Compositional Function Networks: A High-Performance Alternative to Deep Neural Networks with Built-in Interpretability

Fang Li
Oklahoma Christian University
Computer Science Department
fang.li@oc.edu

Abstract

Deep Neural Networks (DNNs) deliver impressive performance but their black-box nature limits deployment in high-stakes domains requiring transparency. We introduce Compositional Function Networks (CFNs), a novel framework that builds inherently interpretable models by composing elementary mathematical functions with clear semantics. Unlike existing interpretable approaches that are limited to simple additive structures, CFNs support diverse compositional patterns-sequential, parallel, and conditional-enabling complex feature interactions while maintaining transparency. A key innovation is that CFNs are fully differentiable, allowing efficient training through standard gradient descent. We demonstrate CFNs' versatility across multiple domains, from symbolic regression to image classification with deep hierarchical networks. Our empirical evaluation shows CFNs achieve competitive performance against black-box models (96.24% accuracy on CIFAR-10) while outperforming state-of-the-art interpretable models like Explainable Boosting Machines. By combining the hierarchical expressiveness and efficient training of deep learning with the intrinsic interpretability of well-defined mathematical functions, CFNs offer a powerful framework for applications where both performance and accountability are paramount.

1 Introduction

The past decade has witnessed an unprecedented surge in machine learning capabilities, driven largely by Deep Neural Networks (DNNs) (LeCun, Bengio, and Hinton 2015). DNNs have revolutionized fields from computer vision (Krizhevsky, Sutskever, and Hinton 2012) to natural language processing (Vaswani et al. 2017), often achieving superhuman performance through their ability to learn intricate, hierarchical representations from vast datasets.

However, DNNs' remarkable performance comes at a significant cost: inherent opacity. Their complex, non-linear transformations make it exceedingly difficult to understand why a particular decision is made or how specific inputs influence outputs. This "black-box" problem poses substantial challenges in domains requiring transparency and accountability, such as healthcare, finance, and autonomous systems (Gunning and Aha 2019).

We propose Compositional Function Networks (CFNs) as a novel machine learning paradigm that addresses the interpretability challenge while maintaining the capacity to model complex relationships. CFNs are built upon explicit function composition, constructing sophisticated mappings by systematically combining elementary, interpretable function nodes (e.g.. Gaussian radial basis functions, linear transformations, or sinusoidal waves).

CFNs overcome traditional interpretable models' limitations through flexible composition operators-sequential, parallel, and conditional to build hierarchical structures capable of representing highly complex functions. The parameters learned by each function node retain their semantic meaning, allowing direct interpretation of the model's internal logic. This interpretability is not an afterthought but an intrinsic property of the CFN architecture.

Our contributions are threefold:

1. We introduce the formal framework of CFNs, detailing their interpretable function nodes, composition mechanisms, and theoretical underpinnings.
2. We develop a taxonomy of CFN architectural patterns enabling domain-specific adaptations while preserving interpretability, from symbolic regression to deep hierarchical structures for computer vision.
3. We empirically demonstrate that CFNs achieve competitive performance with black-box models while maintaining interpretability advantages, with notable computational efficiency even in CPU-only implementations.

The remainder of this paper is organized as follows: Section 2 positions our work within interpretable machine learning literature. Section 3 introduces CFNs' foundational components and training methodology. Section 4 explores diverse architectural patterns enabled by CFNs. Section 5 presents experimental results, including benchmarks and case studies. Finally, Section 6 discusses implications, limitations, and future research directions.

2 Related Work

The quest for interpretable machine learning has generated diverse approaches that we organize into three categories: post-hoc explanation methods, inherently interpretable models, and compositional approaches.

2.1 Post-hoc and Inherently Interpretable Methods

Post-hoc methods explain black-box models without modifying their structure. These include LIME (Ribeiro, Singh, and Guestrin 2016), which approximates complex models with simpler ones around specific predictions, SHAP (Lundberg and Lee 2017), which assigns feature importance values using game theory, and gradient-based attribution methods (Selvaraju et al. 2017). While valuable, these approaches provide approximations rather than true insights into the model's decision-making process and can sometimes produce misleading explanations (Adebayo et al. 2018).

In contrast, inherently interpretable models are transparent by design. Traditional approaches include linear models and decision trees, which offer transparency but limited expressiveness. More advanced models include Explainable Boosting Machines (EBMs) (Nori et al. 2019) and Neural Additive Models (NAMs) (Agarwal et al. 2021), which implement Generalized Additive Models (GAMs) using gradient boosting and neural networks, respectively. While powerful, these models are fundamentally additive and may struggle with complex feature interactions. We benchmark against both EBMs and XGBoost (Chen and Guestrin 2016), the latter representing the upper bound of black-box model performance on structured data.

2.2 Compositional and Symbolic Approaches

Several approaches share philosophical similarities with CFNs. Symbolic regression via genetic programming (Koza 1992) discovers explicit mathematical formulas but can be computationally intensive. Neural-Symbolic systems (Garcez and Lamb 2022) combine neural learning with symbolic reasoning but often involve complex integration challenges. Sum-Product Networks are deep probabilistic models with less semantically rich components than the diverse mathematical functions in CFNs.

Recent efforts to improve deep learning transparency include Concept Bottleneck Models (Koh et al. 2020), which force networks to learn through human-defined concepts, and Network Dissection (Bau et al. 2017), which aligns hidden units with semantic concepts. However, these approaches typically operate on traditional neural architectures without fundamentally changing their black-box nature.

Our proposed CFNs offer a novel synthesis: they combine the hierarchical learning capabilities of DNNs with the intrinsic interpretability of well-defined mathematical functions within an efficient, gradient-based optimization framework. Unlike GAMs, they are not restricted to additive modeling. Unlike symbolic regression, they scale to higher dimensions. And unlike many neural-symbolic systems, they are trained end-to-end seamlessly.

3 Methodology

The CFN framework is built upon the principle of constructing complex functions $F(x)$ through the systematic composition of elementary, interpretable function nodes. This section details the core components, theoretical framing, and training of CFNs.

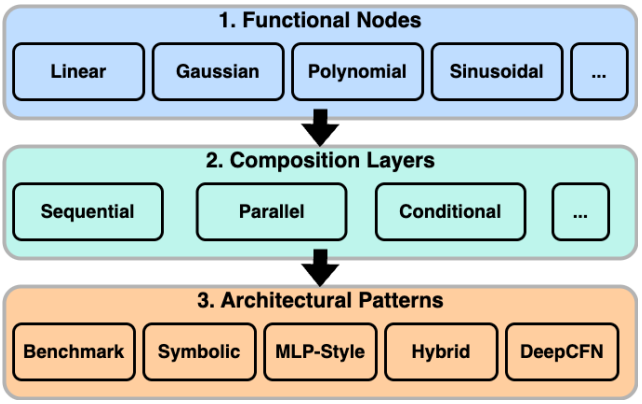

Figure 1: Overview of the Compositional Function Network framework.

3.1 Framework Overview

At a high level, a CFN consists of three key components:

1. **Function Nodes**: Elementary mathematical operations with interpretable parameters.
2. **Composition Layers**: Mechanisms for combining function nodes in different ways.
3. **Architecture**: The overall structure that defines how the layers are arranged.

The power of CFNs comes from their ability to build complex functions through composition while maintaining the interpretability of each component. Figure 1 provides a visual overview of the framework.

3.2 Theoretical Foundation

A CFN is defined by its architecture and a set of parameters Θ. The goal of training is to find the optimal parameters Θ that minimize a given loss function L over a dataset (X, y), potentially with a regularization term $R(\Theta)$ to penalize complexity:

$$\Theta = \arg\min_{\Theta} \mathbb{E}_{(x,y)\sim(X,y)}[L(y, F(x;\Theta))] + \lambda R(\Theta) \quad (1)$$

The expressive power of CFNs stems from their compositional nature. By including function nodes like polynomials or Gaussians, which are known universal function approximators, and combining them through flexible operators, CFNs can theoretically approximate any continuous function to an arbitrary degree of accuracy, similar to RBF networks or multi-layer perceptrons.

To illustrate the concept, consider a simple regression problem: approximating $y = \sin(x) + x^2$. A CFN could model this as:

$$F(x) = f_{sin}(x) + f_{poly}(x) \quad (2)$$

where f_{sin} is a Sinusoidal node with parameters close to $A = 1, \omega = 1, \phi = 0$ and f_{poly} is a Polynomial node with parameters close to $a_2 = 1, a_1 = 0, a_0 = 0$. Unlike a neural network, where the internal representations would be difficult to interpret, the CFN directly learns the interpretable parameters of these mathematical functions.

3.3 Function Nodes

At the heart of CFNs are a library of fundamental function nodes, each representing a distinct mathematical operation $f(x;\theta)$ with interpretable parameters θ. Our framework provides two complete implementations—PyTorch and NumPy—with the PyTorch version offering a particularly rich set of specialized nodes.

Basic Function Nodes Both implementations include these foundational nodes:

- **Linear:** $f(x) = Wx+b$, mapping inputs through a linear transformation.
- **Gaussian:** $f(x) = \exp(-\frac{|x-c|^2}{2w^2})$, producing localized activations around center c with width w.
- **Sigmoid:** $f(x) = (1 + \exp(-s(x \cdot d + o)))^{-1}$, creating smooth transitions along direction d.
- **Exponential:** $f(x) = A\exp(bx + c)$, where scale factor A, rate b, and offset c are learnable parameters. These nodes are valuable for modeling growth and decay processes, including compound effects.
- **Polynomial:** $f(x) = \sum_{i=0}^{D} a_i(x \cdot d)^i$, modeling polynomial relationships along direction d.
- **Sinusoidal:** $f(x) = A\sin(\omega(x \cdot d) + \phi)$, capturing oscillatory patterns with interpretable frequency, amplitude, and phase.
- **ReLU:** $f(x) = \max(0, x)$, providing standard nonlinear activation.

Advanced Function Nodes The PyTorch implementation extends this foundation with specialized nodes:

- **Image-Processing Nodes:**
  - **Fourier:** Captures frequency components in images through a learnable Fourier basis.
  - **Gabor:** Implements Gabor filters with learnable frequency, orientation, and scale parameters.
  - **SharedPatch:** Applies a shared function node to image patches, similar to convolutional operations but with greater flexibility.
- **Deep Learning Nodes:**
  - **GenericConv:** A convolutional operation with learnable filters, used in the DeepCFN architecture.
  - **Pooling:** Performs average or max pooling for spatial dimensionality reduction.
  - **Dropout:** Implements stochastic regularization to prevent overfitting.
- **Wrapper Nodes:**
  - **Sequential Wrapper:** Encapsulates a sequence of function nodes as a single node, enabling hierarchical model construction.

Each node is implemented to support automatic differentiation—through PyTorch's autograd system in the PyTorch implementation, or through custom gradient computation in the NumPy implementation. Careful initialization is key to training stability; for instance, Linear node weights use He initialization, while other parameters are set to sensible defaults based on their mathematical properties.

The diversity of available function nodes enables CFNs to model a wide range of mathematical relationships. For instance, in image processing tasks, the Gabor and Fourier nodes can capture specific visual patterns with interpretable parameters that directly correspond to visual features like orientation and frequency. This rich function library allows domain experts to select nodes that align with the expected mathematical structure of their problem, providing a strong inductive bias while maintaining interpretability.

3.4 Composition Layers

The power of CFNs stems not just from individual function nodes but from how they interact through composition layers. These layers organize function nodes into meaningful computational structures while maintaining interpretability.

Sequential Composition Sequential composition provides the most straightforward approach, where functions are applied in a chain: $F(x) = f_n(\dots f_1(x)\dots)$. This mirrors the layer-by-layer processing found in traditional neural networks but with each transformation being semantically meaningful. For example, a sequential layer might first extract polynomial features and then apply a sigmoid transformation to produce probability outputs. Our implementation ensures proper dimension checking between adjacent functions, maintaining a clean data flow throughout the computational pipeline.

Parallel Composition Parallel composition enables multiple function nodes to process the same input simultaneously, creating rich feature representations. The outputs can be combined through various operations:

- **Sum**: For additive models where each function contributes independently
- **Product**: To capture interaction effects between different feature transformations
- **Concatenation**: When preserving all outputs separately is desired
- **Weighted Sum**: Where the importance of each function's output is learned during training

This approach is particularly valuable for modeling complex dependencies while maintaining the ability to assess each component's contribution. Our implementation enforces appropriate dimensional constraints based on the combination method, ensuring that operations like summation only occur between compatible outputs.

Conditional Composition The most sophisticated composition mechanism is conditional composition, which implements a mixture-of-experts approach. For a set of condition nodes $\{c_i\}$ and expert nodes $\{g_i\}$, the output is:

$$F(x) = \sum_{i=1}^{N} \frac{c_i(x)}{\sum_{j=1}^{N} c_j(x) + \epsilon} \cdot g_i(x) \tag{3}$$

Here, each condition node produces a non-negative weight (typically via sigmoid activation), and these weights are normalized to sum to one. A small epsilon term ($\epsilon = 10^{-10}$) is added to the denominator to prevent numerical instability. The condition nodes act as "selectors" that determine which experts should handle the input. This creates an adaptive model that can apply different strategies across

the input space-for instance, using linear approximations in simple regions and more complex functions in others.

Through these composition mechanisms, CFNs can represent highly sophisticated functions while preserving a clear computational graph. Each mechanism creates a different pattern of information flow, allowing model designers to construct architectures that match their domain knowledge about the underlying problem structure.

3.5 Training Process

CFNs are trained using standard gradient-based optimization, managed by dedicated Trainer classes in both our PyTorch and NumPy implementations. Our training implementations include several essential features to ensure stable and effective learning:

- **Optimizer Selection:** Both implementations support Adam optimization with configurable hyperparameters. The NumPy implementation also offers a simpler SGD option.
- **Gradient Clipping:** To ensure numerical stability, especially with nodes like Exponential or Polynomial that can produce large gradients, both implementations support gradient norm clipping.
- **Learning Rate Scheduling:** Step-based learning rate decay is implemented to fine-tune convergence.
- **Regularization:** The NumPy implementation explicitly supports L2 regularization, while the PyTorch implementation leverages the built-in weight decay option in the Adam optimizer.
- **Early Stopping:** To prevent overfitting, both implementations include patience-based early stopping that saves the best model weights.
- **Loss Functions:** The PyTorch version accepts any PyTorch loss function, while the NumPy implementation provides custom implementations of common loss functions with appropriate gradient calculations.
- **Performance Monitoring:** Both trainers track training and validation metrics throughout the learning process.

For image-based tasks, the PyTorch trainer automatically handles reshaping of inputs, detecting 4D tensors (batches of images) and flattening them appropriately for processing by the network.

3.6 Implementation Details

We developed two complementary implementations of the CFN framework:

- **PyTorch Implementation:** Our primary implementation leverages PyTorch's automatic differentiation, GPU acceleration, and neural network primitives. This version provides seamless integration with modern deep learning workflows and enables the creation of complex architectures like DeepCFN. Each function node and composition layer is implemented as a subclass of nn.Module.
- **NumPy Implementation:** We also developed a lightweight, CPU-only implementation using NumPy. This version includes custom implementations of gradient-based optimization with manually implemented backpropagation for each function node. This approach enables detailed inspection of the gradient flow and makes it possible to implement custom optimization techniques.

Both implementations share the same conceptual foundation and node library, following a factory pattern that allows new node types to be registered and instantiated dynamically. The key difference lies in how gradients are computed: the PyTorch version relies on autograd, while the NumPy version explicitly implements forward and backward passes for each node type.

For all experiments, we ensure that model architectures and hyperparameters remain identical across implementations, with the only differences being in the underlying computational framework. This approach allows us to validate the correctness of both implementations while leveraging their complementary strengths.

4 CFN Architectural Patterns

The CFN framework's flexibility allows it to be configured into different architectural patterns for specific problem domains. This section presents a taxonomy of reusable patterns identified through our experimental work.

4.1 Pattern Taxonomy

Tabular Data Pattern Optimizes CFNs for structured tabular data with well-defined features.

- **Structure**: ParallelCompositionLayer with diverse function nodes $\rightarrow$ SequentialCompositionLayer for output transformation.
- **Key Characteristics**: Feature extraction through parallel transformation, followed by simple aggregation.
- **Results**: Achieves 98.4% and 100% accuracy on Breast Cancer and Wine datasets while maintaining full interpretability.

Symbolic Regression Pattern Discovers mathematical relationships in data with known underlying structure.

- **Structure**: Targeted function nodes in simple compositions with regularization favoring simplicity.
- **Key Characteristics**: Prioritizes parameter recovery and explicit mathematical forms.
- **Results**: Accurately recovers sinusoidal patterns and parameters from noisy data.

Mixture of Experts Pattern Models complex functions with region-specific behaviors using conditional composition.

- **Structure**: ConditionalCompositionLayer where condition nodes act as gates and function nodes as specialized experts.
- **Key Characteristics**: Partitions input space into distinct regions with specialized functions for each.
- **Results**: Successfully models spatially varying functions by applying different mathematical forms to distinct regions.

Deep Hierarchical Pattern Extends CFNs to handle complex high-dimensional data like images.

- **Structure**: Hierarchical composition of specialized nodes in a ResNet-inspired architecture.
- **Key Characteristics**: Maintains interpretability while incorporating spatial awareness.
- **Results**: Achieves 93.79% accuracy on CIFAR-10 while preserving node-level interpretability.

Hybrid Pattern Balances interpretability with state-of-the-art performance techniques.

- **Structure**: Core CFN nodes augmented with selected neural network components at computational bottlenecks.
- **Key Characteristics**: Maintains component-level interpretability while incorporating performance-enhancing elements.
- **Results**: Achieves 96.24% accuracy on CIFAR-10 while preserving key interpretability benefits.

4.2 Pattern Selection Guidance

Based on our experiments, we recommend:

- For maximum interpretability: Use the Tabular Data Pattern.
- For physical systems or known mathematical relationships: Use the Symbolic Regression Pattern.
- For functions with region-specific behaviors: Use the Mixture of Experts Pattern.
- For complex perceptual data: Use the Deep Hierarchical Pattern.
- When balancing performance and interpretability: Use the Hybrid Pattern.

These patterns demonstrate how CFNs can be systematically tailored to different problem domains while maintaining their core interpretability principles.

5 Experiments and Results

In this section, we evaluate the performance of CFNs on a range of tasks and datasets, from simple synthetic problems to complex real-world benchmarks.

5.1 Experimental Setup

Datasets We evaluate CFNs on the following datasets: Synthetic Datasets:

- Advanced Regression: A 2D function with different behaviors in concentric regions.
- Spiral Classification: A classic non-linearly separable dataset.
- Physics-informed Regression: Data from $x(t) = A\sin(\omega t + \phi)$.

Real-World Datasets:

- Breast Cancer Wisconsin: Binary classification (569 samples, 30 features)(Dua and Graff 2019).
- Wine Recognition: Multi-class classification (178 samples, 13 features)(Dua and Graff 2019).
- Diabetes: Regression (442 samples, 10 features)(Efron et al. 2004).
- CIFAR-10: Image classification (60,000 32x32 color images in 10 classes)(Krizhevsky 2009).

Data is split into 80% training and 20% validation sets and standardized where appropriate.

Computing Infrastructure All experiments were conducted on a consistent hardware and software platform to ensure fair comparisons. The specifications of the system used are as follows:

- CPU: Intel Core i9-9900K @ 3.60GHz
- GPU: NVIDIA GeForce RTX 3090
- RAM: 32GB DDR4
- Operating System: Ubuntu 24.04 LTS

The PyTorch models were run using CUDA version 11.7.

Models and Training We compare four models:

- Manual CFN (PyTorch): A hand-designed CFN with a generic architecture implemented in PyTorch, leveraging GPU acceleration.
- Manual CFN (NumPy): The identical CFN architecture implemented in our lightweight NumPy framework, running on CPU only.
- XGBoost: A state-of-the-art black-box gradient boosting model.
- EBM: A state-of-the-art interpretable additive model.

For statistical robustness, all benchmark results are averaged over 5 independent runs with different random seeds.

5.2 Benchmark Results

Tabular Data Performance The results show that CFNs are competitive with or exceed the performance of established models. For instance, on the Breast Cancer dataset, the PyTorch-based CFN achieves the highest accuracy and AUC. Notably, our NumPy-based CFN significantly outpaces its PyTorch counterpart running on a GPU for these datasets. This suggests that for datasets of this scale, the overhead of GPU data transfer can outweigh the benefits of parallel computation. Our streamlined NumPy implementation proves more efficient, highlighting that CFNs do not force a trade-off between performance and transparency.

Hardware Efficiency Implications The CPU efficiency of CFNs has broader implications. By leveraging mathematically expressive function nodes rather than many simple neurons, CFNs create more computation-efficient models. Each node performs meaningful operations that would require dozens or hundreds of traditional neurons to approximate. This efficiency has implications for:

- Edge Computing: CFNs can deliver sophisticated modeling on resource-constrained devices without GPU acceleration.
- Sustainable AI: The reduced computational footprint aligns with growing concerns about the environmental impact of Al development.
- Democratization: Reducing dependency on specialized hardware makes advanced machine learning more accessible.

5.3 Illustrative Case Studies

To demonstrate the practical applications and interpretability of CFNs, we present case studies. Each highlights a different strength of the framework.

Case Study 1: Discovering Physical Laws from Data Problem Description: A fundamental challenge in science

Breast Cancer	**CFN (PyTorch/GPU)**	**CFN (NumPy/CPU)**	**XGBoost**	**EBM**
Accuracy	0.987 (±0.004)	0.984 (±0.007)	0.956 (±0.000)	0.974 (±0.000)
AUC	0.998 (±0.001)	0.996 (±0.001)	0.991 (±0.000)	0.996 (±0.000)
Time (s)	1.992 (±0.129)	0.667 (±0.206)	0.209 (±0.000)	60.354 (±0.000)
Wine	**CFN (PyTorch/GPU)**	**CFN (NumPy/CPU)**	**XGBoost**	**EBM**
Accuracy	1.000 (±0.000)	1.000 (±0.000)	0.944 (±0.000)	1.000 (±0.000)
AUC	1.000 (±0.000)	1.000 (±0.000)	1.000 (±0.000)	1.000 (±0.000)
Time (s)	0.455 (±0.226)	0.358 (±0.179)	0.050 (±0.000)	6.978 (±0.000)
Diabetes	**CFN (PyTorch/GPU)**	**CFN (NumPy/CPU)**	**XGBoost**	**EBM**
RMSE	59.114 (±4.090)	51.194 (±0.580)	57.888 (±0.000)	51.441 (±0.000)
Time (s)	1.161 (±0.299)	0.430 (±0.067)	0.060 (±0.000)	5.961 (±0.000)

Table 1. Benchmark Results on Real-World Tabular Datasets.

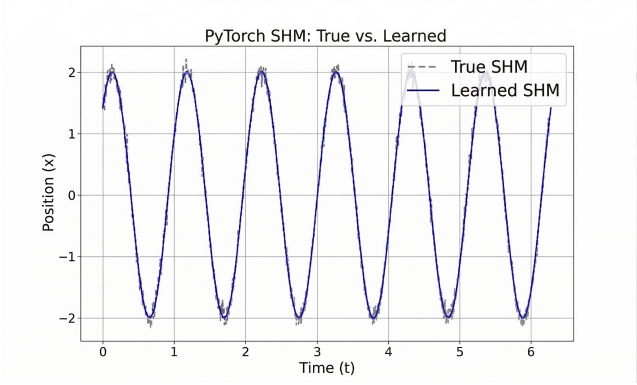

Figure 2: True vs. Learned Simple Harmonic Motion.

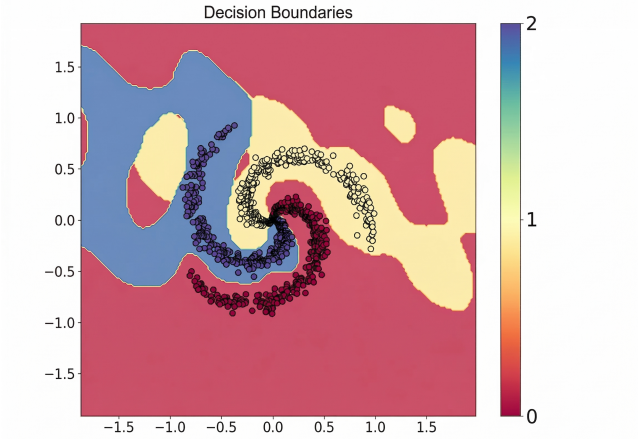

Figure 3: Decision boundary for the spiral dataset.

is to discover the underlying mathematical laws from observational data. We use a synthetic dataset generated from the Simple Harmonic Motion (SHM) equation: $x(t) = A\sin(\omega t + \phi)$, with known parameters ($A = 2.0$, $\omega = 1.5$, $\phi = \pi/4$) and added Gaussian noise.

CFN Architecture: We constructed a CFN with a single SinusoidalFunctionNode within a SequentialCompositionLayer. The model takes time t as input and learns the three interpretable parameters of the sine function: amplitude (A), angular frequency (ω), and phase (ϕ).

Results and Interpretation: After training, the CFN accurately learned the underlying function with parameters:

- Amplitude (A): 1.9988 (True: 2.0)
- Frequency (ω): 1.4999 (True: 1.5)
- Phase (ϕ): 0.7864 (True: $\pi/4 \approx 0.7854$)

This case study exemplifies the power of CFNs for scientific discovery, providing a transparent model that can be validated against domain knowledge.

Case Study 2: Classifying Non-Linear Data with Interpretable Features Problem Description: This case study uses the classic "spiral" dataset to demonstrate how CFNs can model complex decision boundaries while maintaining interpretability.

CFN Architecture: We designed a CFN with a three-stage architecture: a ParallelCompositionLayer with 12 different function nodes (Gaussian, Sigmoid, Sinusoidal, and Polynomial) for feature engineering, followed by a SequentialCompositionLayer with Linear and Sigmoid nodes, and a final SequentialCompositionLayer with a Linear node for classification.

Results and Interpretation: The CFN achieved 99.67% accuracy on the test set. By examining the learned parameters of the function nodes, we can understand how the model transforms the input space to make it linearly separable-a feature absent in black-box models.

Case Study 3: Interpretable Classification on Tabular Data Problem Description: The Breast Cancer Wisconsin dataset involves 30 features computed from digitized images of breast masses, with the goal of classifying a mass as benign or malignant.

CFN Architecture: We employed a CFN with a ParallelCompositionLayer for feature engineering, including a variety of function nodes to extract different types of features,

Model Architecture	Key Innovation	Accuracy
CFN-Conv	Hierarchical Conv Nodes	85.11%
DeepCFN	Optimized Function Composition	93.79%
Hybrid DeepCFN	CFN-NN Integration	96.24%

Table 2. Performance evolution on CIFAR-10.

followed by a SequentialCompositionLayer with a Linear node for classification.

Results and Interpretation: The CFN achieved 99.12% accuracy (it is a separate experiment to the benchmarking) on the test set. The interpretability stems from its architecture-by examining the learned parameters, one can understand which feature ranges or combinations are most indicative of malignancy.

5.4 Scaling to Complex Tasks

For the CIFAR-10 dataset, we developed a series of CFN architectures. Table 2 shows the evolution of our approach.

Our initial CNN-style implementation achieved 85.11% accuracy. The ResNet-styled DeepCFN reached 93.79%, effectively closing the gap with typical ResNet implementations while maintaining interpretability advantages. Most significantly, our hybrid DeepCFN architecture achieves 96.24% accuracy, competitive with state-of-the-art models like WideResNet-28 (Zagoruyko and Komodakis 2016) (96.00%) while preserving core interpretability benefits. This demonstrates that the apparent trade-off between interpretability and performance can be overcome through thoughtful integration of CFN principles with selected neural network techniques.

Case Study 4: High-Performance Image Classification with a Hybrid Deep CFN Problem Description: To test the limits of the CFN framework, we tackled the CIFAR-10 image classification benchmark.

CFN Architecture: We developed the Hybrid DeepCFN architecture, which combines the interpretability of CFNs with modern deep learning techniques. Built on a ResNet-18 equivalent backbone using our specialized nodes, it integrates batch normalization, attention mechanisms (SE blocks), advanced regularization, and width scaling.

Results and Interpretation: The Hybrid DeepCFN achieved 96.24% accuracy on CIFAR-10, competitive with highly-optimized black-box models. While node-level interpretability is partially abstracted by components like SE blocks, the model retains a clear compositional structure, allowing researchers to analyze the flow of information-a level of transparency unavailable in traditional deep learning models.

5.5 Discussion

Strengths and Implications CFNs offer a unique combination of performance and transparency, allowing domain experts to engage directly with the model. Their modularity enables the explicit encoding of prior knowledge, a significant advantage in data-scarce domains. The hardware efficiency also contributes to more energy-efficient AI development.

Limitations Despite their strengths, CFNs face challenges:

- Interpretability-Performance Trade-off: For complex tasks like image classification, achieving state-of-the-art accuracy sometimes requires components that abstract away some direct interpretability, though structural interpretability is maintained.
- Architectural Design: The performance of a CFN depends on selecting appropriate function nodes and architectural patterns. If the underlying function lacks a clear compositional structure or if the library of function nodes lacks optimal primitives, the inductive bias may be too restrictive.

Managing these trade-offs is a key consideration for practitioners and an important direction for future research.

6 Conclusion

This paper introduced Compositional Function Networks (CFNs), a framework for building high-performance, interpretable models through compositions of elementary mathematical functions. Our work makes three key contributions: formalizing the CFN framework, developing architectural patterns for domain-specific adaptations, and demonstrating competitive performance with inherent interpretability.

Our taxonomy of architectural patterns showcases CFNs' flexibility across domains-from discovering physical laws through symbolic regression to handling complex image classification with hierarchical networks. Despite their interpretability advantages, our empirical results show CFNs achieve performance comparable to black-box models while maintaining computational efficiency even without GPU acceleration.

By providing transparency by design, CFNs offer a compelling alternative to opaque models in critical domains where understanding model decisions is paramount. This approach represents a significant step toward trustworthy Al systems that can be readily inspected, understood, and validated.

6.1 Future Research Directions

The CFN framework opens several promising avenues for future research:

Theoretical Foundations Further work is needed to formalize the expressive power of different CFN configurations and establish theoretical guarantees on their approximation capabilities.

Architecture Search for CFNs Developing efficient methods for automated architecture discovery for CFNs could involve genetic algorithms, reinforcement learning, or Bayesian optimization to search the vast space of possible architectures.

Expanded Node Library Future work could explore specialized nodes for unstructured data, such as interpretable attention mechanisms for transformers or recurrent nodes for sequence data.

Formalizing the Interpretability Trade-off Developing quantitative metrics to measure the trade-off between

model complexity and interpretability would provide valuable guidance for practitioners.

Uncertainty Quantification Integrating mechanisms for uncertainty estimation, such as Bayesian variants of function nodes, would provide reliable confidence scores for predictions in high-stakes domains.

Applications to New Domains CFNs show particular promise for scientific and medical applications where interpretability is critical, potentially leading to new scientific insights in fields such as genomics and climate modeling.

Ultimately, we believe the CFN paradigm can help bridge the gap between performance and interpretability, fostering greater trust and broader adoption of artificial intelligence in science, industry, and society.

References

Adebayo, J.; Gilmer, J.; Muelly, M.; Goodfellow, I.; Hardt, M.; and Kim, B. 2018. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, volume 31.

Agarwal, R.; Melnick, L.; Frosst, N.; Zhang, X.; Lengerich, B.; Caruana, R.; and Hinton, G. E. 2021. Neural additive models: Interpretable machine learning with neural nets. In *Advances in Neural Information Processing Systems*, volume 34, 4699–4711.

Bau, D.; Zhou, B.; Khosla, A.; Oliva, A.; and Torralba, A. 2017. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6541–6549.

Chen, T.; and Guestrin, C. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

Dua, D.; and Graff, C. 2019. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.

Efron, B.; Hastie, T.; Johnstone, I.; and Tibshirani, R. 2004. Least Angle Regression. *Annals of Statistics*, 32(2): 407–499.

Garcez, A. d.; and Lamb, L. C. 2022. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics*, 6(4): 611–643.

Gunning, D.; and Aha, D. W. 2019. XAI–Explainable artificial intelligence. *Science Robotics*, 4(37).

Koh, P. W.; Nguyen, T.; Tang, Y. S.; Mussmann, S.; Pierson, E.; Kim, B.; and Liang, P. 2020. Concept bottleneck models. In *International Conference on Machine Learning*, 5338–5348. PMLR.

Koza, J. R. 1992. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images. Technical report.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 1097–1105.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature*, 521(7553): 436–444.

Lundberg, S. M.; and Lee, S.-I. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30.

Nori, H.; Jenkins, S.; Koch, P.; and Caruana, R. 2019. InterpretML: A Unified Framework for Machine Learning Interpretability. In *Proceedings of the 2019 Conference on Fairness, Accountability, and Transparency*, 349–350. ACM.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, 618–626.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, 5998–6008.

Zagoruyko, S.; and Komodakis, N. 2016. Wide Residual Networks. In *Proceedings of the British Machine Vision Conference*.