Signal Classification with a Minimal Quantized Photonic Reservoir

Paul LEMAIRE, Mael ROBERTO

April 2025

1 Introduction

Reservoir Computing (RC) is a bio-inspired computational paradigm rooted in the temporal processing capabilities of recurrent neural networks. Its appeal lies in its simplicity: the reservoir's internal dynamics remain untrained, and only the readout layer is optimized. This makes RC particularly attractive for efficient temporal pattern recognition and classification tasks.

Recent developments in *physical reservoir computing* (PRC) have pushed this paradigm into the domain of real-world hardware implementations. Among these, **photonic reservoirs** stand out for their potential to deliver low-latency and energy-efficient computation, leveraging the intrinsic parallelism and speed of optical systems. Notably, the work of Rontani et al. [1] has demonstrated the power of large-scale photonic reservoirs with untrained dynamics in achieving high performance on benchmark tasks, while remaining compatible with physical constraints.

Our project builds on these insights but explores a complementary approach: rather than focusing on temporal sequences and rich internal recurrence, we simulate a minimal photonic-inspired architecture operating in a static regime. Our reservoir is driven by a single input injection per sample and is restricted to use quantized signals and a sinusoidal squared nonlinearity, $\sin^2(\cdot)$, a characteristic transfer function of photonic devices. This simplification is intended to mimic the behavior of a lightweight photonic circuit that minimizes memory and power consumption.

To compensate for the limited dynamics, we combine the reservoir with HOG-based feature preprocessing and train only a linear readout using ridge regression. Surprisingly, despite these strong constraints, our system achieves an accuracy of **98.86%** on MNIST, approaching early convolutional networks like LeNet-5. This highlights how even minimal, interpretable systems can yield competitive results under suitable optimization, particularly in settings where physical deployability and biological plausibility are prioritized.

Ultimately, our work complements prior contributions to photonic RC by exploring the performance potential of highly simplified analog-inspired systems, reinforcing the broader versatility of the reservoir computing framework.

2 Photonic Reservoir Model

Our goal is to simulate a simplified photonic reservoir system, motivated by real-world optical hardware such as the one illustrated in Figure 1. In such systems, an initial image (e.g., a handwritten digit) is projected by a LED onto a Spatial Light Modulator (SLM). The SLM transforms the image into an optical pattern that is then captured by a camera. This camera output can be interpreted as the state of a photonic reservoir, which is finally used by a classical computer to perform classification.

In our model, this physical behavior is captured through a compact mathematical formulation:

$$\mathbf{x}(t+1) = \left\lfloor I_0 \sin^2\left(\lfloor \mathbf{W}_{\text{res}}\mathbf{x}(t) + \mathbf{W}_{\text{in}}\mathbf{u}(t) \rfloor_8\right)\right\rfloor_{10}$$

$$\mathbf{y}(t) = \mathbf{W}_{\text{out}}\mathbf{x}(t)$$

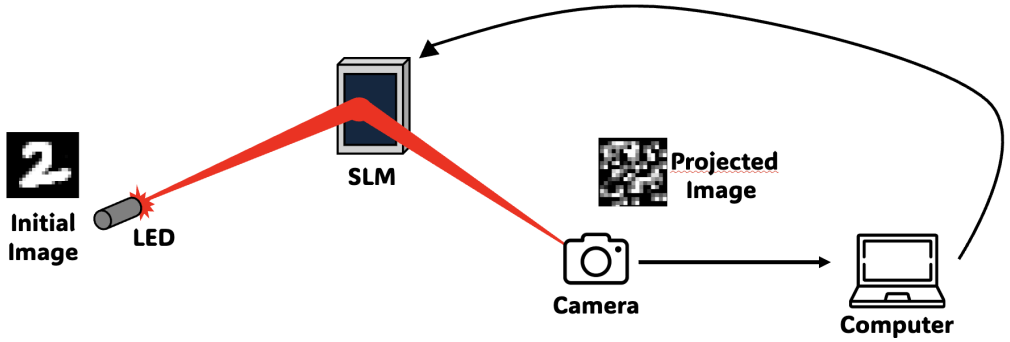

Figure 1: Simplified photonic reservoir computing setup. An image is optically encoded and propagated through a spatially complex medium (SLM), then recorded by a camera for digital processing.

Here, the input $\mathbf{u}(t)$ is a flattened image vector (or its HOG features), $\mathbf{x}(t)$ is the internal state of the reservoir, and $\mathbf{y}(t)$ is the classification output. The inner quantization ($\lfloor \cdot \rfloor_8$) and outer quantization ($\lfloor \cdot \rfloor_{10}$) simulate bit-limited photonic hardware. The $\sin^2$ nonlinearity mimics the transfer function of photonic elements such as lasers or modulators.

Figure 2 shows the internal dynamics of five randomly selected neurons over 30 time steps under random input. This highlights the diversity and nonlinearity of the reservoir states, even under a single-step dynamic.

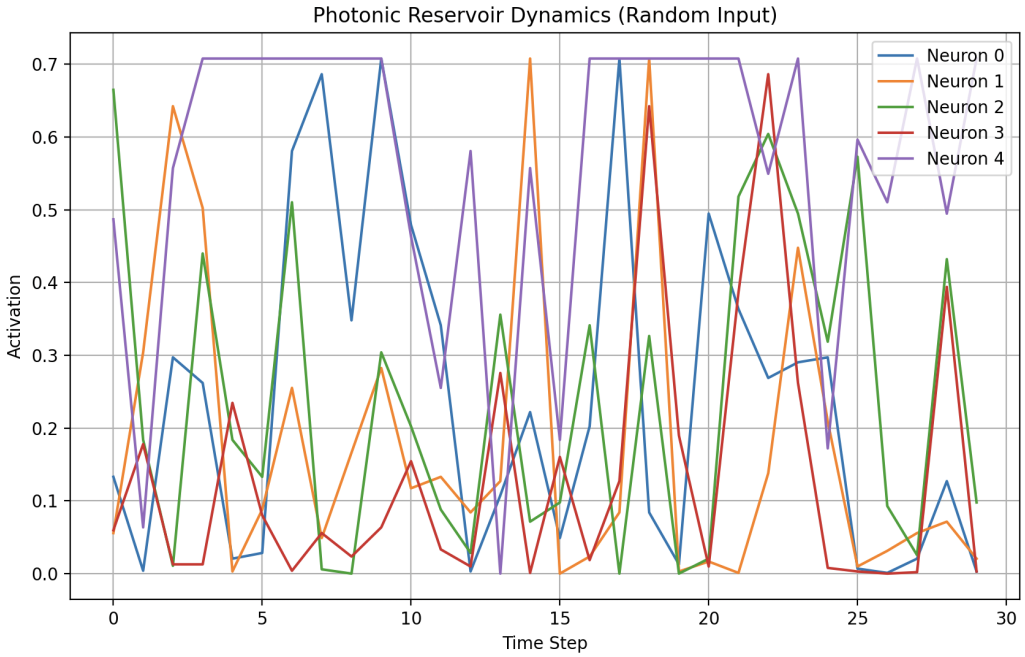

Figure 2: Temporal evolution of the activation of 5 randomly selected neurons under random input. Despite minimal recurrence, the states remain diverse and nontrivial.

This structure allows us to simulate a biologically plausible and hardware-ready reservoir with minimal computational cost. Crucially, our model does not use internal training, only the output weights $\mathbf{W}_{\text{out}}$ are optimized via ridge regression. This makes our architecture simple, efficient, and interpretable.

3 Stability Analysis and Hyperparameter Impact

3.1 Objective and Methodology

In this section, we investigate how different hyperparameters affect the stability of the photonic reservoir. While our final architecture performs a single feedforward step per input image, the reservoir still exhibits rich internal dynamics that can vary significantly depending on its configuration. Understanding how sensitive the system is to small input perturbations provides key insights into its operating regime: whether it behaves in a stable, chaotic, or edge-of-chaos manner.

Stability is a fundamental property in reservoir computing. Systems that are too stable may fail to separate inputs effectively, while highly chaotic ones may be too sensitive to noise and lack generalization. The sweet spot lies at the *edge of chaos*, where information can be both stored and transformed. This motivates a careful analysis of how our physical hyperparameters influence these dynamics.

To do so, we introduce a divergence-based metric inspired by Lyapunov stability analysis. We measure how small perturbations in the input propagate through the reservoir by computing the log distance between the resulting trajectories over time. This *divergence curve* serves as a proxy for assessing the stability and richness of the reservoir's temporal dynamics under various settings.

3.2 Hyperparameters under Study

We focus our analysis on three key hyperparameters that shape the behavior of the photonic reservoir:

- **Input scaling** (γ): this parameter controls the amplitude of the input signal projected into the reservoir through the input weight matrix W_{in}. It determines how strongly the external input $\mathbf{u}(t)$ influences the internal state dynamics. In physical terms, it corresponds to the intensity or contrast of the modulation applied to the input signal.
- **Nonlinearity gain** (I_0): this scalar multiplies the squared sine nonlinearity $\sin^2(\cdot)$ at the output of the activation function. Physically, this models the gain of the optical system, how strongly the nonlinear transformation amplifies the modulated input. It affects the amplitude and saturation level of neuron activations after quantization.
- **Reservoir sparsity** (μ): this controls the density of the recurrent connections in the internal weight matrix W_{res}. A low value of μ corresponds to a sparse, weakly connected reservoir, whereas higher values introduce more interactions and recurrent dynamics. This impacts the level of internal recurrence, which is known to influence memory capacity and dynamical richness.

We explore the following value ranges for each parameter:

$$\gamma \in [0.1, 1.0]$$
$$I_0 \in [0.5, 1.5]$$
$$\mu \in [0.1, 0.9]$$

These intervals were chosen to reflect plausible constraints for physical photonic implementations (e.g., limited modulation depth or gain) while ensuring numerical stability. Empirical testing also indicated that extreme values outside these ranges often led to saturation or instability in the reservoir, making them less informative for meaningful analysis.

3.3 Divergence Curve as a Stability Metric

To evaluate the dynamic stability of the reservoir, we introduce a divergence-based metric. The core idea is to measure how a small perturbation in the reservoir's initial state evolves over time. A stable

reservoir should resist amplification of such perturbations, while an unstable or chaotic reservoir will exhibit rapid divergence in state trajectories.

Method: We begin by applying a fixed input $\mathbf{u}(t)$ to two identical copies of the reservoir. The only difference between the two systems is a small perturbation applied to the initial state of one copy. At each time step t_k, we compute the Euclidean distance between the two reservoir states:

$$D(t_k) = \|\mathbf{x}(t_k) - \tilde{\mathbf{x}}(t_k)\|_2$$

where $\mathbf{x}(t_k)$ and $\tilde{\mathbf{x}}(t_k)$ denote the unperturbed and perturbed trajectories, respectively.

To obtain a scale-invariant measure of divergence, we normalize the distance with respect to the initial distance and take the logarithm:

$$\text{log-divergence}(t_k) = \log\left(\frac{D(t_k)}{D(t_0)}\right)$$

This curve reveals how differences evolve across time and provides a direct proxy for stability.

Interpretation:

- **Flat or decreasing curves** indicate convergence and thus high stability.
- **Increasing curves** suggest divergence and potentially chaotic behavior.
- **Plateaus** or oscillations near zero may reflect dynamics on the edge of chaos, often associated with better memory properties.

We compute and compare these divergence curves across a grid of hyperparameter values to assess their influence on the system's dynamical regime.

3.4 Parameter-wise Impact on Stability

To better understand the effect of individual hyperparameters on the reservoir's stability, we begin by analyzing the full divergence curves for several values of each parameter. These curves are shown in Figure 3. While the trajectories are noisy and can vary from step to step, they generally exhibit a saturation behavior where the divergence quickly converges towards a plateau. This observation supports our decision to summarize each curve using its maximum value as a proxy for instability.

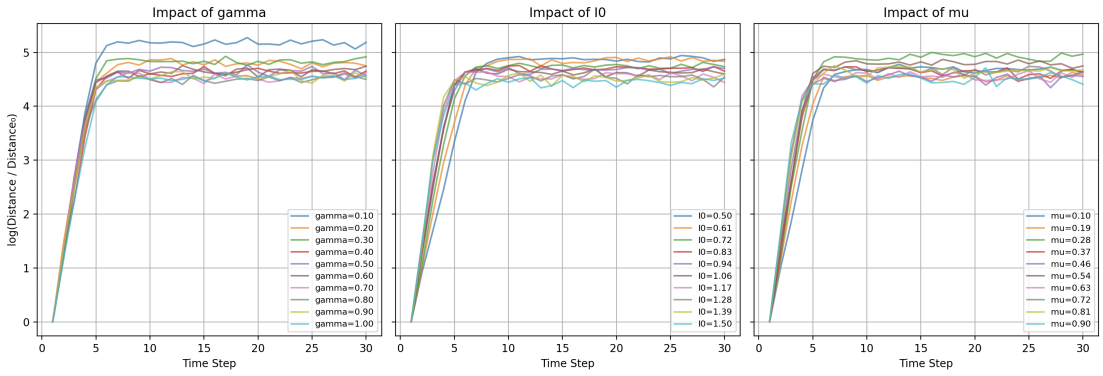

Figure 3: Divergence curves across 30 time steps for different values of γ, I_0, and μ. While noisy, most curves exhibit convergence toward a stable value.

Therefore we turn to Figure 4, which plots the maximum divergence for each value of the three hyperparameters under study. This provides a cleaner, high-level view of each parameter's influence on stability.

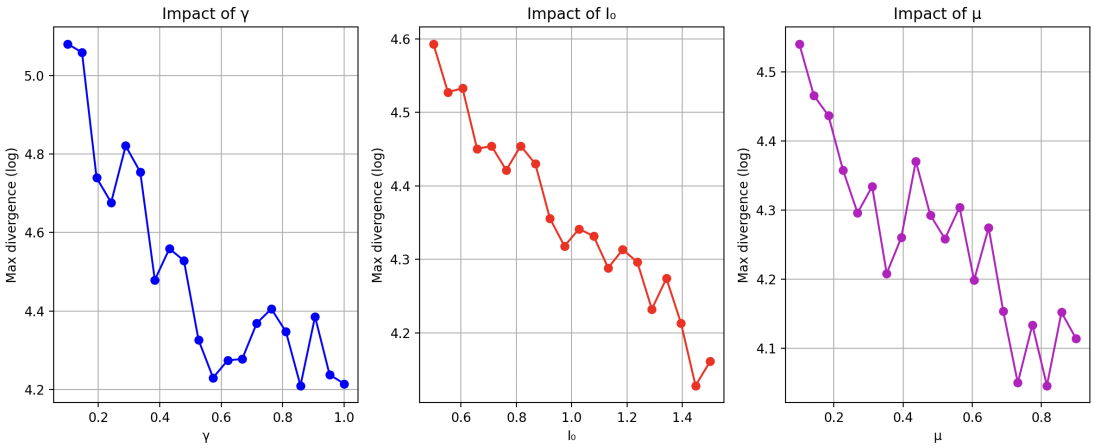

Figure 4: Maximum divergence value (in log scale) for each hyperparameter setting. Lower values correspond to more stable reservoir dynamics.

Input scaling γ: Increasing γ has a clear stabilizing effect. At low values, the reservoir states are more susceptible to perturbations in the input, which leads to higher divergence. As γ increases, the input signal dominates the dynamics and quickly drives the system into a regular activation regime, hence reducing instability.

Nonlinearity gain I_0: Similarly, higher I_0 values generally reduce the reservoir's sensitivity. This may be due to the amplification of the saturated part of the $\sin^2$ nonlinearity, which dampens variations and pushes neuron outputs into flatter regions of the transfer curve.

Sparsity level μ: A denser internal reservoir (higher μ) appears more stable, additional recurrent connections may distribute the influence of any perturbation across more neurons, preventing localized amplification and promoting more robust dynamics.

These results reinforce the idea that parameter tuning can be used not only to improve task performance, but also to shape the internal dynamic landscape of the reservoir in meaningful and predictable ways.

3.5 Hyperparameter Interactions and Bayesian Optimization

The parameter-wise analysis in the previous section highlighted general trends in how γ, I_0, and μ influence stability. However, it also hinted that the relationship may not be strictly separable: Parameter interactions could play a significant role. For instance, while a high γ generally improves stability, this effect might depend on the value of I_0. To investigate such interactions, we visualize the joint effect of two parameters while fixing the third.

Figure 5 shows a heatmap of the maximum divergence as a function of γ and I_0, with μ fixed to an intermediate value. The landscape reveals clear interaction effects: certain regions of the space yield stable behavior only when both parameters are jointly well tuned. For example, low values of γ and I_0 simultaneously result in high instability, whereas either parameter on its own might be benign in a different context. This confirms the need to consider multivariate tuning.

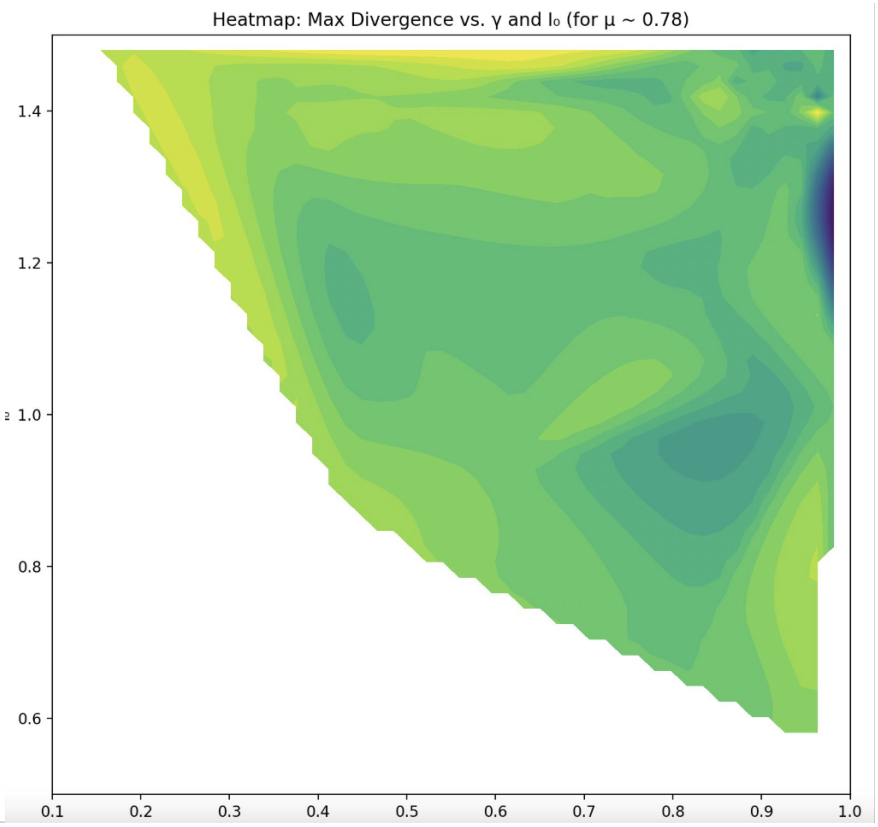

Figure 5: Heatmap of maximum divergence as a function of γ and I_0, with μ fixed at 0.78. Stability is not separable: some combinations cause instability even if one parameter alone would not.

Given this coupled behavior, we move beyond grid search and employ Bayesian optimization to efficiently search the 3D parameter space for the most stable configuration. The goal is to minimize the maximum divergence, using a surrogate model to propose promising hyperparameter combinations and iteratively refine the search.

The optimization was conducted over the following ranges: $\gamma \in [0.1, 1.0]$, $I_0 \in [0.5, 1.5]$, and $\mu \in [0.1, 0.9]$. The optimization process encountered several local minima, confirming the non-convex nature of the stability landscape.

The best parameters found were:

- $\gamma = 0.967$
- $I_0 = 1.416$
- $\mu = 0.778$

With a corresponding minimized maximum divergence of approximately **3.97**, this configuration significantly improves stability compared to the initial settings.

These results offer several insights. First, the optimal γ and I_0 are both near the upper end of their ranges, which aligns with the intuition that stronger input scaling and higher nonlinear amplification stabilize the reservoir. Interestingly, the optimal μ is also relatively high, supporting the idea that a denser internal network, while less biologically plausible, may distribute perturbations more effectively, promoting stable dynamics.

These findings provide a valuable foundation for later performance-oriented tuning. While stability was our primary focus here, the relationship between dynamic stability and task-specific performance remains an open question, which we revisit in a subsequent section.

4 Convergence Time Analysis

To better understand the temporal behavior of our reservoir system, we introduce a new variant of the model: the **leaky reservoir**. This formulation adds a temporal smoothing effect by blending the previous internal state with the newly computed state, mimicking a form of memory integration observed in physical systems.

The updated reservoir dynamics are given by:

$$\mathbf{x}(t_{k+1}) = (1-\alpha)\mathbf{x}(t_k) + \alpha \left[I_0 \sin^2 \left(\mathbf{W}_{res}\mathbf{x}(t_k) + \mathbf{W}_{in}\mathbf{u}(t_k) \right) \right]_8^{10} \tag{1}$$

$$y(t_k) = \mathbf{W}_{out}\mathbf{x}(t_k) \tag{2}$$

The parameter $\alpha \in (0, 1]$ controls the *leak rate*, i.e., the trade-off between retaining memory of past states and incorporating new input:

- When $\alpha = 1$, the reservoir fully updates its state at each step, as in our original non-leaky model.
- When $\alpha \ll 1$, the update is dominated by the previous state, leading to smoother and slower dynamics.

This leaky integration mechanism allows us to investigate the **convergence time** of the reservoir under a constant input signal. Specifically, we measure how quickly the activations settle toward a steady state as a function of α.

4.1 Stability Analysis under Constant Input

To study the qualitative effect of the leak rate α on the system's dynamics, we simulate the reservoir's response to a constant input vector $\mathbf{u}(t_k) = \mathbf{1}$ for all t_k.

Figure 6 shows the temporal evolution of the activation of 10 randomly selected neurons for different values of α. The general trend is as follows:

- For small α (e.g., $\alpha = 0.10$), the system exhibits smooth, stable behavior with a slow transition toward equilibrium.
- As α increases (e.g., $\alpha = 0.46$ or $\alpha = 0.64$), the convergence becomes faster, but the trajectories start to show transient oscillations.
- For α close to 1.0, the reservoir becomes much more reactive and displays irregular, seemingly chaotic behavior. However, this visual complexity does not necessarily imply instability.

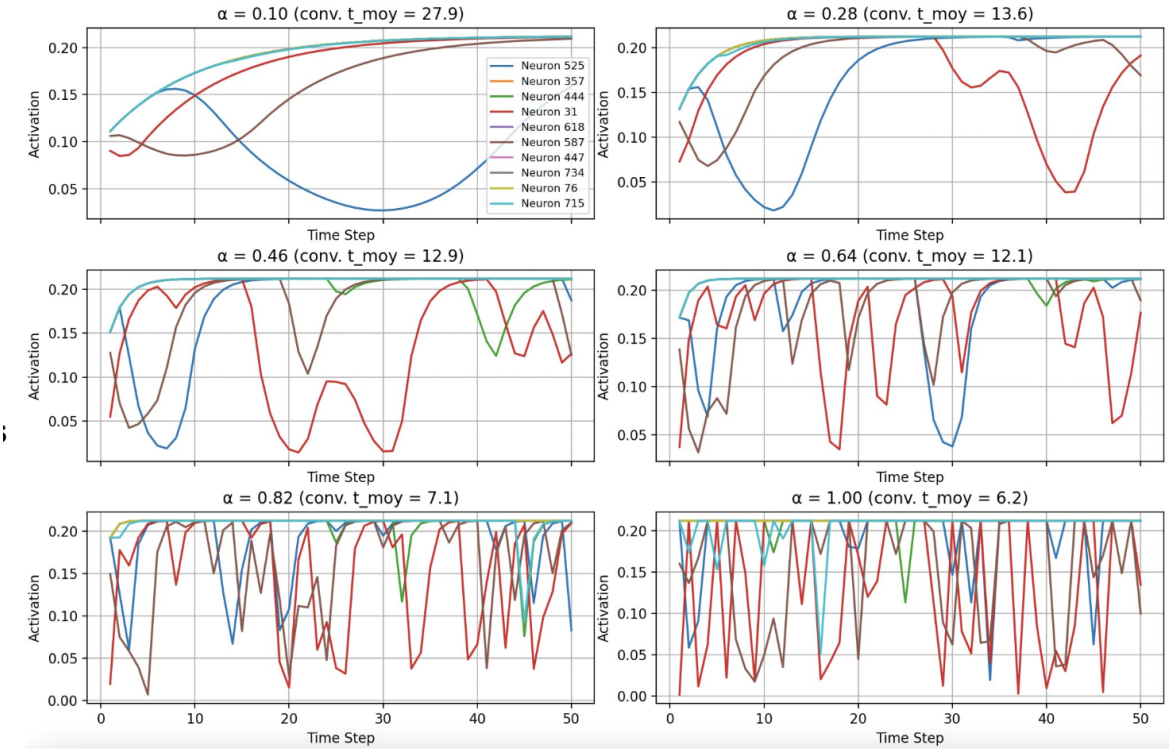

Figure 6: Reservoir activation over time for different values of α under constant input. For each α, we display the activity of 10 randomly selected neurons.

These plots illustrate the trade-off introduced by α: low values ensure smooth and stable convergence but lead to sluggish dynamics, while high values increase reactivity at the cost of stability and regularity.

4.2 Convergence Time Analysis

While visual inspection of neuron dynamics offers useful qualitative insights, it is also important to quantify the convergence speed of the reservoir. To do so, we define the **convergence time** as the number of steps required for all neuron activations to remain within a small tolerance of their final value.

We simulate the reservoir with a constant input for various values of α and compute the average convergence time across neurons. The results are shown in Figure 7.

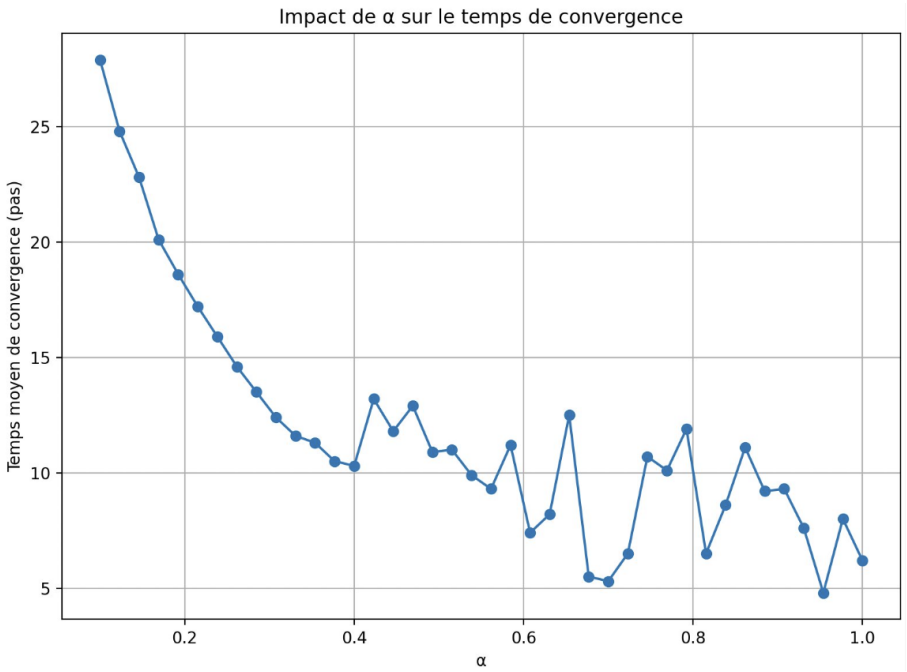

Figure 7: Impact of the leak rate α on the average convergence time.

We observe the following behavior:

- For low values of α (e.g., $\alpha \approx 0.1$), the system converges slowly, requiring around 28 time steps to stabilize.
- As α increases, the convergence time drops significantly, reaching a plateau around 6 to 10 time steps.
- However, for α close to 1, the system becomes visually chaotic and more irregular. This irregularity increases variance in the convergence time estimation.

These results highlight a trade-off between convergence speed and convergence-time stability. A moderate leak rate (e.g., $\alpha \in [0.3, 0.6]$) may provide a good balance between slow fading memory and fast response to inputs.

Note: chaotic behavior at high α does not necessarily imply mathematical instability, it merely reflects less regular neuron trajectories in response to constant input.

5 Classification on MNIST with Photonic Reservoirs

5.1 Problem Setup and Dataset

To evaluate the classification capabilities of our photonic reservoir computer, we use the well-known MNIST dataset. MNIST consists of 60,000 training and 10,000 testing grayscale images of handwritten digits, each of size 28×28 pixels. The goal is to assign each image to one of ten classes (0 through 9).

Although reservoir computing is typically suited for temporal tasks, we frame the static MNIST classification task as a one-shot inference problem. Each image is flattened and injected once into the reservoir. The system must then compute a prediction based solely on the reservoir state after this single update.

This setting allows us to test the representational power of our reservoir as a nonlinear feature map, while respecting physical constraints such as:

- **No recurrent training:** the reservoir weights are fixed and untrained.
- **Single-step dynamics:** only one update is performed per image.
- **Quantized operations:** all internal operations follow realistic bit-depth constraints (8-bit and 10-bit quantization).
- **Photonic transfer function:** the nonlinearity is $\sin^2(\cdot)$, as would be seen in an optical cavity.

This setup models an ultra-fast, energy-efficient photonic processor performing inference in a feedforward fashion, with minimal memory and control overhead. Our aim is to measure how far such a minimal, hardware-inspired system can go on a well-established benchmark like MNIST.

5.2 From Raw Pixels to Features

The input representation is critical in determining the reservoir's ability to separate input classes. Since our reservoir is constrained to a single non-recurrent step, the quality of the features injected plays an outsized role in shaping the output space. We compare two input strategies:

Raw Pixels

Each MNIST image, originally of size 28×28, is flattened into a vector of 784 values by reading the image in column-major order. The values are normalized to the range $[-1, 1]$. This approach provides the reservoir with the full spatial content of the image but without any abstraction: it lacks robustness to translations, rotations, and does not make explicit the local structures such as edges or textures.

HOG Features

To address these limitations, we also consider the use of *Histogram of Oriented Gradients* (HOG), a classical computer vision descriptor designed to encode local edge orientations.

The HOG algorithm operates as follows:

- The image is divided into small spatial regions called **cells**.
- For each cell, the gradient direction at each pixel is computed, typically using simple filters like the Sobel operator.
- These gradients are binned into a histogram over a fixed number of **orientation bins**, capturing the dominant edge directions.
- Histograms from multiple adjacent cells are grouped into **blocks** and normalized to account for illumination or contrast variations.

This process yields a feature vector that emphasizes the spatial distribution of edges and contours rather than raw pixel values. It provides translation-invariant and rotation-aware information that is well suited to tasks like digit classification, where such features are discriminative.

Compared to raw pixels, HOG features act as a learned preprocessing layer: they inject structural priors into the system, such as local coherence and shape information. This is especially useful in our setup, where the reservoir cannot learn such features internally due to its fixed, untrained architecture.

Later experiments show that the use of HOG leads to a substantial increase in classification performance, confirming that meaningful preprocessing can compensate for architectural limitations of the reservoir.

5.3 One-Step Reservoir Injection and Readout

As previously introduced in Section 2, the reservoir performs a single nonlinear transformation of the input using sinusoidal activation and quantized operations. In this section, we analyze how this single update, despite its simplicity, enables surprisingly effective classification.

We focus on a static mode of operation: the image (either raw pixels or HOG features) is injected once into the reservoir, and the resulting state $\mathbf{x}$ is used directly for classification. No recurrence or temporal dynamics are involved beyond this step, making this setup particularly relevant for low-latency photonic inference.

Readout Layer

After computing the state $\mathbf{x}$ for each input, classification is performed using a linear readout layer trained by ridge regression (Tikhonov regularization). The output $\hat{\mathbf{y}}$ is given by:

$$\hat{\mathbf{y}} = \mathbf{W}_{\text{out}}\mathbf{x} \tag{3}$$

Where $\mathbf{W}_{\text{out}}$ is obtained by solving:

$$\mathbf{W}_{\text{out}} = \arg\min_{\mathbf{W}} \|\mathbf{Y} - \mathbf{X}\mathbf{W}^{\top}\|^2 + \lambda\|\mathbf{W}\|^2 \tag{4}$$

Here:

- $\mathbf{X}$ is the matrix of reservoir states for all training inputs.
- $\mathbf{Y}$ is the one-hot encoded label matrix.
- λ is the regularization strength.

Only the readout weights are trained; the reservoir remains fixed and untrained throughout. This reinforces the physical interpretability and low-energy requirements of the system, as no backpropagation is needed.

In the next subsection, we study how the size of the reservoir affects the system's ability to generalize from training data to unseen inputs.

5.4 First Results and Observations

To establish a performance baseline, we first evaluate the reservoir's ability to classify MNIST digits using two types of input representations:

- **Raw Pixels:** Flattened 28×28 grayscale images, normalized to $[-1, 1]$.
- **HOG Features:** Histogram of Oriented Gradients (HOG) features computed with standard parameters.

We train a simple ridge regression readout layer using 60,000 training samples and evaluate on 10,000 test samples. The reservoir contains 1024 neurons and operates in static mode (one-shot injection).

Accuracy Results:

- **Raw Pixels:** 80.40%
- **HOG Features:** 93.29%

Classification Report (HOG):

Digit	Precision	Recall	F1-score	Support
0	0.94	0.98	0.96	980
1	0.98	0.98	0.98	1135
2	0.91	0.93	0.92	1032
3	0.91	0.92	0.91	1010
4	0.93	0.93	0.93	982
5	0.93	0.92	0.93	892
6	0.96	0.96	0.96	958
7	0.92	0.94	0.93	1028
8	0.92	0.90	0.91	974
9	0.92	0.90	0.91	1009
Accuracy		**0.93**		10,000

Key Observations:

- The reservoir shows clear non-linear separation capabilities, despite its minimal structure and untrained internal dynamics.
- The switch from raw pixels to HOG features boosts accuracy by nearly +13%, confirming the reservoir's ability to process structured inputs more effectively.
- The HOG-based model achieves strong performance across all digits, with F1-scores consistently above 0.91. Class `1` reaches near-perfect classification with a 0.98 F1-score.
- Digits `8` and `9` show slightly lower recall, indicating some confusion with similar-looking classes. This aligns with known challenges in handwritten digit classification, especially when using limited dynamic representations.
- These results validate that even a simple readout trained on well-processed inputs can yield satisfying results, highlighting the reservoir's effectiveness as a non-linear feature mapper.

Conclusion: These initial experiments demonstrate that while the internal dynamics of the reservoir remain untrained and relatively simple, its combination with rich feature descriptors like HOG can already yield high classification performance. This sets a strong foundation for future scaling, optimization, and exploration of temporal dynamics.

6 Hyperparameter Optimization for Classification

6.1 Optimization Strategy

While our reservoir architecture is fixed and untrained, several key hyperparameters control its behavior and ultimately influence classification performance. To efficiently navigate this multi-dimensional hyperparameter space, we adopt a **Bayesian optimization** approach.

Bayesian optimization is particularly well-suited for expensive-to-evaluate functions, such as validation accuracy over large datasets. It constructs a probabilistic model (typically a Gaussian Process) of the objective function and uses it to select promising candidates through an acquisition function. This allows for intelligent exploration of the hyperparameter space while minimizing the number of costly model evaluations.

We aim to maximize test accuracy on the MNIST dataset using HOG features, by tuning the following four hyperparameters:

- **Input scaling** $\gamma \in [0.05, 1.0]$ — controls how strongly the input $\mathbf{u}(t)$ influences the reservoir state through the input weights $\mathbf{W}_{\text{in}}$.
- **Nonlinearity gain** $I_0 \in [0.5, 1.5]$ — determines the amplitude of the $\sin^2$ nonlinearity.
- **Reservoir sparsity** $\mu \in [0.1, 0.9]$ — defines the proportion of non-zero connections in the recurrent weight matrix $\mathbf{W}_{\text{res}}$.
- **Ridge regularization** $\lambda \in [10^{-6}, 10^{1}]$ — regularization strength used for training the linear readout.

Together, these parameters balance input sensitivity, nonlinear expressiveness, recurrent structure, and readout robustness. Their interaction governs the capacity of the system to form meaningful internal representations, which motivates the need for joint optimization.

6.2 Optimization Results

The Bayesian optimization yielded a significant boost in classification accuracy, pushing the test performance from **93.29%** (default parameters) to **97.02%** using optimized hyperparameters. This 4-point gain confirms that our model's performance is highly sensitive to careful tuning, even under a minimal architecture with fixed dynamics.

Best Trial:

- **Accuracy:** 97.02%

Best Hyperparameters:

- $\gamma = 0.110$
- $I_0 = 0.992$
- $\mu = 0.360$
- $\lambda = 1.517 \times 10^{-7}$

These values differ markedly from the configuration found in Section 3, which focused on stability. In particular:

- The optimized γ is close to the **lower boundary** of the tested interval, near the threshold for chaotic dynamics ($\gamma \approx 0.1$).
- I_0 and μ are in intermediate ranges, not especially high as in the stability-optimal configuration, suggesting that too much suppression of dynamics is suboptimal for classification.

Implication: Stability $\neq$ Precision These observations reinforce a key takeaway: **stability and classification performance are not aligned objectives**. While stability favors strongly regular dynamics (high γ, high I_0), accuracy peaks near the edge of chaos, where the system can exploit richer internal transformations to enhance separability.

The γ Frontier: The fact that $\gamma = 0.110$ lies very close to the lower bound ($\gamma = 0.1$) raises a natural question: would performance continue to improve for $\gamma < 0.1$? Unfortunately, this domain lies outside our initial design space due to divergence concerns. However, the trend suggests a sharp boundary, indicating a possible sweet spot of even higher performance beyond the tested range. Exploring this region could be an interesting extension.

Exploration Robustness: Figure 8 shows the optimization history over 50 trials. Performance increased rapidly within the first 10 evaluations, plateauing around 97%. This curve indicates that **many configurations achieve near-optimal performance**, confirming that the accuracy boost is not the result of an isolated lucky trial. It also emphasizes that we diverged early from the hyperparameter region that optimized for stability, supporting the idea that chaotic regimes can be more expressive and powerful for classification.

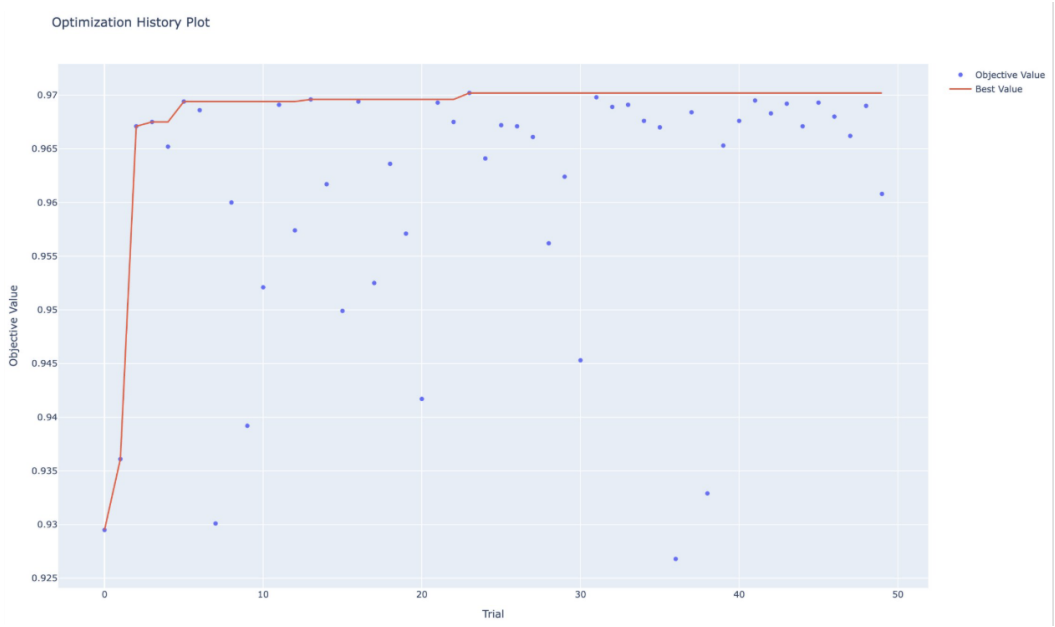

Figure 8: Bayesian optimization history. The red line traces the best accuracy observed up to each trial. The plateau confirms that many configurations achieve 96.9–97% performance.

6.3 Hyperparameter Importance

To better understand what drives the performance of our reservoir-based classifier, we analyze the relative importance of each hyperparameter using the feature attribution methods available in Optuna. This yields a quantitative ranking of how much each parameter contributed to the observed accuracy improvements.

Figure 9 shows the estimated importance of the four optimized hyperparameters: γ, I_0, μ, and λ.

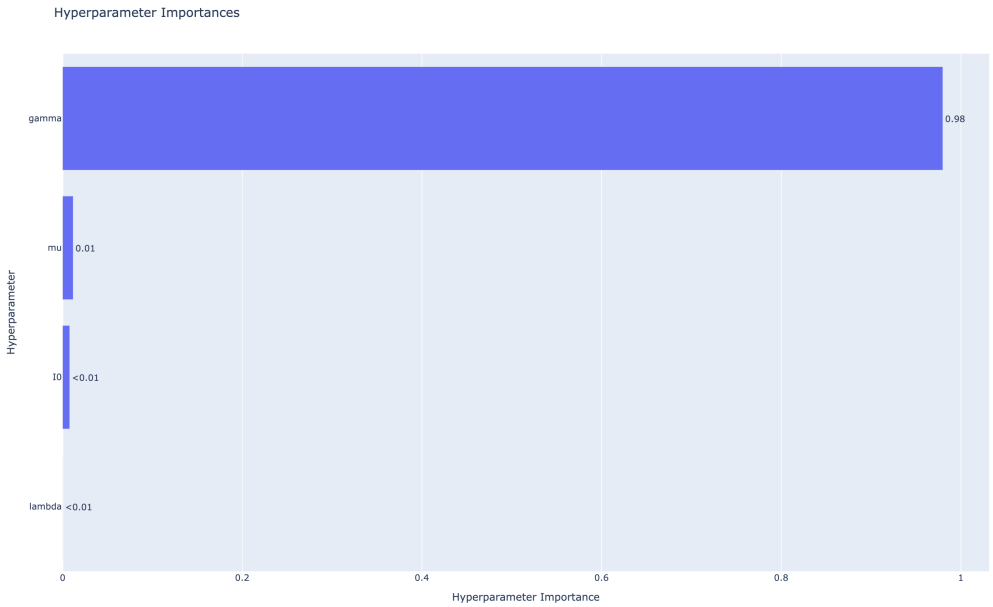

Figure 9: Relative importance of each hyperparameter for classification accuracy (as estimated by Optuna).

Key Takeaways:

- **Dominance of γ (input scaling):** With an importance score of 0.98, γ is by far the most influential hyperparameter. This aligns with prior observations: γ directly controls the signal amplitude at the input of the reservoir and hence determines how nonlinearly separable the reservoir representations are. Too small, and the input is drowned by internal noise; too large, and saturation reduces expressivity.
- **Minor influence of I_0 and μ:** Both parameters, despite having clear effects on the internal dynamics of the reservoir, appear to have a very limited direct impact on classification accuracy. This suggests that, at least in the one-step setting, the input projection plays a much more critical role than the reservoir's recurrent behavior.
- **Negligible effect of λ (readout regularization):** The regularization strength of the ridge regression readout had virtually no impact. This may be due to the high dimensionality of the reservoir features (1024 neurons) combined with good conditioning from HOG preprocessing, leading to little risk of overfitting in practice.

Conclusion: This analysis confirms that **most of the model's predictive power lies in the quality of the input projection**, a remarkable finding given the system's biological inspiration. The result highlights that even in biologically plausible, hardware-friendly architectures, careful tuning of the input-to-reservoir interface can drive near-state-of-the-art results.

In the next section, we study how the number of reservoir neurons impacts generalization and performance scaling.

6.4 Optimization Landscape and Trial History

The results of the Bayesian search not only yielded a highly accurate classifier, but also revealed deeper insights about the underlying optimization landscape.

One striking observation is that the best classification configuration lies in a region characterized by heightened instability, with $\gamma \approx 0.11$, close to the threshold where divergence curves begin to grow rapidly. While we do not formally establish a chaotic regime, this behavior echoes a recurring

theme in reservoir computing: optimal performance often emerges near a boundary between stability and instability, a regime sometimes referred to as the *edge of chaos*. Our results suggest that, rather than seeking maximum stability, tuning the reservoir toward this transition zone can enhance its expressivity, enabling richer internal representations that benefit classification.

Role of Hyperparameters: The importance analysis revealed that γ is by far the most influential parameter for classification performance. This suggests that the input scaling acts as a key control knob for expressive richness. The other parameters, I_0, μ, and λ, show marginal impact within the explored ranges, further highlighting the central role of γ in shaping the reservoir's nonlinear mapping capabilities.

Wider Implications: The fact that high accuracy can be achieved without strong dependence on the recurrent dynamics or the readout regularization (μ and λ) points to a reservoir architecture that is not only effective, but also robust and lightweight. This has direct implications for future hardware implementations: minimal tuning may be sufficient once the expressivity bottleneck (i.e., γ) is properly managed.

Conclusion: This section illustrates that in photonic-inspired reservoirs, the key to performance lies not in maximizing stability but in finding the right **balance between instability and representation power**. It also emphasizes that some parameters, particularly γ, play a dominant role, and should be the primary target of tuning efforts. Overall, our optimization process not only improved accuracy but also clarified the structure of the hyperparameter space and its impact on computation.

7 Impact of HOG Features and Neuron Count

7.1 Why Optimize Input Features?

As observed in previous sections, the quality of the features injected into the reservoir plays a critical role in the system's classification performance. Since our reservoir is untrained and operates in a one-shot regime, it relies entirely on the input projection to generate meaningful internal representations. This makes the preprocessing stage, specifically the generation of HOG features, essential.

To maximize the expressiveness of the input space, we perform a dedicated hyperparameter optimization over the HOG extraction parameters, in conjunction with selected reservoir parameters such as the input scaling γ and photonic gain I_0. This cross-optimization aims to align the structure of the features with the nonlinear mapping capabilities of the reservoir.

Optimization Setup:

- HOG hyperparameters:
  - `pixels_per_cell` $\in \{3, 4, 5, 6, 7\}$
  - `cells_per_block` $\in \{2, 3, 4, 5\}$
  - `orientations` $\in \{4, 5, 6, 7, 8, 9\}$
- Reservoir parameters (fixed sparsity $\mu = 0.7$):
  - $\gamma \in [0.1, 1.0]$
  - $I_0 \in [0.5, 1.5]$
- Number of neurons: $N = 512$ (reduced size for faster computation)
- Optimization method: Bayesian optimization with 300 trials

This reduced reservoir size allows us to explore a wide configuration space efficiently, without incurring high computational costs.

7.2 Results of HOG Optimization (N = 512)

To assess the impact of feature preprocessing on classification performance, we jointly optimized the parameters of the HOG descriptor and the reservoir hyperparameters (γ, I_0) using $N = 512$ neurons.

Performance improvement:

- **Before optimization:** 95.82% accuracy
- **After optimization:** 96.80% accuracy

Best parameters found:

- `pixels_per_cell` = 5
- `cells_per_block` = 5
- `orientations` = 6
- $\gamma = 0.212$
- $I_0 = 0.564$

Interpretation:

- Smaller cells (`pixels_per_cell` = 5) focus on fine-grained spatial structure, allowing the HOG features to better capture local edge variations that are critical for distinguishing handwritten digits.
- A moderate number of orientation bins (6) provides a good trade-off between directional sensitivity and feature compactness.
- The relatively low values of γ and I_0 suggest that once edges are enhanced via HOG, subtle input modulations are sufficient to drive the reservoir effectively. This indicates a synergy between edge-focused inputs and milder nonlinear transformations.

These results confirm the value of optimizing input preprocessing jointly with reservoir dynamics, especially in low-resource scenarios.

7.3 Hyperparameter Importance for HOG Optimization

To better understand which variables most strongly influenced classification accuracy during joint HOG-reservoir tuning (with $N = 512$), we analyzed hyperparameter importance using the built-in feature attribution tools provided by the Bayesian optimization framework (Optuna).

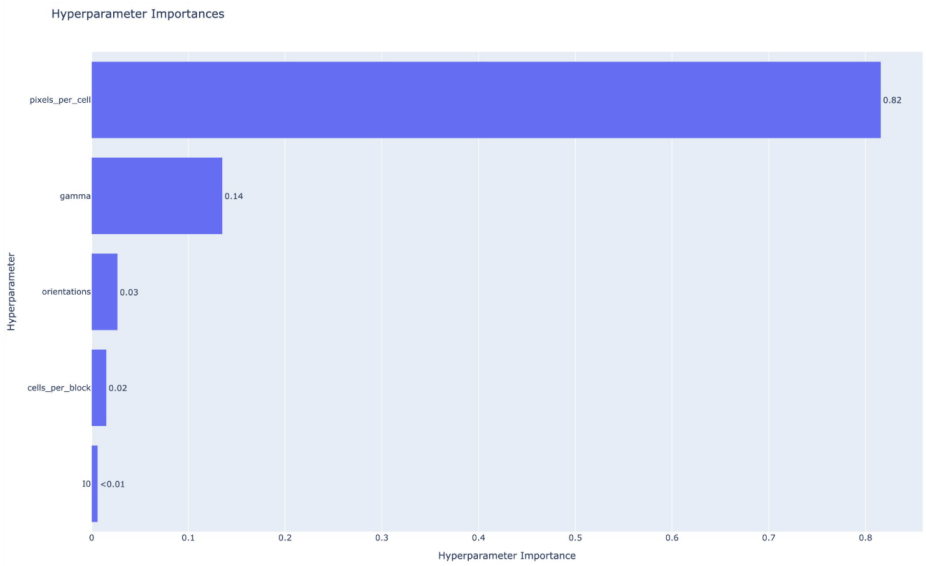

Figure 10: Hyperparameter importance during HOG + reservoir optimization (Optuna).

Key findings:

- The dominant contributor to accuracy is the `pixels_per_cell` parameter, with a relative importance of 0.82. This confirms that spatial granularity in the HOG descriptor plays a critical role in capturing discriminative edge features for MNIST digits.
- The input scaling parameter γ is the second most important variable (0.14), suggesting that even after preprocessing, proper modulation of the input strength into the reservoir remains essential.
- Other parameters such as `orientations`, `cells_per_block`, and I_0 have comparatively minor influence on final performance.

Interpretation: These results support the idea that **preprocessing quality can dominate downstream reservoir tuning**. In low-recurrence regimes or single-step reservoirs, the input representation defines the structure of the feature space. Thus, optimizing edge extraction and spatial encoding has a larger impact than fine-tuning internal dynamics.

This also suggests that in constrained scenarios, computational effort may be better invested in carefully crafting or learning input embeddings rather than heavily tuning the reservoir itself.

7.4 Accuracy Saturation with Increasing Reservoir Size

To understand how reservoir size (N) affects classification performance, we trained models with optimized HOG features and increased the number of neurons progressively. The best observed accuracy was:

- $N = 1024$: 98.04%
- $N = 2048$: 98.48%
- $N = 4096$: 98.73%
- $N = 8192$: 98.86%

This progression clearly demonstrates that increasing the number of neurons enhances performance. However, the marginal gains diminish rapidly, suggesting a law of diminishing returns.

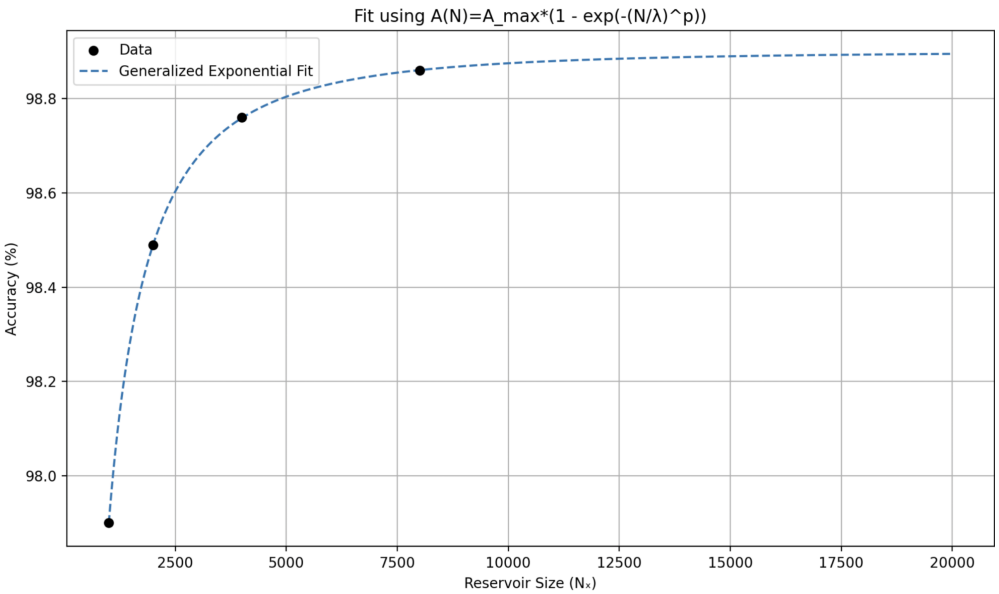

Figure 11: Accuracy as a function of reservoir size N, fitted using a generalized exponential model.

To extrapolate performance beyond our tested range, we fitted the observed data using a generalized exponential model of the form:

$$A(N) = A_{\max} \cdot \left(1 - \exp\left(-\left(\frac{N}{\lambda}\right)^{p}\right)\right)$$

The best-fit parameters were:

$$A_{\max} = 98.90\%$$
$$\lambda = 2.6$$
$$p = 0.256$$

This model captures the empirical trend well and suggests that even with significantly larger reservoirs ($N > 10{,}000$), we are unlikely to exceed 98.90% accuracy. This aligns with the intuition that beyond a certain point, adding neurons does not provide substantial new information, especially under a fixed feature representation (e.g., HOG).

Computation-Time Tradeoff

A key limitation in pushing N further is computational cost. Training and inference times grow at least linearly with reservoir size, and memory usage can become prohibitive for large matrices. During our experiments, training with $N = 8000$ already required around tens of minutes per optimization run on the full MNIST training set of 60,000 images. This exponential increase in compute time quickly made further scaling impractical within our resource constraints.

This implies that while scaling the reservoir is a valid way to improve performance, it quickly becomes inefficient compared to other strategies like:

- Improving input features (as in Section 7),
- Exploring hierarchical or multi-layer reservoirs,
- Using task-specific learned input projections.

In conclusion, the combination of empirical results and theoretical extrapolation suggests that our reservoir size-performance curve has reached its practical limit, with further gains requiring architectural innovations rather than brute-force scaling.

7.5 Conclusion

This section has shown that both input feature design and reservoir scaling play a critical role in achieving high classification performance. Optimizing the HOG feature extraction parameters, especially in synergy with input scaling, led to a substantial accuracy improvement from **95.82%** to **96.8%** for $N = 512$. Hyperparameter importance analysis confirmed that the quality of injected features, particularly the size of the HOG cells, was the most influential factor.

Increasing the number of reservoir neurons from 1024 to 8000 enabled us to reach a final accuracy of **98.86%**, matching early convolutional networks. However, the returns diminished rapidly, and the computational cost became significant: training with $N = 8000$ and 60,000 training images required over **20 minutes per optimization run**. Our fitted exponential model predicts an upper bound of **98.9%** accuracy, suggesting limited gains from further scaling.

These findings highlight the power of simple, quantized reservoir architectures when paired with strong input preprocessing and efficient hyperparameter tuning. They also emphasize that, beyond a certain point, improving performance is less about increasing model size and more about refining how data is represented and injected.

8 Comparison with Other Approaches

8.1 Compared to Classical Models

Classical neural architectures, such as Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs), have long dominated image classification benchmarks. For instance, LeNet-5, a CNN architecture introduced in the 1990s, already achieved around **99.0%** accuracy on MNIST. Modern CNNs can push performance even higher, with near-perfect classification ($> 99.5\%$) using deep architectures and extensive data augmentation.

However, such models come at a cost. CNNs require millions of trainable parameters, backpropagation through multiple layers, and often rely on powerful GPUs for training and inference. Their deployment in resource-constrained settings (e.g., edge devices or embedded systems) remains challenging due to energy consumption, latency, and memory requirements.

In contrast, our quantized photonic-inspired reservoir model offers a radically different approach:

- **No internal training:** Only the output layer is trained, dramatically reducing computational cost.
- **Single feedforward step:** Each image is processed in one pass through a fixed reservoir, avoiding the need for recurrent updates or long inference pipelines.
- **Quantized operations:** All internal computations respect 8-bit and 10-bit limits, simulating low-power hardware constraints.

Despite these limitations, our best configuration reaches an accuracy of **98.86%**, competitive with early CNNs and surpassing many shallow MLP baselines. While we do not challenge the state-of-the-art in raw accuracy, our system demonstrates how minimal, interpretable architectures can perform surprisingly well with proper optimization and feature design.

This tradeoff between **accuracy and efficiency** is at the heart of neuromorphic computing: achieving good enough performance at a fraction of the energy and complexity. For use cases that prioritize speed, simplicity, and deployability over raw precision, such reservoir-based models offer a compelling alternative.

8.2 Compared to State-of-the-Art Reservoir Computing

While our model is rooted in the reservoir computing (RC) paradigm, it adopts a highly constrained design compared to most state-of-the-art (SOTA) RC approaches.

A comprehensive review by Tanaka et al. [2] outlines the landscape of physical reservoir computing, including a broad class of systems that leverage complex temporal dynamics, recurrent feedback loops, and continuous-time evolution to solve a variety of tasks—from speech recognition to chaotic signal prediction. These systems often assume idealized analog behavior, unconstrained signal precision, and multi-step processing pipelines.

In contrast, our model takes a radically minimalist approach: it operates in **single-shot** mode, processes only one static input per sample, and produces an output after a single reservoir update. Additionally, it strictly enforces **quantization constraints**, simulating digital hardware limitations:

- 8-bit quantization at the internal activation level.
- 10-bit quantization on the reservoir output.

These restrictions make our model particularly relevant to low-power photonic or neuromorphic hardware, where bit-depth and latency are critical constraints. Most reviewed systems in [2], while powerful, do not enforce such strict limitations.

Despite its strong architectural constraints, our reservoir—combined with HOG preprocessing and a linear readout—achieves **98.86%** accuracy on MNIST. While Tanaka et al. [2] primarily review systems applied to temporal and analog signal processing tasks, our results show that even in a highly constrained, one-shot regime with strict quantization, competitive classification performance can be achieved. This stands in contrast to many surveyed systems, which rely on richer temporal dynamics, unbounded precision, or recurrent feedback to reach high performance.

Our contribution lies in demonstrating that such a minimalist setup, when paired with strong input encoding and principled hyperparameter optimization, can achieve competitive performance within the RC landscape. This reinforces the idea that simplicity, interpretability, and physical realism need not come at the cost of accuracy—particularly for static classification tasks like MNIST.

9 Training Set Size Impact

9.1 Influence of M on Generalization

To understand how the size of the training set affects the generalization performance of our photonic reservoir, we evaluate classification accuracy for increasing values of M (training set size), while keeping the test set fixed. We perform this analysis for two reservoir sizes: $N = 1000$ and $N = 8000$.

Results for $N = 1000$: As shown in Figure 12, the training accuracy remains close to 100% for all values of M, indicating that the model can easily fit the training data. More interestingly, the test accuracy improves steadily with larger training sets, from approximately 90% with 100 samples to over 98% with 60,000 samples. This suggests that the system does not suffer from overfitting in this regime and clearly benefits from increased training diversity.

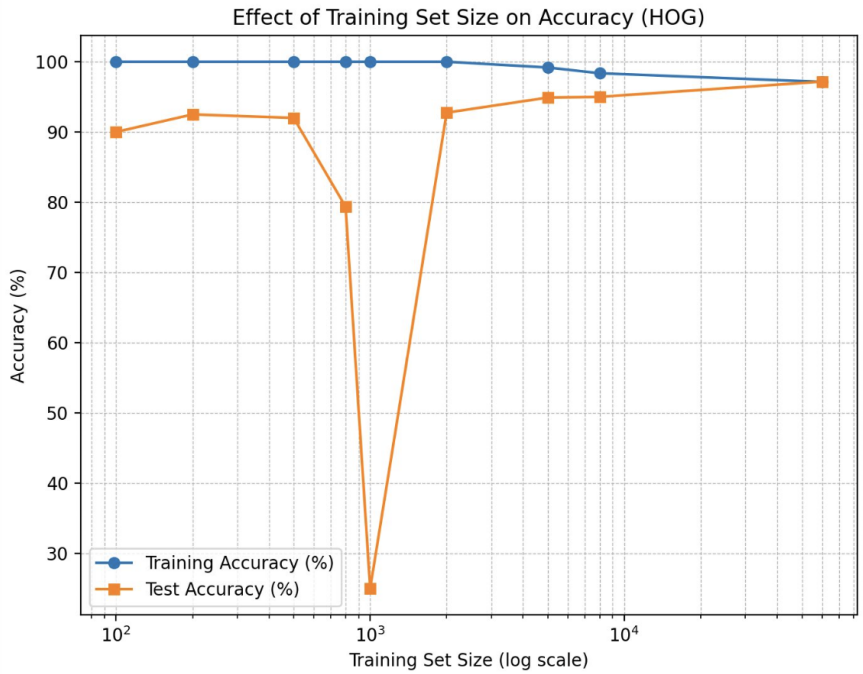

Figure 12: Training and test accuracy vs. training set size for $N = 1000$. Accuracy increases with more training samples.

Results for $N = 8000$: We repeat the experiment with a much larger reservoir of $N = 8000$ neurons. The results, presented in Figure 13, reveal a similar trend: more data leads to better generalization. However, the improvements are smaller and accompanied by increased variance in test accuracy. The effect of increasing M is less pronounced but still noticeable.

The previously reported accuracy of 98.16% for $M = 60{,}000$ is recovered, confirming that adding more training data helps even with large reservoirs. Meanwhile, training accuracy remains perfect, indicating high capacity and no underfitting.

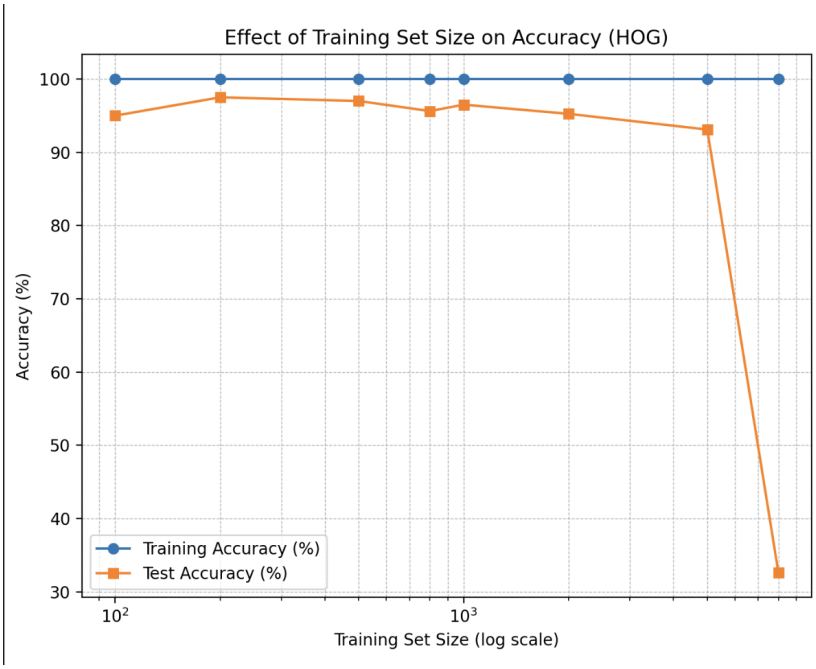

Figure 13: Training and test accuracy vs. training set size for $N = 8000$. Gains in test accuracy are smaller but persist.

These observations suggest that larger training sets systematically improve generalization, with diminishing returns for higher N. The phenomenon visible around $M = 1000$ will be explored in the next subsection.

9.2 Condition Number and $\mathbf{W_{out}}$ Instability

While the previous subsection highlighted an unexpected drop in generalization performance when the training set size M approaches the reservoir size N, a deeper investigation was necessary to determine whether this was a computational artifact or a structural phenomenon.

A Non-Artifactual Dip Around $M = N$ To begin, we increased the granularity of our sampling around $M = 1000$ for a fixed reservoir size of $N = 1000$. The results, shown in Figure 14, reveal a smooth degradation and recovery in test accuracy. This supports the hypothesis that the instability is not a measurement artifact, but rather a consistent phenomenon tied to the relative sizes of M and N.

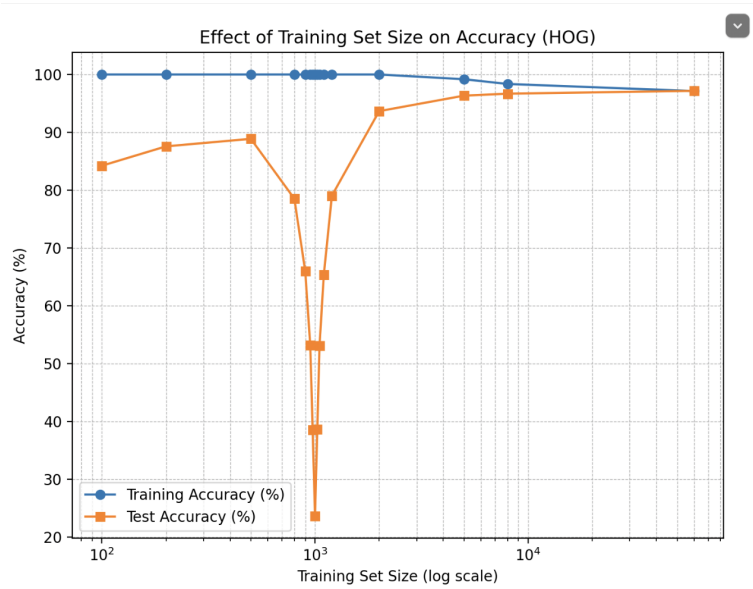

Figure 14: Fine-grained accuracy dip around $M = N = 1000$, indicating a smooth degradation in performance rather than a sharp artifact.

Generalization Collapse Across Reservoir Sizes To verify the generality of this issue, we ran experiments across a range of reservoir sizes N and tracked test accuracy as a function of M. As seen in Figure 15, each curve exhibits a sharp collapse in accuracy when $M \approx N$, regardless of the value of N.

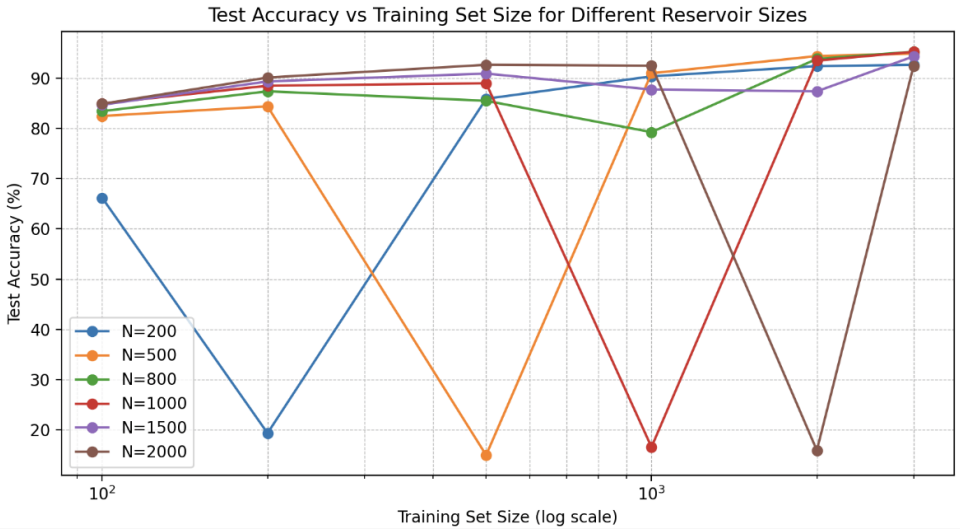

Figure 15: Test accuracy vs. training set size M for various reservoir sizes N. A performance collapse is observed consistently when $M \approx N$.

Numerical Instability in $\mathbf{W}_{\text{out}}$ To further investigate this effect, we computed the ℓ_2 norm of $\mathbf{W}_{\text{out}}$ for each configuration. As shown in Figure 16, this norm spikes sharply when $M \approx N$, consistent with the idea that $\mathbf{X}^\top \mathbf{X}$ becomes ill-conditioned or nearly singular in this regime. The resulting weight explosion explains the sharp degradation in generalization.

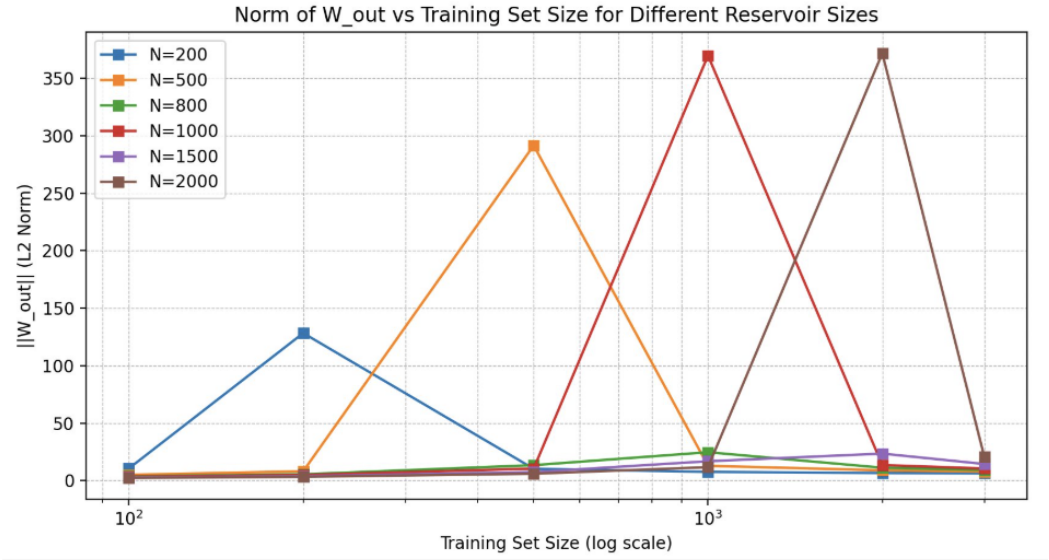

Figure 16: ℓ_2 norm of $\mathbf{W_{out}}$ vs. training set size M for various N. A distinct peak is visible at $M = N$, confirming numerical instability.

Conclusion These results highlight a fundamental limitation in reservoir computing systems: when the number of training examples is close to the reservoir dimension, the least-squares computation of $\mathbf{W_{out}}$ becomes unstable. This results in weight explosions and poor generalization. Practitioners should either ensure $M \gg N$ or employ robust regularization techniques when M and N are close.

Conclusion of Section 9

In summary, increasing the training set size M leads to better generalization performance in reservoir computing models, with no observable overfitting even up to $M = 60{,}000$. However, a critical instability emerges when M approaches the reservoir size N. This leads to a near-singular $\mathbf{X}^\top\mathbf{X}$ matrix during the computation of $\mathbf{W_{out}}$, causing a blow-up in weight magnitude and a dramatic drop in test accuracy. This effect is systematic and should be accounted for during model design. Ensuring $M \gg N$ or applying regularization becomes essential to avoid this pathological behavior.

10 Conclusion and Future Work

In this report, we explored the capabilities and limitations of a photonic-inspired quantized reservoir computing (RC) architecture for image classification using the MNIST dataset. Through a series of experiments, we demonstrated that even a simple, one-step sinusoidal reservoir, when properly tuned, can reach high classification performance up to **98.86% accuracy**, while remaining computationally efficient and compatible with neuromorphic implementation constraints.

We highlighted the importance of both **reservoir dynamics** and **input encoding**, showing that cross-optimizing HOG feature extraction alongside reservoir parameters significantly improves performance. Interestingly, optimal results were obtained near the *edge of stability*, supporting the hypothesis that expressivity in RC often emerges at the boundary of chaotic behavior.

We also identified a critical trade-off between reservoir size and training set size: increasing the number of neurons improves accuracy but leads to rapidly growing computation time, while setting $M \approx N$ causes generalization to collapse due to numerical instability in the readout. These findings stress the importance of careful architectural and data-aware design choices.

Future Work. Several promising directions remain open:

- **Temporal and sequential tasks:** Extending the model to process time-series data would unlock more applications and leverage the reservoir's inherent temporal dynamics.
- **Hardware implementation:** Deploying the model on physical photonic hardware or FPGA-based neuromorphic chips could validate its energy efficiency and real-time capabilities.
- **Improved classifiers:** Replacing the linear readout with more expressive models such as SVMs, decision trees, or small MLPs may enhance accuracy with little additional cost.
- **Deeper architectures:** Stacking multiple reservoirs or exploring hierarchical dynamics could increase feature richness without requiring larger reservoirs.
- **Multi-step dynamics:** Introducing temporal recurrence by running the reservoir over multiple timesteps per input may provide richer internal representations and further boost performance.

Overall, our work shows that simple and energy-efficient RC architectures can compete surprisingly well with more complex models, while offering attractive prospects for low-power AI applications.

References

[1] D. Rontani, T. Griguer, F. De Colle, and J. Albert, "Optimization of a photonic reservoir computer for signal classification," *Journal of Physics: Photonics*, vol. 3, no. 4, p. 044006, 2021.

[2] G. Tanaka, T. Yamane, J.-B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Networks*, vol. 115, pp. 100–123, 2019.