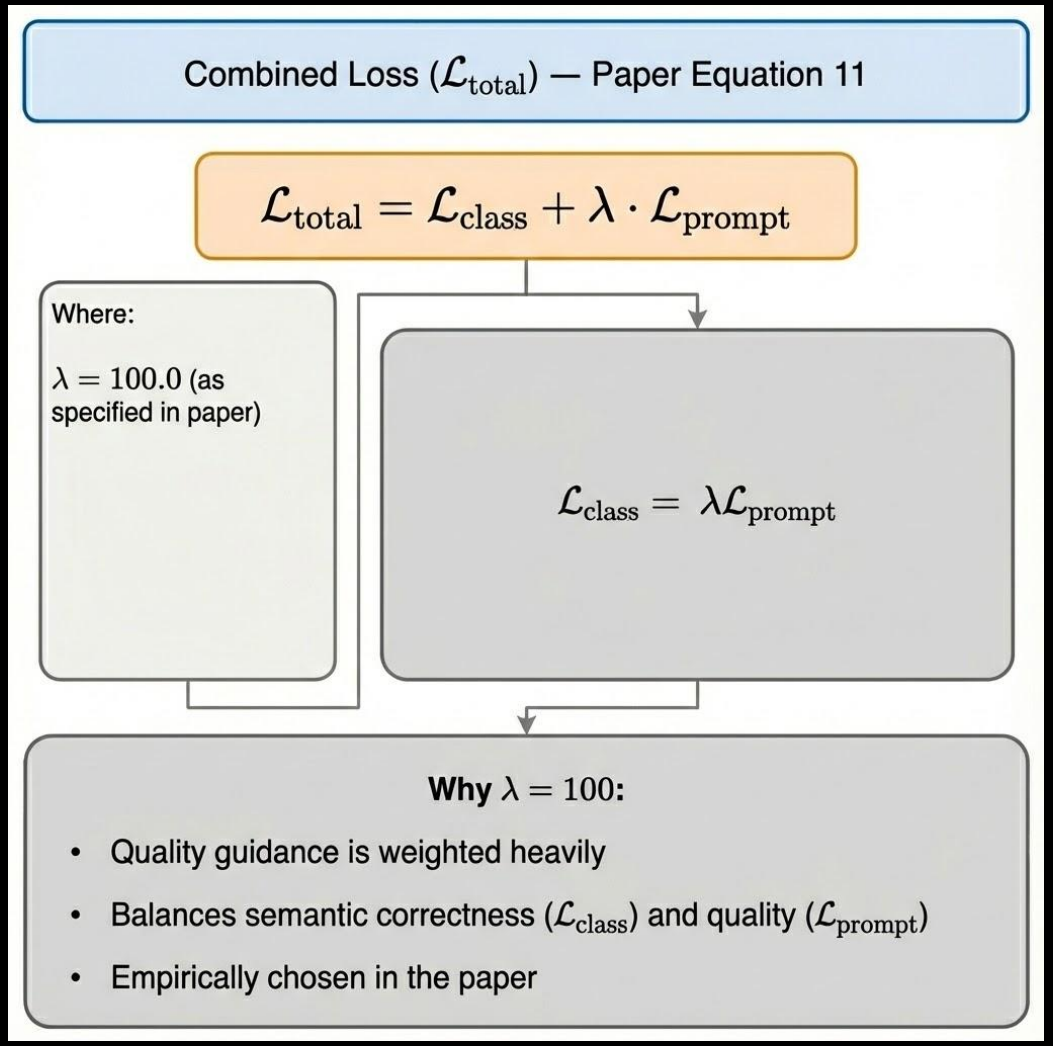


Fig 53. Combined Loss

Stage 3: Training Configuration

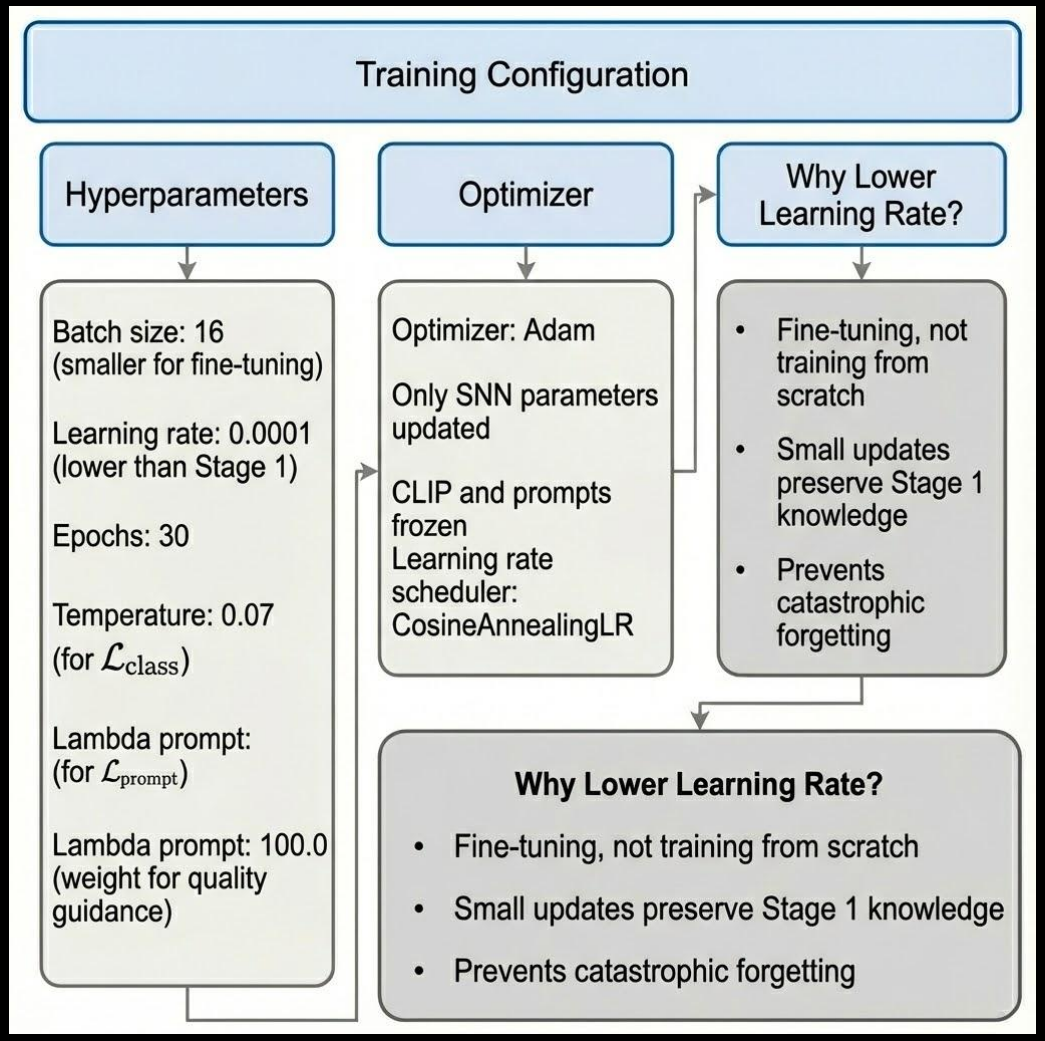


Fig 54. Training Configuration

Stage 3: Training Results

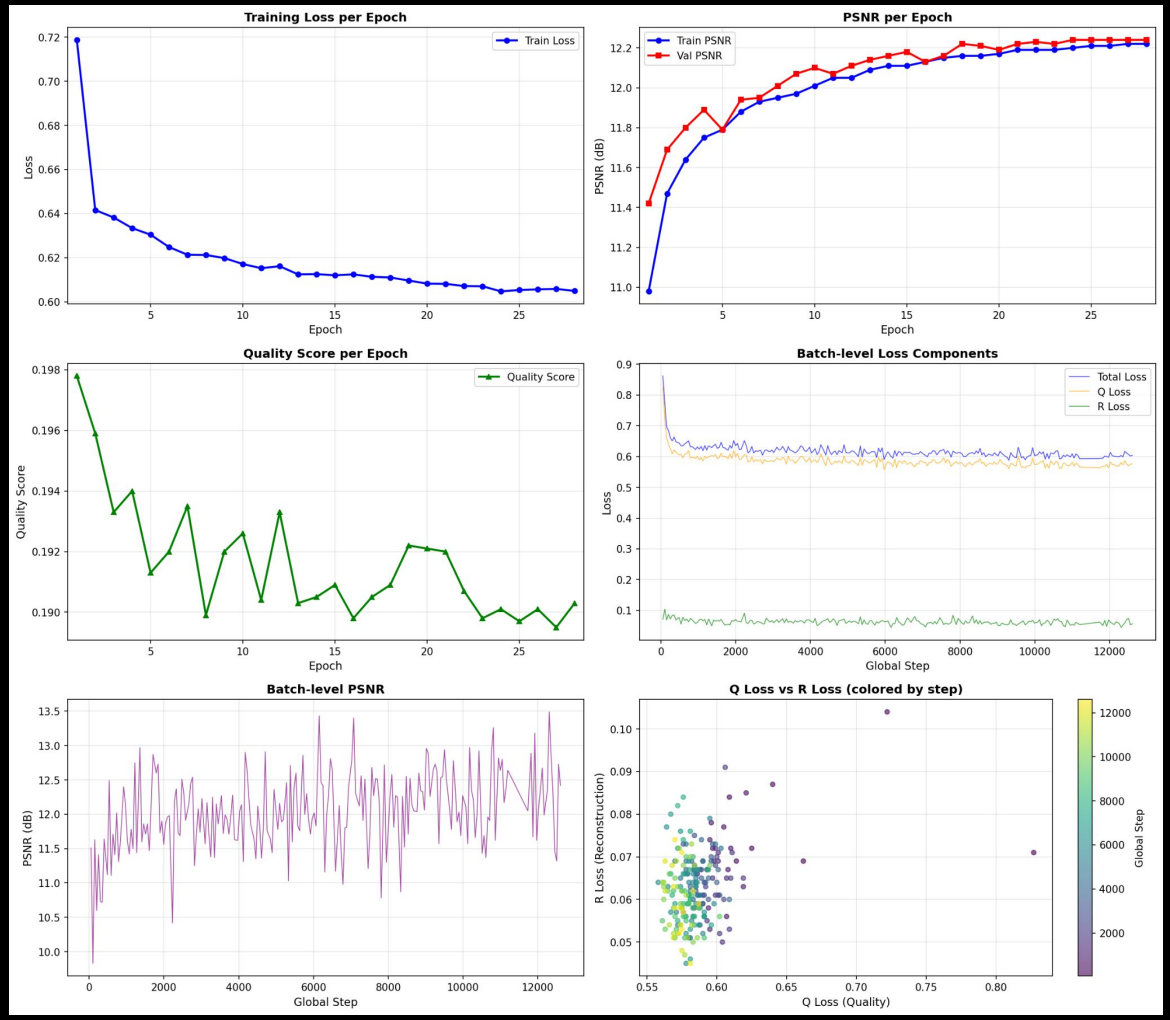


Fig 55. Stage 3 Training Results

Stage 3: Training Results

Epoch [1/30] Batch [50/436] Loss: 70.2653 (Class:1.823 Prompt:0.684) PSNR: 10.67
Epoch [1/30] Batch [100/436] Loss: 70.0059 (Class:2.007 Prompt:0.680) PSNR: 8.57
...
Epoch [1/30] Summary: Train Loss: 69.7114 | Train PSNR: 9.68 dB
Val PSNR: 9.64 dB | Quality Score: 0.2069

Observations:

- Loss decreases over epochs
- Class loss: ~1.8-2.0 (semantic alignment)
- Prompt loss: ~0.67-0.68 (quality guidance)
- PSNR improves from Stage 1 baseline

Final Results

- Stage 1 PSNR: 9.80 dB
- Stage 3 PSNR: ~12.24 dB
- Improvement: +2.44 dB

Quality score:

- Measures similarity to HQ prompt
- Increases over training
- Indicates quality improvement

Stage 3: Training Results

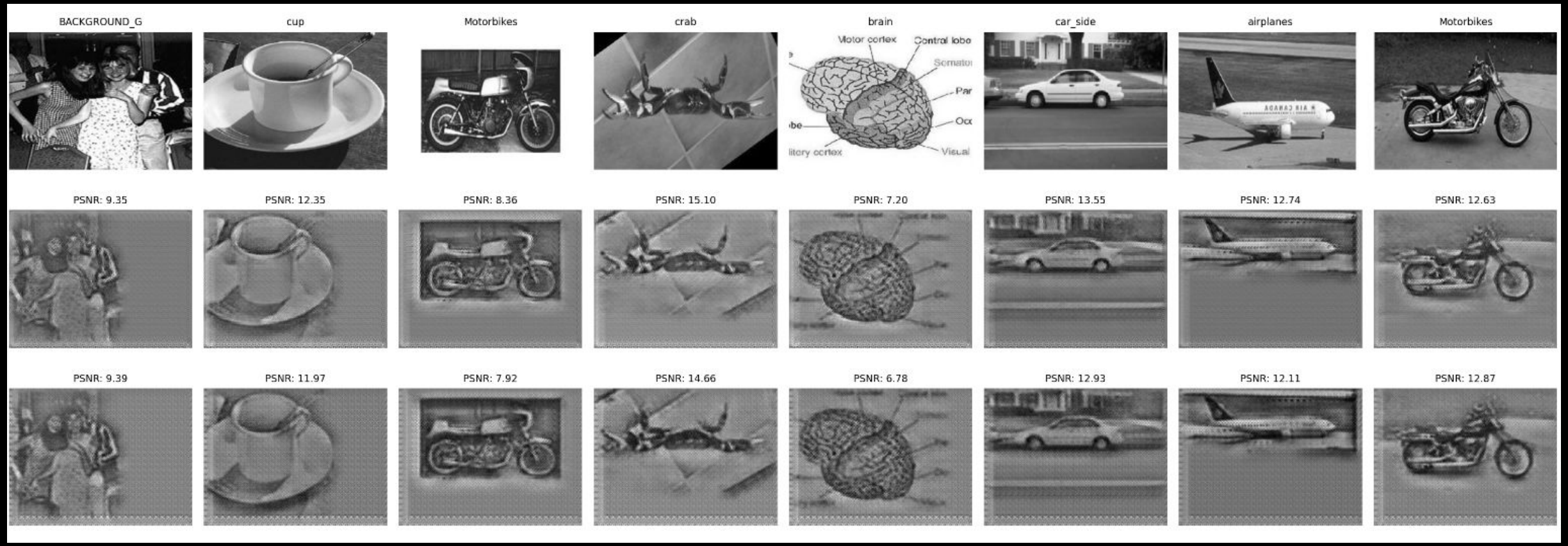


Fig 56. Stage 3 Training Results - 1

Stage 3: Training Results

Latency Results

Batch Size	Mean (ms)	Std (ms)	Max (ms)	Min (ms)	P95 (ms)	P99 (ms)
1	85.985	4.529	105.209	78.673	96.730	99.859
4	97.941	17.987	206.050	82.629	125.513	163.421
8	92.845	8.007	144.867	83.312	104.838	120.790
16	152.339	7.410	183.151	145.907	167.304	174.632
32	378.540	6.124	408.461	372.103	388.908	401.910

Stage 3: Training Results

Latency Results

Batch Size	Throughput (samples/s)	Throughput (images/s)
1	10.83	10.83
4	44.65	11.16
8	91.40	11.43
16	107.41	6.71
32	84.60	2.64

Power Consumption

Idle Power	Active Power	Power Delta
106.46 W	204.11 W	97.65 W

Metric	Batch = 1	Batch = 16
Input Shape	(1, 5, 180, 240)	(16, 5, 180, 240)
Temporal Steps	50	50
Mean Latency	85.99 ms	152.34 ms
Median Latency	85.05 ms	–
Latency per Sample	85.99 ms	9.52 ms
Throughput (samples / second)	10.83	107.41
Throughput (images / second)	10.83	6.71
FLOPs per Inference	24.30 GFLOPs	24.30 GFLOPs (same)
Active Power	204.11 W	(measured at bs=16)
Inference Power (delta)	97.65 W	(active − idle)

Fig 57. Performance Metrics

Stage 3: Key Insights & Learnings

What worked well?

a. Exact paper formulation
   i. Quality improves while maintaining semantics

b. Gradient flow
   i. Removing no_grad() was critical
   ii. Quality guidance now updates SNN

c. Clas text features
   i. Essential for semantic alignment
   ii. Prevents quality improvements from breaking semantics

d. Lambda = 100
   i. Strong quality guidance
   ii. Balances semantic and quality objectives

Stage 3: Key Insights & Learnings

What didn't work?

a. Extra losses
   i. Reconstruction/TV losses hurt quality
   ii. Stick to paper formulation

b. Blocked Gradients
   i. torch.no_grad() prevented quality improvement
   ii. Always check gradient flow

c. Missing class features
   i. Without them, semantic alignment breaks
   ii. Both losses are needed

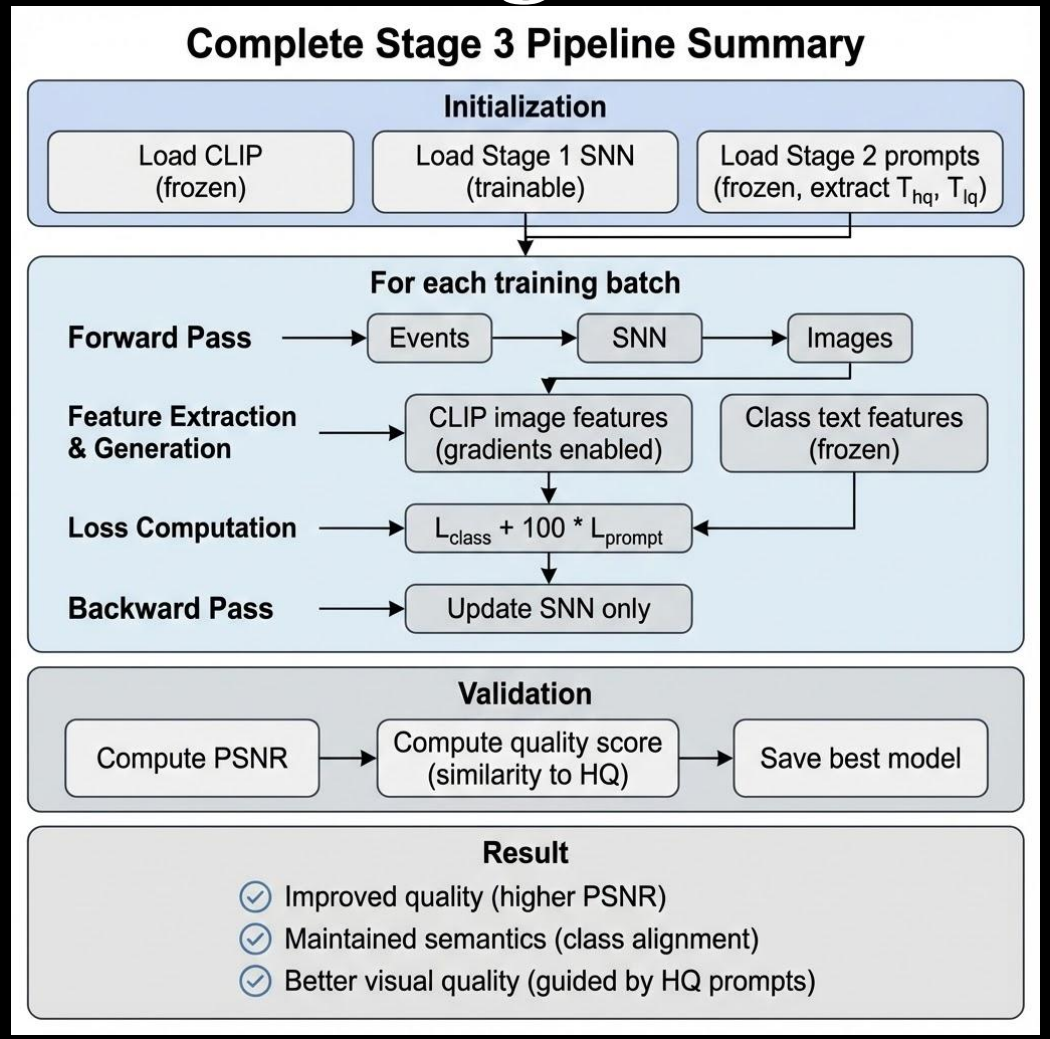


Fig 58. Complete Stage 3 Pipeline Summary

Future Work for SpikeCLIP - SNN

1 - Performance Optimization

DataLoader optimization: increase num_workers and enable pin_memory to reduce GPU idle time

Mixed precision training: use FP16 to speed up training and reduce memory

Model compilation: use torch.compile() for potential speedups

Batch size tuning: find the optimal batch size for throughput

Multi-GPU training: scale to multiple GPUs

2 - Dataset & Generalization

More datasets can be tried

- N-CARS: test on different event camera data
- N-MNIST: evaluate on simpler tasks
- Custom datasets: real-world event camera data
- Synthetic data: generate more training data

3 - Architecture Improvements

SNN Architecture

- Deeper networks: more layers for better reconstruction
- Attention mechanisms: spatial/temporal attention
- Residual connections: improve gradient flow
- Different neuron models: Izhikevich, adaptive threshold

Prompt Learning

- More context tokens: increase `N_CTX` beyond 4
- Multi-level prompts: prompts at different quality levels
- Dynamic prompts: adapt prompts per image
- Prompt ensemble: combine multiple prompt sets

References

[1] SpikeCLIP: Event-Based Image Reconstruction via Spiking Neural Networks with CLIP Guidance. arXiv preprint arXiv:2501.04477, 2025.

[2] A. Radford et al., "Learning Transferable Visual Models From Natural Language Supervision," *Proc. Int. Conf. Mach. Learn.* (ICML), 2021, pp. 8748-8763.

[3] K. Zhou, J. Yang, C. C. Loy, and Z. Liu, "Learning to Prompt for Vision-Language Models," *Int. J. Comput. Vis.*, vol. 130, no. 9, pp. 2337-2348, 2022.

[4] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades," *Frontiers in Neuroscience*, vol. 9, p. 437, 2015.

[5] L. Fei-Fei, R. Fergus, and P. Perona, "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshop* (CVPRW), 2004.

[6] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-based Optimization to Spiking Neural Networks," *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 51-63, 2019.

[7] A. van den Oord, Y. Li, and O. Vinyals, "Representation Learning with Contrastive Predictive Coding," arXiv preprint arXiv:1807.03748, 2018.

[8] G. Gallego et al., "Event-based Vision: A Survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 1, pp. 154-180, 2020.

[9] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Adv. Neural Inf. Process. Syst.* (NeurIPS), vol. 32, 2019.

[10] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014.

[11] J. K. Eshraghian et al., "snntorch: A Python Package for Gradient-based Learning with Spiking Neural Networks," arXiv preprint arXiv:2304.08835, 2021.

[12] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "Events-to-Video: Bringing Modern Computer Vision to Event Cameras," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.* (CVPR), 2019.