Informaticae Abstracta

Año 2025, Vol. 3, No. 2 JUL | DIC 2025
ISSN: 3061-8355

Universidad Autónoma del Estado de México

Informaticae Abstracta
Año 2025, Vol. 3, No. 1 ENE | JUN 2025

Informaticae Abstracta, año 2024, volumen 3, número 2, Julio – Diciembre 2025, es una revista de publicación semestral editada por la Universidad Autónoma del Estado de México, Instituto Literario 100 Ote., Col. Centro, Toluca, Estado de México, C.P. 50000, Tels. (722) 2140855 y 21140795 ext. 10 1217, http://informaticae.uaemex.mx, informaticae@uaemex.mx, Editora Responsable: Ruth Hernández Pérez, Facultad de Ingeniería. Reserva de Derechos al Uso Exclusivo número 04-2022-111113523100-102, ISSN: 3061-8355, ambos otorgados por el Instituto Nacional del Derecho de Autor. Responsable de la última edición Mercedes Lucero Chávez, de la Facultad de Ingeniería, Cerro de Coatepec s/n, Ciudad Universitaria, C.P. 50100, Toluca, Estado de México, fecha de última modificación, 30 diciembre 2025.

Informaticae Abstracta
Año 2025, Vol. 3, No. 2 JUL | DIC 2025

Índice

Informaticae Abstracta. ISSN: 3061-8533
Año 2025, Vol. 3, No. 2, pp. 4-24
JUL | DIC 2025
https://doi.org/10.36677/ia.v3n2.26890

Empirical Characterization of Lipschitz Continuity in k-nets with Precomputed Inner Functions

Huerta Mendoza, U. B.

Información del Artículo

Recibido: 5 de septiembre, 2025
Aceptado: 4 de noviembre, 2025

Palabras clave: Kolmogorov-Arnold networks, Lipschitz continuity, neural network regularity, function approximation, adversarial robustness

Resumen

The application of the Kolmogorov-Arnold representation theorem (KRT) in neural networks has historically been hindered by the non-smooth nature of its inner functions (ψ). While this has led to criticism of the theorem's relevance, the work of Sprecher and later Actor (2018) provided a constructive path forward by developing an algorithm for Lipschitz-continuous universal ψ. However, it remained an open question whether a network built upon this base could preserve that regularity after training.

1. Introduction

The value of the Kolmogorov-Arnold representation theorem (KRT) as basis for learning architectures has been a point of contention for decades. Several critical and survey works have situated KRT's influence as mostly philosophical or inspirational for modern machine learning, warning against misinterpreting it as a direct foundation for universal function approximation in neural networks [1, 2, 3, 4]. The main argument used to hinder KRT applicability is the often-found lack of "smoothness" of the inner ψ function. This is, the inner functions described in KRT can be highly non-smooth, even fractal [5], resulting in learned mappings that are not easily computationally tractable.

At the same time, Neural Networks (NNs) have become the preferred tool to characterize and analyze complex relationships in data, and the foundation of deep learning itself [6]. They've been successfully applied across essentially all scientific domains; a success built upon the well-established Multilayer Perceptron (MLP) architecture. The effectiveness of the MLP's brute force approach to approximating optimal functions for virtually any data source was mathematically formalized by the Universal Approximation Theorem, with no theoretical bounds beyond having a "sufficient" number of layers [7]. This number must be determined empirically for every application; a trial-and-error approach that extends to the selection of the MLP's activation functions.

Nonetheless, the widespread success of deep learning is often shadowed by the "black box"nature of MLPs, that difficult inference tracking due to its layer composition and structure [8]. Additionally, MLPs, like KRT representations, suffer from highly non-smooth outputs, an expected consequence of the universality of its approximation capabilities. This is reflected in high output sensitivity to changes in the inputs of trained models. Small changes in inputs of NNs with the intention of influencing output behavior are known as adversarial perturbations, or adversarial attacks [9].

Both paradigms, universal approximation and universal representation, are expected to deliver robust outputs under such attacks, since they represent a serious concern in the deployment of deep learning models in safety-critical systems such as autonomous driving [10].

One of the preferred methods to assess the robustness of NNs is the Lipschitz constant, defined as the maximum gradient of the function with respect to the distance metric [11]. Lipschitz-constrained neural networks limit the maximum deviation of the learned mapping in response to a change of the input, often at the expense of expressivity [12, 13]. An array of techniques are widely used to enforce this property, ranging from loss-based penalization to architectural constraints, such as 1-Lipschitz layers [14].

Despite skepticism, the applicability of the theorem was significantly advanced by the work of Sprecher [15], showing that the representation could be simplified such that distinct inner functions ψ_{qp} could be replaced by scaled and translated instances of a single, universal inner function ψ. This simplification makes the construction amenable to neural network development. Another notable step toward realizing this was the work of [16], who proposed an algorithm to construct a Lipschitz continuous ψ function via arc-length reparameterization. However, Actor explicitly left the construction of the outer function Φ as future work.

This work presents a Kolmogorov-Arnold neural network (*k-net*) [5] based on the Sprecher-Lorentz reformulation of the KRT that incorporates the precomputed, high-fidelity ψ function from Actor's algorithm parametrized by a Λ coefficient matrix. Throughout this work, we quantify the degree to which the regularity of Actor's precomputed inner function manifests as regularity of a trained network. Also, properties of the learned mapping, as well as standard predictive performance metrics, including its empirical Lipschitz constant, the smoothness of its internal representations, adversarial robustness, and gradient stability are evaluated. We also dissect the roles of ψ and its learnable coefficients Λ through an ablation study.

This work makes several contributions. First, Actor's work is extended by providing the first implementation that take advantage of his Lipschitz-continuous inner function. Second, a suite of internal regularity metrics designed to characterize the preservation of mathematical guarantees of the proposed *k-net* are introduced. Third, the exploration of the Λ coefficients as a viable path towards leveraging theoretical guarantees of the KRT is presented. Finally, a performance-regularity space demonstrates that the proposed architecture occupies a comparable position with MLPs.

This work is positioned as complementary to recent efforts in KRT based neural networks. While Liu et al. [8] focused on learnable B-spline edges for interpretability and Solis-Winkler et al. [5] defined *k-net* emphasizing algebraic symmetry through dual numbers, our work addresses the orthogonal question of whether and how theoretical regularity guarantees can be preserved and leveraged in implementations based on the theorem. The work built upon the rich mathematical foundations established by these and earlier works while pursuing the characterization of regularity in KRT-driven architectures.

This work presents the following structure: Section 2 details the mathematical background of the KRT, Actor's construction, and positions the work within the literature. Section 3 presents the model architecture. Section 4 describes the experimental design, including ablation variants and regularity metrics. Section 5 presents results. Section 6 discusses impli-

cations, and Section 7 concludes with directions for future work.

2. Background and Related Work

The Kolmogorov-Arnold Representation Theorem

The Kolmogorov-Arnold Representation Theorem (KRT), states that any multivariate continuous function can be represented as a finite sum of compositions of continuous univariate functions [17]. Formally, for any continuous function $f \in C([0,1]^n)$ mapping $\mathbb{R}^n \to \mathbb{R}$, there exist continuous univariate functions $\chi_q : \mathbb{I} \to \mathbb{R}$ (outer functions) and $\psi_{p,q} : \mathbb{R} \to \mathbb{R}$ (inner functions), where $p = 1, \dots, n$ and $q = 0, \dots, 2n+1$, such that:

$$f(x_1, \dots, x_n) = \sum_{q=0}^{2n} \chi_q \left(\sum_{p=1}^{n} \psi_{p,q}(x_p) \right) \tag{1}$$

The inner functions $\psi_{p,q}$ are independent of the function f. This theorem proved that a composition of univariate functions suffices for high-dimensional function approximation, opposed to the previous intuition that pointed to the necessity of functions of multiple variables [17].

Given it's universal scope, the original proof constructs inner functions that are often highly irregular: Hölder continuous but not necessarily differentiable, and in some cases exhibiting fractal-like behavior [18]. This poses severe challenges to the attempts to materialize the universal representation guarantees of the theorem computationally.

The applicability of the theorem was significantly advanced by Sprecher [15] and Lorentz [19]. Sprecher showed that all $n(2n+1)$ inner functions could be replaced by shifted and scaled versions of a single universal function ψ, and Lorentz that all outer functions could be replaced by a single function Φ. The reformulation becomes:

$$f(x_1, \dots, x_n) = \sum_{q=0}^{2n} \Phi \left(\sum_{p=1}^{n} \lambda_p \psi(x_p + q\epsilon) \right) \tag{2}$$

where λ_p are rationally independent constants satisfying $\sum_p \lambda_p = 1$, and ϵ is a small shift parameter. This Sprecher-Lorentz formulation forms the foundation of the proposed implementation.

This formulation separates the representation task into three stages. First, the input vector $\mathbf{x} \in \mathbb{R}^n$ undergoes $Q = 2n+1$ shifted projections through a universal inner function ψ. Second, these projections are weighted by coefficients $\Lambda = \{\lambda_{qp}\}$ to produce intermediate features $\{Z_q\}_{q=0}^{2n}$. Third, a shared outer network Φ processes each feature independently before summation yields the final prediction.

Kolmogorov-Arnold Neural Networks (*k-nets*)

The Kolmogorov-Arnold Neural Networks (*k-nets*) are new compositional mappings derived from KRT that allow the selection of the inner layer functions for neural network construction, while maintaining rigorous adherence to the theorem [5]. They are defined as the following:

Let $f : [0,1]^n \to \mathbb{R}$ be a continuous function for which exists univariate function $\Phi : [0,1] \to \mathbb{R}$, $\phi_{qp} : [0,1] \to \mathbb{R}$ such that $f = \sum_q \Phi_q(\sum_p \phi(x_p)$, $1 \leq q \leq 2n+1$, $1 \leq p \leq n$, then a *k-net* is a composition mapping $Z = Z_0 \circ ... \circ Z_L$, where $Z_i : R^{d_i} \to R^{d_{i+1}}$, and $d_{i-1} = d_i$. The coordinates of each $Z_i = (\zeta_1, ..., \zeta_{d_i})$. The model layer structure is the vector $[d_0, ..., d_L]$ of dimensions of the domain of every Z_i for each layer.

Actor's Lipschitz Continuous Inner Functions

While the Sprecher-Lorentz reformulations streamline computing representations, they do not present a method to secure a universal inner function ψ with robustness guarantees. The original constructions, such as Köppen's refinement of Sprecher's function [20], remain Hölder continuous with unbounded slopes.

Actor [16] addressed this gap through a reparameterization based on the work by Hedberg [21] and Kahane [22]. The center of his argument is that any strictly monotonic continuous function is rectifiable (has finite arc length), and any rectifiable curve admits a Lipschitz-continuous reparameterization. Formally, if $\hat{\psi}$ is a Hölder-continuous inner function, one can construct a Lipschitz-continuous equivalent by:

$$\psi = \hat{\psi} \circ \sigma^{-1} \tag{3}$$

where $\sigma(x)$ is the normalized arc-length parameterization of $\hat{\psi}$:

$$\sigma(x) = \frac{1}{L} \sup_{P \in \mathcal{P}(x)} \sum_{i=1}^{\ell} \sqrt{(t_i - t_{i-1})^2 + (\hat{\psi}(t_i) - \hat{\psi}(t_{i-1}))^2} \tag{4}$$

where $\mathcal{P}(x)$ is the set of all partitions of $[0, x]$ and L is the total arc length over $[0, 1]$ [16].

The algorithm that computes ψ starts with Köppen's version of Sprecher's function which produces a Hölder-continuous base function $\tilde{\psi}^k$, that is refined through iterative interval subdivision. At refinement level k, the construction generates γ^k points through recursive application of a fractal-like refinement rule parameterized by the base γ and dimension n. Then, reparametrized. The resulting function is Lipschitz continuous. Figure 1 reproduces Actor's result showing the difference between the Hölder function and the Lipschitz reparameterization. The algorithm begins with Köppen's refinement of Sprecher's inner function. The the reader is referred to [16] for further details.

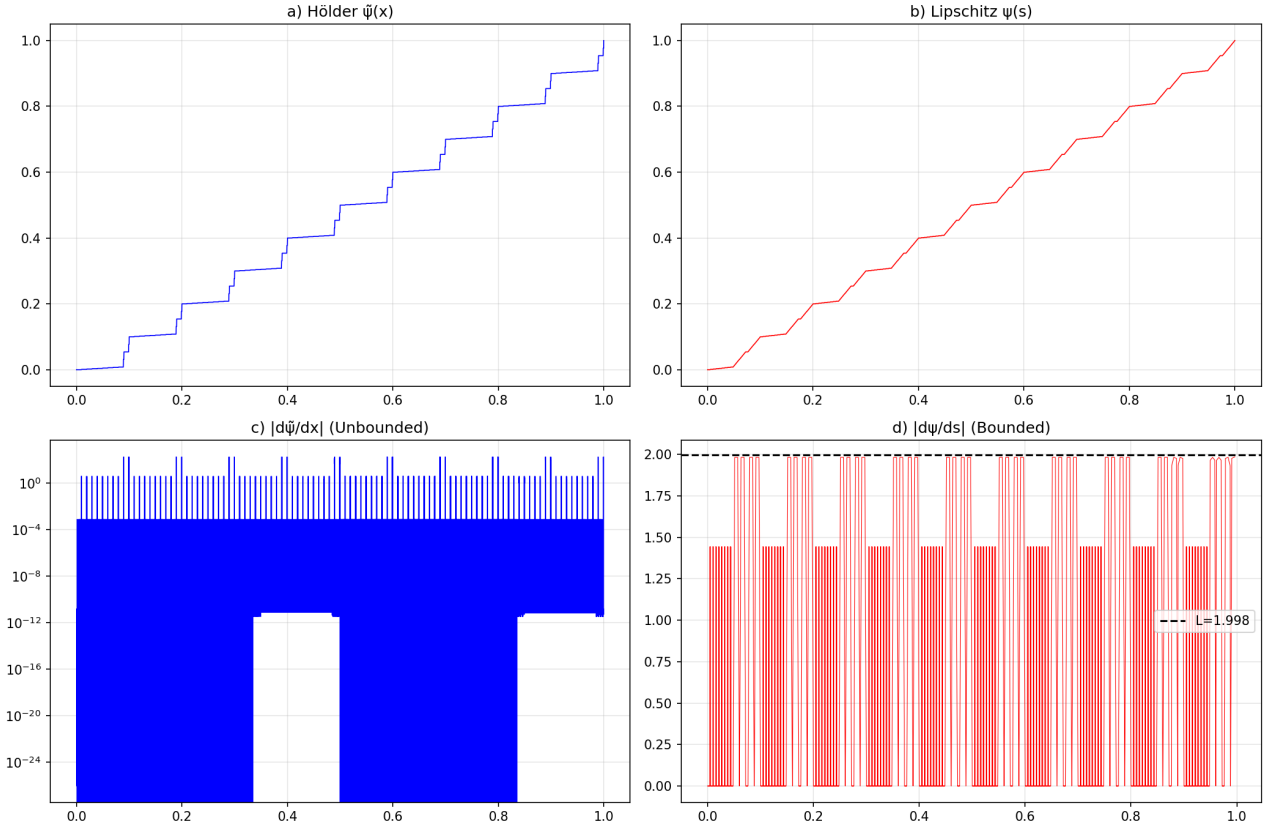

Figura 1: Comparison of inner functions ψ at refinement level $k = 6$. Arc-length reparameterization yields controlled slopes, ensuring Lipschitz continuity with constant $L_\psi \approx 1{,}998$.

Recent Implementations

The past years have seen renewed interest in Kolmogorov-Arnold inspired architectures, with distinct emphases.

KANs (Kolmogorov-Arnold Networks). Liu et al. [8] introduced an architecture where learnable univariate functions parameterized as B-splines are placed on the edges rather than nodes of the network. Their work emphasized interpretability and demonstrated strong performance on symbolic regression. However, their learnable B-spline functions lacked guarantees of regularity and is not a direct translation of the KRT.

***k-nets* with Dual Numbers.** Solis-Winkler et al. [5] recently presented the *k-net* neural network, that formalizes the training process within the algebraic framework of hyperdual numbers. One of the main contributions is demonstrating how forward-mode automatic differentiation via dual number arithmetic provides a mechanism for computing Jacobians while maintaining the KRT decomposition. Their inner functions are also B-splines, and their outer functions use simple activations (ReLU, sigmoid). This is a mathematically rigorous translation of the KRT into a deep learning architecture.

The architecture proposed in this article complements these efforts by addressing a different question. While [8] prioritize interpretability through learnable edge functions and [5]

emphasize algebraic symmetry and mathematical fidelity to the KRT, this work investigates whether and how theoretical regularity guarantees can be preserved in architectures derived from the KRT. The nodes of the inner layer in this *k-net* are neither learnable [8] nor dual extensions of splines [5], but rather a deterministic map with Lipschitz guarantees [16].

3. Model Architecture

k-net instance with precomputed inner function ψ

The *k-net* to be analyzed takes the Sprecher-Lorentz reformulation of the KRT as basis. The forward pass decomposes any continuous multivariate function through a composition of univariate functions according to

$$f(\mathbf{x}) = \sum_{q=0}^{2n} \Phi(Z_q), \quad \text{where} \quad Z_q = \sum_{p=1}^{n} \lambda_{qp} \cdot \psi(x_p + q\epsilon) \tag{5}$$

Architecture Components

Precomputed Inner Function ψ

The inner function $\psi : [0, 2] \to [0, 1]$ is the foundation of the architecture. The *k-net* use ψ as a fixed, deterministic map stored as a checkpoint lookup table. The precomputed ψ used in all experiments exhibits a Lipschitz constant $L_\psi \approx 1{,}988$. This guarantees controlled slope throughout the domain.

Trainable Coefficients Λ

The coefficient matrix $\Lambda \in \mathbb{R}^{Q \times n}$ with $Q = 2n + 1$ determines how input dimensions contribute to each projection. They must satisfy approximate rational independence to ensure the transformation $\mathbf{x} \mapsto \{Z_q\}$ achieves sufficient coverage of the latent space as prescribed by the Sprecher-Lorentz formulation.

Outer Network Φ

The outer function $\Phi : \mathbb{R} \to \mathbb{R}$ consists of a MLP that maps each intermediate projection Z_q to a scalar that is subsequently summed to produce the final output. The network implements a four-layer architecture with topology $1 \to 256 \to 128 \to 64 \to 1$. Hidden layers employ GELU (Gaussian Error Linear Unit) activation functions, chosen for their smooth gradient and effectiveness in regression tasks.

The same Φ network is applied to all Q projections, to reflect the structure of the Sprecher-Lorentz formulation, that specifying a single outer function rather than Q distinct functions. Sharing also confers the same parameters of outer network Φ to all projections.

The final prediction emerges from summation across all outer network outputs according to $f(\mathbf{x}) = \sum_{q=0}^{2n} \Phi(Z_q)$. Figure 2 illustrates the architecture, distinguishing precomputed components (shown in blue) from trainable parameters (shown in orange).

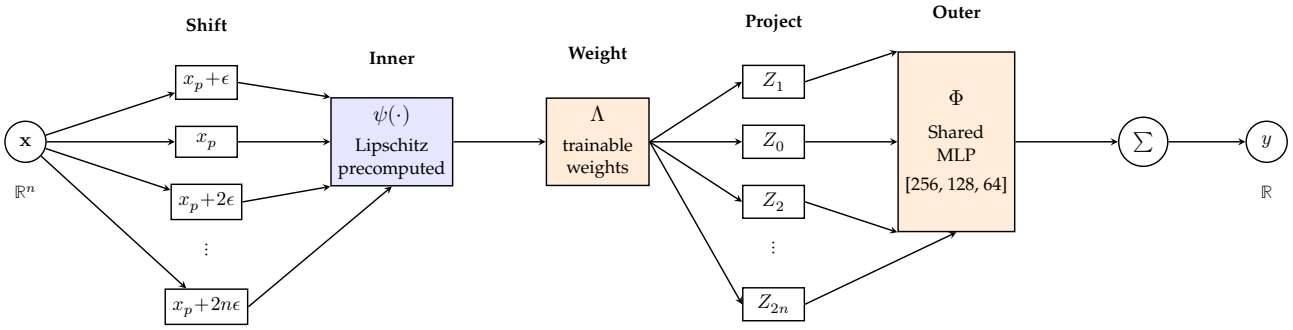

Figura 2: *k-net* architecture implementing the Sprecher-Lorentz formulation. Input $\mathbf{x} \in \mathbb{R}^n$ undergoes $Q = 2n+1$ shifted projections through the precomputed Lipschitz function ψ, weighted by trainable coefficients Λ to produce latent features $\{Z_q\}$. A shared univariate MLP Φ transforms each Z_q independently, and outputs are summed to produce the final prediction y. Blue indicates precomputed components; orange indicates trainable parameters.

Training Protocol

The training uses the AdamW optimizer with weight decay $\lambda = 10^{-4}$ to discourage excessive parameter magnitudes in Φ while preserving expressivity. The learning rate strategy applies $\alpha_\Lambda = 10^{-3}$ to the coefficient matrix and $\alpha_\Phi = 5 \times 10^{-4}$ to the outer network.

Learning rate scheduling utilizes the ReduceLROnPlateau strategy, monitoring loss with patience of 20 epochs and reduction factor of 0.5. Early stopping with patience of 50 epochs is implemented to prevent overfitting.

Regularization combines dropout in the outer network with weight decay in the optimizer. Batch normalization is avoided to preserve the structure of the summation $\sum_{q=0}^{2n} \Phi(Z_q)$, as it would introduce adaptive scaling that violates the additive decomposition prescribed by the theorem.

All experiments were conducted using PyTorch 2.0 on an NVIDIA RTX 2060 Super GPU with 8 GB VRAM. Training batch size was fixed at 64 samples with gradient accumulation employed when necessary. Mixed-precision training (FP16) was not utilized to maintain numerical precision in the function lookups and coefficient updates.

Code Availability

The complete implementation of the proposed *k-net* model, including the precomputed Lipschitz inner function ψ, training scripts, and evaluation routines, is publicly available at: https://github.com/redlab-math/koppen-knets.

4. Experimental Design

This section details the experimental setup for characterizing robustness metrics on *k-net*, including the benchmark functions and model configurations.

Benchmark Functions

The variants were evaluated on a suite of six two-dimensional ($n = 2$) benchmark functions, selected to represent a diverse range of properties. The functions are categorized as follows:

- **Smooth:** Quadratic ($f(\mathbf{x}) = x_1^2 + x_2^2$) and Branin functions, which feature smooth but potentially non-convex landscapes.
- **Non-Convex:** Rosenbrock and Six-Hump Camel functions, characterized by narrow valleys and multiple local minima.
- **Multimodal:** Rastrigin and Ackley functions, which possess numerous local optima, presenting a significant challenge for optimization and function representation.

500 training samples and 500 test samples by drawing points uniformly from the function's domain for each function were generated, then normalized to $[0, 1]^2$. The study was performed using 5-fold cross-validation to ensure robustness of the results. During training, 15 % of the training set was reserved as an internal validation set for early stopping.

Model Configurations

Four model variants are compared to assess the impact of *k-net* components. The number of trainable parameters was kept approximately constant across *k-net* variants and the MLP baseline to ensure a fair comparison.

- **k-net-Complete:** This is the full implementation of the model as detailed in Section 3. It uses ψ loaded from an HDF5 checkpoint and a set of trainable scaling coefficients Λ, implemented as a matrix of shape $(Q \times n) = (5 \times 2)$. The outer network Φ is a univariate MLP with architecture $1 \to 256 \to 128 \to 64 \to 1$, GELU activation functions, and dropout rate of 0.1. Total parameters: 41,739 (Λ: 10, Φ: 41,729).
- **k-net-FixedPsi:** This variant ablates the Lipschitz-continuous ψ function by replacing it with the standard hyperbolic tangent activation function during forward propagation. All other components, including the trainable Λ coefficients (10 params) and the outer network Φ (41,729 params), remain identical to k-net-Complete. Total parameters: 41,739.
- **k-net-FixedLambda:** This variant ablates the trainability of the scaling coefficients. It uses the same architecture as k-net-Complete, including the precomputed Lipschitz ψ, but the Λ matrix is registered as a non-trainable buffer instead of a matrix. The values of

Λ are initialized using the formulation $\lambda_p = \sqrt{p+1}/2n - 1$ plus small Gaussian noise, but are not updated during training. Total parameters: 41,729 (only Φ is trainable).

- **MLP Baseline:** A standard Multi-Layer Perceptron serves as baseline. Its architecture consists of an input layer (2 inputs), three hidden layers with sizes [256, 128, 64] and GELU activations, dropout rate of 0.1, and a single output neuron. This configuration was chosen to have a number of trainable parameters comparable to the *k-net* variants. Total parameters: 41,985.

Evaluation Metrics

The models were evaluated on two fronts: predictive performance and internal regularity. Performance metrics were calculated on the test set, while regularity metrics were estimated to ensure an unbiased assessment of the represented function properties.

Predictive Performance

The predictive accuracy was measured using three standard regression metrics:

- **Coefficient of Determination (R^2):** Measures the proportion of the variance in the dependent variable that is predictable from the independent variables.
- **Root Mean Squared Error (RMSE):** The square root of the average of the squared differences between prediction and actual observation.
- **Mean Absolute Error (MAE):** The average of the absolute differences between prediction and actual observation.

Internal Regularity

Four properties were measured to characterize the preservation of mathematical guarantees in the trained networks and to quantify the regularity and stability of the represented functions:

- **Empirical Lipschitz Constant (L_{emp}):** Estimated as the 95th percentile of the ratio $\|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|/\|\mathbf{x}_i - \mathbf{x}_j\|$ over 500 randomly sampled pairs of points from the test set. The 95th percentile was used rather than the maximum to avoid sensitivity to outliers while still capturing near-worst-case behavior. A lower value indicates a more regular, Lipschitz-bounded, function.
- **Internal Smoothness:** Measures the stability of the projection space Z and is specific to the *k-net* architectures. The $k = 10$ nearest neighbors in the input space X were identified for each point, then compute the variance of their corresponding projections $Z_q = \sum_p \lambda_{qp} \cdot \psi(x_p + q\epsilon)$ across all Q projections. The metric is the mean variance across all test points. A lower value indicates more stable internal representations, suggesting that nearby points in input space produce similar internal projections.

- **Adversarial Robustness:** Measured by applying a small random perturbation ($\epsilon = 0{,}01$ in L^2 norm) to 100 randomly sampled test inputs and calculating the sensitivity ratio $\|f(\mathbf{x}+\delta) - f(\mathbf{x})\|/\|\delta\|$. A smaller ratio indicates greater robustness to input variations.
- **Gradient Coefficient of Variation (Grad. CV):** Calculated by tracking the L^2 norm of parameter gradients $\|\nabla_\theta L\|$ during each training batch. The Grad. CV is the ratio of the standard deviation to the mean of these gradient norms across all training epochs: $\text{CV} = \sigma(\|\nabla_\theta L\|)/\mu(\|\nabla_\theta L\|)$. A lower value suggests a more uniform gradient landscape during optimization, which can be beneficial for training stability.

5. Results

The experimental protocol comprised 120 independent training runs (4 model variants $\times$ 6 functions $\times$ 5 folds). The findings are organized into three subsections addressing predictive performance (Section 5), internal regularity (Section 5), and performance vs regularity (Section 5).

Predictive Performance

Table 1 presents the aggregate predictive performance metrics across all benchmark functions and folds. Each entry represents the mean and standard deviation computed over 30 experimental runs per model variant (6 functions $\times$ 5 folds).

Tabla 1: Mean $\pm$ standard deviation of predictive performance metrics across all benchmark functions. Parameter counts refer to trainable parameters only. Performance metrics computed on held-out test sets.

Model	R^2	RMSE	MAE	Train Time (s)	Parameters
MLP	0.906 ± 0.178	7.71 ± 10.81	5.12 ± 6.48	9.9 ± 3.8	41,985
k-net-Complete	0.844 ± 0.151	31.40 ± 55.46	23.27 ± 40.55	18.9 ± 1.9	41,739
k-net-FixedPsi	0.692 ± 0.250	56.41 ± 114.83	40.11 ± 81.38	17.9 ± 5.9	41,739
k-net-FixedLambda	0.437 ± 0.300	92.45 ± 181.82	63.44 ± 122.86	9.1 ± 3.8	41,729

k-net-Complete achieves a mean R^2 of 0.844 across the benchmark suite, demonstrating successful operationalization of the Sprecher-Lorentz formulation with the scaled Actor's Lipschitz-continuous inner function. The architecture captures 84.4 % of the variance in the target functions. For context, the MLP baseline achieves $R^2 = 0{,}906$, establishing that k-net-Complete attains 93.2 % of this benchmark while maintaining its theorem-grounded structure. The performance gap of $\Delta R^2 = 0{,}062$ relative to the MLP quantifies the distance between this implementation and the de-facto standard, suggesting opportunities for refinement in the outer network capacity and optimization techniques.

The ablation variants isolate the contributions of the two architectural components. Replacing the inner function ψ with a $\tanh$ activation (k-net-FixedPsi) yields $R^2 = 0{,}692$, corresponding to an 18.0 % degradation compared to k-net-Complete ($\Delta R^2 = -0{,}152$). This difference achieves statistical significance under paired t-test ($t_{29} = 4{,}54$, $p < 0{,}001$, Cohen's

$d = 0{,}748$), indicating a medium-to-large effect size. The result provides evidence that the specific mathematical properties encoded in ψ contribute substantially to expressiveness beyond what generic smooth activations provide.

Fixing trainable Λ coefficients to their theoretical initialization (k-net-FixedLambda) produces a performance collapse to $R^2 = 0{,}437$, representing a 48.2 % degradation from k-net-Complete ($\Delta R^2 = -0{,}407$, $t_{29} = 8{,}91$, $p < 0{,}001$, Cohen's $d = 1{,}739$). The large effect ($d > 1{,}5$) and high significance level establish that Λ trainability constitutes an indispensible element. The KRT theorem guarantees that suitable coefficients exist within the space $\mathcal{L} = \{\Lambda \in \mathbb{R}^{Q\times n} \mid \sum_p \lambda_{qp} = 1\}$, yet provides no algorithmic guidance for identifying them. The initialization $\lambda_p \propto \sqrt{p+1}/(2n-1)$, formulated to achieve the integral independence requirement occupies an arbitrary position within this vast coefficient space. The 48.2 % performance gap quantifies the distance between this arbitrariness and the suitable values. This results demonstrates that gradient-based optimization over Λ successfully bridges the theorem guarantees and implementations.

The error metrics (RMSE, MAE) exhibit substantial variance reflecting the diversity of target function scales across the benchmark suite. The MLP achieves lower mean error (RMSE = 7.71 compared to k-net-Complete's 31.40), a difference attributable to both the heterogeneity of the test functions and the constraints imposed by this translation of the theorem. Standard deviations of comparable magnitude to the means (RMSE standard deviation of 10.81 for MLP, 55.46 for k-net-Complete) indicate considerable variation across function classes, motivating the per-category analysis presented in Section 5.

Regarding efficiency, k-net-Complete requires a mean training time of 18.9 $\pm$ 1.9 seconds per fold, representing a 91 % increase relative to the MLP baseline (9.9 $\pm$ 3.8 seconds). k-net-FixedLambda achieves training times (9.1 $\pm$ 3.8 seconds) statistically indistinguishable from the MLP baseline, confirming that gradient computation for Λ contributes minimally to the overhead.

Internal Regularity

Table 2 presents the internal regularity metrics that quantify properties of the learned mappings beyond predictive accuracy. These metrics interrogate whether the regularity of ψ manifests in the trained networks.

Tabla 2: Mean $\pm$ standard deviation of internal regularity metrics across all benchmark functions. Lower values indicate more desirable regularity properties for all metrics. Metrics on test sets following the protocol specified in Section 4.

Model	$L_{\text{emp}} \downarrow$	**Smoothness** $\downarrow$	**Adv. Sens.** $\downarrow$	**Grad. CV** $\downarrow$
k-net-FixedLambda	3.26 $\pm$ 0.75	0.002 $\pm$ 0.000	1.82 $\pm$ 0.67	0.358 $\pm$ 0.118
k-net-FixedPsi	4.35 $\pm$ 1.21	0.001 $\pm$ 0.000	2.79 $\pm$ 0.93	0.322 $\pm$ 0.076
k-net-Complete	5.39 $\pm$ 1.04	0.029 $\pm$ 0.016	3.86 $\pm$ 1.10	0.332 $\pm$ 0.119
MLP	5.84 $\pm$ 1.19	0.171 $\pm$ 0.102	3.88 $\pm$ 1.40	0.512 $\pm$ 0.171

The empirical Lipschitz constant L_{emp} quantifies the rate of output change relative to input perturbations across the test set. k-net-Complete achieves $L_{\text{emp}} = 5{,}39$ compared to the MLP's $L_{\text{emp}} = 5{,}84$, representing a 7.7 % reduction. This difference achieves statistical significance under paired t-test ($t_{29} = -3{,}21$, $p = 0{,}003$), indicating that the Sprecher-Lorentz decomposition imposes regularity constraints even with trainable parameters. The ordering of all four architectures by Lipschitz constant inversely correlates with predictive performance (Spearman $\rho = -0{,}89$, $p < 0{,}001$ across all 120 experiments). The correlation suggests that the current implementation has not yet converged to the theoretical optimal. Potential explanations include insufficient capacity in the outer network Φ or local minima in the optimization landscape.

k-net-FixedLambda exhibits the lowest L_{emp} (3.26) with the poorest predictive performance, reflecting its stiffness. This architecture defaults to a highly regular but constrained mapping. Conversely, k-net-Complete achieves substantially improved performance at the cost of a moderately increased Lipschitz bound. The observation that k-net-Complete maintains a lower L_{emp} than the MLP architecture suggests that its structure provides an inductive bias toward regularity.

The internal smoothness metric reveals that k-net-Complete achieves a smoothness value of 0.029, approximately six times lower than the MLP's 0.171. This metric quantifies variance in the internal projection space $Z_q = \sum_p \lambda_{qp} \psi(x_p + q\epsilon)$ across neighborhoods in the input space X. *k-net* structure, where each Z_q emerges from weighted sums of univariate evaluations, naturally produces stable and continuous internal representations. The MLP exhibits higher local variance in its approximations.

The FixedLambda and FixedPsi k-net models achieve smoothness values approaching zero (0.002 and 0.001 respectively), indicating that fixed coefficients or simplified activations yield maximally stable internal representations at the cost of reduced expressivity. This pattern suggests that the smoothness-performance relationship reflects the adaptability of the projection stage. Trainable Λ coefficients introduce essential variability in the Z-space for capturing complex target functions while maintaining smoothness.

The adversarial sensitivity metric, quantifying output change per unit input perturbation ($\epsilon = 0{,}01$ in L^2 norm), reveals minimal difference between k-net-Complete (3.86) and MLP (3.88). Both architectures respond similarly to small adversarial perturbations, indicating that internal smoothness in the projection space Z does not directly translate to improved robustness. The absence of correlation between internal smoothness and adversarial sensitivity (Pearson $r = 0{,}08$, $p = 0{,}38$ across models) demonstrates that these metrics capture orthogonal properties of the represented functions. k-net-FixedLambda again exhibits the lowest adversarial sensitivity (1.82), consistent with its pattern of maximal regularity coupled with minimal expressivity. The reduced sensitivity in ablated variants reflects their inability to capture fine-grained features, resulting in smoother boundaries that are less sensitive to perturbations but also less accurate.

The gradient coefficient of variation (Grad. CV) quantifies training stability through the

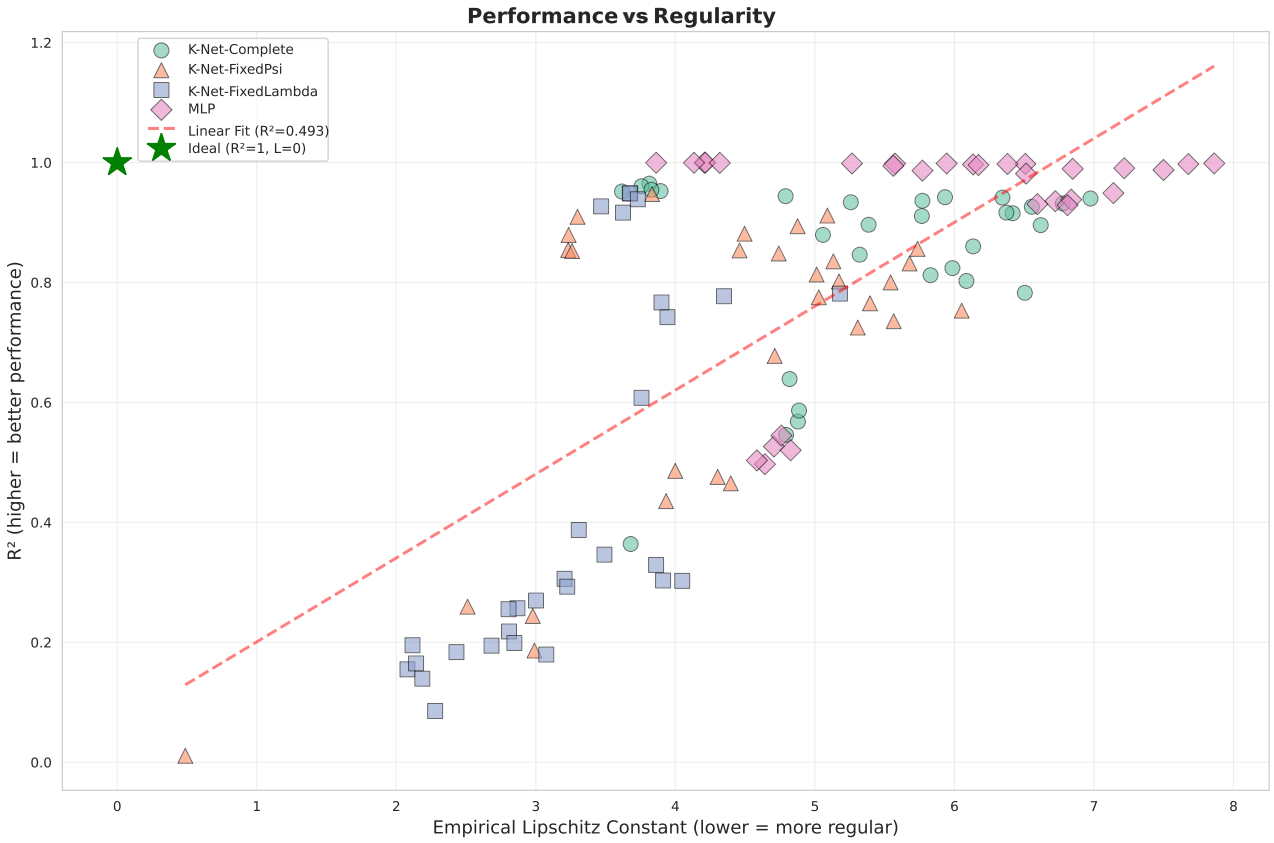

Figura 3: Performance-regularity characterization across 120 experimental runs.

ratio of standard deviation to mean of parameter gradient norms across training epochs. k-net-Complete achieves Grad. CV = 0.332, representing a 35 % reduction compared to the MLP's Grad. CV = 0.512. This improvement suggests more uniform gradient flow throughout training, attributable to the structured parameter space. The Λ coefficients provide a mechanism for adjusting projection contributions, while the shared univariate network Φ constrains the dimensionality of the optimization landscape relative to the MLP's layers. k-net-FixedPsi demonstrates slightly lower Grad. CV (0.322), reflecting simplified gradient dynamics associated with the $\tanh$ activation compared to the lookup evaluation of ψ.

Performance vs Regularity

The negative correlation between R^2 and L_{emp} (Pearson $r = -0{,}56$, $p < 0{,}001$) characterizes the implementations across both *k-net* variants and the MLP baseline. A linear regression fit yields $R^2 = -0{,}237 \cdot L_{\text{emp}} + 1{,}049$ (model $R^2 = 0{,}31$), though this relationship masks heterogeneity across architectures and function classes. k-net-Complete and the MLP baseline occupy adjacent regions of the performance–regularity space, with both achieving $R^2 > 0{,}84$ while maintaining $L_{\text{emp}} < 6{,}0$. Table 3 present performance by function category, revealing how both architectures respond to the particularities of each target function.

Both architectures achieve high performance on smooth functions (quadratic, Branin), with the MLP attaining near-perfect fits ($R^2 = 0{,}999$) and k-net-Complete achieving $R^2 =$

Tabla 3: Mean R^2 and L_{emp} decomposed by function category for k-net-Complete and MLP baseline. Each cell aggregates 10 experimental runs (2 functions per category $\times$ 5 folds).

Category	**Model**	R^2 **Mean**	L_{emp} **Mean**
Smooth	k-net-Complete	0.935	4.52
	MLP	0.999	4.98
Non-Convex	k-net-Complete	0.879	6.15
	MLP	0.992	6.78
Multimodal	k-net-Complete	0.719	5.52
	MLP	0.727	5.76

$0{,}935$. The performance gap widens on non-convex functions (Rosenbrock, Six-Hump Camel), where the MLP achieves $R^2 = 0{,}992$ compared to k-net-Complete's $R^2 = 0{,}879$. The current k-net-Complete captures smooth function classes but encounters greater challenges on non-convex landscapes.

The most notable pattern emerges on multimodal functions (Rastrigin, Ackley), characterized by numerous local optima and rugged optimization landscapes. Here the performance gap nearly vanishes, with k-net-Complete achieving $R^2 = 0{,}719$ compared to the MLP's $R^2 = 0{,}727$, a difference of only 1.1 %. This indicates that *k-net*'s fixed inner layer do not disadvantage the architecture on highly complex, multimodal functions. This near-equivalence, combined with superior regularity metrics, suggests that *k-nets* may be well-suited for complex optimization regimes.

k-net-Complete maintains lower L_{emp} than the MLP across all categories, with improvements of 9.2 %, 9.3 %, and 4.3 % for smooth, non-convex, and multimodal classes respectively. This pattern corroborates that the dual layer structure imposes regularity constraints independent of target function.

Statistical Validation

T-tests comparing k-net-Complete against each ablation variant and the MLP baseline were conducted to assess the significance of performance differences. Given three pairwise comparisons, applied Bonferroni correction was applied with significance threshold $\alpha^* = 0{,}05/3 = 0{,}0167$.

The comparison between k-net-Complete and k-net-FixedPsi on R^2 yields a mean difference of +0.152 favoring k-net-Complete. The paired t-test produces $t_{29} = 4{,}54$ ($p = 0{,}0001$) with Cohen's $d = 0{,}748$, indicating a medium-to-large effect. This result establishes that replacing Actor's ψ with generic $\tanh$ activation significantly degrades performance.

k-net-Complete versus k-net-FixedLambda demonstrates an even larger effect. The mean difference of +0.407 favoring k-net-Complete corresponds to paired t-test statistics of $t_{29} = 8{,}91$ ($p < 0{,}0001$) with Cohen's $d = 1{,}739$, indicating a large effect. This result establishes that fixed Λ lead to performance collapse.

The comparison between k-net-Complete and the MLP baseline yields a mean difference of -0.062 favoring the MLP. The paired t-test produces $t_{29} = -4{,}80$ ($p < 0{,}0001$) with Cohen's $d = -0{,}381$, indicating a small-to-medium effect. The modest effect size, substantially smaller than between variations of *k-net* ($d = 1{,}74$ for the Λ ablation), demonstrates that the performance difference between the theorem-based architecture and the empirical MLP baseline reflects optimization opportunities rather than fundamental constraints.

All three comparisons achieve significance under Bonferroni-corrected thresholds. The effect sizes provide context. The impact of fixing Λ ($d = 1{,}74$) exceeds the k-net-Complete versus MLP difference ($|d| = 0{,}38$) by more than a factor of four, indicating that components within the theorem-based design exert stronger influence on performance than the choice between shallow or deep architectures.

For Lipschitz bound comparisons, k-net-Complete achieves significantly lower L_{emp} than the MLP baseline with mean difference of -0.448 (paired t-test $t_{29} = -3{,}21$, $p = 0{,}0030$). The comparison between k-net-Complete and k-net-FixedLambda reveals mean difference of +2.139, with k-net-FixedLambda exhibiting lower empirical Lipschitz constant (paired t-test $t_{29} = 15{,}87$, $p < 0{,}0001$). This confirms that trainable Λ permits a small Lipschitz bound increase for substantially improved expressivity.

6. Discussion

The results establish *k-net* as a theoretically-grounded neural net that preserves a great level of regularity of Actor's ψ. In this section, we interpret the findings within the context of approximation theory, explore the possible origins of the observed differences, and address the question motivating this research line: if the Kolmogorov-Arnold representation theorem guarantees exact representation, why does k-net-Complete achieve $R^2 = 0{,}844$ rather than $R^2 = 1{,}0$?

Trainable Coefficients

The most interesting finding from the ablation analysis concerns the 48.2 % performance collapse when fixing the scaling coefficients Λ (k-net-FixedLambda $R^2 = 0{,}437$ versus k-net-Complete's $R^2 = 0{,}844$). This result opens a path toward designing new, more accurate, and shallower, algorithmic translations of the Kolmogorov-Arnold representation theorem.

The Sprecher-Lorentz formulation guarantees the existence of rationally independent coefficients λ_p such that any continuous function can be exactly represented. The initialization follows $\lambda_p = \sqrt{p+1}/(2n-1)$, normalized, with small Gaussian perturbations to break symmetry. These initial values satisfy the integral independence condition of the theorem, and their evolving magnitudes çalibrate"the represented function by scaling ψ until a given error bound is reached. The expressivity of the outer network Φ is more sensible to the modest parameter count of Λ than to its 41,729 trainable parameters. A properly scaled Actor's ψ contribute with at least half of the representation capacity of the whole architecture. Z_q can-

not completely compensate inner layer stiffness through its own adaptation despite being a parametrized analytic function.

When Λ is fixed, the feature space $\{Z_q\}_{q=0}^{2n}$ becomes entirely determined by the input and the precomputed ψ function. Φ must learn the target function using only these five fixed features. In contrast, trainable Λ allow to discover which linear combinations of $\psi(x_p + q\epsilon)$ prove most informative for the specific task.

We can formalize this as a representational capacity argument. The space of functions representable by k-net-FixedLambda is

$$\mathcal{F}_{\text{fixed}} = \left\{ f \mid f(\mathbf{x}) = \sum_{q=0}^{2n} \Phi(Z_q^{\text{fixed}}),\ \Phi \in \mathcal{H} \right\} \tag{6}$$

where $Z_q^{\text{fixed}} = \sum_p \lambda_p^{\text{init}} \psi(x_p + q\epsilon)$ and $\mathcal{H}$ denotes the hypothesis set of the outer MLP. With trainable Λ, the representational space expands to

$$\mathcal{F}_{\text{trainable}} = \left\{ f \mid f(\mathbf{x}) = \sum_{q=0}^{2n} \Phi\left(\sum_p \lambda_{qp} \psi(x_p + q\epsilon) \right),\ \Lambda \in \mathbb{R}^{Q \times n}, \Phi \in \mathcal{H} \right\} \tag{7}$$

The trainable formulation enables learning in the projection stage itself, whereas the fixed formulation constrains the model to a predetermined feature representation. The results ($R^2 = 0{,}844$ versus $R^2 = 0{,}437$) quantify the magnitude of this representational constraint.

The theorem establishes that suitable fixed coefficients exist within the admissible space but provides no constructive procedure for identifying them. The presented training paradigm allow effective coefficients to be discovered through optimization starting from random initialization. The 48.2 % performance gap suggests that the landscape of the coefficient space $\Lambda \in \mathbb{R}^{Q \times n}$ exhibits high non-convexity, necessitating a complete optimization routine with high numerical precision due to the rational nature of the coefficients ($\sum_p \lambda_p = 1$) , rather than relying on fixed, trial-and-error initialization schemes. This finding stablish a direction for future work regarding the structure and optimization Λ.

The Role of the Lipschitz-Continuous Inner Function

The second more interesting result is the 22 % performance improvement of the Complete k-net model over k-net-FixedPsi ($R^2 = 0{,}844$ versus $R^2 = 0{,}692$). This establishes that the properties encoded in Actor's Lipschitz-continuous ψ function contribute substantially to model expressiveness beyond what generic smooth activations provide. Both ψ and $\tanh$ satisfy Lipschitz continuity with bounded derivatives. They appear interchangeable. However, three fundamental differences emerge from analysis.

First, their Lipschitz constants differ. The hyperbolic tangent exhibits a maximum value of $L_{\tanh} = \max_x |\tanh'(x)| = 1$ achieved at $x = 0$, while Actor's construction yields $L_\psi \approx 1{,}998$ measured from the reparameterized function. The larger Lipschitz constant of ψ permits steeper local variations, potentially enabling finer-grained feature discrimination.

Second, the hyperbolic tangent implements a monotonic sigmoid approaching ± 1 asymptotically. In contrast, ψ emerges from arc-length reparameterization of Köppen's fractal-like function, exhibiting periodic structure and multi-scale character. This may provide richer representational capacity aligned with the theorem's universal guarantees.

Third, the hyperbolic tangent represents a smooth activation with no intrinsic connection to the representation theorem, while ψ was constructed to satisfy its conditions. This ensures that ψ encodes the richness required for universal function decompositionThe performance gap between the Complete and FixedPsi k-net models exhibits variation across function categories. FixedPsi achieves $R^2 = 0{,}91$ (only 2.7 % below Complete) on smooth functions, but the gap widens to 71 % on non-convex topology. This suggests that the simplified periodic structure of ψ provides sufficient regularizing capacity for smooth mappings but lacks the representational power required for highly nonlinear boundaries.

Architectural Differences and Parameter Counts

k-net-Complete (41,739) and the MLP baseline (41,985) approximately equivalent parameter counts where chosen to allow a fair comparison. However, this is not a direct equivalence due to architectural differences. The parameter distribution differs fundamentally between architectures. k-net-Complete allocates 10 parameters to the trainable Λ matrix and 41,729 parameters to the outer network Φ implementing architecture $[1 \to 256 \to 128 \to 64 \to 1]$. The MLP baseline distributes 41,985 parameters across architecture $[2 \to 256 \to 128 \to 64 \to 1]$.

The outer network Φ in k-net-Complete operates as a shared univariate function applied independently to each of the $Q = 5$ projections. The same 41,729 parameters process five distinct inputs $Z_0, Z_1, \ldots, Z_4$ sequentially during forward propagation. This enforces universality, Φ cannot specialize to the position or role of each projection but must function effectively across all inputs. In contrast, each parameter in the MLP serves exactly one position in the computational graph. The first layer processes raw inputs x_1, x_2 directly, middle layers perform hierarchical feature extraction with position-specific specialization, and the final layer produces output through potentially specialized structure.

The activation function evaluation count also differs substantially. k-net-Complete evaluates three GELU activations per application of Φ, applied five times per forward pass, yielding 15 total activation evaluations per sample. The MLP baseline evaluates four GELU activations per forward pass. *k-net* thus performs nearly four times as many activation computations per sample, potentially contributing to the observed 91 % training time overhead alongside other architectural factors.

The Performance-Regularity Relationship

Several factors plausibly contribute to the observed gap. Architectural capacity in the inner Λ coefficients represent one candidate factor. The theorem requires n coefficients λ_p

replicated across projections with small shifts, whereas this implementation learns a full $(Q \times n)$ matrix. This over-parameterization may introduce optimization challenges or permit departure from the theoretical optimal structure.

Additionally, local minima in the non-convex optimization landscape affect gradient-based training. The parameter space of k-net-Complete spans both Λ and Φ, creating a complex loss surface. In this case, standard training procedures potentially could not guarantee convergence to global optima.

Finite sample effects might further contribute to deviation from theoretical guarantees. The Kolmogorov-Arnold representation theorem applies to continuous functions defined on compact domains, while this implementation trains on finite samples (500 training points per function) with potential generalization gaps between training and test distributions.

The positioning of k-net-Complete and the MLP baseline in adjacent regions of the performance–regularity space suggests that current implementations face similar challenges in simultaneously optimizing for predictive accuracy and regularity. Neither architecture achieves the theoretical ideal of ($R^2 = 1{,}0$, $L_{\text{emp}} \to 0$), indicating room for improvement through fine-tuned architectures, optimization procedures, or training paradigms.

Multimodal Convergence

An intriguing pattern concerns the negligible performance gap on multimodal functions. k-net-Complete achieves $R^2 = 0{,}719$ compared to MLP's $R^2 = 0{,}727$ on Rastrigin and Ackley functions, representing only a 1.1 % difference. This contrasts with the 6.4 % gap on smooth functions and 11.3 % gap on non-convex functions, suggesting that *k-net*'s and MLP's optimization routing might stuck in the same local minimum on complex functions.

An alternative interpretation suggests that the minimalist architecture of *k-net* could act as beneficial regularization against overfitting. Multimodal functions create risk of memorizing spurious patterns. *k-net* enforces discovery of latent representations that generalize across all projections. This may prevent overfitting to local artifacts that an MLP architecture could memorize.

Discriminating these hypotheses requires additional experiments. Varying training set size would test the regularization hypothesis. *k-net*'s advantage should increase with smaller training sets where overfitting risk increments. Loss landscape visualization comparing *k-net* and MLP in weight space would directly assess multimodality. These experiments remain subjects for future research.

Limitations and Scope

Several constraints circumscribe the scope and generalization of this conclusions. The benchmark suite evaluates only two-dimensional functions ($n = 2$). The Kolmogorov-Arnold representation theorem manifests its most remarkable in high-dimensional problems, asser-

ting that $2n + 1$ univariate functions suffice regardless of input dimensionality n. Whether *k-net*'s regularity amplify, diminish, or remain constant as n grows represents an open question.

The six benchmark functions represent idealized test cases with known closed forms. Real-world regression tasks exhibit additional complexities including noise, missing data, heteroscedastic variance, and data-generating processes that may alter performance and regularity.

This implementation fixes the outer network architecture at Φ: $[256, 128, 64]$ and the number of projections at $Q = 2n + 1 = 5$ based on an interpretation of the theoretical prescription. Systematic exploration of alternative design choices including different Φ, varying Λ, residual connections could reveal improved performance.

The current study focuses on ablating Actor's Lipschitz ψ and trainable Λ coefficients. Broader comparison with other recent Kolmogorov-Arnold network implementations such as Liu et al.'s learnable B-splines or Solís-Winkler et al.'s dual number formulation would provide valuable context for positioning this architecture within the emerging landscape of KRT-inspired models. However, such comparisons introduce confounding variables including different training protocols, datasets, and hyperparameters that complicate causal attribution of performance differences.

Future Directions

Extending evaluation to high-dimensional regimes ($n \geq 10$) would test the models capacity to harness the central guarantee of the Kolmogorov-Arnold representation theorem: that univariate decompositions suffice even in very high dimensions. If *k-net*'s advantages remain constant or improve with increasing n, the architecture becomes compelling for high-dimensional problems where MLPs face dimensionality challenges.

7. Conclusion

This work addresses a foundational question at the intersection of approximation theory and machine learning concerning whether theoretical regularity guarantees from the Kolmogorov-Arnold representation theorem can be preserved and leveraged in practical, trainable neural architectures. The results suggest that Actor's Lipschitz-continuous inner function successfully propagates its properties through empirical characterization spanning 120 independent experiments across diverse function classes, manifesting as superior regularity while maintaining competitive performance.

A *k-net* based on the Sprecher-Lorentz reformulation of the Kolmogorov-Arnold representation theorem that combines Actor's Lipschitz-continuous inner function ψ with trainable scaling coefficients (Λ) and a shared univariate outer network (Φ) is introduced. The presented ablation study establishes three facts:

First, trainable coefficients prove indispensable for representing functions. Fixing Λ to its initialization values causes a 48.2 % performance collapse (R^2 declining from 0.844 to 0.437), gradient-based optimization is required to discover effective ψ scalings in this architecture.

Second, Actor's Lipschitz-continuous ψ substantially outperforms generic activations. Replacing the carefully constructed ψ with standard `tanh` degrades performance by 22 % (R^2 declining from 0.844 to 0.692). This confirms that the specific mathematical properties of the theorem-grounded inner function yield benefits for expressivity and regularity that are not captured by generic smooth functions.

Third, this *k-net* variant successfully achieves its primary design objective of preserving and leveraging theoretical regularity guarantees. k-net-Complete attains $R^2 = 0{,}844$ versus the MLP's $R^2 = 0{,}906$ while demonstrating superior functional properties. The architecture exhibits a 7.7 % lower Lipschitz constant ($L_{\text{emp}} = 5{,}39$ versus 5.84), maintains a 35 % more uniform gradient landscape (Grad. CV of 0.332 versus 0.512), and achieves internal projection stability (smoothness metric of 0.029 versus 0.171) nearly an order of magnitude better than the baseline. These improvements manifest consistently across all benchmark functions, confirming that the Kolmogorov-Arnold structure imposes beneficial constraints on representations. Notably, on challenging multimodal functions characterized by rugged optimization landscapes, the performance gap essentially vanishes (k-net-Complete achieves $R^2 = 0{,}719$ versus MLP's $R^2 = 0{,}727$, only 1.1 % difference).

Referencias

[1] F. Girosi y T. Poggio, *Representation Properties of Networks: Kolmogorov's Theorem Is Irrelevant*, 1989. DOI: 10.1162/neco.1989.1.4.465

[2] L. Xing, *Kolmogorov superposition theorem and its applications*, 2015. DOI: 10.25560/30762

[3] V. Ismailov, *Addressing Common Misinterpretations of KART and UAT in Neural Network Literature*, 2024. DOI: 10.48550/arXiv.2408.16389

[4] S. Somvanshi, S. A. Javed, M. M. Islam, D. Pandit y S. Das, *A Survey on Kolmogorov-Arnold Network*, 2024. DOI: 10.48550/arXiv.2411.06078

[5] A. Solis-Winkler, J. R. Marcial-Romero y J. A. Hernández-Servín, «Symmetry in the Algebra of Learning: Dual Numbers and the Jacobian in k-nets,» *Symmetry*, vol. 17, n.º 8, 2025, ISSN: 2073-8994. DOI: 10.3390/sym17081293 dirección: https://www.mdpi.com/2073-8994/17/8/1293

[6] J. Berner, P. Grohs, G. Kutyniok y P. Petersen, «The Modern Mathematics of Deep Learning,» en *Mathematical Aspects of Deep Learning*, P. Grohs y G. Kutyniok, eds. Cambridge University Press, 2022, págs. 1–111.

[7] K. Hornik, M. Stinchcombe y H. White, «Multilayer feedforward networks are universal approximators,» *Neural Networks*, vol. 2, n.º 5, págs. 359-366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8 dirección: https://www.sciencedirect.com/science/article/pii/0893608089900208

[8] Z. Liu, Y. Wang, S. Vaidya et al., «KAN: Kolmogorov-Arnold Networks,» *arXiv preprint arXiv:2404.19756*, 2024.

[9] M.-M. Zühlke y D. Kudenko, «Adversarial Robustness of Neural Networks from the Perspective of Lipschitz Calculus: A Survey,» *ACM Comput. Surv.*, vol. 57, n.º 6, feb. de 2025, ISSN: 0360-0300. DOI: 10.1145/3648351 dirección: https://doi.org/10.1145/3648351

[10] M. Bojarski et al., *End to End Learning for Self-Driving Cars*, 2016. arXiv: 1604.07316 [cs.CV]. dirección: https://arxiv.org/abs/1604.07316

[11] A. Virmaux y K. Scaman, «Lipschitz regularity of deep neural networks: analysis and efficient estimation,» en *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi y R. Garnett, eds., vol. 31, Curran Associates, Inc., 2018. dirección: https://proceedings.neurips.cc/paper_files/paper/2018/file/d54e99a6c03704e95e6965532dec148b-Paper.pdf

[12] L. Li, T. Xie y B. Li, «SoK: Certified Robustness for Deep Neural Networks,» en *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, págs. 1289-1310. DOI: 10.1109/SP46215.2023.10179303

[13] L. Béthune, T. Boissin, M. Serrurier, F. Mamalet, C. Friedrich y A. Gonzalez Sanz, «Pay attention to your loss : understanding misconceptions about Lipschitz neural networks,» en *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho y A. Oh, eds., vol. 35, Curran Associates, Inc., 2022, págs. 20 077-20 091. dirección: https://proceedings.neurips.cc/paper_files/paper/2022/file/7eb3d8ae592966543170a65e6b698828-Paper-Conference.pdf

[14] B. Prach, F. Brau, G. Buttazzo y C. H. Lampert, «1-Lipschitz Layers Compared: Memory Speed and Certifiable Robustness,» en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, págs. 24 574-24 583.

[15] D. A. Sprecher, «On the structure of continuous functions of several variables,» *Transactions of the American Mathematical Society*, vol. 115, págs. 340-355, 1965.

[16] J. Actor, *Computation for the Kolmogorov Superposition Theorem*, 2018.

[17] A. N. Kolmogorov, *On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables*. American Mathematical Society, 1961.

[18] J. Schmidt-Hieber, «The Kolmogorov–Arnold representation theorem revisited,» *Neural networks*, vol. 137, págs. 119-126, 2021.

[19] G. G. Lorentz, «Metric Entropy, Widths, and Superpositions of Functions,» *The American Mathematical Monthly*, 1962.

[20] M. Köppen, «On the training of a kolmogorov network,» en *International Conference on Artificial Neural Networks*, Springer, 2002, págs. 474-479.

[21] T. Hedberg, «The Kolmogorov Superposition Theorem, Appendix II in Topics in Approximation Theory, HS Shapiro.,» *Lecture notes in Math*, vol. 187, págs. 267-275, 1971.

[22] J.-P. Kahane, «Sur le Theoreme de Superposition de Kolmogorov,» 1975.

Informaticae Abstracta. ISSN: 3061-8533
Año 2025, Vol. 3, No. 2, pp. 25-43
JUL | DIC 2025
https://doi.org/10.36677/ia.v3n2.27474

De la Filosofía del Derecho a la Computación: Un Enfoque Computacional Multi-Agente Inspirado en la Teoría Pura de Kelsen

Vidal-López, R.

Información del Artículo

Recibido: 12 de septiembre, 2025
Aceptado: 15 de noviembre, 2025

Palabras clave: Sistemas multi-agente, Arquitectura BDI, Razonamiento jurídico, Teoría Pura del Derecho, Lógica híbrida intuicionista

Resumen

La resolución de conflictos normativos en el ámbito jurídico requiere un análisis riguroso de normas, hechos y valores, así como la capacidad de adaptarse a interpretaciones cambiantes. Este artículo presenta un modelo computacional multi-agente basado en la arquitectura BDI (*Beliefs*, *Desires*, *Intentions*), que integra la Teoría Pura del Derecho de Hans Kelsen y elementos de lógica híbrida intuicionista para simular procesos de razonamiento jurídico. Implementado en el lenguaje lógico-funcional Mercury, el modelo permite que los agentes generen y ajusten argumentos frente a ataques normativos y cambios en sus creencias, replicando la dinámica adaptativa del razonamiento judicial. Se emplea el caso clásico *Pierson v. Post* como estudio preliminar para ilustrar el comportamiento del sistema. Esta propuesta, de carácter descriptivo e interdisciplinario, contribuye al desarrollo de sistemas inteligentes aplicados al derecho y abre nuevas vías para el análisis computacional de decisiones normativas complejas.

1. Introducción

El avance de la inteligencia artificial y su creciente aplicación en el ámbito jurídico han abierto nuevas oportunidades para desarrollar sistemas que apoyen la toma de decisiones en escenarios legales complejos [1], [2]. Se han propuesto diversos modelos de razonamiento legal, como los sistemas basados en casos [2] y los enfoques basados en lógica deóntica para representar sistemas normativos [3]. Aunque estos modelos han demostrado eficacia en la representación de normas jurídicas, presentan limitaciones al momento de simular interacciones dinámicas entre actores jurídicos con intereses y perspectivas divergentes.

Una alternativa prometedora para superar estas limitaciones es el uso de sistemas multi-agente, que permiten modelar actores como jueces, abogados y partes litigantes en escenarios jurídicos simulados [4]. Al integrarse con modelos normativos basados en lógica y teoría del derecho, estos agentes pueden representar normas, valores y principios de justicia de forma más dinámica [5]. En particular, la arquitectura BDI (Beliefs, Desires, Intentions) ofrece una base sólida para simular cómo estos actores toman decisiones y generan argumentos.

En respuesta a este panorama, este trabajo propone un modelo computacional que combina la arquitectura BDI con la Teoría Pura del Derecho de Hans Kelsen, apoyado en la lógica híbrida intuicionista (*IHL*) para manejar incertidumbre y estados intermedios en el razonamiento jurídico [6].

Como estudio de caso se emplea *Pierson v. Post* [7], un caso clásico de derecho de propiedad que plantea un conflicto normativo entre persecución y posesión. La controversia —si la mera persecución de un animal salvaje confiere derechos de propiedad— ejemplifica tensiones entre reglas legales formales y principios de equidad. Esto lo convierte en un escenario ideal para probar el modelo.

La simulación se centra en el agente *Tomkins*, quien actúa como juez y es capaz de generar argumentos normativos en función de sus creencias y deseos. Aunque se planea incorporar otros agentes (*Livingston*, *Banks*, *Tropolle*), esta fase inicial se enfoca en validar el sistema para un solo agente, dejando para etapas futuras la automatización de ataques lógicos entre múltiples actores. Por ahora, tales interacciones se simulan manualmente.

El modelo se implementó en el lenguaje lógico-funcional Mercury [8], [9], elegido por su capacidad de manejar estructuras lógicas complejas con eficiencia y precisión.

Este artículo tiene como objetivo describir esta implementación inicial y explorar su potencial para simular juicios legales desde una perspectiva normativa. La estructura del artículo se organiza como sigue: primero se presenta el marco teórico; luego, la metodología; posteriormente, los resultados preliminares; y finalmente, las conclusiones y líneas futuras de investigación.

2. Marco Teórico

El presente trabajo propone un modelo computacional para la resolución de disputas legales que se fundamenta en una combinación de enfoques teóricos provenientes del derecho, la inteligencia artificial y las ciencias de la computación. Este modelo, se apoya en varias teorías clave, incluyendo la **Teoría Pura del Derecho** de Hans Kelsen, los agentes BDI (*Belief-Desire-Intention*), la lógica híbrida intuicionista (*IHL*), el uso de ontologías basadas en la teoría de topos y el lenguaje de comunicación entre agentes ACL (*Agent Communication Language*) de FIPA. Estos conceptos proporcionan una base sólida para el desarrollo de un sistema multiagente que simula la toma de decisiones en un entorno legal, como *Pierson v Post*. Aunque actualmente se ha implementado solo para el agente *Tomkins*, este marco es escalable a múltiples agentes en simulaciones futuras.

El fundamento legal del modelo se basa en la **Teoría Pura del Derecho** de Hans Kelsen, que describe el derecho como un sistema normativo autónomo y jerárquico, separado de influencias externas como la moral y la política [10]. Esta teoría se basa en dos principios centrales: la existencia de una *norma fundamental* (*Grundnorm*) y la jerarquía normativa, ambos esenciales para estructurar sistemas normativos tanto en contextos legales como computacionales. La *Grundnorm* es asumida como válida y no requiere justificación adicional, sirviendo como el punto de partida lógico para todo el sistema jurídico. Formalmente, esta jerarquía puede representarse como:

$$\textit{Grundnorm} = n_0$$

Las normas derivadas $(n_1, n_2, \ldots, n_k)$ están subordinadas jerárquicamente, con su validez dependiendo de una norma superior. Esto puede representarse mediante un grafo dirigido acíclico $G = (N, A)$, donde:

- N es el conjunto de normas ($n_i \in N$).
- A es el conjunto de relaciones de subordinación ($a_{ij} \in A \implies n_i \to n_j$).

La función de validación de una norma puede definirse como:

$$\textit{validarnorma}(n_j) = \begin{cases} \text{Verdadero}, & \text{si } n_j \text{ es derivada de } n_i \text{ y } \textit{validarnorma}(n_i), \\ \text{Falso}, & \text{en otro caso.} \end{cases}$$

Este enfoque asegura que la validez de cada norma dentro del sistema pueda ser trazada hacia la *Grundnorm*, garantizando coherencia lógica y normatividad en el sistema.

La perspectiva [11] de Kelsen, que define el análisis jurídico en términos de validez normativa sin implicar valores morales, es ideal para un modelo computacional, ya que permite formalizar las normas jurídicas en un marco lógico riguroso y preciso. En esta implementación inicial, el modelo propuesto utiliza la jerarquía normativa de Kelsen para representar normas como nodos en un grafo dirigido. Los agentes computacionales verifican la validez normativa a través de una función recursiva basada en la *validarnorma*. Por ejemplo, si el agente A_i necesita decidir si una norma n_j es válida, ejecuta:

$$\textit{validarnorma}(n_j) \implies \exists n_i \text{ tal que } (n_i, n_j) \in A$$

En términos computacionales, esto se implementa mediante recorridos en profundidad (*Depth-First Search*, **DFS**) sobre el grafo G, asegurando que todas las relaciones normativas sean verificadas [12]. Este formalismo permite representar no solo la jerarquía, sino también dinámicas de adaptación cuando se introducen nuevas normas o se invalidan existentes.

Por ejemplo, en el caso *Pierson v. Post*, el agente *Tomkins* utiliza la función *validarnorma* para determinar si las normas relacionadas con la propiedad sobre el zorro cumplen con los criterios establecidos por la *Grundnorm*. Esto asegura que sus decisiones sean consistentes y justificadas en términos normativos, manteniendo un enfoque objetivo dentro del marco jurídico predefinido.

La *Teoría Pura del Derecho* permite que las decisiones de los agentes sean trazables y explicables, aspectos cruciales en el contexto jurídico. Al eliminar ambigüedades y garantizar coherencia normativa, esta teoría se convierte en una herramienta clave para superar las limitaciones de modelos tradicionales que carecen de transparencia y justificabilidad, como los basados en aprendizaje automático [13]. En este trabajo, la *Teoría Pura del Derecho* de Kelsen proporciona los fundamentos normativos necesarios para el desarrollo de un modelo computacional adaptativo y preciso, ejemplificado en el agente *Tomkins*.

En cuanto a la arquitectura de los agentes, el modelo sigue el enfoque BDI (*Belief-Desire-Intention*), ampliamente utilizado en el desarrollo de agentes autónomos en sistemas multiagente [4]. Este marco conceptual permite modelar el comportamiento adaptativo de los agentes, simulando procesos de toma de decisiones racionales mediante tres componentes clave: *creencias* (B), que representan el conocimiento o percepción del agente sobre el estado del mundo; *deseos* (D), que son las metas que el agente aspira a alcanzar; y *intenciones* (I), que corresponden a los compromisos del agente para realizar acciones dirigidas al logro de esos deseos.

Formalmente, un agente BDI puede representarse como una tupla:

$$\text{Agente} = (B, D, I),$$

donde:

- B es el conjunto de **creencias**, que incluye información sobre el entorno y otros agentes.
- D es el conjunto de **deseos**, o metas deseadas, aunque no necesariamente factibles.
- I es el conjunto de **intenciones**, que representa metas seleccionadas para guiar las acciones del agente.

El comportamiento del agente se modela mediante la relación:

$$(B, D) \rightarrow I,$$

donde las creencias B y los deseos D influyen en la generación de intenciones I a través de reglas normativas o estratégicas. Este proceso es iterativo y dinámico, ya que las intenciones se adaptan continuamente en función de las observaciones del entorno y los resultados de acciones previas. Por ejemplo, el agente actualiza sus creencias mediante:

$$B' = B \cup \textit{observaciones},$$

donde B' representa las creencias actualizadas tras incorporar nueva información del entorno.

En este modelo, el agente *Tomkins* actúa como un actor jurídico en el caso *Pierson v. Post*, empleando sus creencias B sobre los hechos del caso (por ejemplo, *"Post perseguía al zorro"* o *"Pierson mató al zorro"*), sus deseos D (como *"resolver el caso"* o *"promover el bienestar público"*), y generando intenciones I como *"evaluar los derechos de propiedad"* o *"emitir un juicio basado en normas"*.

Esta arquitectura es particularmente adecuada para el contexto jurídico porque:

- **Racionalidad práctica:** Permite modelar cómo los agentes toman decisiones basadas en sus creencias y metas normativas, lo que refleja razonamientos humanos en escenarios legales [14].
- **Adaptabilidad:** Los agentes pueden ajustar sus decisiones dinámicamente a medida que se incorporan nuevas evidencias o cambian las condiciones del entorno.
- **Transparencia:** La estructura explícita de B, D e I facilita la trazabilidad y justificación de las decisiones, lo cual es crucial en sistemas normativos.

El uso de BDI en este trabajo permite generar argumentos adaptativos que simulan procesos de razonamiento jurídico. Por ejemplo, en el caso *Pierson v. Post*, *Tomkins* analiza las creencias iniciales sobre la posesión del zorro, evalúa escenarios posibles basados en sus deseos y produce decisiones normativas consistentes con los principios del derecho. Este enfoque demuestra la capacidad del modelo para manejar conflictos normativos en sistemas complejos y dinámicos.

Para manejar la incertidumbre en las creencias y los hechos disponibles, el modelo incorpora la **Lógica Híbrida Intuicionista (IHL)**, una extensión de la lógica modal que permite representar estados intermedios de conocimiento y creencias en un entorno dinámico. La lógica intuicionista se distingue de la lógica clásica al rechazar el principio del tercero excluido,

admitiendo que una proposición p no siempre es verdadera (p) o falsa (¬p), sino que puede encontrarse en un estado de conocimiento incompleto [6]. Además, la lógica modal, base de la IHL, introduce los operadores $\Diamond$ y $\Box$ para razonar sobre **posibilidades** y **necesidades**, respectivamente. El operador $\Diamond p$ indica que la proposición p es posible, mientras que $\Box p$ expresa que p es necesariamente verdadera. Este marco lógico es particularmente adecuado para el sistema legal, donde la información a menudo es incompleta, incierta o está sujeta a cambios.

Formalmente, un modelo de lógica híbrida intuicionista puede representarse como una tupla:

$$M = (W, R, V, @_w),$$

donde:

- W: Conjunto de **mundos posibles**.
- R: Relación de **accesibilidad** entre los mundos, que define cómo se conectan los distintos estados de conocimiento.
- V: Función de **valoración**, que asigna a cada proposición el conjunto de mundos donde es verdadera.
- $@_w$: Operador que permite referirse a proposiciones en mundos específicos, facilitando el razonamiento sobre escenarios pasados, alternativos o hipotéticos.

La semántica de sus principales operadores se describe como sigue:

- $\Box p$: p es verdadera en un mundo w si lo es en todos los mundos accesibles desde w:

$$M, w \models \Box p \iff \forall w'(R(w, w') \Rightarrow M, w' \models p).$$

- $\Diamond p$: p es verdadera en un mundo w si lo es en al menos un mundo accesible desde w:

$$M, w \models \Diamond p \iff \exists w'(R(w, w') \wedge M, w' \models p).$$

- $@_w p$: p es verdadera en un mundo específico w, lo que permite razonar sobre evidencia particular en ese mundo:

$$M, w \models @_v p \iff M, v \models p, \text{ donde } v \in W.$$

En el modelo propuesto, la IHL permite que los agentes, como *Tomkins*, gestionen estados intermedios de conocimiento y adapten sus creencias y acciones conforme surge nueva información. Por ejemplo, en el caso *Pierson v. Post*, el agente puede enfrentar un estado de conocimiento incompleto respecto a la proposición *"Post poseía al zorro"*, representado como:

$$\neg(\Box p \vee \neg\Box p),$$

indicando que la verdad de p no puede determinarse completamente en el estado actual.

La IHL también proporciona herramientas para razonar sobre escenarios futuros y evaluar la necesidad o posibilidad de ciertos hechos normativos. Por ejemplo:

- **Posibilidad futura:** El agente puede evaluar si es posible que un derecho de propiedad sea reconocido:
$$\Diamond(\textit{Pierson adquiere la propiedad del zorro}).$$
- **Necesidad normativa:** Puede determinar si es necesario un principio para la resolución del conflicto:
$$\Box(\textit{la captura física establece la propiedad}).$$

Este enfoque permite que los agentes adapten sus razonamientos legales de manera continua y justificada, manteniendo coherencia con los principios normativos subyacentes al sistema jurídico. La incorporación de la IHL no solo mejora la capacidad del modelo para manejar incertidumbre, sino que también enriquece el análisis de escenarios hipotéticos, fortaleciendo la trazabilidad y justificabilidad de las decisiones tomadas por los agentes.

El modelo también incorpora **ontologías basadas en la teoría de topos** para estructurar los conceptos legales como posesión, propiedad y normativa. La **teoría de topos**, desarrollada por Grothendieck y Lawvere [15], surge como una generalización de la teoría de conjuntos y ofrece un marco matemático que encapsula tanto relaciones lógicas como estructuras geométricas. Un *topos* (significa "lugar" o "posición") es una categoría $\mathcal{T}$ que posee propiedades que lo hacen un modelo unificado para representar conceptos abstractos y relaciones complejas.

Formalmente, un *topos* se define como una tupla:

$$\mathcal{T} = (\mathrm{Obj}(\mathcal{T}), \mathrm{Mor}(\mathcal{T}), \Omega),$$

donde:

- $\mathrm{Obj}(\mathcal{T})$: Conjunto de **objetos**, que representan las entidades abstractas en el dominio, como los conceptos legales de posesión y propiedad.
- $\mathrm{Mor}(\mathcal{T})$: Conjunto de **morfismos**, que denotan relaciones entre objetos, representadas como $f : A \to B$, donde $A, B \in \mathrm{Obj}(\mathcal{T})$.
- Ω: El **clasificador de subobjetos**, que actúa como un dominio lógico dentro del topos, permitiendo interpretar proposiciones sobre los objetos.

En este marco, el clasificador de subobjetos Ω permite modelar proposiciones como morfismos. Por ejemplo, un subobjeto de A, que puede interpretarse como un subconjunto en la teoría de conjuntos, es un morfismo $f : A \to \Omega$, donde Ω codifica las verdades lógicas del sistema.

Para una proposición P, su evaluación en un objeto A se define como:

$$P(A) = \{x \in A \mid P(x) \text{ es verdadero en } \Omega\}.$$

En el contexto de este trabajo, las **ontologías basadas en la teoría de topos** proporcionan un marco riguroso para estructurar relaciones entre actores legales (como demandantes o jueces) y conceptos jurídicos (como posesión y propiedad). Por ejemplo, las relaciones entre posesión y propiedad pueden representarse como morfismos en un topos:

$$\text{Posesión}(D, O) \rightarrow \text{Propiedad}(D, O),$$

donde:

- Posesión(D, O): Representa la posesión efectiva del objeto O por el actor D.
- Propiedad(D, O): Denota la propiedad legal del objeto O por D.
- Los morfismos codifican transiciones legales, como transferencias de derechos o disputas sobre la validez de la posesión.

El uso de esta teoría permite extender el modelo a entornos más complejos, donde múltiples agentes interactúan y las relaciones legales entre ellos pueden representarse de manera precisa y adaptable. Además, la estructura jerárquica y lógica de los topos refleja de forma natural la organización normativa descrita en la *Teoría Pura del Derecho* de Kelsen, lo que facilita la coherencia normativa en los sistemas multiagente.

Este marco extensible es particularmente útil en simulaciones jurídicas complejas, como el caso *Pierson v. Post*, donde conceptos como la posesión y la propiedad pueden evaluarse dinámicamente dentro de un sistema normativo. Por ejemplo, el clasificador Ω puede utilizarse para verificar si un subobjeto, como la posesión efectiva, cumple con los criterios normativos requeridos para derivar una propiedad legal. Esto proporciona no solo una representación formal de las relaciones legales, sino también una base lógica para razonar sobre ellas.

Finalmente, se utiliza el **Lenguaje de Comunicación entre Agentes (ACL)** de *FIPA* para facilitar la interacción y coordinación en sistemas multi-agente distribuidos. Aunque actualmente el agente *Tomkins* no interactúa con otros agentes, el ACL sería esencial en el modelo ampliado para permitir la comunicación entre agentes BDI que compartan o negocien creencias, deseos e intenciones, como en el caso de una simulación jurídica completa con múltiples actores [16]. Estos agentes podrían utilizar el ACL para realizar actos de lenguaje como informar, proponer o negociar, lo que resulta fundamental para simular procesos como la negociación y la deliberación entre las partes en conflicto [17]. Los conceptos de actos de habla fueron extensamente desarrollados por John R. Searle, quien en su obra *"Speech Acts: An Essay in the Philosophy of Language"* [18], exploró la capacidad de las palabras para realizar acciones en lugar de solo transmitir información [19].

La semántica de ACL se puede representar utilizando operadores modales de necesidad ($\Box$) y posibilidad ($\Diamond$), que permiten expresar propiedades clave de los actos comunicativos. Por ejemplo, el acto de *informar* implica que el agente emisor a:

- Cree necesariamente ($\Box$) que el contenido de su mensaje ϕ es verdadero.
- Considera posible ($\Diamond$) que el agente receptor b pueda incorporar ϕ a su conjunto de conocimiento.

Formalmente, el acto de *informar* se puede modelar como:

$$\text{informar}(a, b, \phi) \iff \Box\phi \wedge \Diamond(\text{Sabe}(b, \phi)),$$

donde:

- α es el agente emisor.
- b es el agente receptor.
- ϕ es la proposición transmitida.
- $\Box\phi$ representa que α necesariamente cree que ϕ es verdadera.
- $\Diamond(\text{Sabe}(b, \phi))$ indica que α considera posible que b llegue a saber que ϕ.

Este uso de los operadores modales proporciona una base lógica sencilla pero robusta para modelar la semántica de ACL, enfocándose en los aspectos de certeza y posibilidad en los actos de comunicación entre agentes. Además, los operadores modales pueden extenderse para capturar dinámicas más complejas de interacción en futuras implementaciones del modelo.

En este modelo, la comunicación entre agentes no es solo un intercambio de datos, sino un acto racional basado en sus estados internos (creencias, deseos, intenciones) y en la dinámica de su entorno.

Aunque actualmente el agente *Tomkins* no interactúa con otros agentes, el ACL sería fundamental en el modelo ampliado para habilitar una comunicación eficaz en simulaciones jurídicas más complejas con múltiples actores. Por ejemplo, agentes BDI podrían utilizar el ACL para negociar acuerdos, resolver conflictos o coordinar estrategias legales, realizando actos de lenguaje como *proponer*, *rechazar* o *aceptar*. Esto no solo facilitaría la colaboración entre agentes, sino que también permitiría modelar dinámicamente procesos como la deliberación y la negociación jurídica [17].

La implementación de ACL en el modelo proporciona un marco robusto y estandarizado para la interacción multi-agente, asegurando interoperabilidad y extensibilidad para futuras investigaciones.

Este conjunto de teorías y herramientas permite construir un modelo escalable y adaptable para simular decisiones jurídicas basadas en normas y principios, como se ha comenzado a implementar en el agente *Tomkins* para el caso *Pierson v Post*. El marco teórico descrito proporciona un soporte sólido para futuras expansiones que permitirían una simulación detallada de interacciones entre agentes en un juicio legal completo.

3. Metodología

Esta sección describe el proceso de implementación del modelo de agentes BDI utilizando el lenguaje de programación *Mercury*, un lenguaje lógico-funcional especialmente adecuado para modelar sistemas normativos. Su sistema de tipos fuerte, inferencia de modos y optimización en tiempo de compilación contribuyen a una implementación precisa y coherente, aspectos esenciales al simular el comportamiento de actores jurídicos en contextos normativos.

El modelo desarrollado emplea una arquitectura BDI (*Belief-Desire-Intention*) para representar de manera estructurada las creencias, deseos e intenciones de los agentes involucrados en el caso *Pierson v. Post*. En esta etapa inicial, la implementación se centró exclusivamente

en el agente *Tomkins*, quien desempeña el rol de juez. Aunque el objetivo final es simular el juicio completo con cuatro agentes (Tomkins, Livingston, Banks y Tropolle), esta fase se enfocó en construir una base sólida para su posterior expansión.

Una característica fundamental de *Mercury* que fortalece la robustez del modelo es su sistema de tipos estático, que permite definir estructuras de datos complejas y garantizar relaciones internas consistentes. Este control desde la fase de compilación previene errores de tipo y asegura integridad lógica en entornos sensibles como el jurídico. La definición del tipo *agent*, mostrado a continuación, encapsula los componentes esenciales de un agente BDI.

```
:- type agent --->
    agent(
        beliefs :: list(belief),
        desires :: list(desire),
        arguments :: list(argument)
    ).
```

Listing 1: Definición del tipo *agent*.

Esta estructura permite integrar de forma cohesionada los elementos internos del agente y gestionarlos con precisión. La capacidad de Mercury para trabajar con tipos algebraicos facilita una representación concisa pero detallada de cada componente del estado mental del agente, lo cual es indispensable en la simulación de razonamiento jurídico.

El agente *Tomkins* es inicializado con un conjunto de *beliefs*, *desires* e *intentions*, definidos como tipos algebraicos. En el siguiente fragmento, se muestran las proposiciones relevantes al caso y la forma en que se modelan las creencias.

```
% Definicion de proposiciones y creencias.
:- type proposition --->
    f1; % Post estaba persiguiendo al zorro.
    f2; % Post no habia capturado ni herido al zorro.
    f3; % Pierson mató al zorro para arruinar el deporte de Post.
    f4; % Los zorros matan ganado.
    f5; % Fomentar la caza reducirá el número de zorros.
    f6; % Reducir el número de zorros protege el ganado.
    f7. % Si la caza se desalienta, no se inflige sufrimiento innecesario.

% Definicion de las creencias del agente.
:- type belief ---> belief(proposition); disbelief(proposition); unknown(proposition).
```

Listing 2: Definición de tipos para representar proposiciones y creencias.

Este modelo permite que Tomkins adopte posturas afirmativas, negativas o neutrales sobre cada proposición, lo que refleja la ambigüedad y complejidad de los entornos legales. La fuerte tipificación impide que se representen estados inválidos, asegurando que la simulación se mantenga dentro de parámetros normativos coherentes.

Los deseos del agente modelan los objetivos que este aspira a lograr, y las intenciones representan los planes elegidos para cumplir dichos objetivos. Ambos se definen también como tipos algebraicos. A continuación se presentan los valores, condiciones y estructuras que permiten expresar estos elementos.

```
% Condiciones para los deseos.
:- type condition ---> ownership_plaintiff;
    no_ownership_defendant; malicious_intent_plaintiff.

% Valores que el agente busca maximizar.
```

```
:- type value ---> less_litigation;
    public_benefit; economic_benefit.

% Deseos que vinculan un nombre, un valor y una condición.
:- type desire ---> desire(name, value, condition).

% Argumento compuesto por creencias, acción, metas y valores.
:- type argument ---> argument(
    beliefs::list(belief), action::action,
    goals::list(proposition), values::list(value)
).
```

Listing 3: Definición de deseos y argumentos.

Esta representación permite al agente alinear sus objetivos con valores normativos como el beneficio público o la reducción de litigios. La semántica declarativa de Mercury favorece una implementación clara de estos componentes.

Uno de los aspectos clave del modelo es la generación automática de argumentos que respalden las intenciones del agente. Cada argumento propuesto es evaluado en función de su compatibilidad con las creencias y deseos del agente. El siguiente predicado decide si un argumento puede integrarse al conjunto de intenciones del agente:

```
:- pred should_add_argument(agent::in, argument::in) is semidet.
should_add_argument(Tomkins, Arg) :-
    tomkins_belief_in_argument(Tomkins^beliefs, Arg),
    tomkins_desire_value_in_argument(Tomkins^desires, Arg).
```

Listing 4: Predicado para evaluar un argumento propuesto.

Si el argumento es coherente con el estado mental del agente, se incorpora a su conjunto de intenciones, lo cual ilustra el proceso de razonamiento interno basado en metas y consistencia lógica [4].

La siguiente función muestra cómo se filtran los argumentos válidos y se actualiza el estado del agente:

```
:- func add_arguments_to_tomkins(agent) = agent.
add_arguments_to_tomkins(Tomkins) = TomkinsUpdated :-
    Arguments = [arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10],
    ValidArguments = list.filter(should_add_argument(Tomkins), Arguments),
    TomkinsUpdated = Tomkins^arguments := ValidArguments.
```

Listing 5: Función para actualizar el conjunto de argumentos del agente.

Este mecanismo permite al agente modificar dinámicamente su razonamiento conforme cambian sus creencias o prioridades, lo que simula con precisión la actividad judicial frente a evidencia o argumentos en disputa.

En esta etapa, los ataques a las creencias del agente se simulan manualmente, pero ya permiten observar cómo dichos desafíos alteran la validez de sus argumentos. Esta capacidad de adaptación es crucial para representar la dinámica del razonamiento en escenarios legales complejos.

En resumen, la implementación en *Mercury* del modelo BDI proporciona una estructura lógica y tipada que permite representar con precisión la toma de decisiones en contextos jurídicos. La metodología empleada garantiza coherencia, extensibilidad y claridad, estableciendo una base sólida para incorporar agentes adicionales y mecanismos automatizados de ataque y defensa argumentativa en futuras etapas del proyecto.

4. Resultados

El modelo propuesto, basado en una arquitectura BDI (Belief-Desire-Intention) e implementado en el lenguaje *Mercury*, demuestra cómo es posible representar procesos de decisión jurídica mediante un enfoque normativo y racional. Para validar su funcionamiento, se utilizó el caso clásico *Pierson v. Post*, simulando cómo un agente, en este caso *Tomkins* (juez), toma decisiones a partir de sus creencias, deseos e intenciones.

Este enfoque se inspira en la Teoría Pura del Derecho de Hans Kelsen, lo que implica que las decisiones del agente se guían por la validez formal de las normas, sin recurrir a factores morales o subjetivos externos. El agente opera dentro de un marco normativo bien definido, lo cual garantiza decisiones objetivas y legalmente coherentes.

El siguiente fragmento de código muestra la inicialización del agente *Tomkins* con un conjunto de creencias y deseos normativos:

```
Tomkins = agent(
    [belief(f1), belief(f2), belief(f3), belief(f4), unknown(f5), belief(f6)],
    [desire(clear_law, less_litigation, ownership_plaintiff),
     desire(unclear_law, less_litigation, ownership_no_possession),
     desire(trade_restricted, economic_benefit,
         ↪ malicious_intent_productive_activity_defendant)],
    []
).
```

Listing 6: Inicialización del agente Tomkins con creencias y deseos.

Las creencias reflejan proposiciones del caso, como la persecución del zorro por parte de Post (`f1`) o la amenaza que representan los zorros para el ganado (`f4`). Los deseos de Tomkins están vinculados a objetivos jurídicos como reducir el litigio y promover beneficios públicos o económicos. La lógica híbrida intuicionista (IHL) permite representar estados de creencia inciertos, como `unknown(f5)`, lo cual añade flexibilidad al razonamiento legal del agente.

Una vez definido su estado mental inicial, Tomkins evalúa argumentos construidos a partir de creencias, acciones, metas y valores. A continuación, se presentan dos argumentos con enfoques contrapuestos:

```
:- func arg1 = argument.
arg1 = argument(
    [belief(f4), belief(f5), belief(f6)],
    encourage_fox_hunting,
    [f6],
    [public_benefit]
).

:- func arg2 = argument.
arg2 = argument(
    [belief(f7)],
    discourage_fox_hunting,
    [f7],
    [humaneness]
).
```

Listing 7: Argumentos evaluados por Tomkins.

El argumento `arg1` defiende la caza del zorro como forma de proteger al ganado, en consonancia con el valor del beneficio público. En contraste, `arg2` se basa en la compasión hacia

los animales, abogando por desalentar la caza. Esta dualidad permite observar cómo Tomkins selecciona argumentos según su alineación con creencias y valores normativos.

El siguiente paso es evaluar cuáles de estos argumentos son coherentes con el estado mental del agente:

```
:- pred should_add_argument(agent::in, argument::in) is semidet.
should_add_argument(Tomkins, Arg) :-
    tomkins_belief_in_argument(Tomkins^beliefs, Arg),
    tomkins_desire_value_in_argument(Tomkins^desires, Arg).
```

Listing 8: Evaluación de argumentos por el agente Tomkins.

Solo los argumentos que satisfacen las creencias y valores deseados serán integrados al razonamiento del agente:

```
:- func add_arguments_to_tomkins(agent) = agent.
add_arguments_to_tomkins(Tomkins) = TomkinsUpdated :-
    Arguments = [arg1, arg2, ..., arg10],
    ValidArguments = list.filter(should_add_argument(Tomkins), Arguments),
    TomkinsUpdated = Tomkins^arguments := ValidArguments.
```

Listing 9: Actualización de intenciones del agente.

Este mecanismo permite que el agente actualice dinámicamente sus intenciones de acuerdo con su marco normativo, lo que resulta esencial en contextos jurídicos donde nuevas pruebas o argumentos pueden surgir.

Además, el modelo permite simular ataques entre agentes. Estos ataques desafían creencias fundamentales, obligando a reevaluar los argumentos asociados. El siguiente ejemplo presenta un ataque contra la creencia `f1`:

```
:- func attack1a = attack.
attack1a = attack(
    [belief(f1)],
    [ownership_plaintiff],
    [disbelief(f1)]
).
```

Listing 10: Ataque contra la creencia f1.

El impacto de un ataque se procesa mediante un mecanismo de actualización de creencias:

```
:- pred apply_attack(agent::in, attack::in, agent::out) is det.
apply_attack(!Agent, Attack) :-
    Attack = attack(QuestionedBeliefs, Conditions, ArgumentBeliefs),
    list.foldl((pred(Belief::in, A0::in, A::out) is det :-
        (if belief_in_list(Belief, A0^beliefs) then
            (if list.member(desire(_, _, Condition), A0^desires) and
                list.member(Condition, Conditions) then
                    list.delete_all(A0^beliefs, Belief, TempBeliefs),
                    list.append(TempBeliefs, ArgumentBeliefs, NewBeliefs),
                    A = A0^beliefs := NewBeliefs
            else
                A = A0)
        else
            A = A0)
    ), QuestionedBeliefs, !Agent).
```

Listing 11: Aplicación del ataque al agente.

Una vez modificado el sistema de creencias, el agente puede reevaluar los argumentos disponibles y actualizar sus intenciones:

```
TomkinsUpdated = add_arguments_to_tomkins(TomkinsUpdatedAttack),
TomkinsReevaluatedArguments = list.map(argument_name, TomkinsUpdated^arguments)
io.print("= Nuevas intenciones del agente Tomkins: =\n"),
io.print(TomkinsReevaluatedArguments, !IO).
```

Listing 12: Reevaluación de intenciones tras ataque.

Este flujo simula el razonamiento jurídico dinámico, donde un argumento previamente válido puede perder fuerza al cambiar el contexto normativo del agente. Por ejemplo, si Tomkins ya no cree en la posesión por persecución, dejará de apoyar la asignación de propiedad al demandante.

El modelo también ha sido probado con simulaciones adicionales, incorporando otros agentes como *Banks* y *Anthony*, quienes aportan distintas perspectivas jurídicas. Estas simulaciones mostraron cómo el sistema se adapta a diferentes valores normativos y contextos, ofreciendo decisiones coherentes pero variables, según la configuración interna de cada agente.

En conjunto, los resultados muestran que la arquitectura BDI implementada en *Mercury* es capaz de representar razonamientos legales complejos, incluyendo el análisis de argumentos, la adaptación frente a ataques y la toma de decisiones en entornos normativos cambiantes. El uso de lógica formal, tipos algebraicos y razonamiento simbólico proporciona un marco sólido y escalable para el estudio computacional del derecho.

Además, la capacidad de modificar dinámicamente las creencias permite simular contextos legales realistas, donde la información puede ser ambigua, contradictoria o incompleta. Esto no solo resulta útil desde una perspectiva académica, sino que abre caminos hacia el desarrollo de herramientas automatizadas para análisis jurídico, enseñanza del derecho y diseño de sistemas normativos artificiales.

5. Discusión

La resolución computacional de disputas jurídicas sigue siendo un desafío para la inteligencia artificial, debido a la naturaleza dinámica, argumentativa y normativamente compleja del derecho. En este contexto, el modelo propuesto basado en agentes BDI implementado en *Mercury* representa un avance significativo por su capacidad para simular razonamientos jurídicos transparentes, adaptables y teóricamente fundamentados.

A diferencia de enfoques basados en aprendizaje automático como Lex Machina, Kira Systems o CaseMine, que se centran en el análisis de grandes volúmenes de datos mediante técnicas estadísticas, nuestro modelo no opera como una "caja negra". Por el contrario, cada decisión del agente es rastreable y justificable, ya que se construye explícitamente sobre estructuras normativas y principios argumentativos definidos. Esta trazabilidad es crucial en entornos jurídicos, donde la legitimidad de una decisión depende tanto de su contenido como de su justificación.

En comparación con marcos argumentativos formales como Carneades o Reason-Based Logic (RBL), que han sido fundamentales en el análisis lógico de argumentos jurídicos, nuestro enfoque introduce un mayor grado de adaptabilidad. El agente Tomkins no solo genera

argumentos alineados con sus creencias y deseos, sino que puede reevaluarlos activamente ante ataques y cambios de contexto. Esto permite representar la evolución del razonamiento jurídico a lo largo de un juicio, algo que los modelos con creencias estáticas difícilmente logran simular con fidelidad.

Una de las contribuciones más destacadas del modelo es su capacidad para gestionar conflictos normativos y estados de incertidumbre mediante la integración de lógica híbrida intuicionista (IHL). Este enfoque, inspirado en trabajos como los de Shams [20] y Governatori [21], permite representar creencias intermedias o desconocidas sin comprometer la coherencia del sistema. Así, los agentes pueden operar en entornos donde las normas son ambiguas, contradictorias o en constante cambio, reflejando mejor la realidad del derecho contemporáneo.

Pese a sus ventajas, el modelo presenta desafíos en términos de escalabilidad. A medida que se incorporan más agentes, normas y creencias, la complejidad computacional crece. Aunque *Mercury* ofrece un rendimiento optimizado para programación lógica-funcional, serán necesarias estrategias de optimización para extender el modelo a escenarios más complejos. Además, la incorporación de normas dinámicas, como sugiere García [22], permitiría una representación más granular de los sistemas jurídicos donde las reglas evolucionan con el contexto.

La simulación del caso *Pierson v. Post* demuestra la aplicabilidad del modelo a un escenario clásico. Sin embargo, para ampliar su utilidad, sería recomendable desarrollar una biblioteca de casos con distintas estructuras normativas y valorativas. Esto permitiría explorar cómo los agentes ajustan sus estrategias frente a precedentes, doctrinas o valores diversos, abriendo posibilidades para una inteligencia artificial jurídica más contextual y versátil.

En definitiva, el valor del modelo no reside únicamente en su implementación técnica. La verdadera aportación está en su arquitectura conceptual: una síntesis entre la Teoría Pura del Derecho de Kelsen, la lógica formal y la teoría de agentes BDI. Esta combinación permite capturar la dimensión normativa del derecho, su estructura argumentativa y su capacidad de adaptación. Mientras que cualquier programador podría replicar el modelo en *Mercury*, su diseño refleja una comprensión profunda de los desafíos filosóficos y estructurales del razonamiento jurídico. Esta integración interdisciplinaria ofrece una vía prometedora para el desarrollo de sistemas jurídicos artificiales más transparentes, adaptables y normativamente coherentes.

6. Conclusiones

Este trabajo ha presentado un modelo computacional innovador para la resolución de disputas legales, integrando la Teoría Pura del Derecho de Kelsen, agentes *BDI*, lógica híbrida intuicionista (*IHL*) y una ontología basada en *topoi*. La implementación en *Mercury* ha permitido simular cómo un agente jurídico puede generar, revisar y adaptar argumentos normativos en función de sus creencias y deseos, dentro de un marco estructurado y justificable.

A través del estudio del caso *Pierson v. Post*, se demuestra que el modelo no solo es funcional, sino que ofrece una aproximación transparente y dinámica al razonamiento jurídico, capaz de responder a ataques y transformaciones en el entorno normativo. Esta capacidad

adaptativa representa una diferencia clave frente a enfoques más estáticos o automatizados que no explicitan la lógica de sus decisiones.

Más allá de su valor técnico, el modelo destaca por su aporte conceptual: propone una forma de representar computacionalmente el derecho sin sacrificar su estructura normativa ni su dimensión argumentativa. Esto lo convierte en una herramienta con potencial no solo en simulaciones jurídicas, sino también en educación legal, diseño institucional y análisis normativo en contextos complejos.

Aunque la implementación actual se limita a un solo agente, se sienta una base sólida para extender el modelo a escenarios multiagente y casos más diversos. La posibilidad de incorporar dinámicas complejas, contradicciones normativas y valores divergentes abre un camino fértil para futuras investigaciones interdisciplinarias.

En este sentido, los investigadores, desarrolladores y teóricos del derecho pueden considerar este enfoque como una plataforma para modelar, experimentar y entender los procesos normativos de forma más rica y formalizada. La IA aplicada al derecho no debe limitarse a predecir resultados, sino que puede ayudarnos a reflexionar sobre cómo y por qué se toman ciertas decisiones legales. Este trabajo es un paso en esa dirección.

Referencias

[1] T. Bench-Capon y G. Sartor, «A model of legal reasoning with cases incorporating theories and values,» *Artificial Intelligence*, vol. 150, n.º 1, págs. 97-143, 2003, ISSN: 0004-3702.

[2] K. D. Ashley, *Tutorial on AI and Legal Reasoning*, 2005.

[3] J.-J. C. Meyer, R. J. Wieringa et al., «Deontic logic in computer science normative system specification,» *HeinOnline*, vol. 3, pág. 93, 1993.

[4] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[5] C. S. W. V. Andrés García-Camino Juan A. Rodríguez-Aguilar, «Constraint rule-based programming of norms for electronic institutions,» *Autonomous Agents and Multi-Agent Systems*, vol. 18, n.º 1, págs. 186-217, sep. de 2008, ISSN: 1573-7454.

[6] T. Braüner y V. de Paiva, «Intuitionistic hybrid logic,» *Journal of Applied Logic*, vol. 4, n.º 3, págs. 231-255, 2006, Methods for Modalities 3 (M4M-3), ISSN: 1570-8683. DOI: https://doi.org/10.1016/j.jal.2005.06.009 dirección: https://www.sciencedirect.com/science/article/pii/S1570868305000443

[7] N. Y. C. of Appeals, *Pierson v. Post, 3 Cai. R. 175 (N.Y. 1805)*, Disponible en https://www.nycourts.gov/reporter/archives/pierson_post.htm (última consulta: noviembre 2024), 1805.

[8] Z. Somogyi, F. Henderson y T. Conway, «The execution algorithm of mercury, an efficient purely declarative logic programming language,» *The Journal of Logic Programming*, vol. 29, n.º 1, págs. 17-64, 1996, High-Performance Implementations of Logic Programming Systems, ISSN: 0743-1066.

[9] F. Henderson et al., «The Mercury language reference manual,» 1996.

[10] M. S. Green, «Hans Kelsen and the Logic of Legal Systems,» *Alabama Law Review*, George Mason Law & Economics Research Paper No. 03-50, vol. 53, págs. 365-413, 2003.

[11] H. Alessa, «The role of Artificial Intelligence in Online Dispute Resolution: A brief and critical overview,» *Information & Communications Technology Law*, vol. 31, n.º 3, págs. 319-342, 2022. DOI: 10.1080/13600834.2022.2088060 eprint: https://doi.org/10.1080/13600834.2022.2088060. dirección: https://doi.org/10.1080/13600834.2022.2088060

[12] R. E. Tarjan y U. Zwick, «Finding strong components using depth-first search,» *European Journal of Combinatorics*, vol. 119, pág. 103 815, 2024, Dedicated to the memory of Pierre Rosenstiehl, ISSN: 0195-6698. DOI: https://doi.org/10.1016/j.ejc.2023.103815 dirección: https://www.sciencedirect.com/science/article/pii/S0195669823001324

[13] D. Restrepo Amariles y P. M. Baquero, «Promises and limits of law for a human-centric artificial intelligence,» *Computer Law & Security Review*, vol. 48, pág. 105 795, 2023, ISSN: 0267-3649.

[14] A. S. Rao y M. P. Georgeff, «BDI Agents: From Theory to Practice,» en *International Conference on Multiagent Systems*, 1995.

[15] G. E. Reyes, «A Topos-Theoretic Approach to Reference and Modality,» *Notre Dame Journal of Formal Logic*, vol. 32, n.º 3, págs. 359-391, 1991. DOI: 10.1305/ndjfl/1093635834

[16] IEEE Foundation for Intelligent Physical Agents (FIPA), *Design Process Documentation Template*, http://www.fipa.org/, IEEE FIPA DPDF Working Group, 2001.

[17] K. Greenwood, T. B. Capon y P. McBurney, «Towards a computational account of persuasion in law,» en *Proceedings of the 9th International Conference on Artificial Intelligence and Law*, ép. ICAIL '03, Scotland, United Kingdom: Association for Computing Machinery, 2003, págs. 22–31, ISBN: 1581137478.

[18] J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*. Cambridge: Cambridge University Press, 1969. dirección: https://www.cambridge.org/core/books/speech-acts/D2D7B03E472C8A390ED60B86E08640E7

[19] L. Villoro, *Creer, Saber y Conocer*, 2da. Edición, S. XXI, ed. Siglo XXI, 1989, ISBN: 9682316944.

[20] J. P. W. W. V. Zohreh Shams Marina De Vos, «Practical reasoning with norms for autonomous software agents,» *Engineering Applications of Artificial Intelligence*, vol. 65, págs. 388-399, 2017.

[21] G. Governatori y A. Rotolo, «BIO logical agents: Norms, beliefs, intentions in defeasible logic,» *Autonomous Agents and Multi-Agent Systems*, vol. 17, n.º 1, págs. 36-69, 2008.

[22] A. García-Camino, J. A. Rodríguez-Aguilar, C. Sierra y W. Vasconcelos, «Constraint rule-based programming of norms for electronic institutions,» *Autonomous agents and multi-agent systems*, vol. 18, n.º 1, págs. 186-217, 2009.

7. Aplicación del modelo del caso *Pierson v. Post* en Mercury

```
:- module test1.
:- interface.
:- import_module io.

% Prototipos de funciones y tipos.
:- pred main(io::di, io::uo) is det.

:- implementation.
:- import_module list, bool, string.
:- import_module pair.

% Definición de proposiciones utilizadas en el caso legal.
:- type proposition --->
    f1;  % Post estaba en persecución del zorro.
    f2;  % Post no había capturado ni herido al zorro.
    f3;  % Pierson mató al zorro para arruinar el deporte de Post.
    f4;  % Los zorros matan ganado.
    f5;  % Fomentar la caza reducirá el número de zorros.
    f6;  % Reducir el número de zorros protege al ganado.
    f7.  % Si se desalienta la caza, no se inflige sufrimiento innecesario a los animales
        ↪ .

% Las creencias pueden ser positivas, negativas o desconocidas respecto a las
    ↪ proposiciones.
:- type belief --->
    belief(proposition);
    disbelief(proposition);
    unknown(proposition).

% Condiciones bajo las cuales los deseos son válidos.
:- type condition --->
    ownership_plaintiff;
    no_ownership_defendant;
    no_ownership_no_possession;
    ownership_no_possession;
    no_ownership_plaintiff;
    no_ownership_possession;
    malicious_intent_productive_activity_defendant;
    malicious_intent_plaintiff;
    malicious_intent_defendant;
    fewer_foxes_farmers_protected;
    no_fewer_foxes_no_farmers_protected;
    reduced_animal_suffering;
    no_reduced_animal_suffering;
    ownership_pursuit;
    no_pursuit_no_ownership.

% Valores que un agente puede intentar maximizar o mantener.
:- type value --->
    less_litigation;
    economic_benefit;
    public_benefit;
    humaneness.

% Nombres de deseos que reflejan estrategias o metas legales.
:- type name --->
    clear_law ;
    unclear_law ;
```

```
    trade_restricted ;
    malice_condemned ;
    malice_condoned ;
    less_threat_to_others ;
    more_threat_to_others ;
    more_suffering ;
    less_suffering .

% Acciones que un agente puede tomar basadas en el escenario legal.
:- type action --->
    encourage_fox_hunting ;
    discourage_fox_hunting ;
    find_ownership_established ;
    find_no_ownership ;
    find_for_plaintiff ;
    find_for_defendant ;
    not_find_for_defendant .

% Los deseos combinan nombres, valores y condiciones.
:- type desire --->
    desire(name, value, condition).

% Los argumentos son combinaciones de creencias, acciones, metas y valores.
:- type argument --->
    argument(
        beliefs :: list(belief),
        action :: action,
        goals :: list(proposition),
        values :: list(value)
    ).

% Los ataques están definidos para desafiar las creencias de los agentes.
:- type attack --->
    attack(
        question :: list(belief),
        conditions :: list(condition),
        argument :: list(belief)
    ).

% Estructura del agente actualizada.
:- type agent --->
    agent(
        beliefs :: list(belief),
        desires :: list(desire),
        arguments :: list(argument)
    ).

% Función para añadir argumentos válidos a un agente basándose en creencias y deseos.
:- func add_arguments_to_\textit{Tomkins}(agent) = agent.
add_arguments_to_\textit{Tomkins}(\textit{Tomkins}) = \textit{Tomkins}Updated :-
    Arguments = [arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10],
    ValidArguments = list.filter(should_add_argument(\textit{Tomkins}), Arguments),
    \textit{Tomkins}Updated = \textit{Tomkins}^arguments := ValidArguments.

% Punto de entrada para el programa Mercury.
main(!IO) :-
    % Configuración inicial del agente con creencias y deseos.
    \textit{Tomkins} = agent(
        [belief(f1), belief(f2), belief(f3), belief(f4), unknown(f5), belief(f6), unknown
            ↪ (f7)],
        [desire(clear_law, less_litigation, ownership_plaintiff), desire(unclear_law,
            ↪ less_litigation, ownership_no_possession), desire(trade_restricted,
            ↪ economic_benefit, malicious_intent_productive_activity_defendant)],
```

```
        []
    ),
    % Actualiza el agente con argumentos válidos y muestra los resultados.
    \textit{Tomkins}Updated = add_arguments_to_\textit{Tomkins}(\textit{Tomkins}),

    % Imprime las creencias, deseos y argumentos de \textit{Tomkins} después de la
        ↪ actualización.
    io.print("= Creencias del agente \textit{Tomkins}: =\n", !IO),
    io.print(\textit{Tomkins}Updated^beliefs, !IO),
    io.print("\n\n", !IO),

    io.print("= Deseos del agente \textit{Tomkins}: =\n", !IO),
    io.print(\textit{Tomkins}Updated^desires, !IO),
    io.print("\n\n", !IO),

    io.print("= Intenciones (argumentos) del agente \textit{Tomkins}: =\n", !IO),
    \textit{Tomkins}Arguments = list.map(argument_name, \textit{Tomkins}Updated^arguments
        ↪ ),
    io.print(\textit{Tomkins}Arguments, !IO),
    io.print("\n\n", !IO),

    % Aplica un ataque y reevalúa las creencias del agente.
    Attack1 = attack1a,
    apply_attack(\textit{Tomkins}Updated, Attack1, \textit{Tomkins}UpdatedAttack),

    % Reevalúa los argumentos después del ataque.
    \textit{Tomkins}Reevaluated = add_arguments_to_\textit{Tomkins}(\textit{Tomkins}
        ↪ UpdatedAttack),
    \textit{Tomkins}ReevaluatedArguments = list.map(argument_name, \textit{Tomkins}
        ↪ Reevaluated^arguments),

    % Imprime el estado de \textit{Tomkins} después de reevaluar el ataque.
    io.print("= Creencias de \textit{Tomkins} después de la reevaluación: =\n", !IO),
    io.print(\textit{Tomkins}Reevaluated^beliefs, !IO),
    io.print("\n\n", !IO),

    io.print("= Deseos de \textit{Tomkins} después de la reevaluación: =\n", !IO),
    io.print(\textit{Tomkins}Reevaluated^desires, !IO),
    io.print("\n\n", !IO),

    io.print("= Intenciones (argumentos) de \textit{Tomkins} después de la reevaluación:
        ↪ =\n", !IO),
    io.print(\textit{Tomkins}ReevaluatedArguments, !IO),
    io.print("\n\n", !IO).
```

Listing 13: Aplicación del modelo del caso *Pierson v. Post* en Mercury

Informaticae Abstracta. ISSN: 3061-8533
Año 2025, Vol. 3, No. 2, pp. 44-61
JUL | DIC 2025
https://doi.org/10.36677/ia.v3n2.27649

Software Libre para Solución de Líneas de Transmisión

Pérez Merlos, J.C.; Salgado Gallegos, M.; Albarrán Trujillo, S.E.

Información del Artículo

Recibido: 22 de septiembre, 2025
Aceptado: 20 de noviembre, 2025

Palabras clave: Línea de transmisión, software libre, Carta de Smith, ROE, coeficiente de reflexión

Resumen

La carta de Smith es una herramienta gráfica útil que permite dar solución a circuitos de alta frecuencia y líneas de trasmisión evitando complejos cálculos usando regla y compás. Hoy en día, existen apoyos tecnológicos de software para facilitar el desarrollo de soluciones de esta área de microondas, muchos de estos tienen costo pero otros son libres e incluso en línea. En este trabajo se muestra el uso del software libre SmithChart para dar solución a un problema de línea de transmisión, también se realizan los cálculos en forma analítica, los resultados muestran que con pocos movimientos en el software se obtienen resultados similares a los calculados analíticamente.

1. Introducción

La carta de Smith es una herramienta para el análisis de las líneas de transmisión en cualquier circuito de microondas. La carta es una representación, en el plano del coeficiente de reflexión, de la resistencia y reactancia normalizadas que permite obtener diversos parámetros que caracterizan las líneas de transmisión, así como resolver problemas de adaptación de impedancias de forma gráfica, sin necesidad de recurrir al cálculo complejo [1]. Es una herramienta gráfica ampliamente utilizada en telecomunicaciones, especialmente en las áreas del conocimiento de diseño de circuitos de comunicaciones, líneas de transmisión, ingeniería de antenas y microondas, inventada en la década de los 30´s del siglo pasado, desarrollada casi simultáneamente por Mizuhashi Tosaku (1937) y Phillip Hagar Smith (1939) [2].

Zozaya (2018), comenta que esta carta permite el cálculo gráfico de numerosos parámetros técnicos de interés cuyo cálculo analítico comprende operaciones tediosas con números complejos que, si bien en la primera mitad del siglo pasado eran difíciles de llevar a cabo, hoy día con la enorme capacidad de cómputo de cualquier PC o teléfono inteligente, mantiene una importante vigencia formando parte sustancial del currículo de Ingeniería en Telecomunicaciones en todo el mundo [2].

La carta de Smith (ver Figura 1), se usa para calcular impedancias de entrada, ROE, coeficientes de reflexión y otros datos sólo con una regla y un compás, sin usar funciones trigonométricas o hiperbólicas, lo que facilita los cálculos. Aunque se han deducido sus ecuaciones para líneas sin pérdidas (Z0 real), es posible extender su uso a líneas con bajas pérdidas [3].

En el estudio de la transmisión de ondas, las estructuras a través de la cuales se propaga la señal desde el generador hasta un punto de destino (o carga) se denominan líneas de transmisión. En este contexto, para el diseño de circuitos resulta interesante conocer la oposición que presentan los distintos tipos de líneas al paso de las ondas que se pretende estudiar; la medida de esta oposición es conocida como impedancia [1].

De acuerdo con Giraldo (2014), mencionado por Acuña (2018), indica que las líneas de transmisión son medios materiales uniformes diseñados, construidos y empleados para transportar señales eléctricas tal que se garanticen niveles de atenuación, distorsión y res-

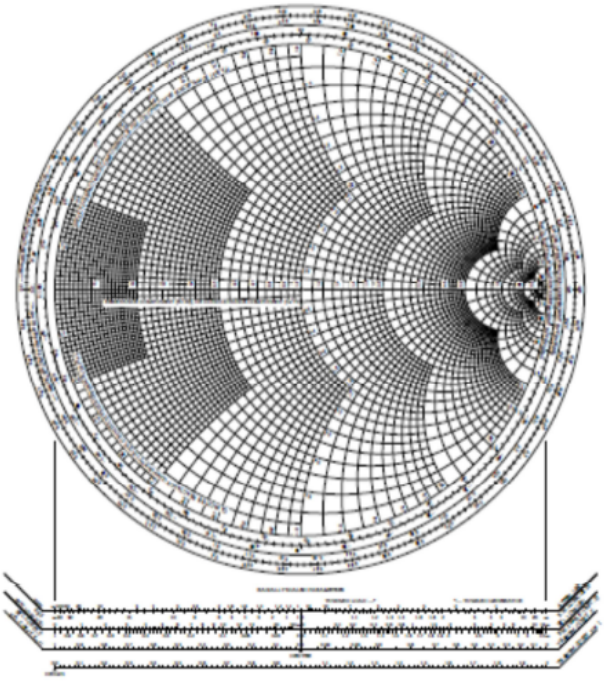

Figura 1: Gráfica o carta de Smith. Fuente: [3].

puesta en frecuencia adecuados para los niveles de potencia, alcance y espectro de frecuencia de las señales conducidas. Para el caso de señales inalámbricas, el medio de transmisión es la atmósfera y el vacío, los cuales, no pueden ser modificados y, por eso, las señales a transmitirse por esos medios se acondicionan a sus características [4].

En la Figura 2, se muestra el modelo eléctrico de una línea de transmisión la cual consiste de una resistencia modelando las pérdidas por calor, una inductancia en serie debido a los conductores y un capacitor en paralelo con una conductancia modelando las pérdidas del dieléctrico entre conductores.

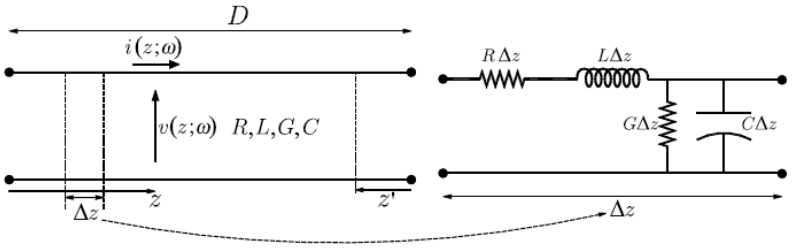

Figura 2: Modelo eléctrico de una línea de transmisión. Fuente: [5].

De acuerdo con Caspers (2012), la carta de Smith es una herramienta muy valiosa e importante que facilita la interpretación de las mediciones de los parámetros S. En su trabajo ofrece una breve descripción general del por qué y, aún más importante, del cómo usarla. Muestra las amplias posibilidades de uso de la carta en diversas aplicaciones de alta frecuencia [6].

Pieper y Dellsperger (2001), comentan que una ventaja reconocida del uso de la carta de Smith es que evita el uso de álgebra compleja para resolver problemas de transmisión. En

su trabajo combinan la potencia de la PC con su estética visual, velocidad y precisión para familiarizar a los estudiantes con el uso de la calculadora gráfica de líneas de transmisión de A. Smith. Enfatizan que la versión demo del programa carta de Smith para PC, disponible en internet, está limitada a menos de 6 puntos de datos. A pesar de sus limitaciones para la resolución de problemas de diseño extensos, la versión demo proporciona una excelente herramienta para la instrucción de nivel introductorio de la carta de Smith [7].

Las ecuaciones de Maxwell relacionan los campos eléctricos y magnéticos con las cargas y corrientes que los crean. La solución general de las ecuaciones, en el caso variable en el tiempo, es en forma de ondas que pueden estar ligadas a una estructura, como es el caso de una línea de transmisión o guía de ondas, o bien libre en el espacio como ocurre con las producidas por las antenas. Toda onda se caracteriza por su frecuencia y su longitud de onda, ambas relacionadas por la velocidad de propagación en el medio que habitualmente en antenas tiene propiedades del vacío [8].

En otro orden de ideas, Albiter et al. (2019), enfatiza que el sistema educativo se ha visto afectado o impactado como consecuencia de la explosión tecnológica de la informática, con un nuevo enfoque de enseñanza denominado Pensamiento Computacional (PC en lo sucesivo), como consecuencia de lo anterior, dentro de la educación superior el uso de los diferentes paquetes informáticos es demasiado múltiple, dentro de este contexto se encuentra el software de simulación el cual se ha ido integrando en la última década como parte de las didácticas de la enseñanza educativa [9].

Pareja Aparicio citado por Albiter (2019), comenta que la simulación sirve como punto intermedio entre los conceptos teóricos y la realidad. Cuanto mejor sea la expresión que defina a la realidad, mejores serán los resultados, porque serán más reales y, a su vez, puede reducir los costos de fabricación, facilitando las tareas de diseño [9].

Asimismo, Rodríguez (2014), opina que utilizar una herramienta de software libre implica ejecutarlo, distribuirlo, estudiarlo, modificarlo y mejorarlo sin restricciones. Su utilización a escala mundial aumenta cada día y el potencial de beneficio que representa para al sector educativo y de investigación es considerable. Por un lado, reduce los costos para las instituciones académicas; y por otro, les permite a los estudiantes fomentar su curiosidad científica fuera del aula [10].

Añade también, que varios son los motivos que dan importancia a la utilización de software libre en la educación. Por un lado, la significativa reducción de costos para las instituciones educativas y, por otro, la posibilidad que adquiere el estudiante de profundizar su conocimiento hasta donde su curiosidad lo lleve. Lo anterior cobra sentido si se tiene en cuenta que, en el transcurso de su formación, un estudiante de ingeniería debe desarrollar actividades como laboratorios y prácticas que requieren el uso de software para el análisis y la simulación de datos, pero si dicho software es de pago, sólo puede ser utilizado en la institución educativa, lo que limita el tiempo de uso. En cambio, si el software es libre, el estudiante puede seguir trabajando indefinidamente fuera de la institución, lo que fortalece la construcción de su conocimiento [10].

Herrera (2013), hizo una revisión de software libre y dice que éste está orientado a determinados grupos de usuarios, sobre los que se exige un conocimiento mínimo del entorno informático donde se usa determinada herramienta de software, su orientación es innovadora, es decir, prima el entender cómo funciona y sus posibles usos, sobre la practicidad o

facilidad de uso que determinada herramienta tenga [11].

Zamanillo et al. (2012), muestra en su trabajo el desarrollo de una herramienta educativa gratuita para el diseño de filtros de microondas. El software, denominado FILMAT 2.0, se basa en un programa previo desarrollado para el diseño de filtros de elementos concentrados y el cual fue desarrollado en MATLAB. Se generó una aplicación independiente, lo que permitió que el software funcione sin necesidad de una licencia de MATLAB, simplemente con un entorno de ejecución gratuito. Los resultados son comparables en calidad a los de los simuladores comerciales. La aplicación fue desarrollada bajo la filosofía de ”amigable con el usuario”, es fácil de usar y puede ser utilizada por usuarios de cualquier nivel [12].

Liu et al. (2017), mencionan que en las últimas décadas los trabajos de investigación relacionados a las antenas y dispositivos de lata frecuencia, han contribuido al desarrollo de software comercial como herramientas para el diseño de las propias antenas como: CTS Studio, Ansoft HFSS, ADS-Momentum, Altair-Feko, Sonnet Suits, etc,. Estos softwares proporcionan métodos de optimización local (método quasi-Newton, método de programación cuadrática secuencial) y global (algoritmos genéticos y método de grupo de hormigas PSO) sobre un diseño específico. Su trabajo muestra un diseño de antena de microcinta haciendo uso de las facilidades del software “Antenna Design Explorer (ADE)” [13].

Georgescu et al. (2025), comentan una metodología integral y eficaz para la adaptación de impedancia en circuitos analógicos, combinando cálculos analíticos, información gráfica mediante el diagrama de Smith y automatización asistida por computadora con MATLAB. La herramienta desarrollada, basada en MATLAB, genera eficientemente múltiples soluciones de adaptación de impedancia equivalente, simplificando significativamente el proceso de diseño y permitiendo una rápida evaluación y creación de prototipos para ingenieros de RF y microondas [14].

Badii y Oloomi (1998), presentan una aplicación de líneas de transmisión (TLA) MATLAB Toolbox capaz de resolver problemas de emparejamiento o acoplamiento de ramales simples y dobles en líneas de transmisión mediante el método gráfico del diagrama de Smith. El paquete gestiona tanto los casos de ramales cortos como los de ramales abiertos y proporciona una solución analítica y gráfica. Para cada uno de los problemas mencionados, el software genera el diagrama de Smith correspondiente y un resumen de los resultados finales [15].

De acuerdo con Zuniga (2014), MMANA-GAL es un programa para diseñar y analizar antenas. Es gratuito y fue escrito por Alexandr Schewelev, Igor oncharenko y Makoto Mori. La versión pro es el programa GAL-ANA. El nombre MMANA-GAL viene de abreviaciones del nombre de sus tres programadores: MM por Makoto Mori, G por Goncharenko, y AL por Alexandr [3].

Se puede decir, que cada vez se está utilizando más software para circuitos de microondas o alta frecuencia, estos son costosos comúnmente, pero también hay software libre que permite realizar soluciones de líneas de transmisión o para los diseños de antenas como es el caso de SmithChart o MMana-Gal respectivamente etc. Pero comúnmente la información es compleja y las ayudas muy limitadas, en este trabajo se presenta el software libre SmithChart para la solución de problemas con líneas de transmisión.

2. Metodología

Para el desarrollo de la solución del problema de líneas de transmisión, se siguió la siguiente metodología:

- Investigación documental
- Selección del problema a resolver
- Manejo del software SmithChart
- Solución del problema de manera analítica
- Solución del problema con el software SmithChart
- Pruebas y análisis de resultados
- Conclusiones

3. Desarrollo

Descripción de la carta de Smith

La carta de Smith es un gráfico compuesto por círculos denominados círculos de resistencia constante, indicados en la Figura 3, señalados por el número 4, en el centro con el número 1 (1,0) representa la impedancia característica de una línea de transmisión que puede ser de 50, 75 o 100 Ohms comúnmente; la parte extrema izquierda número 2, representa un corto circuito; al otro extremo el número 3, representa un circuito abierto en una línea de transmisión.

Otros círculos que también contiene, son los llamados círculos de reactancia constante, representados en la Figura 4, haciendo referencia con el número 1 a las reactancias inductivas y con el número 2 a las reactancias capacitivas.

Es decir, esta carta representa un esquema de números complejos tanto de manera rectangular como polar, y permite ver los estados de una línea de transmisión de manera abstracta como se muestra en la Figura 5, siendo el número 1 la impedancia característica de la línea y el número 2 la impedancia de carga, cuando $Z_0 = Z_L$ se estaría en un punto en el centro de la carta, si $Z_0 \neq Z_L$ se estaría en cualquier punto dentro de la carta, si Z_L es un corto se estaría en el extremo izquierdo de la carta en el número 2 del círculo de la Figura 3, pero si Z_L es infinita o abierta se estaría en el punto 3 también de la Figura 3.

A esta función de la carta se denomina carta de Smith de impedancias, pero también tiene la contraparte que son círculos de admitancia que ayudan para cuando existen elementos en paralelo, y es más fácil trabajar en el sistema de admitancias.

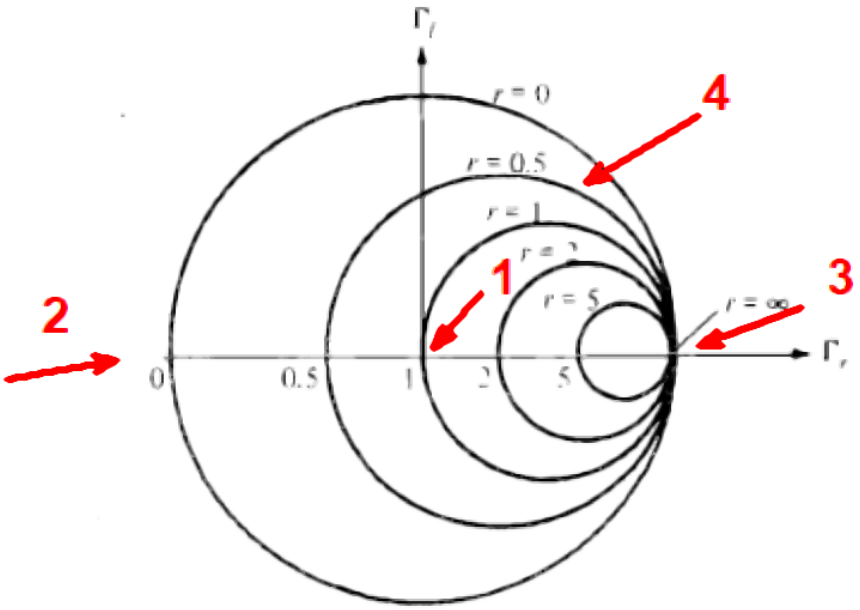

Figura 3: Carta de Smith círculo de resistencia constante. Fuente: [16].

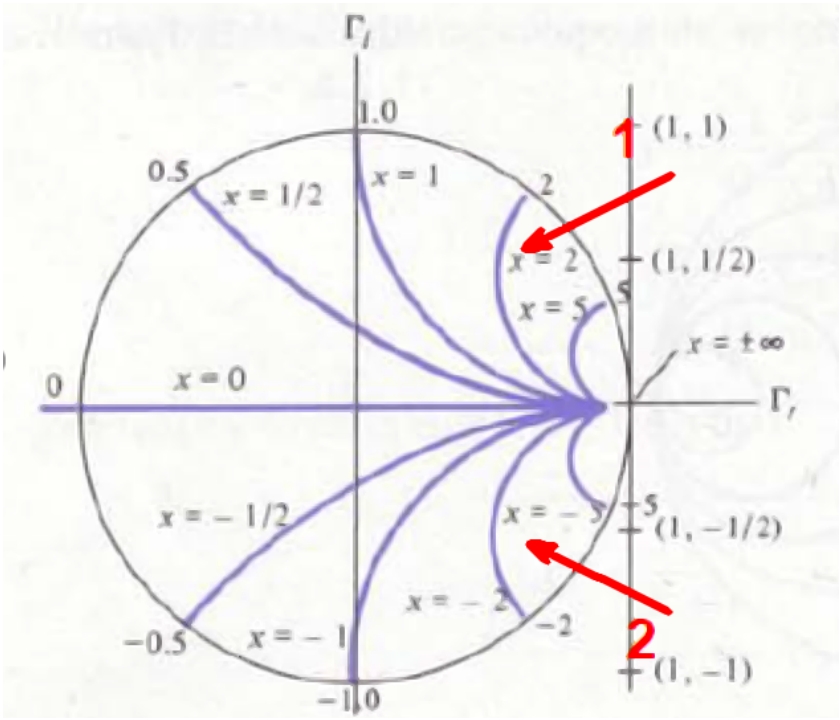

Figura 4: Carta de Smith circulo de resistencia constante. Fuente: [16].

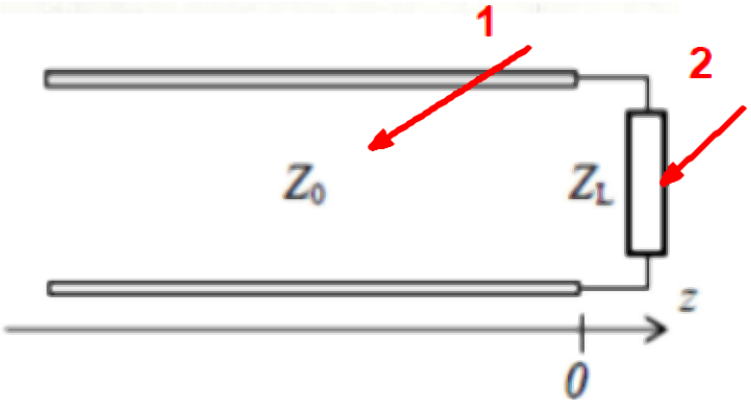

Figura 5: Línea de transmisión genérica que omite los parámetros de resistencias, bobinas, capacitancias y conductancias. Fuente: [3].

Software SmithChart

El software SmithChart es libre, facilita el proceso de solución de problemas con el uso de la carta de Smith, presenta una pantalla inicial como se indica en la Figura 6, es la interfaz de usuario donde se ingresan datos y se reciben los resultados en el diagrama de Smith mostrado en la parte derecha. La ayuda que tiene es escasa, pero aún así es una herramienta muy útil.

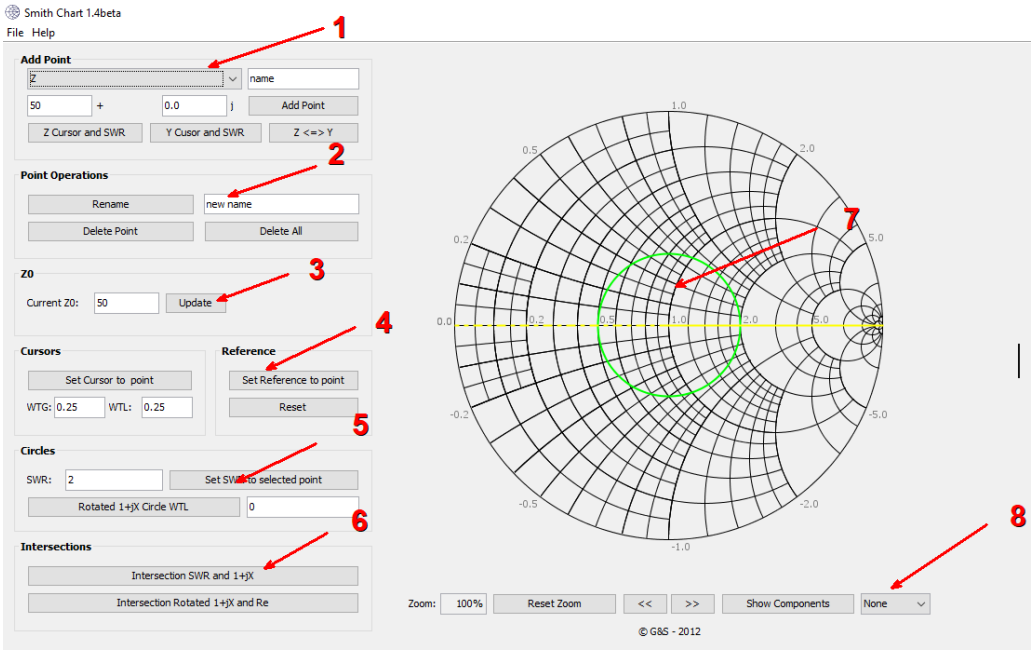

Figura 6: Pantalla de inicio del software SmithChart. Fuente: Elaboración propia.

En el indicador 1, se pueden ingresar los datos de varios parámetros como la impedancia de carga, o la admitancia y pueden ser normalizados o no; en el número 2, se muestran botones para editar o borrar datos del gráfico; el número 3, es uno de los más importantes para indicar la impedancia característica de la línea, ya que con este referente se hacen todos los cálculos; el indicador 4, permite colocar el cursor en un punto para ser una referencia; con

el número 5, se realiza el círculo de la relación de onda estacionaria ROE; el número 6, permite colocar los círculos unitarios de impedancia y admitancia para cálculos de acoplamiento; el número 7, es el gráfico de Smith donde se muestran los resultados y, finalmente, el número 8 permite la selección de viajar hacia el generador desde la carga o desde la carga hacia el generador en una línea de transmisión.

4. Pruebas y resultados

Matthew y Sadiku (2003) proponen la solución del problema de manera analítica y en la carta de Smith tradicional, en esta propuesta se desarrolla la solución al problema usando el software SmithChart, el problema planteado es: Una carga de valor $100 + j150$ Ohms, está conectada a una línea de transmisión sin pérdidas de 75 Ohms encuentre [16]:

a) Coeficiente de reflexión Γ

b) La ROE

c) La admitancia de la carga Y_L

d) La impedancia de entrada a $0{,}4\lambda$ de la carga

e) Las localizaciones de V_{max} y V_{min} con respecto a la carga, si la línea es de $0{,}6\lambda$ de longitud

f) La impedancia de entrada Z_{ent} en el generador

Solución:

Paso 1. Normalizar la impedancia de carga Z_L, esto se hace de manera analítica dividiendo el valor de la impedancia de carga entre la impedancia característica de la línea, como se indica en la Ec. (1).

$$Z_{LN} = \frac{Z_L}{Z_0} \tag{1}$$

$$Z_{LN} = \frac{100 + j150}{75} = 1{,}33 + j2$$

Paso 2. Localizar en el diagrama de Smith la impedancia de carga normalizada Z_{LN}

En la Figura 7, se muestran los resultados: la impedancia de carga normalizada que coincide con lo calculado en la Ec. (1), además de la relación de onda estacionaria ROE (Voltaje Standing Wave Ratio, VSWR) y el coeficiente de reflexión, esto de manera inmediata.

Paso 3. Para calcular el coeficiente de reflexión se utiliza la Ec. (2), que relaciona la impedancia de carga y la impedancia característica, este resultado es similar al del software:

$$|\rho| = \frac{Z_L - Z_0}{Z_L + Z_0} \tag{2}$$

$$|\rho| = \frac{100 + j150 - 75}{100 + j150 + 75} = 0{,}659\angle 40^\circ$$

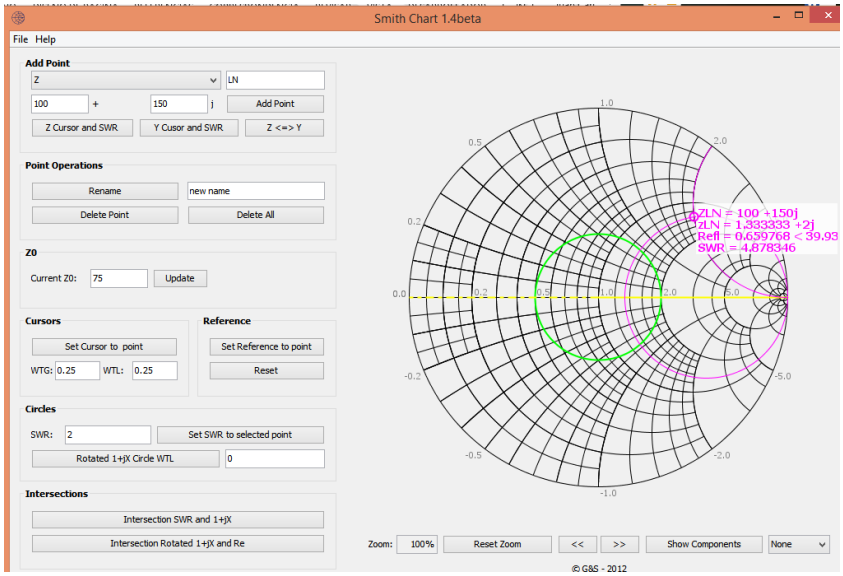

Figura 7: Localización de la impedancia de carga en el software SmithChart. Fuente: Elaboración propia.

Paso 4. La ROE, se puede calcular partir del coeficiente de reflexión como se indica en la Ec. (3).

$$\text{ROE} = S = \frac{1 + |\rho|}{1 - |\rho|} \tag{3}$$

$$\text{ROE} = \frac{1 + 0{,}659}{1 - 0{,}659} = 4{,}865$$

Los resultados analíticos concuerdan con los datos gráficos del software, como se indica en la Figura 8.

Paso 5. Para obtener la admitancia de carga Y_L, en la carta de Smith se prolonga una línea desde la impedancia de carga Z_{LN} que pasa por el centro hasta el lado opuesto del círculo de coeficiente de reflexión constante, se debe seleccionar el círculo de resistencia y de reactancia que se crucen con el círculo del coeficiente de reflexión. En el software, hay que colocar el punto de impedancia de carga y presionar un botón para calcular la admitancia de carga (ver Figura 9) y el resultado se muestra en la Figura 10.

El resultado muestra un valor de admitancia de carga normalizada (en Mhos o Siemens) y des-normalizada de:

$$Y_{LN} = 0{,}2307 - j0{,}3461$$

$$Y_L = 0{,}003077 - j0{,}004615$$

Este valor está normalizado a la impedancia característica de la línea del problema, que es de 75 Ohms, para des-normalizar se puede hacer con la Ec. (4), para encontrar los valores reales.

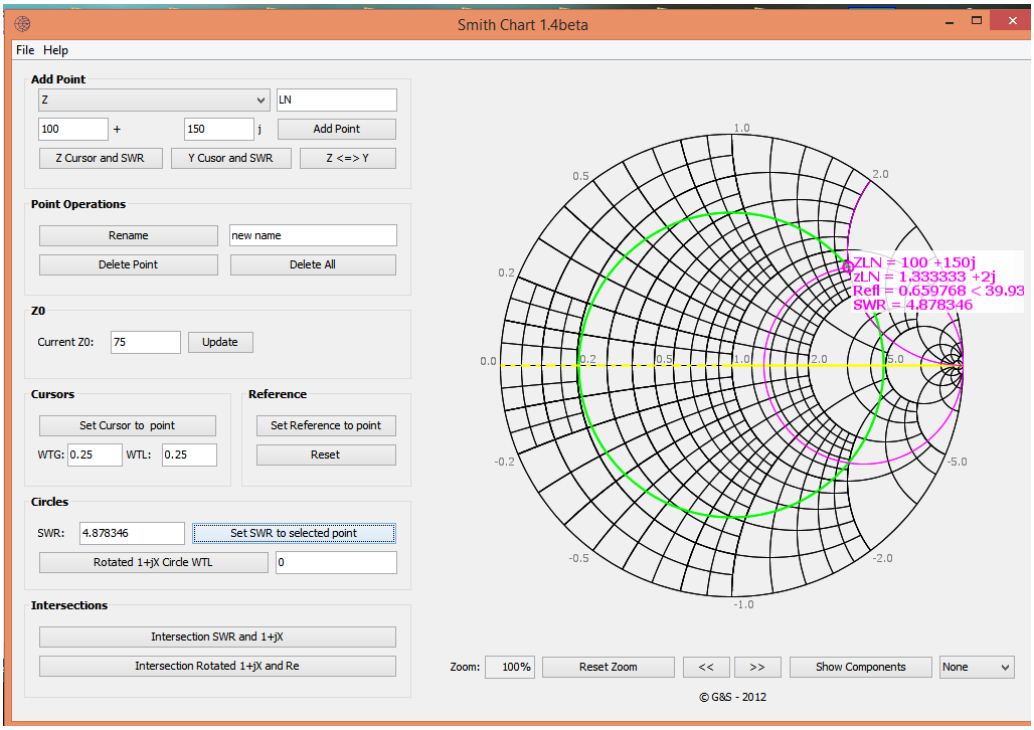

Figura 8: Localización del coeficiente de reflexión y la ROE en el software SmithChart. Fuente: Elaboración propia.

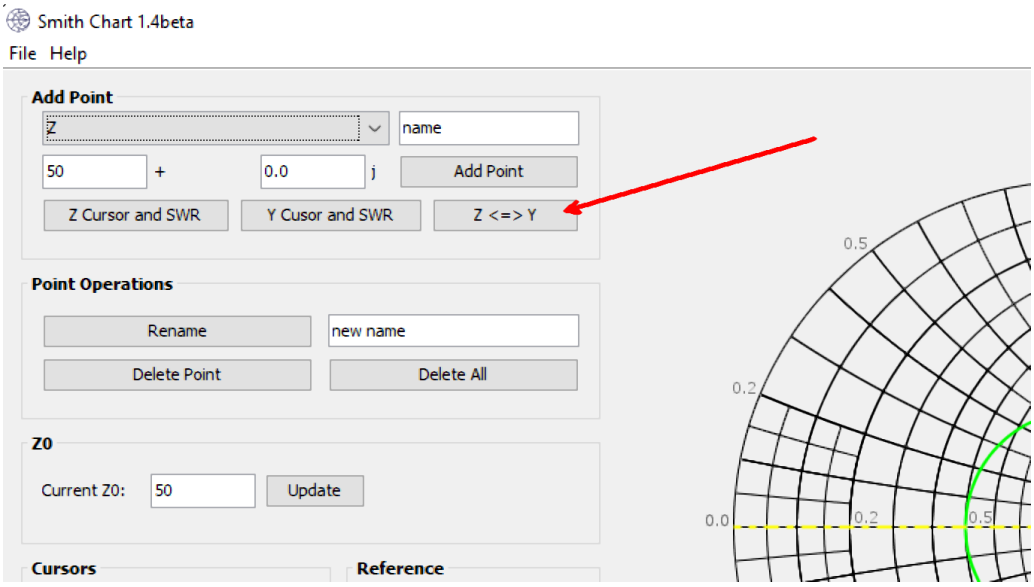

Figura 9: Localización del coeficiente de reflexión y la ROE en el software SmithChart. Fuente: Elaboración propia.

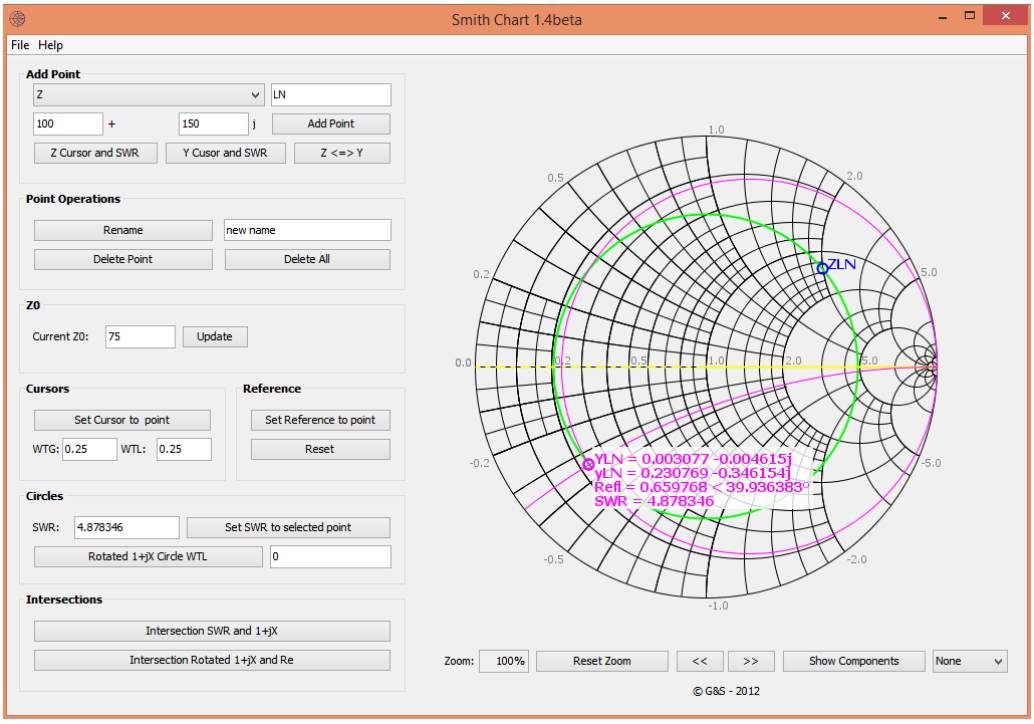

Figura 10: Localización de la admitancia de carga normalizada y des-normalizada en el software SmithChart. Fuente: Elaboración propia.

$$Y_L = Y_0 Y_{Ln} \tag{4}$$

Y considerar que la admitancia de carga es el inverso de la impedancia como se indica en la Ec. (5).

$$Y_0 = \frac{1}{Z_L} \tag{5}$$

Calculando de manera analítica y aplicando la Ec. (5) el resultado es:

$$Y_L = \frac{1}{75}(0{,}228 - j0{,}35) = 0{,}00304 - j0{,}00467 \text{ Mhos}$$

O de otra manera se puede utilizar la Ec. (6).

$$Y_L = \frac{1}{Z_L} \tag{6}$$

$$Y_L = \frac{1}{100 + j150} = 0{,}00304 - j0{,}00467 \text{ Mhos}$$

Que coincide con los datos obtenidos en el software.

Paso 6. Para obtener la impedancia de entrada, si la línea es de $0{,}4\lambda$ desde la carga, se procede de la siguiente forma. Se sabe que una vuelta en la carta de Smith corresponde a $360°$ y es igual a $0{,}5\lambda$, y 1λ es igual a $720°$ con esto como referencia se puede decir que:

$$0{,}4\lambda = 0{,}4 \times 720 = 288°$$

Por lo que, en la carta de Smith, a partir del punto de impedancia de carga normalizada Z_{LN} se gira hacia el generador $288°$ en el sentido de las manecillas del reloj, sobre el círculo de coeficiente de reflexión constante hasta el punto de cruce (ver Figura 11), ahí se lee la impedancia de entrada normalizada Z_{entN}.

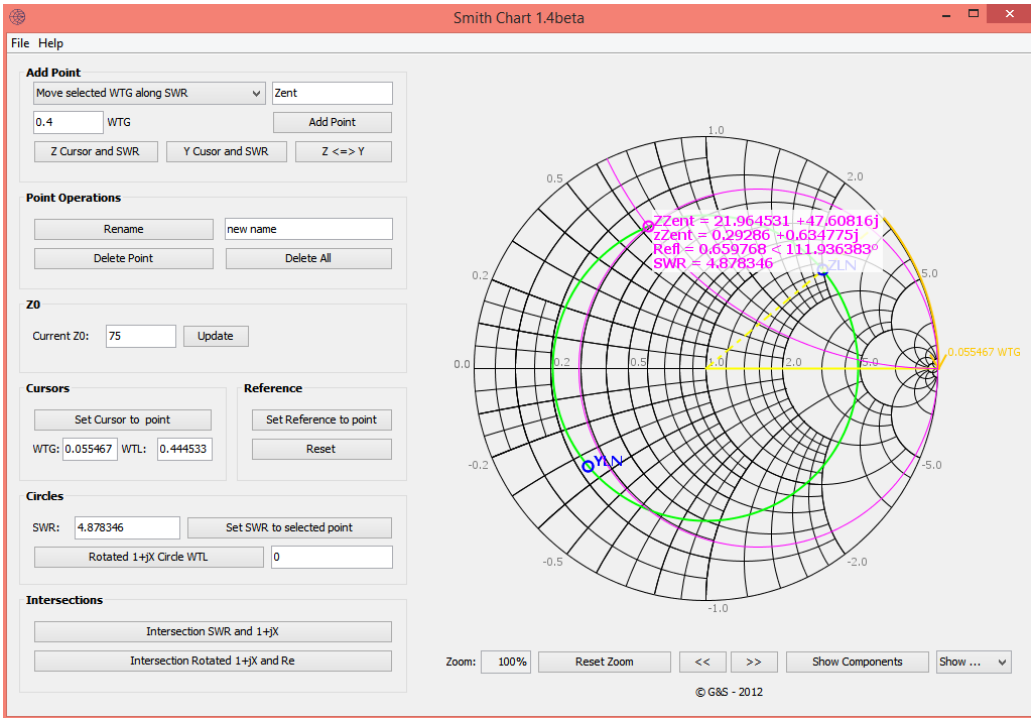

Figura 11: Localización de la impedancia de entrada normalizad y des-normalizada para una distancia de $0{,}4\lambda$ en SmithChart. Fuente: Elaboración propia

El software proporciona el resultado tanto normalizado como des-normalizado de la impedancia de entrada, en este caso, la impedancia de entrada normalizada es:

$$Z_{entN} = 0{,}29286 + j0{,}6347 \text{ Ohms}$$

Si se desea des-normalizar la impedancia de entrada se puede aplicar la Ec. (7).

$$Z_{ent} = Z_0 Z_{entN} \tag{7}$$

$$Z_{ent} = 75(0{,}29 + j0{,}63) = 21{,}75 + j47{,}25 \text{ Ohms}$$

Que coincide con los datos del software SmithChart.

Analíticamente se puede calcular la impedancia de entrada con una fórmula que incluye una tangente hiperbólica como se muestra en la Ec. (8), aquí es donde el software ahorra este tipo de cálculos y gama que se describe en la Ec. (9).

$$Z_{ent} = Z_0 \left[\frac{Z_L + jZ_0 \tanh(\Upsilon l)}{Z_0 + jZ_L \tanh(\Upsilon l)} \right] \tag{8}$$

Donde:

Υ= constante de propagación y es un número complejo
l= distancia en mts
Z_0= Impedancia característica de la línea
Z_L= Impedancia de carga

$$\Upsilon = \alpha + j\beta \tag{9}$$

Donde:
α= constante de atenuación
β= constante de fase
Cuando la línea es sin pérdidas, se puede aplicar, la fórmula de la Ec. (10) donde la tangente se aproxima a la tangente hiperbólica y como no hay pérdidas el factor α se hace cero:

$$Z_{ent} = Z_0 \left[\frac{Z_L + jZ_0 \tan \beta l}{Z_0 + jZ_L \tan \beta l} \right] \tag{10}$$

A la relación βl se le conoce como la longitud eléctrica y se expresa en grados eléctricos (que son la mitad de los grados geométricos que se toman en cuenta en la carta de Smith) o radianes. Aplicando la Ec. (10) se tiene:

$$Z_{ent} = 75 \left[\frac{100 + j150 + j75 \tan 144^\circ}{75 + j(100 + j150) \tan 144^\circ} \right]$$

De forma rectangular el resultado se:

$$Z_{ent} = 21{,}9 + j47{,}61 \text{ Ohms}$$

De forma polar el resultado es:

$$Z_{ent} = 54{,}41 \angle 65{,}25^\circ$$

Que coincide con lo graficado y obtenido en el software de manera más fácil, la diferencia radica en los decimales que se toman en cuenta.

Paso 7. Para obtener el primer máximo y mínimo de voltaje desde la impedancia de carga normalizada, sobre el círculo de coeficiente de reflexión constante se viaja hacia el generador hasta el primer cruce con el eje los reales, ahí es el primer máximo de voltaje.

$$\text{El primer } V_{max} = \frac{40^\circ}{720^\circ}\lambda = 0{,}055\lambda$$

El segundo máximo de voltaje es sumando una vuelta a la carta de Smith que equivale a sumar $0{,}5\lambda$ al valor anterior como se indica a continuación.

$$\text{El segundo } V_{max} \text{ está } = 0{,}055\lambda + \frac{\lambda}{2} = 0{,}55\lambda$$

El primer mínimo está $0{,}25\lambda$ del primer máximo por lo que es igual a:

$$\text{primer } V_{min} = 0{,}055\lambda + \frac{\lambda}{4} = 0{,}3055\lambda$$

En el software SmithChart se realiza como sigue:

La carga está a $40°$ grados del eje de los reales, este valor es igual a $0{,}055\lambda$ y se integra en los datos del software como indica la Figura 12.

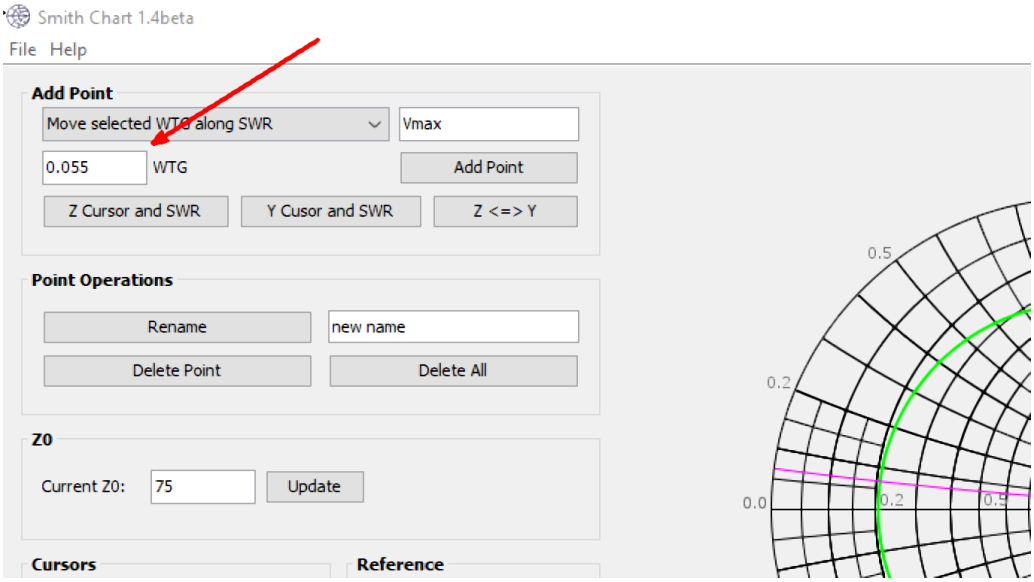

Figura 12: Ingreso del punto a cuarenta grados o $0{,}055\lambda$ de la carga hacia al generador en SmithChart. Fuente: Elaboración propia.

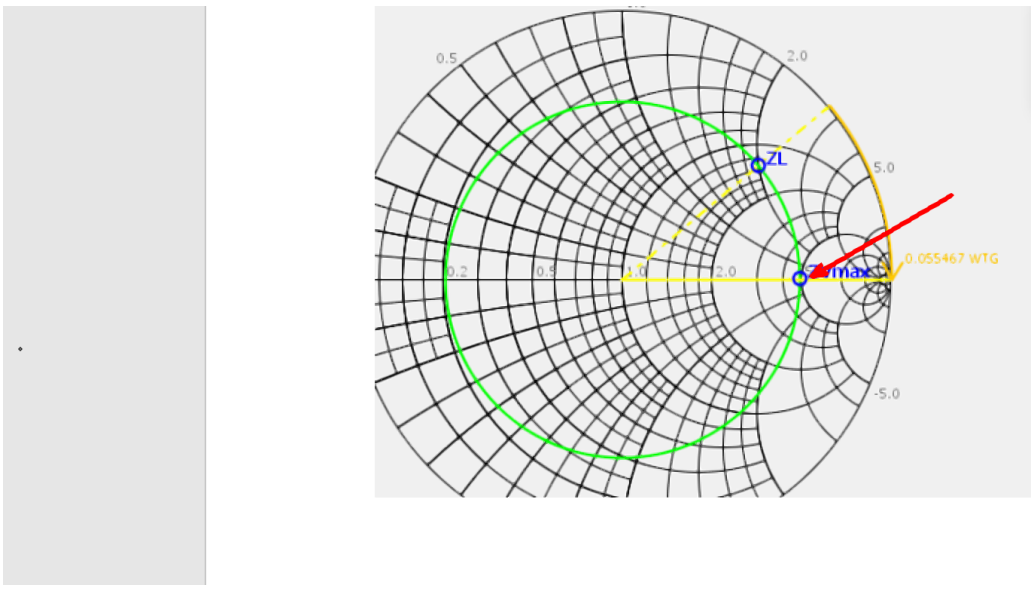

Figura 13: Localización del primer máximo de voltaje en SmithChart. Fuente: Elaboración propia

El resultado de la Figura 13 muestra el primer máximo de voltaje a $0{,}055\lambda$ de la carga que es el cruce con el eje de los reales.

De la misma forma se ingresa el valor del primer mínimo que está a $0{,}3055\lambda$ desde la carga, en el software como un punto se ingresa el dato, y se selecciona hacia el generador como se observa en las Figuras 14 y 15.

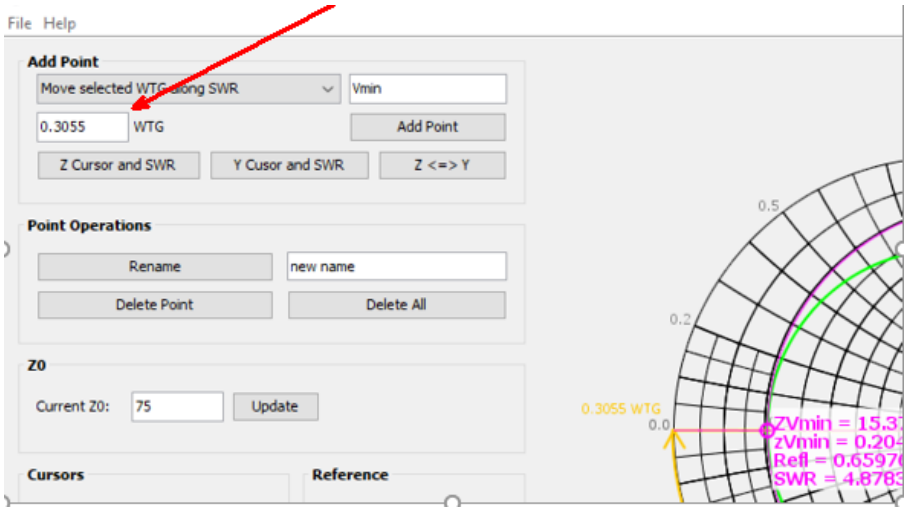

Figura 14: Ingreso del punto $0{,}3055\lambda$ desde la carga hacia al generador en SmithChart. Fuente: Elaboración propia.

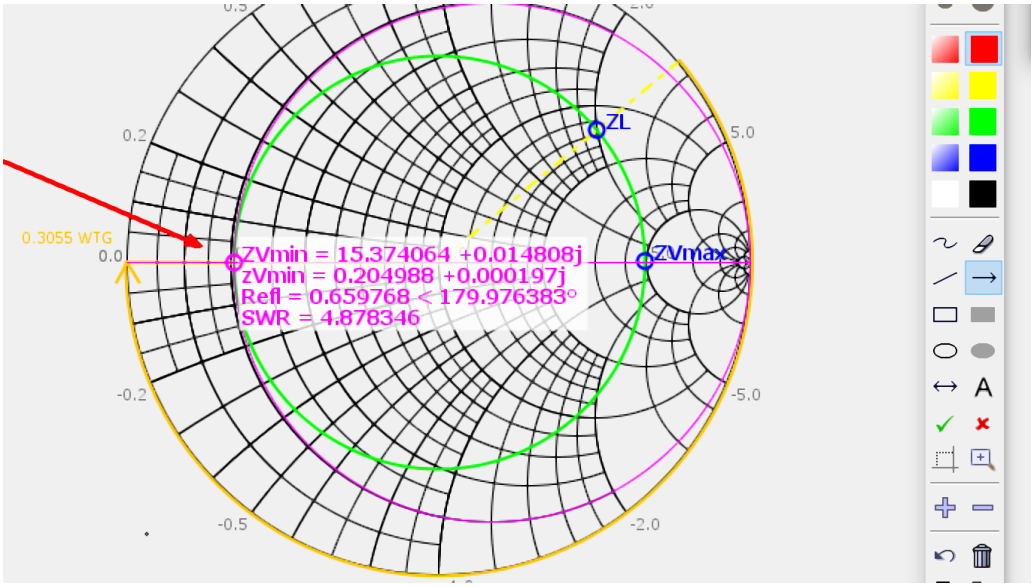

Figura 15: Gráfico de localización del primer mínimo de voltaje en SmithChart. Fuente: Elaboración propia.

Paso 8. Para obtener la impedancia de entrada si la línea es de $0{,}6\lambda$ desde la carga, se procede de la misma forma que el punto 6; se sabe que una vuelta en la carta de Smith corresponde a $360°$ y es igual a $0{,}5\lambda$, y 1λ es igual a $720°$ con esto como referencia se puede decir que:

$$0{,}6\lambda = (0{,}6)720° = 432° = 1 \text{ revolución} + 72°$$

Se parte del punto Z_{LN} y se gira hacia el generador a lo largo del círculo de coeficiente de reflexión constante, hasta avanzar $72°$ o $0{,}1\lambda$ hasta encontrar el punto de cruce, ese punto es la impedancia de entrada.

Analíticamente se hace lo siguiente:

$$Z_{entN} = 1{,}8 - j2{,}2 \text{ Ohms}$$

Aplicando la Ec. (7).

$$Z_{ent} = 75(1{,}8 - j2{,}2) = 135 - j165 \text{ Ohms}$$

Se puede comprobar calculando la distancia eléctrica y observando que los grados utilizados en la Ec. (10) son eléctricos la mitad que los grados geométricos que son $432°$:

$$\beta l = \frac{2\pi(0{,}6\lambda)}{\lambda} = 360°(0{,}6) = 216°$$

Sustituyendo valores en la Ec. (10) se tiene:

$$Z_{ent} = 75\left[\frac{100 + j150 + j75\tan 216°}{75 + j(100 + j150)\tan 216°}\right]$$

$$Z_{ent} = 135 - j165 \text{ Ohms}$$

$$Z_{entN} = 1{,}8 - j2{,}2 \text{ Ohms}$$

$$Z_{ent} = 75(1{,}8 - j2{,}2) = 135 - j165 \text{ Ohms}$$

Los resultados de la Figura 16, en el software SmithChart muestran que la impedancia de entrada normalizada es de 1.781177-2.209 y la des-normalizada de 133.588-162 que son más precisos que los calculados analíticamente, por el truncamiento, pero son muy semejantes.

5. Conclusiones

El software libre permite diseñar y simular antenas con líneas de transmisión, o acoplamientos de manera rápida y eficiente.

El software SmithChart facilitó la obtención de resultados sobre el problema de la línea de transmisión, los resultados con los cálculos analíticos son similares.

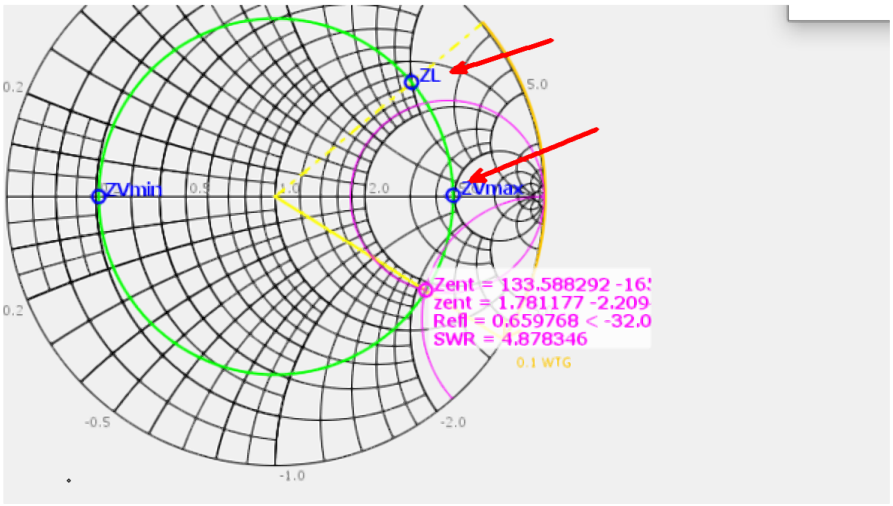

Figura 16: Localización de la impedancia de entrada cuando la línea es de $0{,}6\lambda$ en SmithChart. Fuente: Elaboración propia.

En el problema resuelto se obtuvo un coeficiente de reflexión de 0.659 y una ROE de 4.865 aunque es un ejemplo didáctico se puede decir que es un mal resultado ya que el coeficiente de reflexión debe tender a 0 y la ROE un valor menor a 2 para considerase línea acoplada.

Existe software propietario y comúnmente es de un costo elevado, el software libre es una opción para instituciones y jóvenes estudiantes de bajos recursos.

Direcciones de correo de los autores

Pérez Merlos, J.C.: jcjcjc63@yahoo.com

Salgado Gallegos, M.: msalgadog@uaemex.mx

Albarrán Trujillo, S.E.: Universidad Autónoma del Estado de México sealbarrant@uaemex.mx

Referencias

[1] A. A. Müller, M. J. Pérez-Peñalver y E. Sanabria-Codesal, *MSEL*. 2016.

[2] A. Zozaya, «Smith Chart ad hoc using GNU Octave for didactics purposes,» *Ingeniería UC*, vol. 25, n.º 3, págs. 381-387, 2018.

[3] D. Morales Escobar, *Líneas de transmisión: Guía para el alumno*. 2014.

[4] L. L. Acuña Carpio, *Líneas de transmisión*. Lima Perú: Escuela Profesional de Electrónica y Telecomunicaciones, Dpto. Telecomunicaciones e Informática, 2018.

[5] E. Gago-Ribas, C. Dehesa-Martínez y M. González-Morales, «Complex analysis of the lossy-transmission line theory: A generalized smith chart,» *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 14, n.º 1, págs. 173-194, 2006.

[6] F. Caspers, «RF engineering basic concepts: the Smith chart,» *arXiv preprint*, 2012, arXiv:1201.4068.

[7] R. Pieper y F. Dellsperger, «Personal computer assisted tutorial for Smith charts,» en *Proc. 33rd Southeastern Symposium on System Theory (Cat. No. 01EX460)*, 2001.

[8] . C. Aznar, J. R. Robert, J. M. R. Casals, L. J. Roca, S. B. Boris y M. F. Bataller, *Antenas*. Univ. Politécnica de Catalunya, 2004, vol. 3.

[9] J. J. Albiter, M. R. V. Mendoza y C. E. J. Dorantes, *El pensamiento computacional en la electrónica: La importancia del software de simulación en la comprensión del principio de funcionamiento de los componentes electrónicos*. Alfaomega, 2019. dirección: http://doi.org/10.17993/3ctic.2019.84.85-113

[10] W. R. Rodríguez Dueñas, «Software libre para educación e investigación en ingeniería,» *Revista Educación en Ingeniería*, vol. 9, n.º 18, págs. 12-22, 2014. DOI: 10.26507/rei.v9n18.383

[11] L. R. C. Herrera, «Herramientas de software libre para aplicaciones en ciencias e ingeniería,» *Revista Politécnica*, n.º 32, 2013.

[12] J. M. Zamanillo Sainz de la Maza, . Fernández Ríos y M. I. Zamanillo Sainz de la Maza, «FILMAT 2.0 Software de diseño de filtros de microondas en tecnología microtira para aplicaciones docentes,» inf. téc., 2012.

[13] B. Liu, A. Irvine, M. O. Akinsolu, O. Arabi, V. Grout y N. Ali, «GUI design exploration software for microwave antennas,» *Journal of Computational Design and Engineering*, vol. 4, n.º 4, págs. 274-281, 2017.

[14] A.-F. Georgescu et al., «Automatic Procedure and the Use of the Smith Chart in Impedance Matching in Analog Circuits,» *Electronics*, vol. 14, n.º 14, pág. 2746, 2025.

[15] V. Badii y H. M. Oloomi, «Transmission line application MATLAB Toolbox based on the graphical design methods of the Smith chart,» *Computer Applications in Engineering Education*, vol. 6, n.º 1, págs. 23-30, 1998.

[16] S. Matthew y O. Sadiku, *Elementos de electromagnetismo*. España: Crítica, 2003.

Informaticae Abstracta. ISSN: 3061-8533
Año 2025, Vol. 3, No. 2, pp. 62-74
JUL | DIC 2025
https://doi.org/10.36677/ia.v3n2.27650

Arquitectura y Sincronización de un Sistema de Monitoreo Térmico con Notificaciones Push mediante Aplicaciones Web Progresivas (PWA) empleando una Raspberry Pi

Javier Salas-García; Otniel Portillo-Rodríguez; Adriana H. Vilchis-González

Información del Artículo

Recibido: 1 de octubre, 2025
Aceptado: 30 de noviembre, 2025

Palabras clave: Raspberry Pi, Temperatura, PWA, Notificaciones, IoT, Python, PHP, MariaDB

Resumen

Este trabajo explica el desarrollo de un sistema de monitoreo térmico, denominado «Monitor térmico», basado en una Raspberry Pi y una Aplicación Web Progresiva (PWA). El objetivo central es superar las restricciones de los sistemas operativos móviles modernos que limitan la ejecución de procesos en segundo plano para sitios web convencionales. A diferencia de las aplicaciones nativas distribuidas por tiendas oficiales como App Store o Play Store, esta solución permite al usuario recibir notificaciones push proactivas en su dispositivo móvil incluso cuando la página web del sistema no se encuentra abierta. Se detalla la integración del sensor MAX6675, el almacenamiento persistente en MariaDB y la implementación del protocolo Web Push mediante llaves criptográficas VAPID para asegurar la entrega de alertas de sobrecalentamiento. El artículo ofrece una metodología que escala desde implementaciones locales gratuitas, abordando el manejo de direcciones IP dinámicas, hasta el despliegue en entornos de hosting profesional, proporcionando una alternativa robusta y segura para el monitoreo proactivo sin las barreras de los ecosistemas cerrados.

1. Introducción

Construir sistemas para la supervisión remota de dispositivos electrónicos suele ser costoso debido a la dependencia de plataformas propietarias o aplicaciones móviles complejas de desarrollar. Sin embargo, el uso de Tecnologías Web Progresivas (PWA) permite crear sistemas de comunicación bidireccional donde no solo se monitorea información en tiempo real, sino que también se pueden ejecutar acciones sobre un hardware remoto directamente desde la pantalla de un celular. El presente artículo utiliza el desarrollo de un controlador de temperatura, denominado de ahora en adelante como el «Monitor térmico», como un pretexto práctico para aterrizar conceptos de ingeniería de software y electrónica. El objetivo es demostrar que, mediante una Raspberry Pi y estándares web abiertos, es posible implementar una solución reproducible que va más allá de un simple sensor: es una arquitectura completa para el envío de notificaciones y el control remoto de procesos, eliminando los costos de licencias y las barreras de las tiendas de aplicaciones tradicionales.

A pesar del creciente interés académico en las aplicaciones web, existe una brecha notable entre la teoría y la ejecución práctica en la literatura científica actual. Diversas investigaciones se centran en revisiones exhaustivas de la arquitectura PWA [1], o en comparaciones de su potencial frente a aplicaciones nativas [2]. No obstante, estos trabajos suelen limitarse a describir el «qué» —es decir, las ventajas y la estructura teórica— sin profundizar en el «cómo» implementar estas soluciones de manera granular. La falta de guías detalladas que aborden

desde la configuración de llaves criptográficas VAPID hasta la sincronización física de sensores con lógica de control persistente representa un obstáculo para la comunidad científica y técnica. Este artículo se distingue al proporcionar una metodología de implementación rigurosa y completamente reproducible, resolviendo los retos de configuración técnica que son omitidos en las fuentes académicas tradicionales.

Para que el usuario cuente con un sistema de alerta proactivo, se diseñó el «Monitor térmico» como una Aplicación Web Progresiva (PWA). Esta elección tecnológica es clave para resolver el principal problema de los dispositivos móviles modernos: las estrictas restricciones de ahorro de energía y gestión de memoria que suspenden la actividad de los navegadores cuando una pestaña no está visible. El objetivo fundamental del sistema es garantizar que el usuario reciba notificaciones push incluso cuando la página web del monitor no se encuentra abierta en el dispositivo, logrando un comportamiento idéntico al de una aplicación industrial nativa pero sin la necesidad de depender de los procesos de revisión y costos de las tiendas oficiales (App Store o Play Store). Para asegurar la viabilidad técnica en diversos contextos, el artículo detalla tres rutas de despliegue: el uso de herramientas gratuitas para una implementación local básica, la contratación de servicios de túnel para obtener direcciones permanentes y la migración hacia un servidor de hosting para una solución robusta.

Para reproducir este proyecto, se deben obtener los archivos de código desde el repositorio oficial. El código completo se encuentra disponible en GitHub mediante la dirección `https://github.com/JavierSalasGarcia/notificaciones_PWA_raspberry`, el cual contiene toda la estructura necesaria para el funcionamiento del servidor web, la lógica de control en Python y los esquemas de la base de datos. En la Sección 5 se detalla exhaustivamente el procedimiento técnico para la obtención de estos archivos y su correcta ubicación dentro del sistema de archivos de la Raspberry Pi.

Este artículo explica cada detalle del proceso. Primero se describen las piezas que hacen posible que el navegador se comporte como una aplicación. Después se muestra cómo unir las piezas físicas con el código de manera segura, detallando la conexión del sensor y el manejo de los datos. También se explica cómo configurar el sistema para que los avisos lleguen correctamente a teléfonos iPhone y Android, y finalmente se detallan los pasos para que todo quede funcionando en internet de forma permanente.

2. La Web Moderna y las Aplicaciones Progresivas

En los últimos años, la forma en que se usa el internet ha cambiado mucho. Antes, los sitios web solo servían para leer información, y si se requería una aplicación que enviara avisos al celular, se tenía que descargar de una tienda oficial. Esto era un problema porque crear aplicaciones para diferentes tipos de teléfonos es tardado y difícil. Las Aplicaciones Web Progresivas (PWA) nacieron para solucionar esto. Una PWA es básicamente un sitio web que puede instalarse en el celular y funcionar casi igual que una aplicación normal [1].

Para que un sitio web se convierta en una PWA, se necesitan cuatro piezas principales que se encuentran en el código del «Monitor térmico». La primera pieza es el «Manifiesto», que es el archivo `Repositorio/public/manifest.json`. Este archivo es como una carta de presentación para el teléfono donde se escribe el nombre de la aplicación y se eligen los

colores y el icono que se verá en la pantalla de inicio. La segunda pieza es el «Service Worker», un programa en JavaScript ubicado en `Repositorio/public/sw.js` que corre en el fondo del teléfono para recibir alertas incluso si la página está cerrada [3]. La tercera pieza es la seguridad con HTTPS, que asegura que la información de temperatura viaje protegida y que el navegador confíe en los permisos otorgados.

La cuarta pieza fundamental, que no se encuentra directamente en el código descargado del repositorio, es la carpeta llamada «vendor». Esta carpeta es de gran importancia porque contiene todas las librerías externas que permiten que el sistema de notificaciones funcione. Se tiene que tener esta carpeta porque en ella están los programas que manejan el cifrado y la comunicación con los servidores de Google y Apple. Para crearla, se utiliza una herramienta llamada Composer que descarga automáticamente miles de archivos necesarios. No se incluye en el repositorio de GitHub debido a que la enorme cantidad de archivos que genera haría que la descarga fuera innecesariamente pesada y lenta. Sin la carpeta «vendor», un iPhone no tendría forma de detectar las funciones de envío de mensajes, por lo que su creación es un paso obligatorio para que el sistema sea profesional. Además, el «Monitor térmico» incluye una sección de ayuda en la página principal que guía al usuario con instrucciones claras para «instalar» la aplicación, diferenciando entre los pasos para sistemas Android y los requisitos específicos de los equipos de Apple.

3. Almacenamiento de Datos y Conexión de Hardware

Para que el sistema sea de utilidad, se deben conectar las piezas físicas con el código de computadora de manera segura. El cerebro del «Monitor térmico» es una Raspberry Pi 4 conectada a un sensor MAX6675 para medir la temperatura con gran precisión [4]. Para conectar estos dos componentes, se deben usar los pines de la Raspberry y el sensor mediante cables de puente, tal como se muestra en el diagrama de conexiones de la Figura 1. Es fundamental distinguir entre las dos formas de contar los pines en la Raspberry Pi. La primera es la numeración física o de placa (BOARD), que simplemente numera los pines del 1 al 40 según su posición física en dos columnas. La segunda es la numeración del procesador (BCM), la cual se basa en el nombre funcional de cada pin según el diseño interno del equipo. El «Monitor térmico» utiliza exclusivamente la numeración BCM porque es el estándar compatible con la mayoría de las librerías de programación profesionales. Para la conexión SPI del sensor MAX6675, se deben identificar los pines físicos: el Reloj (SCK) se conecta al pin BCM 11 (Pin físico 23), la salida de datos (SO) al pin BCM 9 (Pin físico 21) y la selección del chip (CS) al pin BCM 8 (Pin físico 24). La alimentación del sensor debe realizarse estrictamente con el pin de 3.3V (Pin físico 1) para no dañar los componentes, compartiendo la tierra común (GND) en el pin físico 9. Para identificar el pin número uno, se observa la esquina más alejada de los puertos USB, junto al pequeño foco rojo de encendido; a partir de ahí, los pines nones están en la fila interior y los pares en la fila exterior. La conexión física requiere precisión para evitar daños permanentes en la placa, por lo que se debe seguir un mapeo riguroso basado en la función de cada componente. En la Tabla 1 se presenta la relación exacta entre los pines del procesador y su ubicación física, facilitando el cableado del sensor y los actuadores.

Tabla 1: Mapeo de conexiones entre la Raspberry Pi y los componentes externos.

Dispositivo y Señal	Pin Físico (Raspberry Pi)	Pin BCM (Código)
Sensor MAX6675		
SCK (Reloj)	23	11
MISO (Datos)	21	9
CS (Selección)	24	8
VCC (3.3V)	1	-
GND (Tierra)	9	-
Actuadores		
LED de Control	12	18
Relevador (Calefactor)	16	23

Contar con este mapeo específico previene cortocircuitos eléctricos y asegura que el controlador de Python se comunique con la interfaz de hardware correcta desde el primer intento.

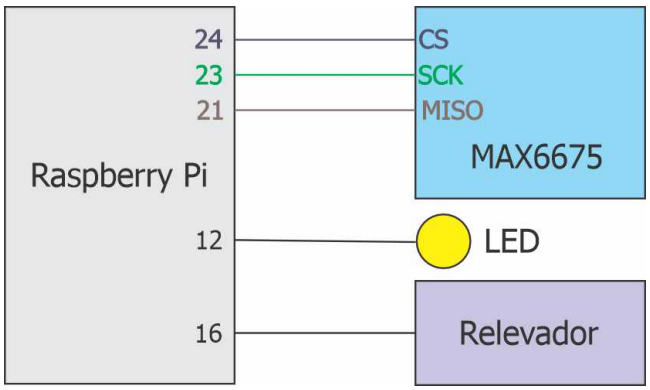

Figura 1: Diagrama de conexiones de pines entre el sensor, actuadores y Raspberry Pi (Elaboración propia).

Además del sensor, el sistema tiene dos salidas importantes operadas mediante pines GPIO. La primera va hacia un LED que sirve para probar que desde el celular se pueden enviar órdenes al sistema, verificando que la comunicación no es solo de lectura sino también de escritura de valores. La segunda salida va hacia un relevador que controla un elemento calefactor. En lugar del relevador, se puede usar otro LED para simular su funcionamiento durante las pruebas. El objetivo del «Monitor térmico» es controlar la temperatura de un recipiente que se debe mantener mucho más caliente que el ambiente. Por esta razón, cuando se alcanza la temperatura deseada, el sistema simplemente apaga el relevador y el recipiente se enfría solo por el contacto natural con el aire. Toda esta lógica requiere asegurar que los cables estén firmemente conectados, ya que un cable flojo podría enviar datos falsos que activarían el relevador de forma innecesaria o peligrosa.

La lógica de las notificaciones y el control de potencia es sumamente precisa para proteger el proceso sin molestar al usuario con avisos innecesarios. El sistema se basa en tres números configurables: la temperatura deseada, el umbral de operación y el offset de sobrecalentamiento. Por ejemplo, si se elige una temperatura deseada de 60 °C y un umbral de

2 °C, el sistema establece un rango aceptable de entre 58 °C y 62 °C. El «Monitor térmico» activará el relevador del calefactor en cuanto la temperatura baje de 58 °C y lo apagará únicamente cuando se alcancen los 62 °C, permitiendo que la temperatura baje de forma natural (ver Figura 2).

Para las alertas, se utiliza el tercer valor llamado offset. Si este fuera de 3 °C, el sistema sumará este valor al límite superior del rango aceptable. En el ejemplo anterior, el límite de peligro sería de 65 °C (60 + 2 + 3). Al llegar a esa temperatura crítica, se enviará una notificación de alerta al celular una sola vez para avisar sobre el sobrecalentamiento. El sistema mantendrá silencio hasta que la temperatura baje y vuelva a entrar en el rango de operación segura (58 °C a 62 °C), momento en el cual enviará un segundo aviso para informar que la situación se ha normalizado. Esta configuración de tres parámetros, detallada visualmente en la Figura 2, asegura un control industrial profesional y una comunicación eficiente con el usuario. Toda esta información se guarda en la base de datos MariaDB, que sirve como puente permanente entre el programa de Python (`main_controller.py`) y la página web en PHP (`dashboard.php`). Para que la comunicación sea exitosa, los archivos de Python `main_controller.py` y la librería `max6675.py` deben permanecer siempre en la misma carpeta, la cual se recomienda ubicar en la ruta personal del usuario para una ejecución sencilla.

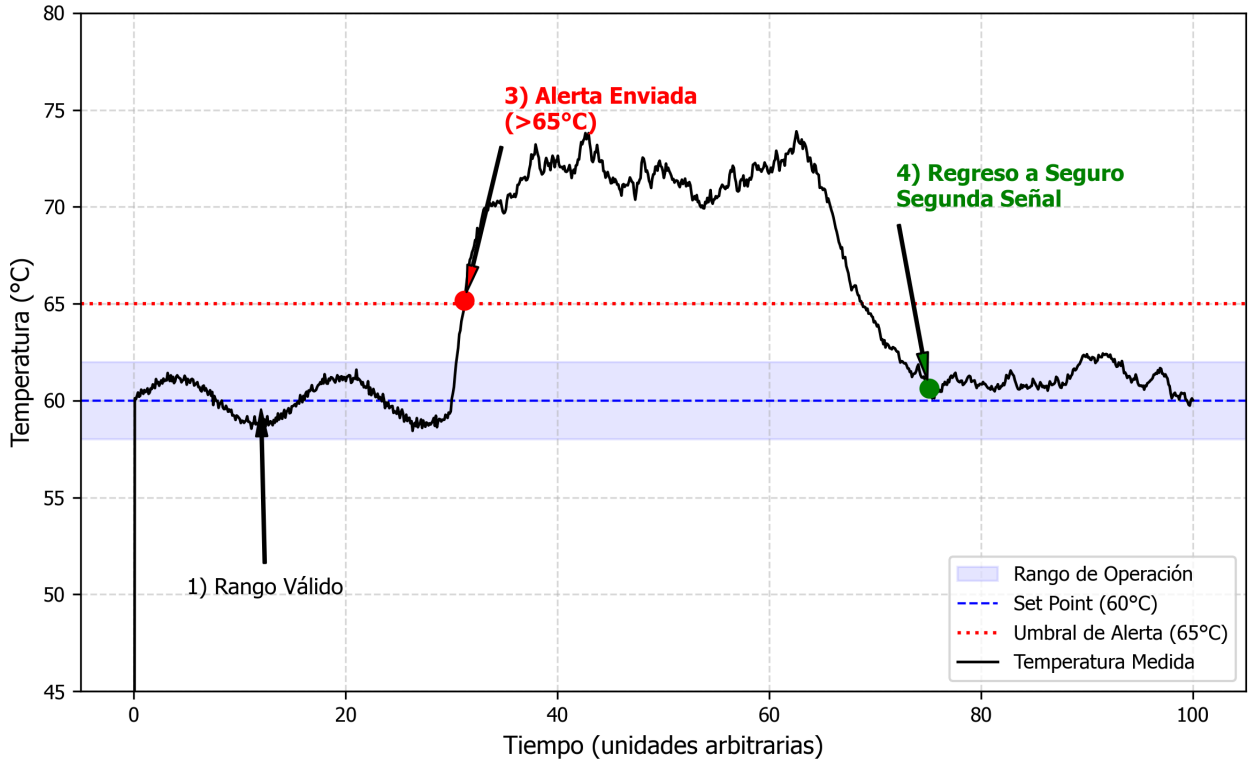

Figura 2: Dinámica de la lógica de control térmico e histéresis del sistema. Se ilustran los cuatro casos críticos: 1) operación en rango válido, 2) salida de umbral operativa, 3) cruce del umbral de alerta ($> 65°C$) con envío de notificación y 4) recuperación del sistema con envío de señal de normalización (Elaboración propia).

4. Seguridad y el Reto de los Teléfonos iPhone

Para que el «Monitor térmico» sea confiable, se utilizan las denominadas «llaves VAPID» [5]. Estas llaves son códigos secretos guardados en el archivo `config.php` (generado previamente a partir de su respectiva plantilla `.sample`) que permiten que el teléfono verifique que los avisos vienen del equipo autorizado. Es posible generar llaves propias usando herramientas en línea o comandos de terminal para asegurar que el sistema sea único y privado. Un requerimiento técnico fundamental para la seguridad es la generación de llaves VAPID únicas, las cuales actúan como la firma criptográfica exclusiva de las notificaciones. Para realizar este proceso paso a paso, se debe ejecutar en la terminal la orden `npx web-push generate-vapid-keys`. Una vez procesado, el sistema devolverá en pantalla dos bloques de texto: la «Public Key» y la «Private Key». La secuencia de configuración consiste en copiar el valor de la Public Key y pegarlo cuidadosamente dentro del archivo `sw.js` en la raíz de la web, así como en la variable correspondiente dentro de `dashboard.php`. Posteriormente, se debe tomar la Private Key y guardarla en el archivo de configuración `config.php`. Este procedimiento secuencial garantiza que solo el servidor autorizado pueda enviar mensajes a los teléfonos suscritos, impidiendo que terceros puedan suplantar la identidad del sistema. Además, los datos se protegen con cifrado de extremo a extremo, lo que significa que nadie puede leer la temperatura mientras viaja por internet [6]. Este nivel de seguridad es posible gracias a las librerías guardadas en la carpeta «vendor», las cuales permiten que el sistema cumpla con las reglas estrictas de los teléfonos modernos.

En el caso de los teléfonos iPhone, existe un reto especial debido a que Apple bloquea las notificaciones de sitios web automáticos para ahorrar batería [7]. Para que un iPhone permita recibir avisos del «Monitor térmico», el usuario debe realizar un proceso manual sencillo que se explica detalladamente en la opción de ayuda de la aplicación. Es fundamental que el usuario abra la dirección del sistema utilizando únicamente el navegador Safari, ya que otros exploradores como Chrome o Firefox no tienen permitido activar estas funciones en iOS. El proceso requiere tocar el botón de compartir (el icono del cuadrado con la flecha) y seleccionar la opción de agregar a la pantalla de inicio. Una vez que se abre la aplicación desde el nuevo icono creado, el usuario debe pulsar explícitamente una opción claramente identificada como «Activar Alertas» dentro de la configuración de la app para otorgar los permisos finales. Durante este proceso, el sistema detecta automáticamente si se trata de un equipo Android o iOS y guarda esta información en la tabla de suscripciones de MariaDB, lo que permite gestionar el envío de mensajes de forma más eficiente y personalizada.

5. Instalación del Sistema y Conexión a Internet

Para instalar el «Monitor térmico», primero se debe abrir la terminal de comandos en la Raspberry Pi. La secuencia de comandos comienza con la actualización del sistema mediante `sudo apt update`. Luego, se instalan los programas del servidor con el comando `sudo apt install apache2 mariadb-server php libapache2-mod-php php-mysql composer`. Para el control del hardware se requiere instalar las herramientas de Python con `sudo apt install python3-pip` y posteriormente asegurar la presencia de las li-

brerías de conexión mediante `pip install mysql-connector-python requests`. Finalmente, es obligatorio escribir `sudo raspi-config`, entrar a «Interfacing Options», seleccionar «SPI» y marcar «Yes» para activar la comunicación con el sensor. Antes de iniciar las pruebas del software, se recomienda confirmar que la comunicación física entre el procesador y el sensor está habilitada correctamente. Una vez realizada la activación en el menú de configuración del sistema y tras haber reiniciado el equipo, se debe ejecutar el comando `ls /dev/spi*` en la terminal. Si la activación fue exitosa, el sistema responderá mostrando nombres como `spidev0.0` o `spidev0.1`. En caso de no obtener respuesta, significa que el bus de datos no es visible para el sistema operativo y cualquier intento de correr el código de Python resultará en un error de hardware. Esta verificación sencilla permite descartar fallos de configuración básica antes de avanzar a etapas más complejas de programación.

Una vez que el servidor web ha sido instalado y la ruta base `/var/www/html` ha sido creada por el sistema, se debe proceder a la obtención y organización minuciosa de los archivos de código. Para asegurar que la estructura del sistema coincida exactamente con lo direccionado en el repositorio oficial, se debe abrir una terminal y ejecutar la descarga mediante la orden `git clone https://github.com/JavierSalasGarcia/notificaciones_-PWA_raspberry Repositorio`. En caso de optar por una descarga manual en formato ZIP, se debe utilizar el comando `unzip` para extraer el contenido. Una vez obtenida la carpeta `Repositorio`, es indispensable crear el directorio de destino para la interfaz web con el comando `sudo mkdir -p /var/www/html/monitor`. La correcta transferencia de los archivos se realiza moviendo únicamente el contenido de la subcarpeta `public` del repositorio hacia la nueva ruta del servidor mediante el comando `sudo cp -r Repositorio/public/* /var/www/html/monitor/`. De forma paralela, para que la lógica de control funcione sin errores de direccionamiento, la carpeta de Python debe ubicarse en la raíz del usuario mediante la orden `mv Repositorio/python /home/pi/`. Este procedimiento garantiza que tanto el servidor web como el controlador de hardware localicen sus archivos de configuración y datos en las rutas previstas, asegurando la integridad del sistema.

Para que la lógica de control funcione sin errores, se debe seguir la siguiente organización jerárquica de archivos:

```
/var/www/html/monitor/ (Interfaz Web)
  |-- config.php.sample (Plantilla de ajustes)
  |-- config.php (Archivo configurado)
  |-- dashboard.php (Panel de control)
  |-- vendor/ (Librerias externas)
  |-- sw.js (Service Worker)

/home/pi/python/ (Logica de Control)
  |-- main_controller.py.sample (Plantilla de logica)
  |-- main_controller.py (Cerebro configurado)
  |-- max6675.py (Libreria del sensor)
```

Esta separación es intencional para mantener el código de control fuera del alcance del servidor web público. Para evitar errores y facilitar la personalización, el repositorio no incluye archivos de configuración definitivos, sino plantillas denominadas «archivos sample». El primer paso para configurar la web es entrar a la carpeta del servidor y crear una copia

de la plantilla mediante el comando `cp config.php.sample config.php`. Una vez creada la copia, se debe abrir mediante el terminal utilizando el editor `nano`. Al ejecutar `sudo nano config.php`, se abrirá una interfaz con comentarios que guían al usuario sobre dónde capturar los datos de la base de datos y las llaves VAPID. Una vez realizados los cambios, la información se almacena presionando la combinación de teclas `Ctrl + O`, confirmando con la tecla `Entrar`, y se finaliza el proceso pulsando la combinación `Ctrl + X`.

Un paso obligatorio para habilitar las notificaciones es la creación de la carpeta «vendor» mediante el gestor de dependencias Composer. Debido a que el repositorio no incluye estas librerías por su gran volumen, el sistema web no funcionará hasta que se descarguen manualmente. Para realizar esta tarea, se debe utilizar la terminal para entrar al directorio del servidor escribiendo la orden `cd /var/www/html/monitor/`. Una vez dentro de la ruta correcta, se debe ejecutar la instrucción `composer install`. En caso de que el sistema devuelva un error de permisos, se debe anteponer la palabra `sudo` al comando anterior. Durante este proceso, se observará en la pantalla una lista de descargas; el programa lee automáticamente el archivo `composer.json` y obtiene las versiones exactas de las librerías de cifrado y protocolos push necesarias para comunicarse con los servidores de Google y Apple. Al finalizar, la carpeta «vendor» aparecerá en el directorio, permitiendo que las funciones de seguridad de la PWA se activen de inmediato.

El acceso continuo al «Monitor térmico», aunque cambie la dirección de internet de la casa, se logra mediante el DNS Dinámico. Normalmente, los proveedores de internet cambian el número de identificación (dirección IP) del módem cada pocos días. Si esto pasa, el usuario perdería la conexión con su Raspberry Pi desde fuera de casa. Para evitarlo, se utiliza un servicio que permite crear un nombre fijo como «mi-monitor.ddns.net». Se debe instalar el programa `ddclient` en la Raspberry con el comando `sudo apt install ddclient`. Este programa vigila constantemente si el número de IP cambió y, si lo hace, le avisa de inmediato al servidor de nombres para que el nombre fijo siempre dirija al usuario a su equipo de forma automática y transparente.

Para que el sistema sea visible en el navegador de cualquier dispositivo, se debe asegurar el correcto despliegue en el servidor web Apache. Tal como se detalló en el proceso de obtención de archivos anterior, la ubicación de los documentos en la ruta `/var/www/html/monitor/` es crítica para que Apache pueda servirlos correctamente. Esta organización permite que al escribir la dirección de la Raspberry seguida de la palabra «monitor», el navegador encuentre de inmediato el panel de control del sistema.

Un desafío técnico importante es que las notificaciones PWA exigen estrictamente el uso de conexiones seguras HTTPS, lo cual es difícil de configurar en la red de una casa. La diferencia fundamental reside en que HTTP (HyperText Transfer Protocol) envía la información en texto plano, lo que permite que sea interceptada, mientras que HTTPS (Protocolo Seguro de Transferencia de Hipertexto) utiliza certificados SSL/TLS para cifrar los datos del sensor y proteger la privacidad del usuario. Los navegadores modernos prohíben el uso de cámaras, ubicación y notificaciones push si no se detecta esta seguridad, a lo cual llaman «Contexto Seguro». Para solucionar este problema de forma profesional y gratuita, el «Monitor térmico» utiliza una herramienta llamada ngrok.

Esta utilidad fue seleccionada porque permite crear un túnel seguro desde internet hacia la Raspberry Pi sin necesidad de abrir puertos en el módem. Al activar la cuenta con el

comando `ngrok config add-authtoken` y ejecutar `ngrok http 80`, el sistema generará una dirección web especial que comienza con «https://». Es en este momento cuando el usuario podrá ver el candado verde en su navegador móvil; esto sucede porque ngrok actúa como un intermediario que ya posee los certificados de seguridad válidos que exigen los teléfonos Android e iPhone. Aunque existen otras herramientas como LocalTunnel o Cloudflare, se eligió ngrok por su gran estabilidad y facilidad de configuración para proyectos de desarrollo técnico.

El uso de ngrok en su modalidad gratuita conlleva una característica importante que debe comprenderse para mantener el acceso a largo plazo. Cada vez que el programa se detiene o se pierde la conexión a internet, la plataforma genera una dirección web completamente nueva y aleatoria. Esto implica que el enlace que se haya guardado o instalado en el teléfono dejará de funcionar de inmediato, mostrando un error de página no encontrada. Para recuperar el control, se debe observar la nueva dirección generada en la terminal de la Raspberry Pi y actualizarla manualmente tanto en los archivos de configuración del sistema como en el navegador del dispositivo móvil para volver a instalar el acceso directo.

Para escenarios donde se requiere disponibilidad permanente y profesional, existen alternativas que evitan el cambio constante de la dirección web. La opción más directa es utilizar una cuenta con nombre de dominio fijo, lo cual permite reservar una dirección que nunca cambiará a pesar de los reinicios del equipo. De igual manera, se pueden implementar soluciones como los túneles de Cloudflare, que operan de forma similar pero sobre un dominio propio registrado. Estas opciones transforman el proyecto de un prototipo local a una aplicación real capaz de operar durante meses sin intervención manual.

Para que la temperatura se mida de forma constante, se debe activar el programa cerebro. Siguiendo la misma lógica de orden, se debe entrar a la carpeta de Python y copiar la plantilla del controlador mediante `cp main_controller.py.sample main_controller.py`. Posteriormente, se debe editar mediante el comando `sudo nano main_controller.py`, asegurando que el nombre del servidor, usuario y clave coincidan plenamente con los de la interfaz web. Si esta información no es idéntica en ambos archivos, el sistema no podrá comunicar el estado real del sensor con el celular del usuario. Para asegurar que este control no se detenga por reinicios del equipo, se recomienda utilizar el programa de tareas automáticas crontab. Se debe abrir la configuración con `crontab -e` y añadir al final la línea `@reboot python3 /home/pi/python/main_controller.py`. Esta alternativa garantiza que el «Monitor térmico» inicie su vigilancia de forma autónoma.

6. Migración a Hosting Web y Escalabilidad

Para transformar el prototipo en una solución de nivel profesional, resulta conveniente migrar la arquitectura web hacia un servicio de hosting comercial. Esta transición ofrece ventajas significativas, como la obtención de una dirección IP fija y un nombre de dominio propio (por ejemplo, www.monitor-termico.com), eliminando la dependencia de herramientas de túnel temporal como ngrok y asegurando que las funciones de Aplicación Web Progresiva operen bajo certificados SSL/TLS nativos. El proceso de migración comienza con el respaldo de la información local. En la terminal de la Raspberry Pi, se debe ejecutar el comando de ex-

portación `mysqldump -u root -p pwa_monitor > respaldo.sql`; tras escribir la contraseña, se generará un archivo que contiene toda la estructura de tablas y los registros históricos. Este archivo debe descargarse a una computadora personal para su posterior carga en el servidor remoto.

Una vez obtenido el respaldo, se debe acceder al panel de control del hosting (comúnmente cPanel o Plesk) y buscar la sección dedicada a bases de datos MySQL. En esta interfaz, se tiene que crear una nueva base de datos y un usuario con una contraseña segura, asegurándose de asignar todos los privilegios al usuario sobre la base de datos recién creada. Posteriormente, se abre la herramienta phpMyAdmin del hosting, se selecciona la base de datos vacía en la barra lateral y se pulsa la pestaña superior de «Importar». Se debe elegir el archivo `respaldo.sql` y presionar el botón de ejecutar al final de la página. Si el proceso es correcto, el sistema mostrará un mensaje indicando que todas las tablas fueron creadas con éxito en la nube.

La transferencia del código se realiza mediante el administrador de archivos del hosting o un cliente SFTP como FileZilla. Una ventaja clave de esta arquitectura es su capacidad de escalabilidad; el sistema no requiere necesariamente ser instalado en la raíz del dominio principal. Es posible organizar múltiples unidades de monitoreo bajo un mismo dominio utilizando subdirectorios (por ejemplo, `dominio.com/sensor1/`) o subdominios específicos. Esto es posible gracias a que el alcance o «scope» del Service Worker se limita automáticamente a la carpeta donde se encuentra alojado, permitiendo que cada instancia opere de forma independiente sin interferir con otras instalaciones en el mismo servidor. Por lo tanto, una empresa puede gestionar una flota de dispositivos bajo una sola infraestructura web centralizada.

En este punto, es fundamental comprender que la mayoría de los servicios de hosting compartido no permiten ejecutar el comando Composer directamente en la nube. Por esta razón, se debe subir la carpeta «vendor» completa que se generó previamente en la Raspberry Pi hacia el directorio elegido en el servidor remoto, junto con el resto de los archivos y la carpeta `api`. Se debe verificar que todos los recursos de JavaScript y el archivo del Service Worker (`sw.js`) se ubiquen en la misma carpeta del proyecto para que el navegador pueda registrarlos con el alcance correcto. Inmediatamente después de la subida, es obligatorio editar el archivo `config.php` en el hosting para actualizar las credenciales de acceso a la base de datos, asegurando que la conexión apunte al nombre de servidor y usuario asignados por el proveedor.

Finalmente, la lógica de control en la Raspberry Pi debe reprogramarse para que el sistema hardware envíe la temperatura al nuevo servidor. Se debe editar el archivo `main_controller.py` (creado previamente a partir de la plantilla `.sample`) en la Raspberry mediante el comando `sudo nano /home/pi/python/main_controller.py` y localizar la línea que define la dirección del servidor (usualmente escrita como una variable URL). Se debe cambiar `http://localhost/monitor` por la dirección completa del dominio contratado, por ejemplo, `https://www.monitor-termico.com/monitor`, asegurando siempre el uso del protocolo seguro `https`. Una característica fundamental de este cambio es que ahora la Raspberry Pi actúa como un cliente que «empuja» los datos hacia internet, por lo que ya no importa si su dirección IP doméstica cambia constantemente. Para que el control de encendido y apagado siga funcionando, el script de Python realiza una consulta cada segundo a la base de datos del

hosting para verificar si el usuario ha presionado algún botón en la web, garantizando una sincronización perfecta a larga distancia sin configuraciones de red complejas en el módem.

7. Pruebas y Puesta en Marcha del Sistema

Para verificar que el «Monitor térmico» funciona correctamente, se debe realizar una secuencia de pruebas que comienza con la base de datos. Si se utiliza una herramienta visual como phpMyAdmin, el primer paso es entrar a la dirección local del servidor y crear una nueva base de datos llamada «pwa_monitor». Una vez creada, se debe usar la opción de importar para cargar el archivo `setup.sql` (ubicado originalmente en la carpeta `database` del repositorio descargado), el cual creará automáticamente todas las tablas necesarias. Es vital asegurarse de que los datos de acceso en el archivo `config.php` (validado tras su creación desde el archivo `.sample`) coincidan con el usuario y la contraseña configurados en el servidor MariaDB. Si se prefiere usar la línea de comandos para mayor seguridad, se puede crear el usuario y darle permisos escribiendo los comandos SQL `CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'contraseña';` y después `GRANT ALL PRIVILEGES ON pwa_monitor.* TO 'usuario'@'localhost';`. Por defecto, el sistema incluye un usuario administrador con el nombre «admin» y la contraseña «admin123». Por seguridad, se recomienda entrar a la nueva sección de gestión de usuarios ubicada en `users.php` para dar de alta a nuevos operadores y eliminar o cambiar la clave del administrador original. Desde esta interfaz, se pueden registrar múltiples cuentas, lo que permite que diferentes personas supervisen el sistema con sus propios accesos.

Una vez configurada la base de datos, se debe probar la conexión desde los dispositivos móviles (usando Safari en iOS). Al entrar por primera vez, el sistema detecta el dispositivo y permite la instalación en la pantalla de inicio, mostrándose una interfaz gráfica que irá guiando al usuario. Tras abrir la aplicación instalada, se debe pulsar el botón «Activar Alertas».

Durante este paso, es probable que el navegador pregunte si se desea permitir el envío de mensajes. Si por error se presiona el botón de bloquear, la aplicación dejará de recibir avisos. Para solucionar esto, se debe tocar el pequeño candado que aparece junto a la dirección web en el navegador móvil y seleccionar la opción de restablecer los permisos del sitio. Para verificar que el registro fue exitoso, se puede consultar la tabla «suscripciones» en phpMyAdmin; allí se deberá observar una nueva fila con los códigos de seguridad VAPID y el nombre del dispositivo detectado (Android o iOS). El sistema está diseñado para enviar una notificación a todos los dispositivos registrados al mismo tiempo, lo que permite que un mismo usuario reciba alertas en varios celulares o que un equipo de trabajo esté informado simultáneamente sobre cualquier sobrecalentamiento.

Es fundamental entender los riesgos de no gestionar correctamente las librerías con Composer. Si se intenta añadir una nueva función pesada y se olvida ejecutar el comando de instalación, el sistema experimentará fallos diversos. En un explorador web, esto se vería como una pantalla completamente blanca. Para diagnosticar este problema, se debe revisar el archivo de registro de errores en la ruta `/var/log/apache2/error.log`, donde se encontrará una descripción técnica del fallo, como la ausencia del archivo `autoload.php`. Para diagnosticar problemas visuales profundos, se puede presionar la tecla F12 en una computadora para

abrir las herramientas de desarrollo, donde en la pestaña llamada «Application» se podrá verificar si el Manifiesto y el Service Worker se cargaron sin errores. El «Monitor térmico» ha sido diseñado para funcionar de forma óptima con versiones de PHP 7.4 o superiores y con Python 3.x, por lo que es importante confirmar que se tienen instaladas estas ediciones para evitar fallos de compatibilidad en las funciones de cifrado. Cuando el proceso de instalación o la llegada de alertas no se comporta como se espera, se debe abrir el navegador en una computadora y pulsar la tecla F12. En la ventana que aparece, se debe acceder a la pestaña denominada «Application» y buscar en la barra lateral el menú de «Service Workers». Ahí se podrá confirmar visualmente si el archivo está en estado «Running» (en color verde). Si aparece cualquier advertencia en color rojo, indica un fallo en la estructura del código o en el certificado de seguridad que debe corregirse para que las funciones se activen.

8. Conclusiones

El desarrollo del «Monitor térmico» demuestra que es posible crear herramientas de supervisión industrial de alta calidad utilizando herramientas de código abierto y hardware accesible como la Raspberry Pi. El hallazgo principal de este trabajo radica en que la tecnología PWA ofrece una solución real para el envío de notificaciones proactivas en segundo plano, superando los bloqueos técnicos de los sistemas operativos móviles actuales que impiden que una web estándar entregue alertas si el usuario no mantiene la página abierta. Al integrar un Service Worker y el protocolo Web Push, se logra un sistema de vigilancia que informa al instante sobre eventos críticos sin obligar al desarrollador a someterse a las restricciones de las tiendas tradicionales de aplicaciones.

Uno de los hallazgos más importantes es que la seguridad no tiene por qué ser un obstáculo para la sencillez. Mediante el uso de llaves VAPID y el cifrado de datos, el sistema asegura que la información privada esté siempre protegida, cumpliendo con los estándares más exigentes de la web actual. Asimismo, el uso de herramientas de gestión como Composer y Pip facilita enormemente la instalación y el mantenimiento del software, permitiendo que el sistema sea fácil de replicar y actualizar. La solución propuesta para el problema de las direcciones de internet cambiantes asegura que el monitor esté siempre disponible, convirtiéndolo en una herramienta útil para el monitoreo de procesos críticos donde el sobrecalentamiento puede representar un riesgo serio.

Finalmente, se concluye que este enfoque de ingeniería no solo ahorra costos significativos, sino que también otorga una libertad total al desarrollador para adaptar el sistema a sus necesidades específicas. La metodología presentada permite escalar el proyecto desde un prototipo funcional basado en herramientas gratuitas, hasta un sistema de monitoreo de grado profesional alojado en la nube. La capacidad de controlar hardware desde un celular, medir variables en tiempo real y recibir alertas automáticas en cualquier parte del mundo abre un abanico de posibilidades para la automatización en talleres, laboratorios y hogares. El «Monitor térmico» se presenta como un ejemplo claro de cómo la unión de la electrónica moderna con los estándares web abiertos puede generar soluciones potentes, seguras y al alcance de todos para asegurar que los procesos importantes se mantengan siempre bajo control.

Direcciones de correo de los autores

Javier Salas-García: Universidad Autónoma del Estado de México jsalasg@uaemex.mx

Otniel Portillo-Rodríguez: Universidad Autónoma del Estado de México oportillo@uaemex.mx

Adriana H. Vilchis-González: Universidad Autónoma del Estado de México avilchisg@uaemex.mx

Referencias

[1] S. M. Gaikwad, V. Sharma, R. Verma y P. Jain, «Progressive Web App (PWA) - One Stop Solution for All Application Development Across All Platforms,» *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, vol. 5, n.º 2, 2019. DOI: 10.32628/cseit1952290

[2] A. Biørn-Hansen, T. A. Majchrzak y T. M. Grønli, «Progressive Web Apps: The Possible Web-native Unifier for Mobile Development,» en *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST)*, 2017, págs. 344-351. DOI: 10.5220/0006353703440351

[3] W3C. «Push API,» visitado 31 de dic. de 2025. dirección: https://www.w3.org/TR/push-api/

[4] S. Sudasna y S. Inthachot, «The IoT Based Temperature Monitoring and Air Inlet Optimization Controlling for Gasification Stove,» en *2019 IEEE International Conference on Electrical, Computing and Communication Technologies (ICECCT)*, 2019, págs. 1-5. DOI: 10.1109/ECTI-NCON.2019.8692283

[5] M. Thomson y P. Beverloo. «Voluntary Application Server Identification (VAPID) for Web Push,» visitado 31 de dic. de 2025. dirección: https://datatracker.ietf.org/doc/html/rfc8292

[6] M. Thomson. «Message Encryption for Web Push,» visitado 31 de dic. de 2025. dirección: https://datatracker.ietf.org/doc/html/rfc8291

[7] WebKit Team. «Web Push for Web Apps on iOS and iPadOS,» visitado 31 de dic. de 2025. dirección: https://webkit.org/blog/13878/web-push-for-web-apps-on-ios-and-ipados/