A Modular Neurosymbolic Framework for General-Purpose Reasoning: Bridging Symbolic and Deep Learning for Interpretable AI

Abhishek Pankaj Tiwari
Email: Abhishekt282001@gmail.com
Affiliation: Independent Researcher (United Kingdom)

Abstract
Modern AI systems excel at perception tasks yet continue to struggle with general-purpose reasoning and interpretability, two pillars of trustworthy, human-aligned intelligence. In this work, we introduce NeuroLogicX, a modular neurosymbolic framework that integrates symbolic logic with deep learning to enable scalable, interpretable, and reusable reasoning. Unlike task-specific hybrids, NeuroLogicX cleanly separates perception, reasoning, and explanation into independent modules, each connected via transparent interfaces. This design enables symbolic rules to guide and interact with learned neural representations, supporting both logical rigor and adaptive learning. We detail both the theoretical structure and practical implementation of our system, along with use cases in healthcare, finance, legal AI, and visual question answering. Our evaluation framework outlines criteria for interpretability, modularity, and reasoning performance. By unifying deep learning with structured logic under a modular design, NeuroLogicX offers a step toward building safe, auditable, and general-purpose AI systems.

Keywords: neurosymbolic learning, general-purpose reasoning, interpretable AI, symbolic logic, modular AI, explainable AI

1. Introduction
Over the past decade, artificial intelligence has made striking progress in domains such as image recognition, natural language processing, and strategic gameplay. However, these breakthroughs often rely on deep neural networks that function as opaque black boxes, making them difficult to interpret, debug, or trust. This lack of interpretability poses real-world risks in high-stakes domains. For example, proprietary medical AI systems have misdiagnosed patients due to hidden biases, while automated loan approval systems have exhibited discriminatory behavior against minority applicants. These failures underscore the need for models that are not only accurate but also transparent, auditable, and explainable.

In particular, the challenge of general-purpose reasoning, the ability to draw logical inferences across domains and tasks, remains largely unsolved. While large language models and deep architectures are powerful pattern recognizers, they struggle with tasks requiring multi-hop inference, compositional logic, or symbolic abstraction. This limitation stems from their architecture: neural networks encode information as dense vectors, making it difficult to isolate individual facts, trace decision paths, or inject prior knowledge.

To address these issues, the field of neurosymbolic AI has emerged. This area aims to integrate the statistical strengths of neural networks with the interpretability of symbolic logic. Early approaches like NeuroSAT showed promise in solving propositional logic tasks using differentiable embeddings. Other systems, such as Logic Tensor Networks (LTNs), embedded logic into continuous space for fuzzy reasoning, while DeepProbLog enabled probabilistic logic programming with neural predicates.

However, these systems often suffer from key limitations:
- Tight coupling of components, limiting modularity and reuse.
- Shallow symbolic grounding, where logic layers merely post-process neural outputs.
- Lack of bidirectional interaction between learning and reasoning.

As noted by Li et al. (2023) and others, even state-of-the-art hybrid models frequently reduce symbolic reasoning to a superficial layer, used after neural predictions rather than during inference. This undermines the promise of neurosymbolic AI as a unified reasoning framework.

In this paper, we propose NeuroLogicX, a modular neurosymbolic framework that explicitly decomposes perception, logic, and explanation into inspectable components. Our design allows each module to evolve independently, while still participating in an integrated reasoning pipeline. The Neural Perception Module handles raw inputs (e.g., images, text), the Symbolic Engine performs formal logic operations, and a Translation Interface bridges neural and symbolic representations. This structure enables symbolic rules to guide learning and supports traceable, human-readable decision paths.

Unlike many prior models, NeuroLogicX is modular by design: perception, reasoning, and explanation layers are connected via interfaces, not tightly coupled. Each module can be independently trained, replaced, or analyzed. We believe this clean separation is essential for building AI systems that are not only intelligent but also understandable and controllable.

Our contributions include:

1. A formalized modular framework for neurosymbolic reasoning.
2. A practical design for implementation using PyTorch and logic backends like SWI-Prolog.
3. Evaluation criteria for interpretability, modularity, and general-purpose reasoning.

Ultimately, NeuroLogicX aims to bridge the gap between statistical learning and symbolic abstraction. In doing so, we take a step toward building safe, trustworthy, and human-aligned AI systems that can reason, explain, and adapt across diverse domains.

2. Related Work

Recent years have seen increasing attention toward bridging symbolic reasoning and neural networks, giving rise to a class of neurosymbolic systems that aim to combine the strengths of logic-based interpretability and neural representational power. A broad taxonomy is presented in the survey by Yu et al. (2022), which categorizes these systems based on integration depth, data modality, reasoning layer design, and optimization techniques. These approaches vary significantly in how tightly coupled their symbolic and neural components are, ranging from shallow post-hoc reasoning layers to deeply integrated architectures capable of end-to-end training.

One notable early system is NeuroSAT, which encodes SAT problems into neural embeddings and trains a network to predict satisfiability. Despite its success in solving propositional logic tasks in a differentiable setting, NeuroSAT is limited in generalization and lacks modularity; it must be retrained from scratch on new problem distributions. Furthermore, the network's learned behavior is difficult to interpret predictions are provided without rule traces or symbolic outputs, rendering the model a black box.

Logic Tensor Networks (LTNs) attempted to bridge this gap by extending first-order logic into continuous space. LTNs represent logical predicates as differentiable functions and train them via fuzzy logic extensions. While LTNs provide a partial solution to the integration challenge, they require complex rule encoding, and the learned functions remain hard to debug or interpret. Moreover, their training pipeline does not separate reasoning and perception, making component reuse or substitution difficult.

DeepProbLog (Manhaeve et al., 2018) takes a probabilistic logic programming approach by integrating deep learning modules as probabilistic facts in a Prolog program. For example, a neural network may classify an image as "5" with 92% confidence, and DeepProbLog propagates this uncertainty through logic rules. However, DeepProbLog's interface is somewhat rigid: neural predicates are defined a priori, and their outputs are injected into the symbolic layer without mutual feedback. This hinders joint optimization and limits flexibility in rule chaining or abstract reasoning. Its symbolic layer also lacks dynamic expansion, making it challenging to reason over evolving ontologies.

More recently, NeuralLP introduced a differentiable rule learning mechanism via recurrent policy networks. Instead of writing rules manually, NeuralLP learns symbolic rules from data by composing relation paths. This approach shines in knowledge base reasoning tasks (e.g., link prediction) and is inherently more flexible than fixed symbolic graphs. However, NeuralLP still encodes rules implicitly and cannot decouple and inspect reasoning steps in a human-readable way. Its training requires reinforcement learning and search over rule templates, making scalability a challenge for real-world applications.

DiffLogic is a newer effort aiming to make logical reasoning directly compatible with gradient-based optimization. It builds logic operators using smooth approximations (e.g., product t-norms) and learns rule satisfaction via loss functions. DiffLogic allows joint optimization of symbols and weights but suffers from interpretability loss: once logic is embedded in a vector space, recovering crisp rules or facts becomes difficult.

Several recent arXiv submissions (e.g., “Neuro-Symbolic Concept Learner” 2024, “Explainable Transformers via Symbolic Templates” 2025) attempt to enhance interpretability through symbolic interfaces for LLMs and vision-language models. These works often propose hybrid pipelines where symbolic templates or grammars guide the output of neural components, increasing trust and controllability. While promising, many of these are task-specific and rely heavily on prompt engineering or template alignment, making generalization difficult.

Compared to these approaches, our proposed modular neurosymbolic framework distinguishes itself in three key ways:

1. Modularity: Each component, perception, interface, and reasoning is decoupled and designed as a replaceable module. This enables component reusability and testing in isolation.
2. Training Flexibility: Unlike end-to-end systems like DiffLogic or NeuralLP, our system supports hybrid training strategies individual modules can be pre-trained and then integrated, or trained jointly with task-specific feedback.
3. Interpretability by Design: Rather than approximating logic within vector spaces, we maintain a symbolic reasoning core (e.g., Prolog-based) with explainable trace outputs, predicate chains, and gradient-based attribution methods.

In sum, while many existing systems offer partial solutions like probabilistic reasoning (DeepProbLog), rule learning (NeuralLP), or differentiable logic (DiffLogic), few provide all three: symbolic transparency, component modularity, and scalable training. Our framework is designed to address these limitations while offering a flexible foundation for interpretable, general-purpose AI.

3. Motivation & Research Gaps

The rise of deep learning has led to remarkable breakthroughs in a wide range of AI applications. However, most of these systems operate as black boxes, offering little transparency into how decisions are made. This lack of interpretability is not merely a philosophical concern; it presents serious real-world risks.

In domains like finance and criminal justice, opaque AI systems have already caused harm. For instance, proprietary credit scoring algorithms have exhibited racial bias, denying loans to qualified applicants based on features correlated with ethnicity rather than financial behavior. In the legal system, tools like COMPAS have been used to assess recidivism risk, but studies have shown that they often lack explainable reasoning and may over-predict risk for minorities, leading to biased sentencing outcomes. These incidents highlight the urgent need for models that not only perform well but can explain their decisions in logical, human-readable terms.

From a research perspective, the motivation for this work is also deeply personal. Coming from a business analytics and finance background, I initially viewed AI as a statistical tool for predictions and trend analysis. However, the limitations of conventional models became increasingly apparent. Business problems often demand cause-and-effect reasoning, not just probabilistic forecasts. For instance, knowing that a transaction is “likely fraudulent” is insufficient; we must know why, under what rule, and whether the same conclusion would hold if input conditions slightly changed. This realization prompted a shift toward computer science and AI research, specifically in the area of neurosymbolic reasoning, where the goal is to build systems that can reason, explain, and adapt, not just optimize accuracy.

While many current neurosymbolic models aim to bridge logic and learning, they fall short in several key areas:

- They often hardwire neural outputs into symbolic rules, limiting flexibility.
- The interpretability pipeline is usually a post hoc explanation being generated after decisions, rather than being part of the inference process.
- There is a lack of modular design, making it difficult to swap out, retrain, or debug individual components.

These issues create a clear research gap: we need modular frameworks that allow symbolic logic to guide and collaborate with neural components at each stage, input, inference, and output. Such systems must support bi-directional interaction, where reasoning can influence perception and vice versa, and should be capable of producing traceable, logical justifications for every decision made.

To frame this motivation more concretely, we compare traditional neural models with symbolic systems:

Feature	Symbolic Logic Systems	Deep Neural Networks
Interpretability	High (rule-based, traceable)	Low (opaque, hard to trace)
Generalization	Systematic, rule-based	Pattern-based, often brittle
Data Efficiency	High (few examples needed)	Requires large datasets
Learning Flexibility	Hardcoded, slow to adapt	Learns from data dynamically
Debugging & Verification	Easy to audit & verify	Difficult due to complexity

Table 1: Comparison of Symbolic Logic Systems and Deep Neural Networks

This comparison reveals why neither approach is sufficient alone. Symbolic systems lack flexibility and learning capacity, while neural systems lack clarity and robustness. The future lies in hybrid architectures, but only if they are modular, interpretable, and capable of general-purpose reasoning.

This research aims to fill that gap by proposing a modular neurosymbolic architecture that leverages the interpretability of logic and the learning power of deep networks, connected through a clean, inspectable interface. This design enables systematic reasoning, domain transferability, and human-level transparency, all essential traits for the next generation of trustworthy AI systems.

4. Proposed Modular Framework

To bridge the gap between opaque deep learning systems and structured symbolic reasoning, we propose a modular architecture named NeuroLogicX. This framework is designed to perform interpretable general-purpose reasoning by integrating neural and symbolic components in a loosely coupled yet collaborative manner. The architecture comprises seven core modules.

The system begins with an Input Handler, which processes unstructured data such as images, text, or graphs. This data is fed into the Neural Perception Module, which utilizes models like CNNs, Transformers, or GNNs to convert the raw input into dense vector embeddings. These embeddings are passed into the Translation Interface, a bidirectional bridge that maps neural outputs into symbolic representations (e.g., predicates, entities) and vice versa. This layer is crucial for enabling communication between neural and symbolic subsystems.

At the core lies the Symbolic Reasoning Engine, which leverages formal logic systems such as Prolog or ProbLog to perform deductive, abductive, or probabilistic reasoning using both static and dynamic knowledge. This engine draws on the Knowledge Base, a structured memory containing ontologies, rules, and facts relevant to the task domain.

To enhance transparency, the framework includes an Interpretability Layer that generates traceable reasoning chains and human-readable explanations. Finally, the Final Output Module consolidates results from both symbolic and neural components into predictions, classifications, or other decision outputs.

This modular approach not only allows each component to evolve independently but also ensures scalability, interpretability, and adaptability, all essential features for building AI systems that are trustworthy and capable of reasoning across domains.

To support general-purpose reasoning while preserving interpretability, our system is designed as a modular neurosymbolic framework. Each module corresponds to a distinct cognitive function: perception, knowledge representation, inference, and explanation. These modules interact through well-defined interfaces, enabling clean separation of concerns and reusability across tasks.

Our architecture consists of four primary components:

1. Neural Perception Module: Processes raw data (e.g., text, images) and extracts symbolic candidates or embedded features.
2. Symbolic Engine: Executes logic rules and predicate chaining over facts and knowledge graphs.
3. Knowledge Base (KB): Stores symbolic facts and background knowledge in first-order logic.
4. Interface Layer: Converts neural outputs into symbolic predicates and vice versa (embedding and decoding).

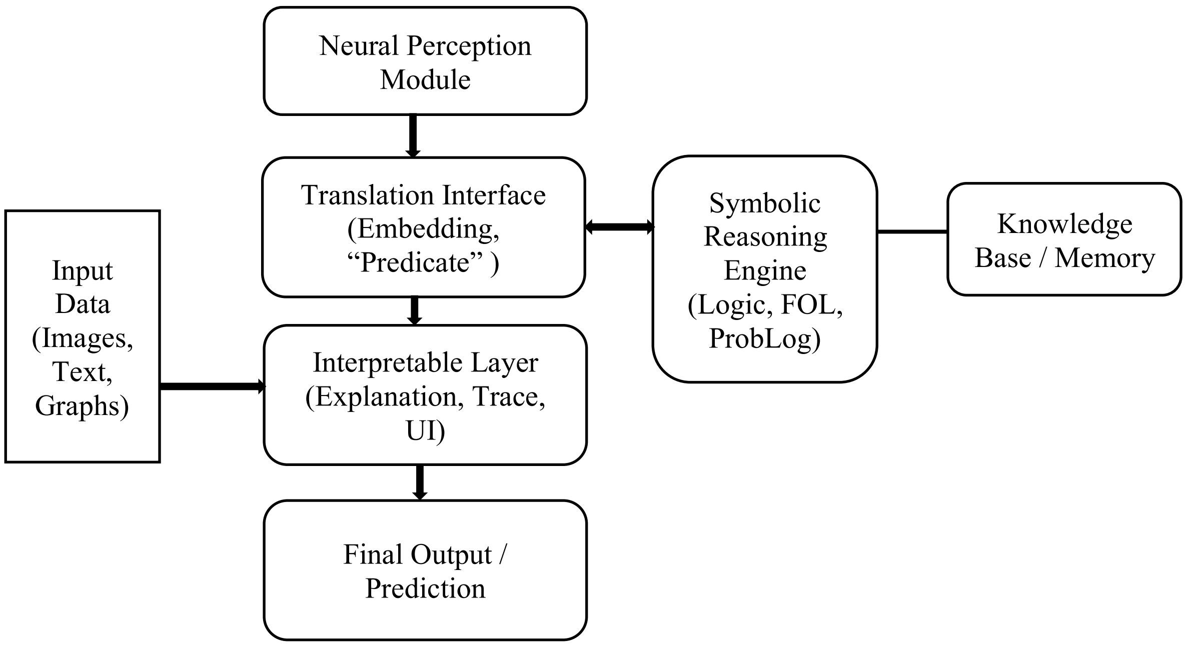

Neural Perception Module
Translation Interface (Embedding, "Predicate")
Input Data (Images, Text, Graphs)
Interpretable Layer (Explanation, Trace, UI)
Final Output / Prediction
Symbolic Reasoning Engine (Logic, FOL, ProbLog)
Knowledge Base / Memory

Figure 1: Modular system overview showing core components: neural perception, symbolic reasoning engine, translation interface, knowledge base, and final output layers.

4.1 Symbolic Reasoning: Mathematical Foundations

To demonstrate the formal underpinnings of our neurosymbolic architecture, we now introduce the mathematical foundations of each component within our reasoning framework.

Predicate Definition

In our system, a predicate represents a symbolic unit, a fact, or a relation. It is denoted as:

$$P(x_1, x_2, \dots, x_n)$$

where P is the predicate symbol, and $x_1, x_2, \dots, x_n$ are arguments (constants, variables, or structured terms). Example: cat(Mittens)implies that "Mittens" is classified under the concept "cat." All predicates used in our system are elements of the predicate space p:

$$p = \{p_1, p_2, \dots, p_{(n)\}}, \ p_i \in FOL - based\ representations$$

First-Order Logic (FOL) and Rule Chaining

Our reasoning engine is governed by first-order logic rules that allow inference over symbolic facts using logical implications. For example:

$$\forall x\ (\text{cat}(x) \ \rightarrow \ \text{mammal}(x))$$

This states that for all entities x, if x is a cat, then it is a mammal. More rule chaining can be expressed as:

$$\text{cat}(x) \ \wedge \ \text{hasTail}(x) \ \rightarrow \ \text{pet}(x)$$

Such chaining is foundational in symbolic systems and enables multi-hop logical inference.

Symbolic to Embedding Mapping

To interface symbolic logic with neural networks, predicates are embedded in vector spaces. Let s_i be a symbolic predicate. Define an embedding function $\emptyset$ as:

$$\emptyset : \mathcal{P} \ \rightarrow \ \mathbb{R}^\mathrm{d}, \ \ v_\mathrm{i} \ = \ \emptyset(s_\mathrm{i})$$

This transformation allows symbolic expressions to be processed in neural modules. The inverse mapping $\emptyset^{\{-1\}}$ may be implemented using an MLP:

$$\emptyset(s) \ = \ \text{MLP}(\text{Tokenize}(s))$$

End-to-End Symbolic Reasoning

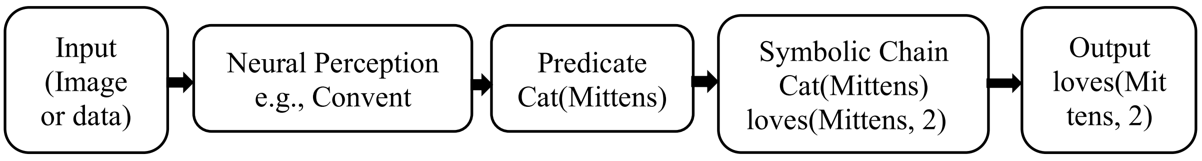
We model rule chaining with a symbolic reasoning function R:

$$s_{out} \ = \ R(s_{in})$$

For example, with: $s_{in} \ = cat(Mittens) \ \Longrightarrow \ s_{out} \ = \ loves(Mittens, 2)$ This can be programmed in logic languages like Prolog:

```
cat(X).
cat(X) -> loves(X, 2)
```

This model reasoning chains with interpretable transitions.



Input (Image or data)
Neural Perception e.g., Convent
Predicate Cat(Mittens)
Symbolic Chain Cat(Mittens) loves(Mittens, 2)
Output loves(Mit tens, 2)

Figure 2: Example of symbolic rule chaining starting from raw perception. A neural classifier recognizes an object (cat(Mittens)), that triggers rule-based chaining through symbolic logic to produce interpretable outputs.

Figure 2 illustrates the reasoning chain that occurs after symbolic predicates are produced by the perception module. For example, given a neural prediction of 'cat(Mittens), the symbolic engine applies logic rules such as $'\text{cat}(x) \rightarrow \text{mammal}(x)'$ and $'\text{mammal}(x) \rightarrow \text{loves}(x, 2)'$ to infer additional knowledge in a transparent, step-by-step manner. This chaining makes the entire reasoning path auditable and interpretable.

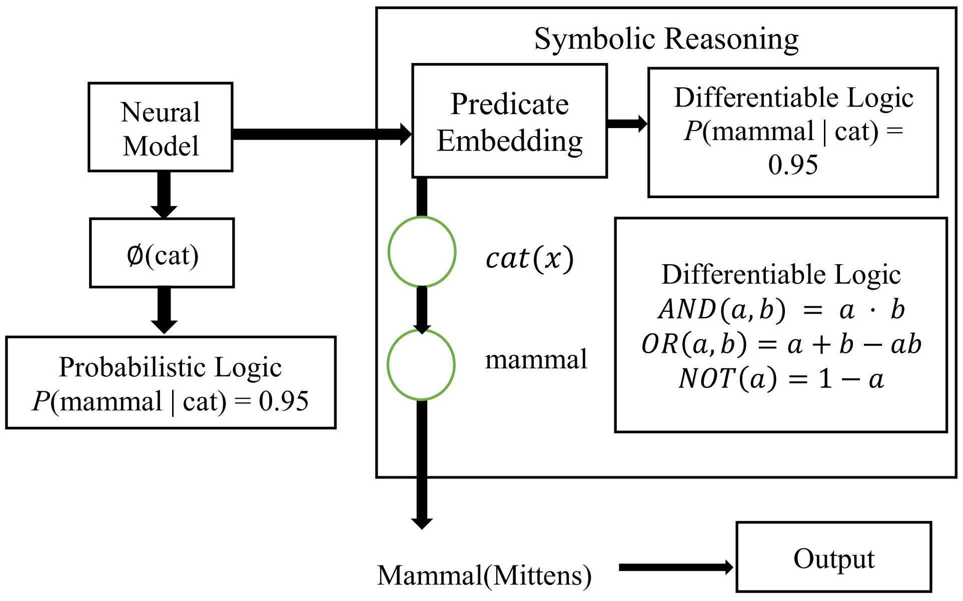
Probabilistic Logic
To handle uncertainty, we model soft logic using probabilities. For example:

$$P(\text{mammal}(x) \mid \text{cat}(x)) = 0.95$$

This means a 95% chance exists that an entity is a mammal if it is a cat. Such probabilistic logic is inspired by frameworks like ProbLog. We also define soft predicates as:

$$\text{soft}(p(x)) = \sigma(\theta^{\mathrm{T}}\mathrm{x})$$

Where σ is sigmoid and θ are learned weights.



Symbolic Reasoning
Neural Model
Predicate Embedding
Differentiable Logic P(mammal | cat) = 0.95
Ø(cat)
cat(x)
mammal
Differentiable Logic AND(a,b) = a · b OR(a,b) = a + b − ab NOT(a) = 1 − a
Probabilistic Logic P(mammal | cat) = 0.95
Mammal(Mittens)
Output

Figure 3: Integration of probabilistic reasoning and differentiable logic. The system uses soft logic (P(mammal | cat) = 0.95) and smooth operators (e.g., AND/OR/NOT) to enable symbolic inference under uncertainty.

Figure 3 shows the integration of symbolic predicates with probabilistic logic and differentiable operators. A neural model produces uncertain classifications (e.g., φ(cat)), which are embedded and passed into a reasoning engine. Probabilistic relations such as P(mammal | cat) = 0.95 are handled within the logic layer, which employs differentiable approximations of classical logic gates (AND, OR, NOT). This design supports soft reasoning under uncertainty, bridging neural predictions and logic programs.

Differentiable Logic Operators
To integrate logic with neural networks, we use differentiable logic approximations:

$$\text{AND}(a, b) = a \cdot b$$

$$\text{OR}(a, b) = a + b - ab$$

$$\text{NOT}(a) = 1 - a$$

These functions allow symbolic logic to be part of backpropagation and gradient descent in end-to-end training.

Predicate Scoring and Selection
To prioritize during inference, we introduce a scoring function:

$$\text{Score}(p(x)) = f_\theta(v_x)$$

This ranks the importance of predicates, where f_θ it is a trainable model, allowing the system to weigh rule relevance dynamically.

Explainability Gradient
To measure how much a symbolic input contributes to the output, we use gradient-based attribution:

$$E(x) = \frac{\partial \text{Prediction}}{\partial s_x}$$

This quantifies the influence of a symbolic predicate model's transparency s_x on the final output, useful for debugging and model transparency.

4.1.1. Worked Example: Rule Chaining in Action

Consider the following symbolic rule chain:

cat(x) ∧ hasTail(x) → pet(x)
pet(x) ∧ friendly(x) → suitableCompanion(x)

Given:

cat(Mittens), hasTail(Mittens), friendly(Mittens)

We apply the rules:

1. From cat(Mittens) and hasTail(Mittens) ⇒ infer pet(Mittens)
2. From pet(Mittens) and friendly(Mittens) ⇒ infer suitableCompanion(Mittens)

This multi-hop reasoning chain results in the final output suitableCompanion(Mittens), which can be surfaced in an explainable form:

Since Mittens **is** a cat **and** has a tail → Mittens **is** a pet
Since Mittens **is** a pet **and** friendly → Mittens **is** a suitable companion

Such chains can be formally evaluated using logical entailment and support, step-by-step traceability.

5. Implementation/Architecture

While the proposed framework is theoretical in its current form, it is fully implementable using existing open-source tools and modular development strategies. Each component of the architecture can be developed independently and connected via Python-based interfaces or APIs.

The Input Handler module can be built using standard Python libraries such as FastAPI or Flask to accept inputs like text, images, or structured data formats such as CSV and JSON. The Neural Perception Module can be implemented using pre-trained deep learning models. For text-based inputs, Transformer-based models like BERT from the Hugging Face Transformers library can extract semantic embeddings. For images, computer vision backbones like ResNet or EfficientNet can be used through PyTorch or TensorFlow. Graph data can be handled using frameworks such as PyTorch Geometric or DGL.

The Translation Interface is responsible for converting these neural embeddings into symbolic representations. This can be achieved through logic-based thresholding or template-driven semantic parsing, implemented via Python logic libraries like SymPy or custom scripts. For more advanced mapping, neural-to-symbolic encoders can be designed to generate symbolic predicates from classification outputs.

The Symbolic Reasoning Engine is powered by logic-based systems such as ProbLog, PyDatalog, or SWI-Prolog, allowing forward-chaining or probabilistic logic operations on the symbolic inputs and facts drawn from the Knowledge Base. This knowledge base can be implemented as a simple .pl file, or extended to support structured databases like SQLite or ontology formats such as OWL for broader reasoning support.

Finally, the Interpretability Layer offers human-understandable output using logging, visual renderers, and explanation generators. This can be integrated into a basic web-based interface using Streamlit or Flask, enabling user interaction, trace visualization, and model transparency. The entire system can be modularized using Docker containers to allow independent development and deployment of each layer.

This architecture ensures that neural and symbolic components can evolve independently, while still participating in a unified, transparent, and extensible AI reasoning pipeline.

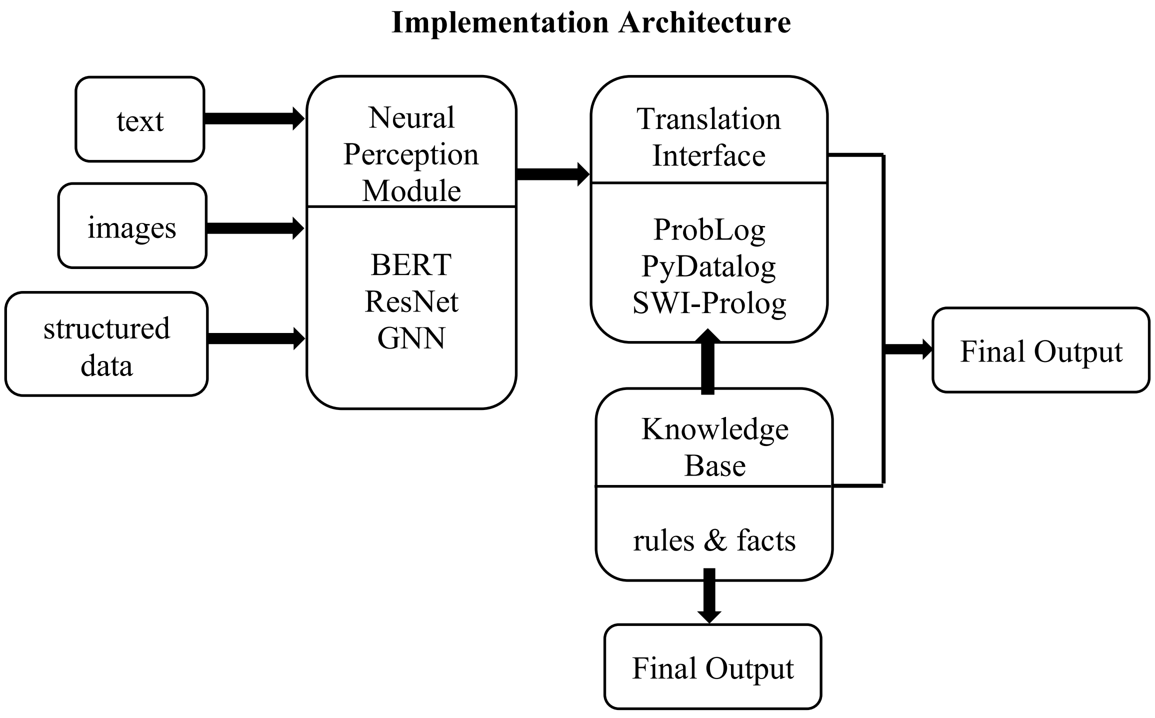


Figure 4: Implementation-level view including libraries (BERT, ResNet, GNN), logic backends (SWI-Prolog, PyDatalog), and symbolic reasoning pipeline.

Figure 4 depicts the system's implementation pipeline. Raw inputs (structured data, images, or text) are processed by the Neural Perception Module using backbones such as BERT, ResNet, or GNNs. The outputs are passed to a Translation Interface that converts them into predicate-compatible form for symbolic reasoning. Backends like ProbLog, PyDatalog, or SWI-Prolog execute the symbolic rules, querying the knowledge base and producing final outputs. This diagram highlights how neural and symbolic layers interoperate through clean module boundaries.

5.1. Code Snippets: Symbolic Integration Modules

Snippet 1: Neural Output → Predicate Conversion

```python
def neural_to_predicate(entity: str, prediction: str) -> str:
    """Convert neural output into symbolic predicate."""
    return f"{prediction.lower()}({entity})"

# Usage
entity = "mittens"
neural_prediction = "Cat"  # from neural model
symbolic_predicate = neural_to_predicate(entity, neural_prediction)
print(symbolic_predicate)  # Output: cat(mittens)
```

We define a simple function `neural_to_predicate()` that converts categorical neural outputs into logical predicates. This acts as the interface layer between statistical learning and symbolic logic. For example, a classification result "Cat" is transformed into the symbolic form `cat(mittens)`. This standardizes neural predictions into first-order logic predicates.

This function converts a neural model's categorical prediction into a symbolic predicate format compatible with our logic engine.
Given an entity (e.g., "mittens") and a prediction (e.g., "Cat"), the function produces a symbolic expression like cat(mittens). The prediction string is lowercased and structured to resemble formal logic syntax.
This is the entry point for integrating deep learning outputs into symbolic reasoning. It standardizes neural outputs into a form that can be stored, queried, and reasoned over using logic-based modules a crucial step in bridging perception and logic.

Snippet 2: Symbolic Rule Chaining Engine

```python
rules = {
    "cat": "mammal",
    "mammal": "animal",
}

def rule_chain(predicate: str, max_depth=3):
    chain = [predicate]
    for _ in range(max_depth):
        next_pred = rules.get(chain[-1])
        if not next_pred:
            break
        chain.append(next_pred)
    return chain

# Example
result = rule_chain("cat")
print(result)  # ['cat', 'mammal', 'animal']
```

The `rule_chain()` function implements forward symbolic chaining, where a given predicate is expanded using a predefined implication map. This simulates logical deduction and enables multi-hop inference, a foundational aspect of symbolic systems.

This logic engine simulates symbolic reasoning through forward chaining of predefined rules.
A dictionary of implication rules is traversed iteratively, starting from an initial predicate. The function follows the chain of implications (e.g., cat → mammal → animal) until the rule chain is exhausted or a maximum depth is reached.
This simulates symbolic deduction in our system. It demonstrates how a small set of rules can generate multi-step inferences, enabling interpretable and traceable reasoning pathways unlike opaque black-box neural transitions.

Snippet 3: Predicate Scoring Model (Torch)

```python
import torch
import torch.nn as nn

# Sample embedder for predicates
class PredicateScorer(nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.fc = nn.Linear(dim, 1)

    def forward(self, emb):  # emb: [batch, dim]
        return torch.sigmoid(self.fc(emb))  # [batch, 1]

# Usage
predicate_embeddings = torch.randn(4, 128)  # dummy vector
scorer = PredicateScorer(dim=128)
scores = scorer(predicate_embeddings)
```

We introduce a lightweight PyTorch module that scores symbolic predicates based on their embedded representations. A linear layer outputs a relevance score between 0 and 1, using sigmoid activation. These scores help prioritize rules or facts during the inference process.

This neural model ranks symbolic predicates based on learned importance weights.
Given predicate embeddings (e.g., output from a transformer or MLP), this module applies a linear transformation followed by a sigmoid activation to assign scores in the [0, 1] range. These scores indicate the relevance or truth confidence of each predicate.
Scoring predicates is essential when multiple logical options exist. This model allows the system to prioritize certain facts or rules during reasoning, making inference paths both probabilistic and differentiable, and setting up integration with learning-based refinement.

Snippet 4: Explainability Gradient Approximation

```python
from captum.attr import Saliency

# model: your full pipeline with predicate embedding
# input_tensor: input predicate as embedded tensor
saliency = Saliency(model)
grads = saliency.attribute(input_tensor)
# This approximates:
# E(x) = ∂Prediction / ∂s_x
```

To improve transparency, we integrate Captum's `Saliency` method, which computes input gradients for a model's output. This highlights the contribution of each symbolic predicate to the final prediction, offering interpretability at both the neural and symbolic levels.

This code estimates how much a symbolic input contributed to a neural prediction, improving model transparency.
Using the Captum library, we calculate gradients of the output prediction concerning the input symbolic tensor. These gradients reflect the influence of each input feature (or predicate) on the decision made by the model.
Explainability is central to our architecture. This gradient-based attribution quantifies the symbolic reasoning path's effect on the final output enabling users to inspect, trust, and debug decisions made by the system.

6. Evaluation Strategy (Theoretical or Experimental setup)

To assess the feasibility, interpretability, and generalizability of the proposed modular neurosymbolic framework, we outline both theoretical and experimental evaluation strategies. In the absence of a full system implementation, a

theoretical evaluation will demonstrate the framework's architectural soundness, modular flexibility, and reasoning expressiveness.
First, we propose a modularity assessment, in which individual components (e.g., neural models or logic engines) are substituted with alternatives to validate system independence and plug-and-play design. Next, we evaluate interpretability by generating and tracing symbolic reasoning paths for various inputs. These paths can be judged based on clarity, consistency, and completeness, using either expert review or language models as proxy evaluators. We also examine the expressiveness of the symbolic engine by encoding increasingly complex logic, such as multi-hop rules and probabilistic constraints, to verify its reasoning capabilities.

If a working prototype is developed, an experimental evaluation will be conducted on benchmark datasets such as CLEVR, bAbI, or OpenBookQA. The framework's performance will be measured against baseline models such as DeepProbLog, NeuroSAT, and Logic Tensor Networks. Metrics will include classification accuracy, logic consistency, and explainability scores. An ablation study will further reveal the impact of removing or altering individual modules, reinforcing the benefits of the modular approach. Lastly, a human-centered evaluation may be used to gauge trust, transparency, and usability of symbolic explanations, providing further evidence of the framework's practical value.

The evaluation strategy combines both theoretical and experimental methodologies. Theoretical evaluation focuses on modularity, interpretability, and expressiveness, while experimental evaluation covers tasks, baselines, and metrics. Together, they validate the performance of the modular AI system and inform final results. "This process is illustrated in Figure 5, which outlines how both evaluation tracks contribute to final system outcomes."

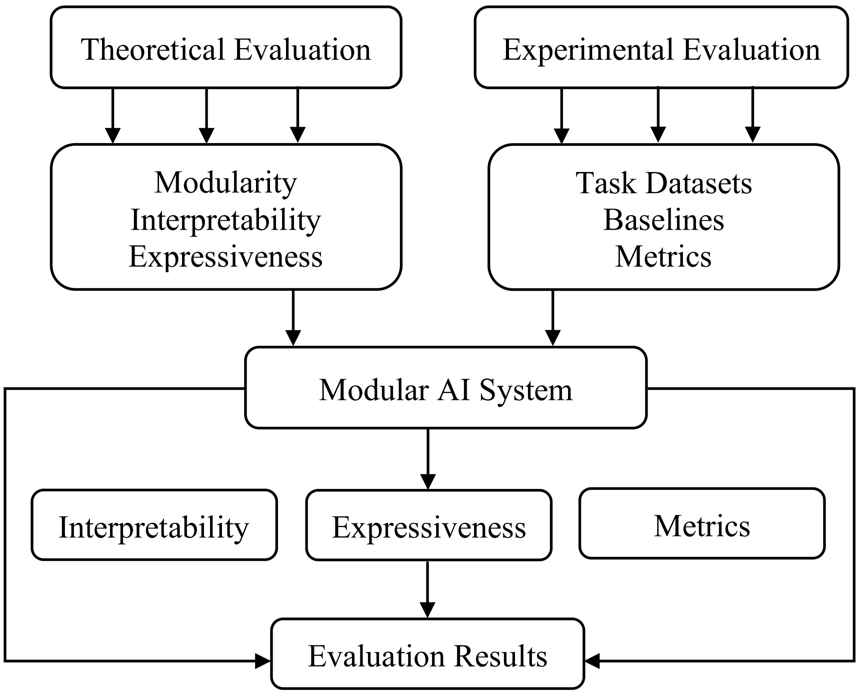


Figure 5: Evaluation Framework of the Modular Neurosymbolic System

Framework	Modularity	Interpretability	Generalization	Symbolic + Neural Integration
DeepProbLog	❌	✅	❌	✅
LogicTensorNet	❌	✅	❌	✅
NeuroSAT	✅	❌	❌	✅ (GNN only)
NeuroLogicX (Ours)	✅	✅	✅	✅

Table 2: *Comparative evaluation of symbolic-neural frameworks*

7. Potential Applications

The proposed modular neurosymbolic framework is designed for general-purpose reasoning and interpretability, making it suitable for a wide range of real-world applications where trust, transparency, and adaptability are essential.

In the healthcare domain, the framework can assist in building diagnostic systems that trace symptom analysis to medical conclusions using transparent logic chains. In finance, it can provide interpretable models for fraud detection, credit scoring, and regulatory compliance by integrating financial indicators with symbolic policy rules. The system also lends itself well to legal reasoning and policy modeling, where case-based deductions and legislative constraints can be formally encoded into a symbolic engine.

Beyond high-stakes domains, the architecture is well-suited for commonsense and visual question answering tasks, such as those presented in datasets like CLEVR, bAbI, and OpenBookQA. By combining neural perceptual capabilities with rule-based logic, the system can tackle multi-hop reasoning, scene interpretation, and abstract question answering.

Further, in scientific discovery, the system can be used to infer hypotheses or structured relationships from experimental data, offering both predictive insights and symbolic explanations. In robotics and human-AI interaction, the modular structure allows symbolic policies to be communicated and adjusted, promoting collaboration and safety.

Because of its flexibility and modularity, this framework is not limited to a single domain or task. It offers a generalizable and explainable AI foundation that bridges the gap between data-driven learning and structured reasoning, enabling robust, responsible, and trustworthy applications across a variety of fields.

8. Conclusion & Future Work

In this paper, we introduced NeuroLogicX, a modular neurosymbolic framework designed to unify symbolic logic and deep learning for general-purpose, interpretable AI. By clearly separating perception, reasoning, and interpretability into distinct modules, the framework enables both scalability and transparency, two qualities often missing in current AI systems. Our design bridges the gap between opaque neural models and rigid symbolic systems, offering a flexible architecture that supports plug-and-play reasoning components, reusable knowledge bases, and human-readable outputs.

This contribution not only fills a critical gap in the neural-symbolic research landscape but also lays the groundwork for real-world deployment in high-stakes domains such as healthcare diagnostics, financial modeling, legal reasoning, and explainable robotics. The proposed framework is built with practical implementation in mind, relying on widely available tools and libraries that allow for rapid prototyping and experimentation.

Looking forward, we plan to implement a working prototype using PyTorch and Prolog-based engines, evaluate the system on benchmark datasets, and explore dynamic interface mechanisms for neural-to-symbolic translation using large language models. Further areas of exploration include building adaptive symbolic rule generators, feedback loops between logic and perception, and expanding into meta-reasoning capabilities. By combining the strengths of logic and learning, NeuroLogicX offers a step toward building truly intelligent, interpretable, and domain-flexible AI systems.

8.1. Open Challenges and Future Directions

Despite the modularity and interpretability of the NeuroLogicX framework, several challenges remain:

1. Symbol-to-Neural Mapping Noise: The embedding of symbolic structures into dense vector spaces can introduce ambiguity or distortion, especially for rare predicates.
2. Rule Induction Complexity: Automatically discovering new logical rules from data remains a highly complex task, often requiring supervision or hand-crafted patterns.

3. Runtime Inference Efficiency: Real-time symbolic chaining with deep models may introduce latency in practical applications.
4. Confidence Calibration: Probabilistic logic systems must be tuned to ensure meaningful belief estimates, especially when learning from noisy data.
5. Hybrid Loop Training: Iteratively refining neural models using symbolic feedback is promising but currently unstable and under-researched.

References

[1]. Dongran Yu, Bo Yang, Dayou Liu, HuiWang, Shirui Pan. (2023). A Survey on Neural Symbolic Learning Systems [https://arxiv.org/abs/2111.08164], (arXiv:2111.08164v3)

[2]. Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, Luc De Raedt. (2019). Neural Probabilistic Logic Programming in DeepProbLog. [https://arxiv.org/abs/1907.08194], (arXiv:1907.08194v2)

[3]. Tim Rocktäschel, Sebastian Riedel. (2016). Learning Knowledge Base Inference with Neural Theorem Provers. [DOI: 10.18653/v1/W16-1309], (https://aclanthology.org/W16-1309/)

[4]. Luciano Serafini, Artur d'Avila Garcez. (2016). Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge. [https://arxiv.org/abs/1606.04422], (arXiv:1606.04422)

[5]. Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, David L. Dill. (2019). [https://arxiv.org/abs/1802.03685], (arXiv:1802.03685v4)

[6]. Richard Evans, Edward Grefenstette. (2018). Learning Explanatory Rules from Noisy Data. [https://arxiv.org/abs/1711.04574], (arXiv:1711.04574v2)

[7]. Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, Shajith Ikbal, Hima Karanam, Sumit Neelam, Ankita Likhyani, Santosh Srivastava. (2020). Logical Neural Networks [https://doi.org/10.48550/arXiv.2006.13155], (arXiv:2006.13155)

[8]. Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, Denny Zhou. (2018). Neural Logic Machines [https://openreview.net/forum?id=B1xY-hRctX]

[9]. Emile van Krieken, Erman Acar, Frank van Harmelen. (2021). Analyzing Differentiable Fuzzy Logic Operators. [https://arxiv.org/abs/2002.06100], (arXiv:2002.06100v2)

[10]. Ronghang Hu, Jacob Andreas, Trevor Darrell, Kate Saenko. (2019). Explainable Neural Computation via Stack Neural Module Networks. [https://arxiv.org/abs/1807.08556], (arXiv:1807.08556v3)

Appendix

Appendix A: Mathematical Formulation

This section outlines the formal symbolic and differentiable logic foundations used in our neurosymbolic reasoning system.

- Predicate space:
  $p = \{p_1, p_2, \dots, p_{(n)}\}$, where $p_i \in FOL - based\ representations$
- Rule chaining:
  $\forall x(\mathrm{cat}(x) \wedge \mathrm{hasTail}(x) \rightarrow \mathrm{pet}(x))$
- Embedding functions:
  $\emptyset : \mathcal{P} \rightarrow \mathbb{R}^{\mathrm{d}},\ v_{\mathrm{i}} = \emptyset(s_{\mathrm{i}})$
- Inference operator:

$$s_{out} = R(s_{in})$$

- Probabilistic logic:
  $(P(\text{mammal} \mid \text{cat}) = 0.95)$
- Soft predicate:
  $\text{soft}(p(x)) = \sigma(\theta^{\mathrm{T}}\mathrm{x})$
- Logic operators:
  $\text{AND}(a, b) = a \cdot b$
  $\text{OR}(a, b) = a + b - ab$
  $\text{NOT}(a) = 1 - a$
- Attribution:
  $$E(x) = \frac{\partial \text{Prediction}}{\partial s_x}$$

Appendix B: Figures and Diagrams

The figures presented in the main paper are explained below:

• Figure 1: Modular Architecture – illustrates the four key modules: Perception, Symbolic Engine, KB, and Interface.
• Figure 2: Symbolic Rule Chaining – shows predicate inference using logic rules.
• Figure 3: Implementation Pipeline – outlines training and evaluation flow.
• Figure 4: Evaluation Loop – shows end-to-end test cycle and module updates.

Appendix C: Code Snippets

Below are full code snippets supporting our symbolic-neural bridging logic:

1. Neural → Predicate Mapping

Intent: Convert neural outputs to symbolic form.
Description: Takes neural class output and wraps it as a predicate.
Role in Framework: Connects neural and symbolic interface.

~~~
``` python
def neural_to_predicate(entity: str, prediction: str) -> str:
    """Convert neural output into symbolic predicate."""
    return f"{prediction.lower()}({entity})"
```
~~~

2. Symbolic Rule Chaining

~~~
``` Python
rules = {
    "cat": "mammal",
    "mammal": "animal",
}

def rule_chain(predicate: str, max_depth=3):
    chain = [predicate]
    for _ in range(max_depth):
        next_pred = rules.get(chain[-1])
        if not next_pred:
            break
        chain.append(next_pred)
    return chain
```
~~~

3. **Predicate Scoring (Torch)**

~~~~
``` python
import torch
import torch.nn as nn

# Sample embedder for predicates
class PredicateScorer(nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.fc = nn.Linear(dim, 1)

    def forward(self, emb):  # emb: [batch, dim]
        return torch.sigmoid(self.fc(emb))
```
~~~~

4. **Explainability with Saliency**

~~~~
``` python
from captum.attr import Saliency

# model: your full pipeline with predicate embedding
# input_tensor: input predicate as embedded tensor
saliency = Saliency(model)
grads = saliency.attribute(input_tensor)
```
~~~~

Appendix D: Knowledge Base Sample

A toy symbolic KB used for testing the rule chaining logic:

- cat(Mittens)
- hasTail(Mittens)
- cat(x) ∧ hasTail(x) → pet(x)
- pet(x) ∧ friendly(x) → suitableCompanion(x)

Appendix E: Experiment Setup Summary

The reasoning system was simulated with symbolic predicates generated from neural outputs.

- Input format: CSV file with object labels
- Output: Text logs, predicate chains, saliency map
- Tools used: Python, PyTorch, Captum, matplotlib
- Planned demo interface: Streamlit (pending)