DOI: 10.15849/IJASCA.251130.09

Int. J. Advance Soft Compu. Appl, Vol. 17, No. 3, November2025
Print ISSN: 2710-1274, Online ISSN: 2074-8523

Petri Nets and Deep Petri Nets for Simulation, Modeling and Analysis of Complex Systems

Aouag Mouna[1], and Djazi Qamar [2]
[1]Department of Computer Science, University Centre
Abdelhafidboussouf Mila-Algeria-
e-mail:aouag.mouna@centre-univ-mila.dz
[2]Department of Computer Science, University Centre
Abdelhafidboussouf Mila-Algeria-
e-mail: djazikamar@gmail.com

Abstract

The growing complexity of modern systems in fields such as artificial intelligence and large-scale simulations demands modeling approaches that are both expressive and formally analyzable. While Petri nets offer a solid foundation for modeling distributed systems, they fall short in representing adaptive behaviors typical of intelligent systems. This work introduces two complementary approaches to bridge this gap. The first extends classical Petri nets into a new formalism called Deep Petri Nets, integrating learning mechanisms inspired by neural networks. Three meta-models are defined using Eclipse Modeling Framework (EMF): the first for Petri nets, the second for neural networks, and the third for Deep Petri Nets, combining structural features of both. The second approach proposes a unified meta-model, NNPN (Neural Network of Petri Net), which merges Petri net and neural network elements into a cohesive model, later transformed into a Deep Petri Net. Model transformations are achieved using Triple Graph Grammar (TGG) and Atlas Transformation Language (ATL) rules, ensuring consistency, traceability, and analytical capability. This dual-method framework supports the modeling of complex systems with both formal structure and adaptive behavior.

Keywords: *Deep Petri Net, Model Transformation with the TGG tool, Neural Network, Neural Network of Petri net, Petri Net.*

1 Introduction

This Petri nets are a widely used and powerful mathematical formalism for modeling, simulating, and analyzing systems where concurrency, synchronization and sequencing are essential. Introduced by Carl Adam Petri in 1962, they have been appreciated for their graphical clarity and solid theoretical foundation. Applications span various domains including computer science, telecommunications, industrial systems, and real-time control [1],[2].Their relevance persists in modern contexts such as cyber-physical systems, intelligent manufacturing, and business process management[3]–[5]. Simultaneously, artificial neural networks (ANNs) have become central to artificial intelligence, especially through advances in deep learning. They excel at pattern recognition, prediction, and classification tasks. However, neural networks often suffer from opacity, making them difficult to interpret or formally analyze [6],[7]. Recent work

Received 6 July 2025; Accepted 13 October 2025

on explainable AI aims to address this issue, but integrating symbolic reasoning with subsymbolic learning remains a promising and active area of research [8],[9]. In this paper, we propose a hybrid modeling approach that combines classical Petri nets, neural networks, and a novel formalism called Deep Petri Nets (DPNs).Our approach leverages the interpretability of Petri nets and the learning capabilities of neural networks .Deep Petri Nets offer a unified model capable of learning and adapting while retaining formal structure and analytical capabilities [10]. This aligns with the growing interesting neuro-symbolic systems as a path toward explainable and structured Artificial Intelligence (AI) [11],[12]. To implement this framework, we present two methods. The first defines three meta-models using the Eclipse Modeling Framework (EMF): the first is for classical Petri nets, the second for neural networks, and the third for Deep Petri Nets, which integrates features from both. The second method introduces an intermediate unified meta-model called Neural Network Petri Net (NNPN), which directly embeds neural components into the Petri net structure. This NNPN meta-model is then transformed into a Deep Petri Net model using ATL (Atlas Transformation Language) rules, facilitating the integration of learning dynamics within the formalism. We automate transformations between meta-models using the Triple Graph Grammar (TGG) approach [13].where applicable, and rely on ATL for fine-grained model refinement. This combination ensures consistency, traceability, and adaptability of hybrid models throughout the development process. This paper is structured as follows :Section 1 presents the motivations and objectives of our work, Section 2 reviews related work and compares it with our approach, Section 3 introduces foundational concepts on Petri Nets, Neural Networks, Deep Petri Nets, and model transformation, Section4 describes the proposed approach ,including both modeling methods, Section 5 discusses two case studies, the Smart Traffic Management System and the Automated Production Workshop, to demonstrate the applicability of the framework., Section 6 addresses scalability and complexity analysis, in section 7 practical benefits of the use of DPN framework, and Section 8 concludes the paper and outlines future works.

2 Related Work

In recent years, there has been a growing interest in integrating formal modeling techniques such as Petri nets with learning-based approaches to better model and analyze complex and adaptive systems. Traditional Petri nets, while effective for representing discrete event systems, often face limitations when dealing with uncertainty, learning capabilities, and continuous adaptation. To overcome these challenges, hybrid models have emerged, combining the interpretability of symbolic systems with the adaptability of neural networks.

The authors in [14] introduced a Deep Fuzzy Petri Net model that integrates fuzzy logic with deep learning techniques. The resulting hybrid framework supports systems that require both rule-based reasoning and data-driven behavior, offering improved interpretability and flexibility over conventional deep neural networks. In [15] proposed enhancements to fuzzy Petri nets for better handling uncertainty in dynamic environments. Their work incorporates fuzzy weights and probabilistic transitions, allowing for more realistic modeling of systems with vague or imprecise input data. In [16] developed a real-time intelligent decision-making framework based on fuzzy Petri nets. Their model combines symbolic reasoning with adaptive learning, supporting dynamic rule evaluation in evolving contexts. In [17] explored the use of Model-Driven Engineering (MDE) to formalize hybrid systems composed of Petri nets and machine learning components. Their approach emphasizes the use of meta-models and

transformation rules to integrate structural modeling with Artificial Intelligence (AI) driven behavior. In [18] presented a method for Petri net based neural network interpretation, where the firing of transitions mimics neuron activations, allowing for traceable decision paths in deep learning applications. This work underlines the importance of explainability in hybrid systems. A more recent contribution by the authors in [19] titled Petri-net-based deep reinforcement learning for real-time scheduling of automated manufacturing systems demonstrates the applicability of hybrid Petri net models in industrial contexts .Their approach leverages deep reinforcement learning within a Petri net structure to optimize task scheduling in real-time, further reinforcing the trend of combining formalism with adaptive learning for complex systems.

Our work differs from these approaches by proposing a more structured and automated integration of Petri nets and neural networks using Model-Driven Engineering techniques. We define in the first method three meta-models for classical Petri nets, neural networks, and Deep Petri Nets. We employ Triple Graph Grammar (TGG) for bidirectional synchronization and ATL (Atlas Trans- formation Language) for rule-based model transformation. In the second method, we introduce an intermediate metamodel Neural Network of Petri Net (NNPN) to streamline the fusion of structural and learning behaviors. This enables simulation, formal verification, and learning within a unified modeling framework, while maintaining traceability, modularity, and scalability.

Table 1: Comparative Analysis of Our Approaches

Authors	**Approach / Model**	**Contribution**	**Key Benefits**
[14]	Deep Fuzzy Petri Net (fuzzy logic + deep learning)	Hybrid framework combining rule-based reasoning and data-driven behavior	Improved interpretability and flexibility over conventional DNNs
[15]	Enhancements to fuzzy Petri nets	Incorporates fuzzy weights and probabilistic transitions	Better handling of uncertainty and imprecise data
[16]	Real-time intelligent decision-making (fuzzy Petri nets)	Combines symbolic reasoning with adaptive learning	Supports dynamic rule evaluation in evolving contexts
[17]	MDE for hybrid systems (PN + ML)	Uses meta-models and transformation rules	Formal integration of structural modeling with AI behavior
[18]	PN-based neural network interpretation	Transition firing mimics neuron activations	Provides traceable decision paths and explainability
[19]	PN-based deep reinforcement learning	Applied to real-time scheduling in manufacturing	Optimizes task scheduling in real time
Our work	Deep Petri Nets (DPN) via MDE (TGG + ATL, NNPN)	Defines three meta-models (PN, NN, DPN) + unified NNPN; automated transformations	Enables simulation, formal verification, adaptive learning, modularity, scalability

3 Background

Our objective is to propose an automatic integration of learning capabilities into classical Petri nets to obtain a Deep Petri Net model using model transformation techniques. The abstract syntax of each formalism (Petri Net, Neural Network, and Deep Petri Net) is described by means of dedicated meta-models. The transformation process is carried out using Triple Graph Grammar (TGG), which takes the source models as input, applies transformation rules, and generates the corresponding Deep PetriNet as output. In the following, we recall some basic concepts about Petri nets, neural networks, and model transformation using TGG.

3.1 Petri Net

Petri net (PN) is a mathematical and graphical model used to represent and analyze the dynamic behavior of discrete event systems [20].
A PN is formally defined as a4-tuple (P, T, F, W) where:
P is a finite set of places.
T is a finite set of transitions.
$F \subseteq (P \times T) \cup (T \times P)$is the flow relation.
$W: F \rightarrow N+$is a weight function.

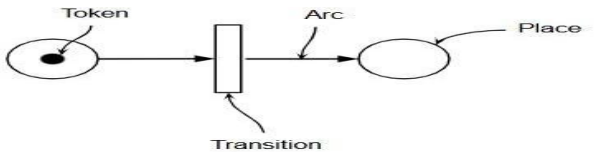

Fig 1 : Petri Net

3.2 Neural Networks

Neural networks (NN) are highly flexible function approximators capable of modeling complex, non-linear relationships in data. Deep neural networks, which consist of many hidden layers, are especially powerful in domains like image recognition, natural language processing, and time-series forecasting [21].
The essential characteristics of modern neural networks include:

- **Input Layer**: Accepts the raw input features.
- **Hidden Layers**: Perform complex transformations via activation functions (e.g.,ReLU, Sigmoid).
- **Output Layer**: Produces the final prediction or classification.
- **Weights and Biases**: Trainable parameters adjusted during the learning process.

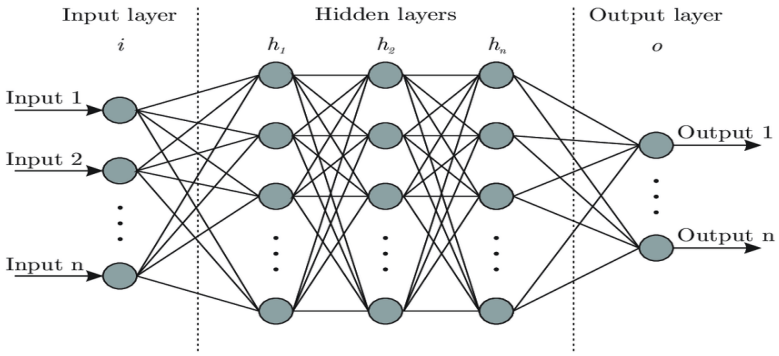

Fig 2: Neural Network

3.3 Deep Petri Nets

At Petri Nets (DPNs) are an advanced extension of classical Petri nets that integrate learning mechanisms inspired by neural networks, while maintaining the formal structure and analytical power of traditional Petri nets. Introduced and formalized in recent works such as in [22], DPNs aim to bridge the gap between symbolic modeling and data-driven machine learning by embedding layers of processing (akin to deep learning layers) into the Petri net structure. According to the authors in [22], a Deep Petri Net is built on the foundation of High-Level Fuzzy Petri Nets (HLFPNs) and is designed to perform both supervised and unsupervised learning. The architecture is composed of multiple concentric rings representing different layers, each containing transitions and places. These rings mimic the depth and connectivity found in neural networks, allowing the model to capture complex and hierarchical behavior. One of the defining features of DPNs is the presence of a supervisory node at the outermost layer. This node monitors parameter changes, helping track and control the learning process. This structural transparency enhances explainability, offering a clear advantage over traditional deep neural networks [23].

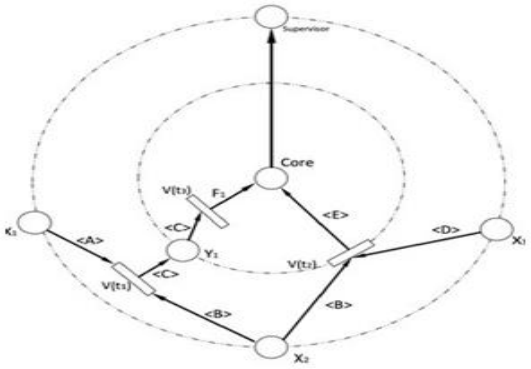

Fig 3: Deep Petri Net

3.3.1 Learning Algorithms within Deep Petri Nets

The learning process within Deep Petri Nets (DPNs) can be specified through three complementary paradigms:

3.3.1.1 Supervised Learning

Arc weights are optimized using gradient-based rules, similar to neural networks, where the loss is computed from the difference between expected and observed markings [24]

3.3.1.2 Unsupervised Learning

Transition firing probabilities adapt through frequency-based reinforcement, inspired by Hebbian learning, allowing the system to capture regular patterns of token flow [25]

3.3.1.3 Reinforcement Learning

DPNs integrate reward signals to optimize transition firing policies using Q-learning–like updates, which are effective in dynamic environments such as traffic or manufacturing systems[26]

3.4 Model Transformation

Model transformation is a fundamental concept in Model-Driven Engineering (MDE) that refers to the automated process of converting one or more source models into one or more target models, in accordance with defined transformation rules and meta-models. As formalized by the authors in [27], a model transformation takes input models conforming to a source meta-model and produces output models that conform to a target meta-model. This process is essential for automating tasks such as model refinement, synchronization, analysis, and code generation, thereby reducing human error and improving development efficiency. Model transformations are commonly classified into:

- **Model-to-Model** (M2M) transformations, which map structural or behavioral elements between models (e.g., using ATL or QVT).
- **Model-to-Text** (M2T) transformations, which generate code or documentation from models (e.g., using Acceleo or Xpand).
- **Text-to-Model** (T2M) transformations, which involve parsing textual representations into structured models.

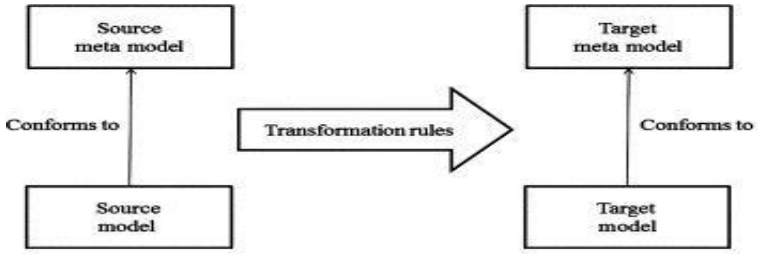

Fig 4: Model Transformation

4 The Proposed Method

In our project, we propose two model-driven engineering (MDE) approaches to transform a classical Petri Net into a Deep Petri Net. To realize this work we use two methods. The first converts the Petri Net into a Neural Network, and transforms the resulting Neural Network into a Deep Petri Net, and the second introduces an intermediate unified meta-model called Neural Network of Petri Net (NNPN), which directly embeds neural components into the Petri net structure. This NNPN meta-model is then transformed into a Deep Petri Net .We define meta-models using EMF and implement transformation rules with ATL to automate the process.

4.1 The First Approach

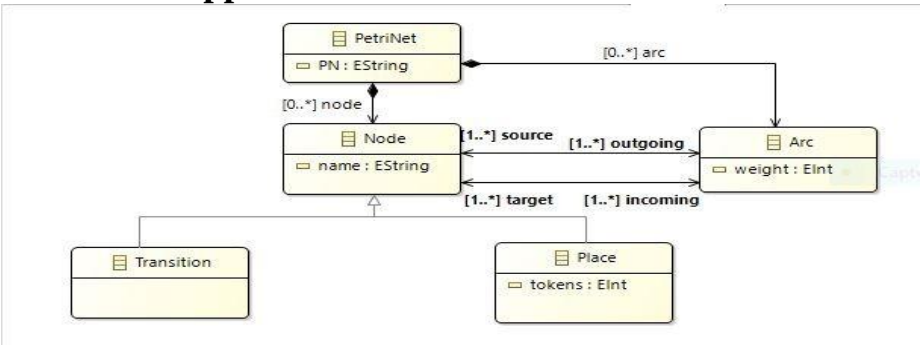

Fig 5: Meta Model of PN

4.1.1 Meta Model of Petri Net

- **The Class Petri Net**: This is the main container class representing an entire Petri Net model. It includes:
  - A name attribute (PN), a collection of Node elements and a collection of Arc elements.
- **Abstract Class Node**: This is a general super class for all elements that can appear in a Petri Net. It has:A name attribute of type EString and two concrete subclasses: Place and Transition
- **Class Place** (sub class of Node): Represents a location capable of holding tokens. It contains: Tokens attribute of type EInt.
- **Class Transition** (sub class of Node): Represents an event or activity that may occur, changing the state of the Petri Net by moving tokens between places.
- **Class Arc**: Represents a directed connection between nodes, describing the flow of tokens. Each Arc includes:A weight attribute (type EInt) to define the number of tokens transferred, an association with a source node and an association with a target node.

Relationships:

- An Arc always has a source and a target node. A Node can have one or more outgoing and incoming Arcs.

4.1.2 Meta Model of Neural Network

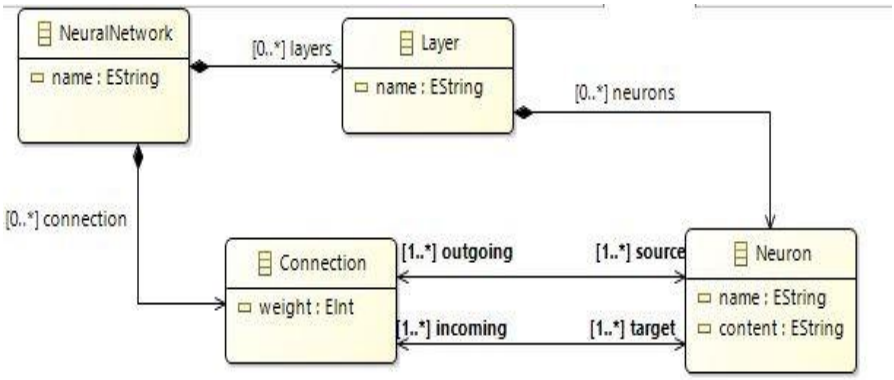

Fig 6: Meta Model of NN

This meta-model represents the structural definition of a Neural Network using a UML-style class diagram.

- **Class Neural Network:** This is the main container class that represents an entire neural network model. It includes:A name attribute of type EString, used to identify the network, a collection of Layer elements and a collection of Arc elements.
- **Class Layer:** Represents a level in the neural network (input layer, hidden layers, output layer). Each Layer contains: A name attribute of type EString and a collection of Neuron elements
- **Class Neuron:** Each Neuron has:A name attribute of type EString and a content attribute of type EString, which may represent internal data such as the activation function.
- **Class Connection:** Represents a directed link between neurons that facilitates

the flow of information. Each Connection includes:A weight attribute of type EInt, representing the strength of the connection, a source neuron and a target neuron**.**

Relationships:

- Each Neuron can have one or more incoming and outgoing Connections. Connections link neurons across layers or within the same layer, depending on the network topology.

4.1.3 ATL Rule For The Step One

The ATL (Atlas Transformation Language) code shown in Figure 7 defines the transformation rules for converting a Petri Net model into a Neural Network model.

Description of the ATL Rules:

- **Rule1 :PetrinetToNeuralnetwork**
  This rule transforms the entire Petri Net into a Neural Network .It simply copies the name attribute from the source model to the target model.

- **Rule2 :PlaceToNeuron**
  This rule converts each Place in the Petri Net into a Neuron. The neuron's name is set to the name of the place and its content is initialized with the token count. The neuron is then assigned to a specific type of layer based on its arc connections: No incoming arcs → assigned to the Input Layer, both incoming and outgoing arcs → assigned to the Hidden Layer and only incoming arcs → assigned to the Output Layer.

- **Rule3:TransitionToNeuron**
  This rule transforms a Transition into a Neuron. The neuron's content is fixed as "Function" to represent processing logic and it is always assigned to the Hidden Layer.

- **Rule4: Arc To Connection**
  This rule transforms each Arc in the Petri Net into a Connection in the neural network. It preserves the weight of the arc and maps the source and target elements directly to the corresponding neurons.

Remark:The outcome of the transformation from the Petri net to the neural network model provides strong support for the validity of the metamodel used in the second approach go to "The Second Approach", known as the Neural Network of Petri Net (NNPN). This intermediate metamodel effectively captures and unifies the structural and behavioral characteristics of both classical Petri nets and neural networks. By analyzing the transformation results, we observe clear alignment between elements of the two original formalisms, which demonstrates that NNPN can serve as a coherent and consistent foundation for integrating learning capabilities within a formal Petri net framework.

```
module PetriToNeural;
create OUT: rDN from IN: petriNet;


rule PetriNetToNeuralNetwork {
    from
        pn: petriNet!PetriNet
    to
        nn: rDN!NeuralNetwork (name <- pn.PN)}

rule PlaceToNeuron {
    from
        p : petriNet!Place
    to
        n : rDN!Neuron (name <- p.name,content <- p.tokens.toString()),
        l : rDN!Layer(
            name <-  if p.incoming->isEmpty()  then 'InputLayer'
                     else if p.incoming->notEmpty() and p.outgoing->notEmpty() then 'HiddenLayer'
                     else 'OutputLayer'
                     endif
                     endif)}

rule TransitionToNeuron {
    from
        t: petriNet!Transition
    to
        n: rDN!Neuron (name <- t.name,content<- 'Function'),
        l: rDN!Layer (name <- 'HiddenLayer')}

rule ArcToConnection {
    from
        a : petriNet!Arc
    to
        c : rDN!Connection (
            weight <- a.weight,
            source <- a.source,
            target <- a.target)}
```

Fig 7: ATL transformation rules from Petri Net to Neural Network

4.1.1 The Meta Model of Deep Petri Net

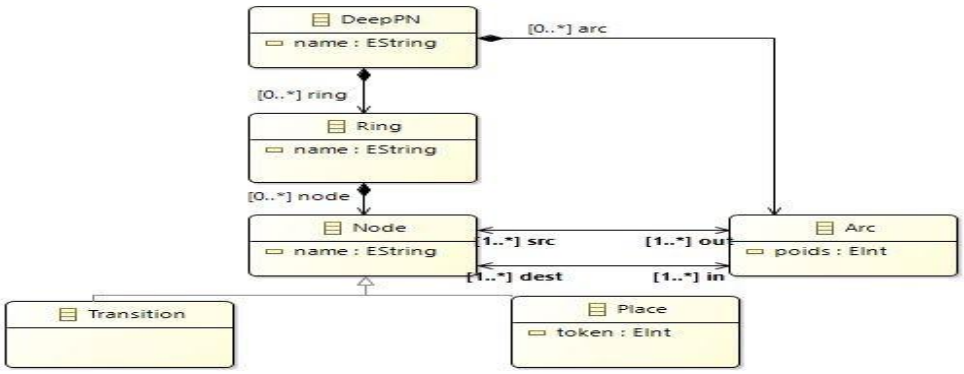

Fig 8: Meta Model of DPN

This meta-model defines the structure of a Deep Petri Net, an advanced formalism that integrates Petri Nets with layered

- **Class DeepPN**: The central class representing a Deep Petri Net. It includes:A name attribute of type EString, a collection of Ring elements and a collection of Arc elements.
- **Class Ring**: Each Ring represents a structural layer in the DeepPN. It contains:A name attribute of type EString and a collection of Node elements (Place or Transition).
- **Class Node**(abstract): An abstract super class representing elements of the net. Each Node has: A name attribute of type EString, one or more outgoing arcs (out) and one or more incoming arcs (in).
- **Class Place** (inherits from Node): Represents a state or condition within the system. It includes: A token attribute of type EInt.
- **Class Transition** (inherits from Node): Represents an event or action.
- **Class Arc**: Models the connection between two Nodes. It includes:A weight attribute of type EInt, indicating the number of tokens involved and a src (source node) and dest (target node) reference.

4.1.4 ATL Rule For The Step Two

Initially, The ATL code shown in Figure 9 defines the transformation rules for converting the resulting Neural Network model into a Deep Petri Net model.

Description of the ATL Rules:

- **Rule1:NeuralNetwork2DeepPN**

This rule transforms the entire Neural Network into a Deep PN. The name attribute is directly copied from the source model.

- **Rule2:Neuron2Place**

This rule converts a Neuron into a Place only if the neuron is not a function. It transfers:-The neuron's name and content to the Place, classifies the resultingPlace into an Ring: No incoming connections → assigned to Outer, no outgoing connections → assigned to Core and both input and output connections → assigned to Intermediate.

- **Rule3:Neuron2Transition**

This rule transforms a Neuron into a Transition and placed in the Intermediate if the following conditions are met:
– It has both incoming and outgoing connections,
– It represents a function.

- **Rule 4:Connection2Arc**

This rule converts a Connection from the neural network into an Arc in the Deep Petri Net.The transformation includes:
– copying the weight as poids, and mapping the source and target neurons to their corresponding nodes in the DPN

```
module rules2;
create OUT : dPN from IN : neuralnetwork;

rule NeuralNetwork2DeepPN {
    from nn : neuralnetwork!NeuralNetwork
    to dpn : dPN!DeepPN (
        name <- nn.name)}

rule Neuron2Place {
    from n : neuralnetwork!Neuron (
        not (n.content ='Function'))
    to
    p : dPN!Place (
        name <- n.name,
        token <- n.content.toInteger()),
    ring : dPN!Ring (
        name <-
            if n.incoming->isEmpty() then 'Outer'
            else if n.outgoing->isEmpty() then 'Core'
            else 'Intermediate'
            endif
            endif)}

rule Neuron2Transition {
    from n : neuralnetwork!Neuron (
        not n.incoming->isEmpty() and not n.outgoing->isEmpty() and n.content ='Function')
    to
    t : dPN!Transition (
        name <- n.name),
    ring : dPN!Ring (
        name <- 'Intermediate')}

rule Connection2Arc {
    from c : neuralnetwork!Connection
    to a : dPN!Arc (
        poids <- c.weight,
        src <- c.source,
        dest <- c.target)}
```

Fig 9 : ATL transformation rules from Neural Network to Deep Petri Net

4.2 The Second Approach

4.2.1 The Meta Model of Neural Network of Petri Net

This meta-model shown in Figure 10 defines a unified structure that integrates elements of both Petri nets and neural networks to support the modeling of systems with adaptive and analytical capabilities. It is composed of the following core components:

- **Class NNPN**: The root element of the model. It includes: A name attribute of type EString, a collection of Layers elements and a collection of Connexion elements

- **Class Layers:** Represents a layer in the neural network structure. It includes:A name attribute of type EString and a collection of Neuron elements.
- **Class Neuron** (abstract): A generic unit within a layer that can represent either a Place or a Transition. It includes: A name attribute of type EString, one or more incoming arcs and one or more outgoing arcs.
- **Class Place** (inherits from Neuron): Represents a state or condition within the system. It includes: A token attribute of type EInt.
- **Class Transition** (inherits from Neuron): Represents an event or action.
- **Class Connexion**: Represents a weighted link between two neurons (analogous to arcs in Petri nets or synapses in neural networks). It includes:A weight attribute of type EInt and references to source and target Neuron.

The Meta Model of Deep Petri Net: As for the Deep Petri net model, it corresponds to the one previously described under the referenced section. Go to "The Meta Model Of Deep Petri Net".

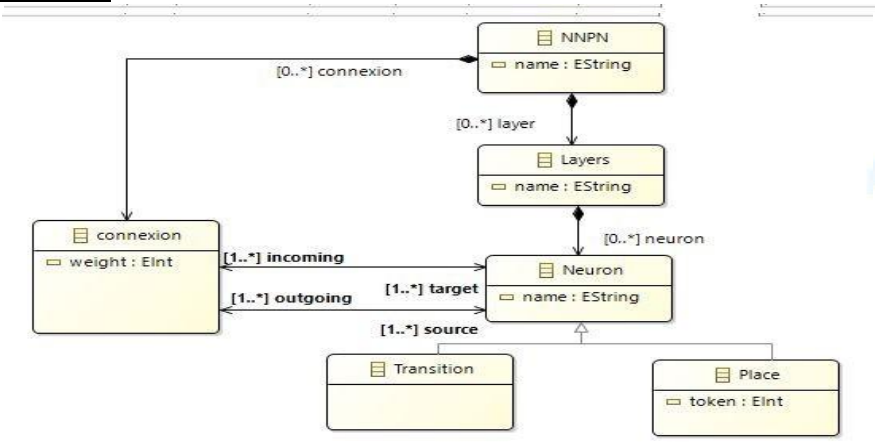

Fig 10 : Meta Model of NNPN

4.2.2 ATL Rule for The Second Approach

The ATL code shown in Figure 11 defines the transformation rules for converting the Neural Network of Petri Net model into a Deep Petri Net model.

Description of the ATL Rules:

- **Rule1:NNPN2DPN**

This rule transforms the root element of the source NNPN model (Neural Network of Petri Net) into the root element of the target DPN model (Deep Petri Net).

– The name attribute is directly copied from the source model.

- **Rule2:Neuron2Place**

This rule converts a neuron of type Place from the NNPN model into a Place in the DPN model. The name and token attributes are transferred and a Ring iscreated and assigned based on the neuron's connections: No incoming connections → Outer, no outgoing connections → Core and both incoming and outgoing connections → Intermediate

- **Rule3:NeuronT2Transition**

This rule transforms a neuron of type Transition into a Transition in the DPN model. The name attribute is copied directly and the transition is placed into a Ring named "intermediate".

- **Rule4:Connection2Arc**

This rule converts a connexion element from the NNPN model into an Arc in the DPN model. The weight is copied as poids and the source and target references are transferred to the arc

```
module rules;
create OUT : dPN from IN : nNPN;

rule NNPN2DPN {
    from nn : nNPN!NNPN
    to dpn : dPN!DPN (
        name <- nn.name)}

rule NeuronP2Place {
    from n : nNPN!Place
    to
    p : dPN!Place (
        name <- n.name,
        token <- n.token),
    ring : dPN!Ring (
        name <-
            if n.incoming->isEmpty() then 'Outer '
            else if n.outgoing->isEmpty() then 'Core'
            else 'Intermediate'
            endif
            endif)}

rule NeuronT2Transition {
    from tr : nNPN!Transition
    to
    t : dPN!transition (
        name <-tr.name
    ),
    ring : dPN!Ring (
        name <- 'intermediate')}

rule Connection2Arc {
    from c : nNPN!connexion
    to a : dPN!Arc (
        poids <- c.weight,
        source <- c.source,
        target <- c.target)}
```

Fig 11 : ATL transformation rules from NNPN to DPN

5 Case Studies

To validate our approaches, we applied it to a real-world examples: the **Smart Traffic Management System** and the **Automated Production Workshop**. These case studies demonstrate how our transformation process enhances the modeling, simulation, and analysis of complex, dynamic behaviors in intelligent systems.

5.1 The First Case Study

5.1.1 The First Approache

5.1.1.1 From Petri Net into Neural Network

In the first transformation step, we convert the initial Petri Net model into a Neural Network model using ATL rules. The input model is provided in XMI format and conforms to the Petri Net meta-model. Figure 12 illustrates the Petri Net representation of the Smart Traffic Management System

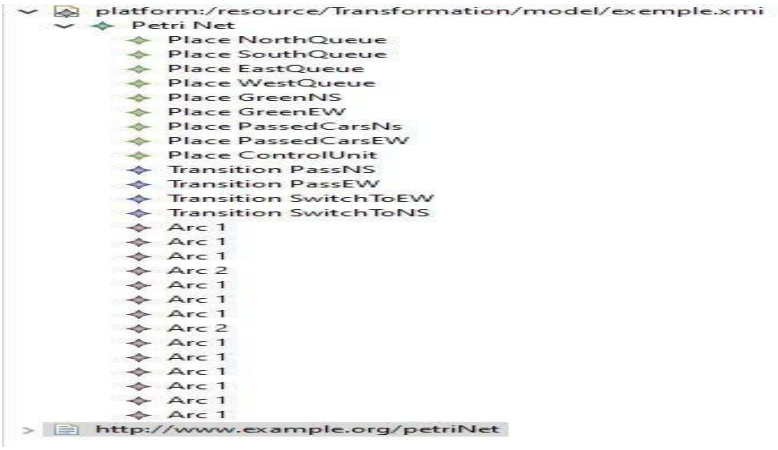

Fig 12: Smart Traffic Management System modeled as a Petri Net

Using the ATL transformation module (see Figure7), the Petri Net model is transformed into a Neural Network model. The resulting Neural Network model is saved in the NNresult.xmi file. Figures 13 show parts of the generated model.

Fig 13:Generated Neural Network model views

5.1.1.2 From Neural Network to Deep Petri Net

In the second transformation step, we take the result of the first transformation Figures 13 as input and apply the ATL transformation rules defined in Figure 9.This transformation converts the intermediate Neural Network model in to a Deep Petri Net (DPN) model. The resulting model is a Deep Petri Net.

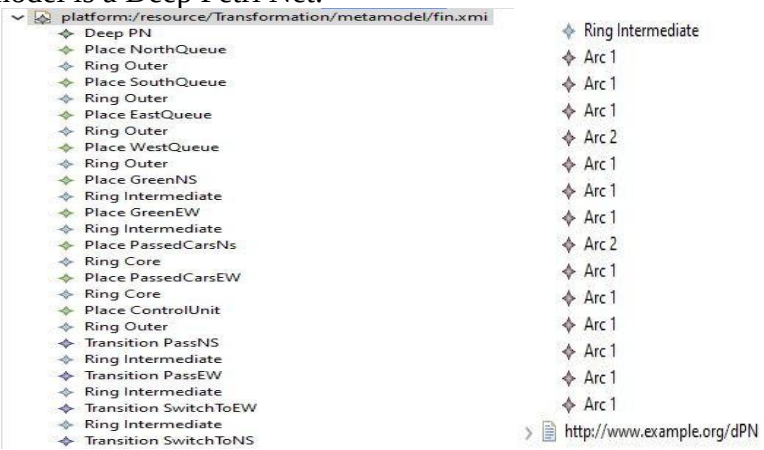

Fig 14:Generated Deep Petri Net model view

5.1.2 The Second Approach

5.1.2.1 From Neural Network of Petri Net to Deep Petri Net

In this transformation, we convert the Neural Network of Petri Net model into a Deep Petri Net model using ATL rules.

Figure 15 illustrates the Neural Network of Petri Net representation of the Smart Traffic Management System.

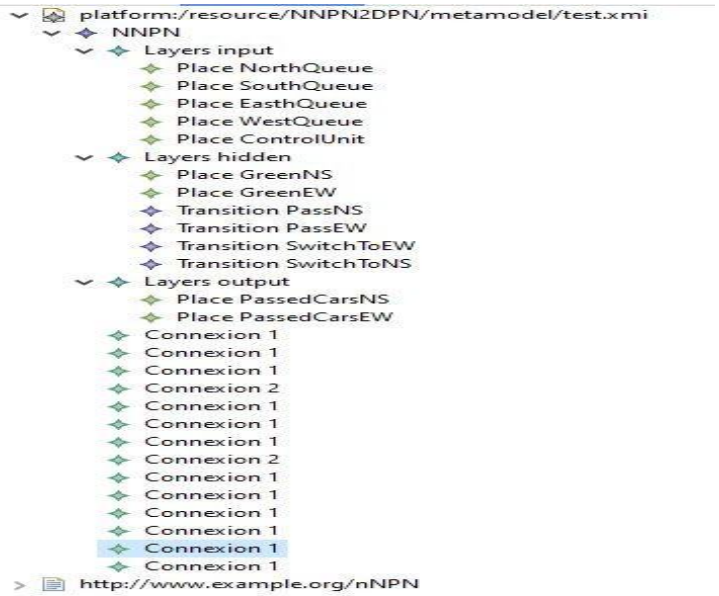

Fig 15: Smart Traffic Management System modeled as a NNPN

Using the ATL transformation module (see Figure11), the NNPN model is transformed into a DPN model. The resulting model is a Deep Petri Net.

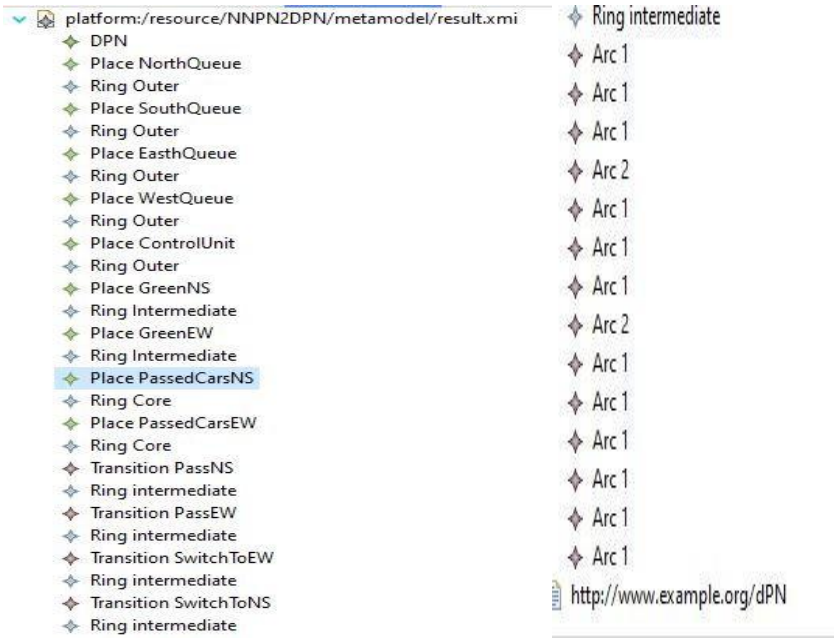

Fig 16: Generated Deep Petri Net model views

5.2 The Second Case Study

5.2.1 The First Approach

5.2.1.1 From Petri Net into Neural Network

For this transformation we generate an XMI file, that encodes the Petri net representation of the Automated Production Workshop.
Then we apply ATL rules to transform this representation into a neural network model

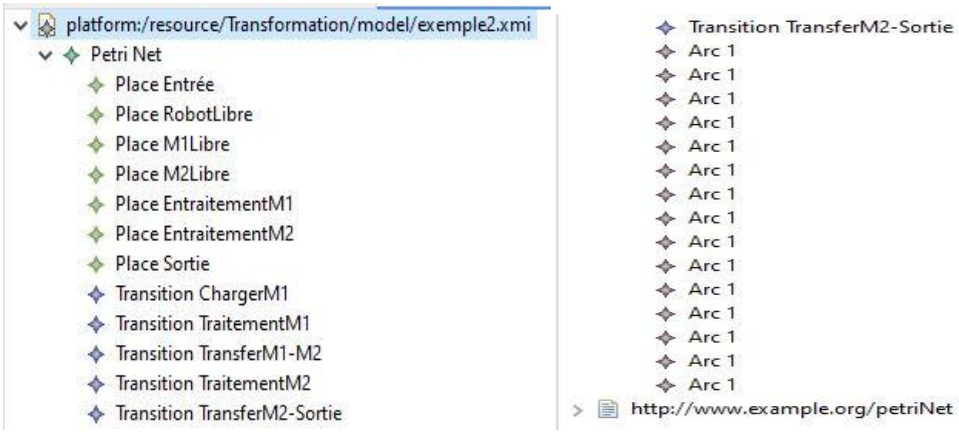

Fig 17: Automated Production Workshop modeled as a Petri Net

The resulting is saved in the XMI file. Figures 18 show the generated model.

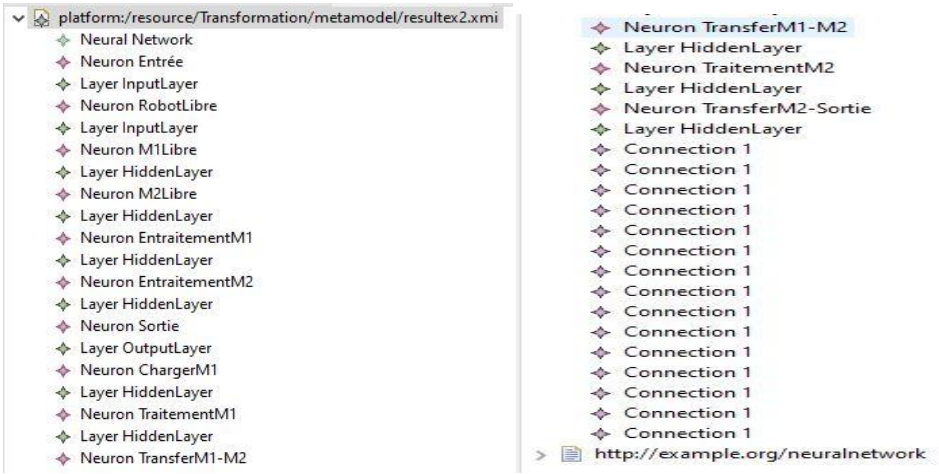

Fig 18: Generated Neural Network model

5.2.1.2 Transformation From The Resulting into Deep petri net

In this transformation, the model generated by the initial transformation is used as input, and ATL rules are applied to produce the final Deep Petri Net model.

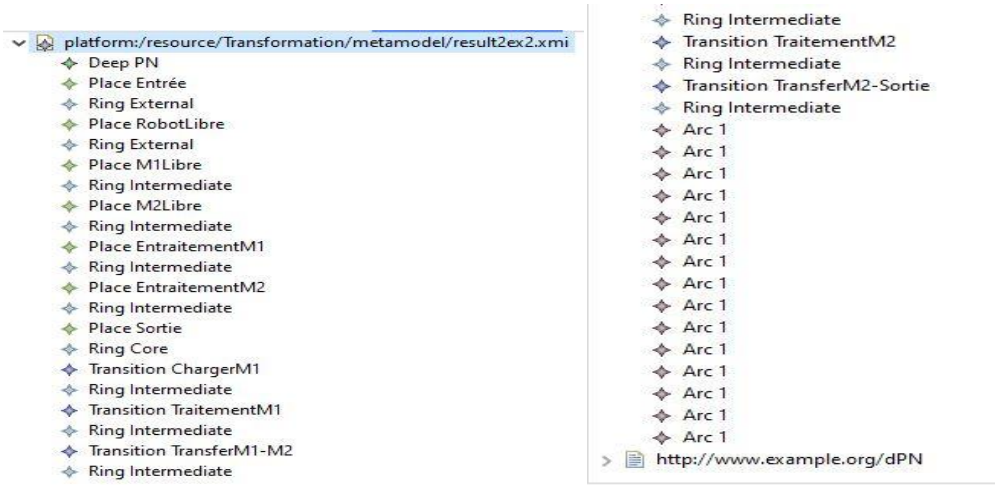

Fig 19: Generated Deep Petri Net model

5.2.2 The Second Approache

In this approach, we generate an XMI file that encodes the Neural Network of Petri Net (NNPN) representation of the Automated Production Workshop.

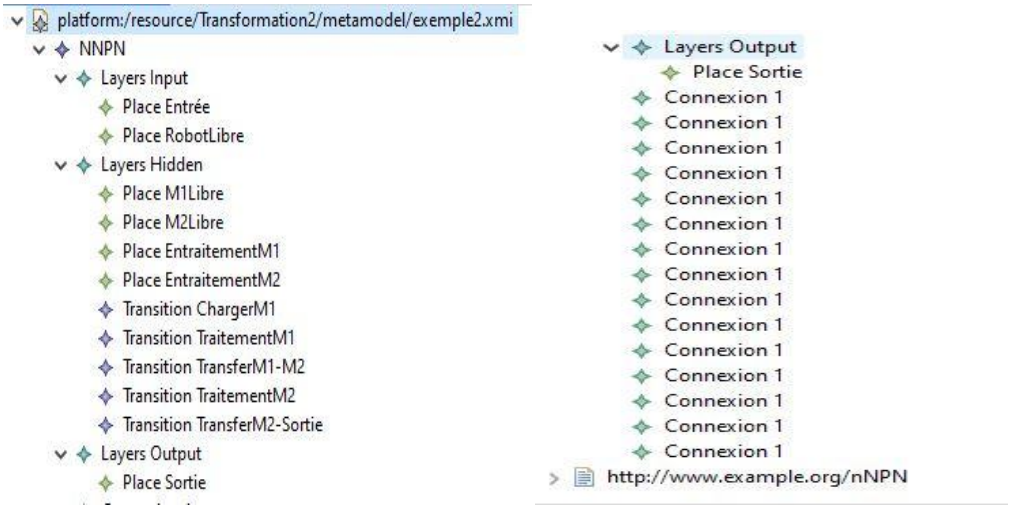

Fig 20: Automated Production Workshop as a NNPN

Then we apply ATL transformation rules to generate the final Deep Petri Net model.

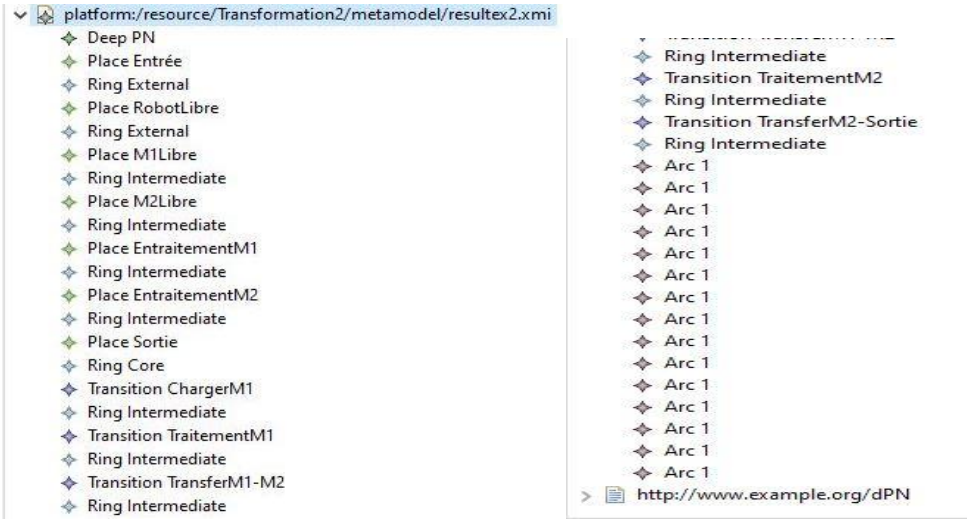

Fig 21: Generated Deep Petri Net model

6. Scalability and Complexity Analysis

The proposed DPN framework scales efficiently to large models through modular ring-based structures and automated transformations with TGG and ATL. Transformation complexity remains polynomial in the number of nodes and arcs, and simulation benefits from reduced recalculations due to adaptive learning. Although reachability analysis inherits the exponential complexity of Petri Nets, the layered DPN representation mitigates state-space explosion, ensuring better scalability compared to classical PN and other hybrid models.

7. Practical Benefits of DPNs

The DPN framework provides measurable advantages: it reduces modeling effort through automated transformations, scales effectively to large models, preserves interpretability by keeping transitions traceable, and improves simulation accuracy by adapting more closely to real-world dynamics.

8. Conclusion and future works

In this paper, we introduced new approaches that combines the formal modeling power of Petri Nets with the learning capabilities of neural networks, resulting in the Deep Petri Net (DPN) model. , we presented a novel approach that bridges the gap between symbolic modeling and subsymbolic learning by integrating the formal rigor of Petri Nets with the adaptive capabilities of neural networks. This integration led to the formulation of the Deep Petri Net (DPN) model a hybrid framework capable of representing both structural logic and dynamic learning processes. Our methodology was structured around two main transformation approaches. The first approach defines three meta-models: the first for classical Petri Nets, the second for artificial neural networks, and the third for Deep Petri Nets. These meta-models were developed using the Eclipse Modeling Framework (EMF), and transformations between them were implemented using the Atlas Transformation Language (ATL). This enabled a step-by-step conversion

from symbolic models to hybrid intelligent systems. The second approach introduces a unified intermediate meta-model, the Neural Network Petri Net (NNPN), which directly integrates neural components into the Petri Net structure. This unified model is subsequently transformed into a Deep Petri Net using ATL rules. The direct embedding approach simplifies the transformation chain while maintaining consistency and cohesion. Looking ahead, several promising directions can be explored to improve and extend the proposed framework. First, enhancing the scalability and performance of the DPN framework is essential. This can be achieved by optimizing transformation rules and exploring modularization strategies to partition complex models into smaller, manageable components. Second, the framework could be extended to support additional learning paradigms such as reinforcement learning and unsupervised learning. These capabilities would increase the adaptability of the system in diverse scenarios. Finally, improving the explainability of the decision-making processes within Deep Petri Nets will be crucial, particularly for applications in safety-critical domains such as healthcare, autonomous vehicles, or industrial automation. In conclusion, the Deep Petri Net framework provides a strong foundation for unifying symbolic reasoning with adaptive learning. Through its dual transformation approaches it opens new possibilities for the design of intelligent, explainable, and traceable systems. We believe this approach represents a valuable contribution to the field of neuro-symbolic artificial intelligence and hybrid system modeling.

References

[1] T.Murata, (1989), "Petri Nets: Properties, Analysis and Applications," *IEEE Transactions on Communications*, vol. 77, no. 4, pp. 541-580.

[2] J .Deseland W.Reisig ,(2001), Lectures on Petri Nets I: Basic Models.

[3] W.M. P. vander Aalst, (2020), Process Mining: Data Science in Action (2nded.).

[4] X. Xu, Y. Lu, B. Vogel-Heuser, and L. Wang,(2021), *Industrial AI and Digital Transformation for Smart Manufacturing : A Review* , Engineering, vol.7 ,no.6,pp.738–75,.

[5] D. Fahland and M. Weidlich, (2023), Conformance Checking and Process Enhancement Using Petri Nets, Foundations of Computing and Decision Sciences, vol. 48, no. 1, pp. 1–27.

[6] Y. LeCun, Y. Bengio, and G. Hinton,(2015) ,"Deep learning," Nature, vol. 521, pp. 436-444.

[7] I. Goodfellow, Y. Bengio, and A. Courville , (2016), Deep Learning, MIT Press.

[8] W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller, (2021), *Explainable AI: Interpreting*, *Explaining and Visualizing Deep Learning*.

[9] D. Gunning and D. Aha, DARPAs ,(2019), *Explainable Artificial Intelligence (XAI) Program,* AI Magazine, vol. 40, no. 2, pp. 44–58.

[10] Y. Liu, H. Zhang, and L. Wang ,(2020), *Deep Fuzzy Petri Nets: A Transparent Neural Reasoning Framework*, Knowledge-Based Systems, vol. 192, pp. 105-354.

[11] A. d'AvilaGarcez, L. C. Lamb, and D. M. (2010), Gabbay, *Neural-Symbolic Cognitive Reasoning*.

[12] T. R. Besold, A. d'AvilaGarcez, S. Bader, H. Bowman, P. Domingos, P. Hitzler et al. (2017), *Neuralsymbolic Learning and Reasoning: A Survey and Interpretation,* Frontiers in Artificial Intelligence and Applications, vol. 291, pp. 1–59.

[13] P. Stevens, (2008), Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions, Software and Systems Modeling, vol. 9, no. 1, pp. 7–20.

[14] L. Hu, Z. Liu, W. Hu, Y. Wang, J.-H. Xu, and F.-Y. Wu, (2020), Petri Net-Based Dynamic Scheduling of Flexible Manufacturing System via Deep Reinforcement Learning with Graph Convolutional Network, Journal of Manufacturing Systems, vol. 55, pp. 1–14.

[15] F. Qiao, Q. Wu, L. Li, Z. Wang, and B. Shi, (2011), A Fuzzy Petri Net-Based Reasoning Method for Rescheduling, Transactions of the Institute of Measurement and Control, vol. 33, no. 3–4, pp. 359–370.

[16] H.-C. Liu, Q.-L. Lin, L.-X. Mao, and Z.-Y. Zhang, (2015), Dynamic Adaptive Fuzzy Petri Nets for Knowledge Representation and Reasoning, inFuzzy Petri Nets for Knowledge Representation, Acquisition and Reasoning, pp. 63–83.

[17] F. Kordon, H. Garavel, and J. Thierry-Mieg, (2021), Formalization of AI Systems with Petri Nets and MDE, Journal of Systems Architecture, vol. 121, pp.102-688.

[18] H. Yin, and al., (2023), Petri Net-Based Interpretability for Deep Learning Models, Information Sciences, vol. 636, pp. 230–245.

[19] Y. Zhao, H. Liu, M. Chen, and X. Wang, (2024), Petri-net-based deep reinforcement learning for real- time scheduling of automated manufacturing systems, Journal of Manufacturing Systems, vol. 74, pp. 995–1008.

[20] J. L. Peterson , (1981), Petri Net Theory and the Modeling of Systems, Prentice Hall ,.

[21] S. Raschka and V. Mirjalili, (2022), Python Machine Learning (4th ed.).

[22] Y.-N. Lin, T.-Y. Hsieh, C.-Y. Yang, V. R. L. Shen, T. T.-Y. Juang, and W.-H. Chen, (2020), Deep Petri Nets of Unsupervised and Supervised Learning, Measurement and Control, vol. 53,no. 7-8, pp.1-11.

[23] H. Qi, M. Guang, J. Wang, and C. Jiang, (2023), A Perspective on Petri Net Learning,

[24] Y. LeCun, Y. Bengio, and G. Hinton,(2015) ,*Deep learning*, .

[25] Y.-N. Lin and al.,(2020) , *Deep Petri Nets of Unsupervised and Supervised Learning*, Measurement and Control.

[26] Y. Zhao and al.,(2024), *Petri-net-based deep reinforcement learning for real-time scheduling of automated manufacturing systems*, Journal of Manufacturing Systems.

[27] T. Mens and P. Van Gorp, (2021), A Taxonomy of Model Transformation Approaches, Software and Systems Modeling (SoSyM), vol. 20, no. 1, pp. 23–44.