RPN 2: On Interdependence Function Learning

Towards Unifying and Advancing CNN, RNN, GNN, and Transformer

Jiawei Zhang
IFM Lab[*]
Department of Computer Science
University of California, Davis
`jiawei@ifmlab.org`
Github: https://github.com/jwzhanggy/tinyBIG
Project Official Website: https://www.tinybig.org

tinyBIG v0.2.0

> *"Nothing in the world stands by itself. Every object is a link in an endless chain and is thus connected with all the other links."*
>
> — DIALECTICAL MATERIALISM (*Alexander Spirkin*)

Abstract

This paper builds upon our previous work on the Reconciled Polynomial Network (RPN) [89]. In our prior research, we introduced RPN as a general model architecture comprising three component functions: data expansion function, parameter reconciliation function, and remainder function. By strategically combining these components functions, we demonstrated RPN's versatility in constructing models for addressing a wide range of function learning tasks on multi-modal data. Furthermore, RPN also unified diverse base models, including PGMs, kernel SVM, MLP, and KAN, into its canonical representation.

The original RPN model was designed under the assumption of input data independence, presuming the independence among both individual instances within data batches and attributes in each data instance. However, this assumption often proves invalid for function learning tasks involving complex, interdependent data such as language, images, time series, and graphs. Ignoring such data interdependence may inevitably lead to significant performance degradation.

To overcome these limitations, we introduce the new **Reconciled Polynomial Network (version 2)**, namely **RPN 2**, in this paper. By incorporating data and structural interdependence functions, RPN 2 explicitly models data interdependence via new component functions in its architecture.

[*] © 2024 IFM Lab. All rights reserved. The RPN 2 project and **TINYBIG V0.2.0** toolkit are developed and maintained by IFM Lab.

This enhancement not only significantly improves RPN 2's learning performance but also substantially expands its unifying potential, enabling it to encompass a broader range of contemporary dominant backbone models within its canonical representation. These backbones include, but are not limited to, convolutional neural networks (CNNs), recurrent neural networks (RNNs), graph neural networks (GNNs), and Transformers. Our analysis reveals that the fundamental distinctions among these backbone models primarily stem from their diverse approaches to defining the interdependence functions. Furthermore, this unified representation opens up new opportunities for designing innovative architectures with the potential to surpass the performance of these dominant backbones.

To evaluate the effectiveness of RPN 2, we conducted extensive empirical experiments on a diverse set of benchmark datasets spanning multiple modalities, including images, language, time series, and graphs. The results demonstrate that RPN 2, with its enhanced interdependence functions, significantly outperforms the previous RPN model across all evaluated benchmarks. Furthermore, by strategically selecting component functions, we developed novel RPN 2-based models that enhance existing backbones in various function learning tasks, such as image classification, language classification, time series forecasting, and graph learning. These findings underscore the versatility and potential of RPN 2 in advancing backbones across diverse domains for addressing different function learning tasks.

The new Reconciled Polynomial Network (version 2), *i.e.,* RPN 2, presents significant improvements over its predecessor. The data interdependence component functions substantially enhance RPN 2's learning effectiveness in complex function learning tasks involving interdependent data. Building on RPN's unification capabilities, RPN 2 integrates a broader range of backbone models into its canonical framework. Importantly, RPN 2 reveals that the primary distinction between existing backbone architectures lies in their definitions of data interdependence functions. This key insight opens avenues for designing new superior backbones, positioning RPN 2 as a powerful framework for advancing function learning development.

To reflect these advancements, we have also upgraded the tinyBIG toolkit into the new **tinyBIG v0.2.0**. This updated version seamlessly incorporates interdependence functions into RPN 2 model design, substantially enhancing its learning capabilities. Additionally, **tinyBIG v0.2.0** also introduces a new family of data compression and fusion functions and expands the existing repertoire of data expansion and parameter reconciliation functions. Detailed information about the updated **tinyBIG v0.2.0** toolkit is available at the project's GitHub repository and dedicated website, with their URLs provided above.

Key Words: Reconciled Polynomial Network; Interdependence Function; Unified Model Architecture; Multi-modal Learning; Function Learning

Contents

1 Introduction

This paper is a follow-up work of our previous study on the Reconciled Polynomial Network (RPN) [89]. In our prior work, we introduced RPN as a general model architecture comprising three component functions: data expansion function, parameter reconciliation function, and remainder function. Inspired by Taylor's Theorem, RPN proposes to disentangle the underlying function to be inferred as the inner product of a data expansion function with a parameter reconciliation function. Together with the remainder function, RPN can accurately approximate the underlying functions that govern data distributions. Our previous research demonstrated RPN's versatility in constructing models with varying complexities, capacities, and levels of completeness, which can also serve as a framework for unifying diverse base models, including PGMs, kernel SVM, MLP, and KAN.

Meanwhile, the previous RPN model was built on the assumption that data instances in the training batches are independent and identically distributed. Moreover, within each data instance, RPN also presumed the involved attributes to be independent as well, treating them separately in the expansion functions. However, as illustrated in Figure 1 (a)-(d), these assumptions often prove invalid for function learning tasks on complex, and interdependent data such as images, language, time series, and graphs. In such data, strong interdependence relationships typically exist among both instances and attributes. Ignoring these data interdependencies, as the previous RPN model does, will significantly degrade learning performance.

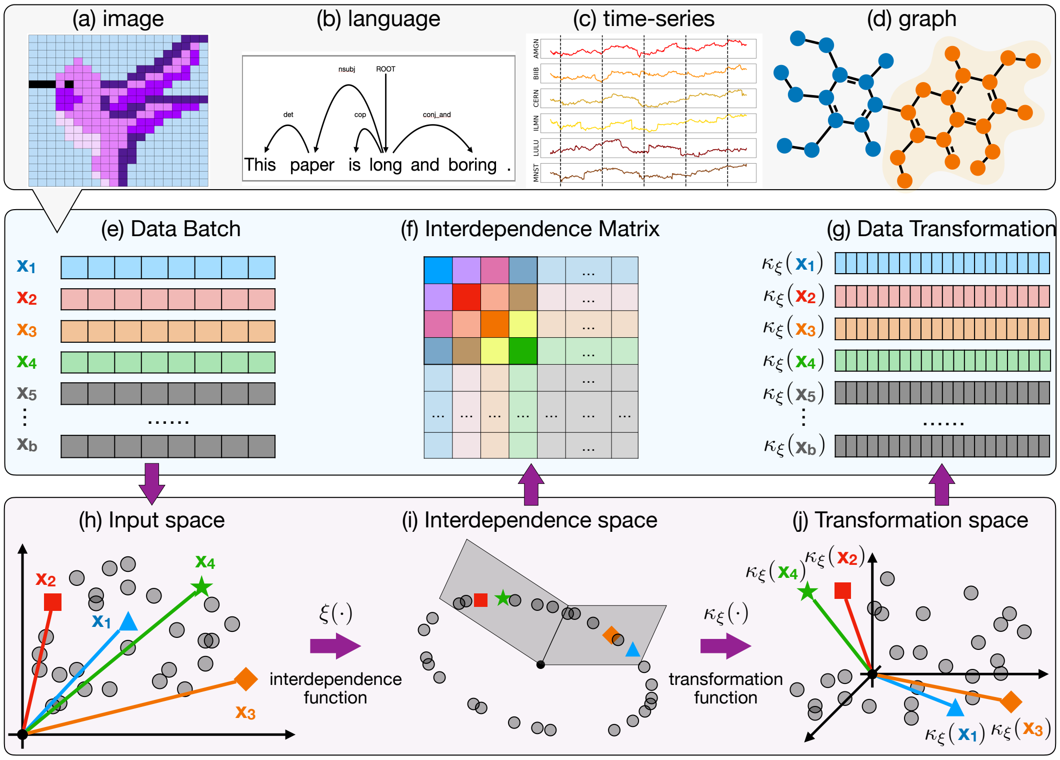


Figure 1: An illustration of data interdependence modeling in RPN 2. Plots (a)-(d) show some examples of interdependent data examples: (a) a colored image of a hummingbird, (b) a sentence and its parsing structure, (c) time-series price data of six stocks, and (d) the Myricetin molecular graph. These data instances in different modalities can all be fed as inputs to the RPN 2 model. Plots (e)-(g) provide the matrices representations of the input data in RPN 2: (e) input data batch, (f) calculated (instance) interdependence matrix, and (g) output data batch after transformation. Plots (h)-(j) indicate the learning space of RPN 2: (h) input data space, (i) interdependence space, and (j) data transformation space used for defining the interdependence function and data transformation function.

In this paper, we propose a redesign of the RPN architecture, introducing the new **RPN 2** (*i.e.,* Reconciled Polynomial Network version 2.0) model. As illustrate by Figure 1, RPN 2 incorporates a novel component, the **interdependence functions**, to explicitly model diverse relationships among both data instances and attributes. While we refer to this component as "*interdependence*", this function actually captures a wide range of relationships within the input data, including structural interdependence, logical causality, statistical correlation, and numerical similarity or dissimilarity.

Technically, as shown in Plots (e)-(j) of Figure 1, RPN 2 employs interdependence functions to generate matrices that capture relationships among data instances and attributes. These functions take data batches as input, and some also incorporate additional structural information, such as underlying topological connections and geometric shapes, to compute comprehensive interdependence matrices. The resulting matrices are typically sparse and are applied efficiently to input data batches (both before and after expansion) using sparse matrix multiplication, optimizing computational resources in terms of both space and time.

The introduction of these interdependence functions significantly enhances RPN 2's ability to model complex function learning tasks involving interdependent data. Moreover, this advancement greatly broadens RPN 2's unifying capacity, enabling it to encompass a wider range of prevalent backbone architectures within its canonical representation, including but not limited to, convolutional neural networks (CNNs), recurrent neural networks (RNNs), graph neural networks (GNNs), and Transformers. Notably, their unified representations reveal that existing backbone architectures primarily differ in their definitions of data interdependence functions. This key insight not only opens new avenues for designing more advanced models but also positions RPN 2 as a powerful framework for further innovation in function learning architecture design.

To evaluate the effectiveness of the new RPN 2 model for deep function learning tasks on interdependent data, this paper presents extensive empirical experiments across a diverse set of benchmark datasets, including image, language, time-series, and graph datasets. Leveraging grid-based geometric structural interdependence functions, RPN 2 effectively captures local interdependence among image patches on benchmarks like MNIST and CIFAR-10. These grid-based functions allow image patches to adopt various shapes, such as cuboids and cylinders, each offering distinct modeling advantages. For language and time-series data, RPN 2 utilizes chain-structured topological interdependence functions, excelling in tasks such as language classification and time-series forecasting. In the case of graph-structured data, RPN 2 demonstrates superior performance by effectively modeling structural relationships within graphs using its interdependence functions. With these diverse interdependence functions, RPN 2 achieves performance comparable to leading models like CNNs, RNNs, and GCNs across these multimodal benchmarks.

What's more, to facilitate the adoption, implementation, and experimentation of the new RPN 2, we have updated the TINYBIG toolkit introduced in our previous paper [89] to the new **TINYBIG v0.2.0**. This updated version incorporates interdependence modeling capabilities in the RPN 2 model design and learning, updating the head and layer modules, and the RPN 2 model architecture. Additionally, **TINYBIG v0.2.0** introduces a new family of data compression and multi-input function functions for embedding data vectors into lower-dimensional spaces. RPN 2 has also updated the existing repertoire of data expansion and parameter reconciliation functions to facilitate the implementation of RPN 2-based models. This updated toolkit enables researchers to rapidly design, customize, and deploy new RPN 2 models across a wide spectrum of function learning tasks on various interdependent datasets.

We summarize the contributions of this paper as follows:

- **RPN 2 for Data Interdependence Modeling**: We redesign the reconciled polynomial network model, introducing RPN 2 with data interdependence modeling capabilities. Equipped with interdependence functions, RPN 2 can learn interdependence relationships among both instances and attributes of the data batch, significantly improving its learning performance for function learning tasks on complex and interdependent data.

- **Data Interdependence Functions**: We present a suite of interdependence functions capable of learning various categories of interdependence relationships among instances and attributes of the input data batch. These functions utilize the input data batches along with their necessary geometric and topological structure information to calculate interdependence matrices comprising scores between pairwise instances and attributes.

- **Backbone Model Unification and Advancing**: The data interdependence functions significantly extend RPN 2's unifying potential, encompassing a broader range of frequently utilized backbone architectures (including CNN, RNN, GNN, and Transformer) within its canonical representation. This unified representation reveals that existing backbone architectures primarily differ in their data interdependence function definitions, opening new avenues for designing superior models and positioning RPN 2 as a powerful framework for advancing future backbone model design.

- **Experimental Investigation**: To evaluate RPN 2's learning performance, we present a series of empirical experiments on numerous real-world benchmark datasets across various function learning tasks, including image classification, language classification, time-series forecasting, and graph classification. The results demonstrate the effectiveness and superior performance of new backbone models designed based on RPN 2 compared to existing dominant backbone architectures.

- **Toolkit Updating**: We update the TINYBIG toolkit to the new **TINYBIG V0.2.0**, incorporating implementations of all data interdependence functions introduced in this paper. Additionally, we introduce a new family of data compression and data fusion functions that can compress and fuse input data into lower-dimensional spaces. The existing repertoire of data expansion and parameter reconciliation functions has also been expanded to include several new component function implementations in the **TINYBIG V0.2.0** toolkit.

Paper Organization: The remaining parts of this paper are organized as follows. Section 2 covers notations, function learning task formulations, and essential background knowledge of the reconciled polynomial network model introduced in our previous paper [89]. Section 3 introduces the new data interdependence concept, several data interdependence examples and modeling approaches. Section 4 provides detailed descriptions of the new RPN 2 model's architecture and design mechanisms. Our library of new data interdependence functions, data compression functions, and other newly added component functions will be presented in Sections 5, 6 and 7, respectively. Section 8 demonstrates how RPN 2 unifies and represents existing backbone architectures. Experimental evaluation of RPN 2's performance on numerous benchmark datasets is provided in Section 9. Interpretations of RPN 2 with the interdependence functions are provided in Section 10 from the theoretic machine learning and biological neuroscience perspectives. Section 11 discusses RPN 2's intellectual merits, limitations, and potential future opportunities. Finally, we introduce related works in Section 12 and conclude the paper in Section 13.

2 Notation System and Background Knowledge

To ensure the self-containment of this paper, we preface the technical descriptions of the novel RPN 2 model with a concise overview of both the function learning task and the original Reconciled Polynomial Network (RPN) introduced in the previous paper [89] in this section. Before introducing the background knowledge, we will also briefly describe the notations that will be used in this paper.

2.1 Notation System

In the sequel of this paper, unless otherwise specified, we adopt the following notational conventions: lower-case letters (*e.g.,* x) represent scalars, upper-case letters (*e.g.,* X) represent variables, lower-case bold letters (*e.g.,* $\mathbf{x}$) denote column vectors, boldface upper-case letters (*e.g.,* $\mathbf{X}$) denote matrices and high-order tensors, and upper-case calligraphic letters (*e.g.,* $\mathcal{X}$) denote sets.

For a vector $\mathbf{x}$, we denote its i-th element as $\mathbf{x}(i)$ or $\mathbf{x}_i$, which will be used interchangeably. We use $\mathbf{x}^\top$ to represent the transpose of vector $\mathbf{x}$. For vector $\mathbf{x}$, its L_p-norm is defined as $\|\mathbf{x}\|_p = \left(\sum_i |\mathbf{x}(i)|^p\right)^{\frac{1}{p}}$. The elementwise product of vectors $\mathbf{x}$ and $\mathbf{y}$ of the same dimension is denoted by $\mathbf{x} \odot \mathbf{y}$, their inner product by $\langle \mathbf{x}, \mathbf{y} \rangle$, and their Kronecker product by $\mathbf{x} \otimes \mathbf{y}$.

For a matrix $\mathbf{X}$, we represent its i-th row and j-th column as $\mathbf{X}(i, :)$ and $\mathbf{X}(:, j)$, respectively. The (i, j)-th entry of matrix $\mathbf{X}$ is denoted as $\mathbf{X}(i, j)$, and its transpose is represented as $\mathbf{X}^\top$. The elementwise and Kronecker product operations extend to matrices $\mathbf{X}$ and $\mathbf{Y}$ as $\mathbf{X} \odot \mathbf{Y}$ and $\mathbf{X} \otimes \mathbf{Y}$, respectively. The Frobenius-norm of matrix $\mathbf{X}$ is represented as $\|\mathbf{X}\|_F = \left(\sum_{i,j} |\mathbf{X}(i, j)|^2\right)^{\frac{1}{2}}$, and its infinity-norm is defined as its maximum absolute row sums, *i.e.,* $\|\mathbf{X}\|_\infty = \max_i \left(\sum_j |\mathbf{X}(i, j)|\right)$. The two-to-infinity subordinate vector norm of matrix $\mathbf{X}$ is defined as $\|\mathbf{X}\|_{2\to\infty} = \sup_{\|\mathbf{z}\|_2=1} \|\mathbf{X}\mathbf{z}\|_\infty$.

For two variables X and Y, we denote their independence as $X \perp\!\!\!\perp Y$, and their conditional independence given a condition C as $X \perp\!\!\!\perp Y|C$. Conversely, their interdependence and conditional interdependence are represented as $X \not\perp\!\!\!\perp Y$ and $X \not\perp\!\!\!\perp Y|C$, respectively. If X exhibits direct dependence on Y, we express this as $X \leftarrow Y$ or $Y \rightarrow X$.

2.2 Function Learning Task

As outlined in the previous RPN paper [89], **function learning**, as the most fundamental task in machine learning, aims to construct a general model comprising a sequence of component functions that infer relationships between inputs and outputs. The term "function" in this context refers to not only the mathematical function components constituting the RPN model but also the cognitive function of RPN as an intelligent system generating the desired output responses from input signals.

In function learning, without prior assumptions about data modalities, the corresponding input and output data can manifest in various forms, including but not limited to continuous numerical values (*e.g.,* continuous functions and time series), discrete categorical features (*e.g.,* images, point clouds and language), probabilistic variables (defining dependency relationships between inputs and outputs), interconnected structures (*e.g.,* grids, graphs and chains) and other forms.

Definition 1 ***(Function Learning)****: Formally, given input and output spaces $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively, the underlying mapping governing the data projection between these spaces can be denoted as:*

$$f : \mathbb{R}^m \to \mathbb{R}^n. \tag{1}$$

Function learning aims to construct a model g as a composition of mathematical function sequences $g_1, g_2, \cdots, g_K$ to project data across different vector spaces, which can be represented as:

$$g : \mathbb{R}^m \to \mathbb{R}^n, \text{ and } g = g_1 \circ g_2 \circ \cdots \circ g_K, \tag{2}$$

where the $\circ$ notation denotes component function integration and composition operators. The component functions g_i can be defined on either the input data or the model parameters.

For input vector $\mathbf{x} \in \mathbb{R}^m$, if the output generated by the model approximates the desired output, i.e.,

$$g(\mathbf{x}|\mathbf{w}, \boldsymbol{\theta}) \approx f(\mathbf{x}), \tag{3}$$

then model g can serve as an approximated of the underlying mapping f for the provided input $\mathbf{x}$.

Notations $\mathbf{w} \in \mathbb{R}^l$ and $\boldsymbol{\theta} \in \mathbb{R}^{l'}$ denote the learnable parameters and hyper-parameters of the function learning model, respectively. By convention, the hyper-parameter vector $\boldsymbol{\theta}$ may be omitted from the model representation, which simplifies the model notation to be $g(\cdot|\mathbf{w})$.

2.3 Reconciled Polynomial Network (RPN) Model

To address the function learning tasks, our previous paper [89] introduced the **Reconciled Polynomial Network** (**RPN**) model as a general framework with versatile architectures. The RPN model comprises three component functions, including data expansion function, parameter reconciliation function, and remainder function. This architecture disentangles input data from model parameters and approximates the target functions as the inner product of the data expansion function with the parameter reconciliation function, subsequently summed with the remainder function.

Formally, given the underlying data distribution mapping $f : \mathbb{R}^m \to \mathbb{R}^n$, we represent the RPN model proposed to approximate function f as follows:

$$g(\mathbf{x}|\mathbf{w}) = \langle \kappa(\mathbf{x}), \psi(\mathbf{w}) \rangle + \pi(\mathbf{x}), \tag{4}$$

where

- $\kappa : \mathbb{R}^m \to \mathbb{R}^D$ is named as the **data expansion function** and D is the target expansion space dimension.
- $\psi : \mathbb{R}^l \to \mathbb{R}^{n \times D}$ is named as the **parameter reconciliation function**, which is defined only on the parameters without any input data.
- $\pi : \mathbb{R}^m \to \mathbb{R}^n$ is named as the **remainder function**.

This tripartite set of compositional functions, *i.e.,* data expansion, parameter reconciliation, and remainder functions, serves as the foundation for the RPN model. By strategically selecting and combining these component functions, we will be able to construct a RPN model to address a wide spectrum of learning challenges across diverse function learning tasks. To enhance RPN's modeling capabilities, in the previous RPN paper [89], we introduced both a wide architecture featuring multi-heads and multi-channels (within each layer), and a deep architecture comprising multiple layers. Furthermore, we equipped RPN with a more adaptable and lightweight mechanism for constructing models with comparable capabilities through nested and extended data expansion functions.

Furthermore, as outlined earlier, the RPN model was based on the assumption that data instances in training batches are independent and identically distributed (*i.i.d.*). It also assumed that the attributes within each instance were independent, treating them separately within its expansion functions. These restrictive assumptions significantly limit RPN's effectiveness in real-world learning tasks involving complex, interdependent data, such as language, images, time series, and graphs. In the following section, we will delve into the concept of data interdependence, which is explicitly modeled in the newly redesigned RPN 2 model, building upon the foundations of RPN.

3 Data Interdependence

This section explores the concept of **data interdependence** in function learning tasks. In practice, interdependence relationships within a data batch can be classified into various granularities, such as *attribute interdependence* and *instance interdependence*, among others. By drawing on real-world data, we will illustrate concrete examples of these relationships. Furthermore, we will explore various metrics and methods for quantifying and modeling interdependence, which will serve as the foundation for developing the robust and effective RPN 2 model to address function learning tasks in complex and interdependent datasets.

3.1 What is Data Interdependence?

Conceptually, the "*Principle of Universal Connection and Development*" discussed in DIALECTICAL MATERIALISM asserts that "*Nothing in the world stands by itself. Every object is a link in an endless chain and is thus connected with all the other links.*" Data collected from the real-world should inherently reflect such universal and extensive connections. Technically, understanding the data interdependence is also critical for conducting accurate analyses, developing robust models, and making correct decisions in the construction of intelligent function learning systems.

DEFINITION 2 ***(Data Interdependence)**: Formally, data interdependence refers to the relationships and interactions between different individual attributes or the entire data instances within a system. In this context, the state or behavior of one element (either an attribute or an instance) can influence or be influenced by others, creating a network of interdependent relationships.*

Instance Interdependence vs. Attribute Interdependence: Data interdependence may manifest in input data at various granularities, such as attribute interdependence among the attributes within each data instance and instance interdependence among the instances within the data batch. From the perspective of function learning tasks, the distinction between attribute and instance interdependence often becomes blurred, as certain data elements can be regarded as either attributes or instances—or even both. For example, image frames may be considered as individual instances in image classification tasks but can serve as attributes in video content understanding tasks. Similar ambiguities arise in various modalities, such as point clouds, languages, graphs, and time series data.

From a technical implementation perspective, the differences between attribute and instance interdependence are primarily a matter of measuring interdependence across different dimensions of the input data batch (*e.g.,* rows for instance interdependence and columns for attribute interdependence). By leveraging data batch transposition and reshaping techniques, a unified implementation can effectively calculate interdependence across various dimensions of the data batch, enabling consistent modeling of both attribute and instance interdependence.

To maintain generality, we will illustrate interdependence using examples of general vectors sampled from a vector space below and explore methods to measure these dependencies. In specific applications, these vector variables can represent either attributes or instances within their respective vector spaces, offering a versatile approach to interdependence analysis.

3.2 Data Interdependence Quantitative Measurements

Data interdependence among vector variables representing attribute or instance vectors (*i.e.,* the columns and rows of the input data batch) can be quantified using various methods, including statistical and numerical approaches. Statistical interdependence measurements, grounded in probability theory and statistics, explicitly model and quantify uncertainty in the interdependence calculation. Conversely, numerical interdependence measurements, based on linear algebra and computational theory, often assume the data interdependence relationships to be deterministic instead. In addition to statistical and numerical approaches, many other quantitative methods also exist, such as topological and geometric measurements, which assess data interdependence relationships from the perspective of topological and geometric structures. These distinct approaches give rise to different categories of quantitative metrics for defining data interdependence functions, all of which will be introduced in detail in the following Section 5.

3.2.1 Statistical Data Interdependence Metrics

Formally, given variables $X_1, X_2, \cdots, X_k$ representing vectors $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k \in \mathbb{R}^d$ from a d-dimensional vector space (each representing either an attribute or an instance), statistical interdependence measurements assume each follows certain distributions, such as the multivariate Gaussian distribution:

$$X_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \mathbf{\Sigma}_i), \forall i \in \{1, 2, \cdots, k\}, \tag{5}$$

where the notation $\mathcal{N}(\boldsymbol{\mu}_i, \mathbf{\Sigma}_i)$ denotes a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}_i \in \mathbb{R}^d$ and variance matrix $\mathbf{\Sigma}_i \in \mathbb{R}^{d \times d}$.

The joint distribution of these k variables also follows the multivariate Gaussian distribution, *i.e.,*

$$\begin{bmatrix} X_1 \\ X_2 \\ \cdots \\ X_k \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \\ \cdots \\ \boldsymbol{\mu}_k \end{bmatrix}, \begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{\Sigma}_{1,2} & \cdots & \mathbf{\Sigma}_{1,k} \\ \mathbf{\Sigma}_{2,1} & \mathbf{\Sigma}_2 & \cdots & \mathbf{\Sigma}_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{\Sigma}_{k,1} & \mathbf{\Sigma}_{k,2} & \cdots & \mathbf{\Sigma}_k \end{bmatrix} \right), \tag{6}$$

where $\mathbf{\Sigma}_{i,j} = Cov(X_i, X_j)$ denotes the covariance matrix of variables X_i and X_j (for $i, j \in \{1, 2, \cdots, k\}$ and $i \neq j$), and $\mathbf{\Sigma}_{j,i} = Cov(X_j, X_i) = \mathbf{\Sigma}_{i,j}^{\top}$. The diagonal variance matrix $\mathbf{\Sigma}_i$ of variable X_i (for $\forall i \in \{1, 2, \cdots, k\}$) can also be calculated in a similar way as $\mathbf{\Sigma}_i = Cov(X_i, X_i)$, which is symmetric by default.

Statistically, two variables X_i and X_j are independent if and only if their corresponding covariance matrix is zero, i.e., $\mathbf{\Sigma}_{i,j} = \mathbf{0}$ (or equivalently $\mathbf{\Sigma}_{j,i} = \mathbf{0}$). If variables $X_1, X_2, \ldots, X_k$ are statistically jointly independent, all the off-diagonal block matrices of the joint covariance matrix in Equation (6) will be zeros, rendering the joint covariance matrix to be diagonal.

Based on these above descriptions, various statistical metrics can measure interdependence among the vector variables, such as correlation coefficients and mutual information metrics.

RV Coefficient based Interdependence Metric: In statistics, the RV coefficient, a multivariate generalization of the squared Pearson correlation coefficient, measures the *linear* dependence between two variables. It ranges from 0 to 1, with 1 indicating perfect linear dependence (or similarity) and 0 indicating no linear dependence. The RV coefficient for pairs of variables can be directly calculated based on their covariance matrices, as introduced above.

Given variables X_i and X_j, as well as their variance and covariance matrices $\mathbf{\Sigma}_i$, $\mathbf{\Sigma}_j$, $\mathbf{\Sigma}_{i,j}$ and $\mathbf{\Sigma}_{j,i}$, their RV-coefficient is defined as:

$$RV(X_i, X_j) = \frac{tr\left(\mathbf{\Sigma}_{i,j}\mathbf{\Sigma}_{j,i}\right)}{\sqrt{tr\left(\mathbf{\Sigma}_i^2\right) tr\left(\mathbf{\Sigma}_j^2\right)}} \in \mathbb{R}, \tag{7}$$

where the notation $tr(\cdot)$ denotes the *trace* of the input matrix.

Mutual Information (Gaussian) based Interdependence Metric: In addition to the RV coefficient, mutual information measures the amount of information one random variable contains about another, defining the *non-linear* interdependence relationships of variables.

For random variables X_i and X_j following a multivariate Gaussian distribution, based on their variance and covariance matrices, their mutual information metric is calculated as:

$$MI(X_i, X_j) = \frac{1}{2} \log\left(\frac{det\left(\mathbf{\Sigma}_i\right) det\left(\mathbf{\Sigma}_j\right)}{det\left(\mathbf{\Sigma}\right)} \right) \in \mathbb{R}, \tag{8}$$

where $\mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_i & \mathbf{\Sigma}_{i,j} \\ \mathbf{\Sigma}_{j,i} & \mathbf{\Sigma}_j \end{bmatrix}$ is the covariance matrix of the joint variables $\begin{bmatrix} X_i \\ X_j \end{bmatrix}$ and $det(\cdot)$ denotes the *determinant* of the input matrix.

3.2.2 Numerical Data Interdependence Metrics

Unlike statistical metrics, numerical metrics measure deterministic interdependence relationships among variables without prior assumptions about their distributions. For variables $X_1, X_2, \ldots, X_k$ representing vectors sampled from a vector space (either attribute or instance vector spaces), numerical methods can quantify interdependence using vector similarity or distance metrics. The simplest form of numerical interdependence in vector space is linear interdependence.

Assuming these vector variables take values $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k \in \mathbb{R}^d$, respectively, they are linearly interdependent if there exist scalar coefficients $\alpha_1, \alpha_2, \ldots, \alpha_k \in \mathbb{R}$, not all zero, such that:

$$\alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_k \mathbf{x}_k = \mathbf{0}, \tag{9}$$

where $\mathbf{0}$ denotes the zero vector of the corresponding vector space. These coefficients $\alpha_1, \alpha_2, \ldots, \alpha_k$ can be calculated using the Gaussian elimination method, which illustrates the linear interdependence relationships among the vectors. Gaussian elimination, also known as *row reduction*, is a method for solving systems of linear equations. It involves a sequence of row-wise reduction operations performed on the corresponding matrix of coefficients.

To demonstrate how Gaussian elimination reveals interdependence among vectors, consider an example with $\mathbf{x}_1 = [2, -1, -1]^\top$, $\mathbf{x}_2 = [3, -4, -2]^\top$ and $\mathbf{x}_3 = [5, -10, -8]^\top$. Equation (9) defined on these three vectors can be rewritten as:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \alpha_1 \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix} + \alpha_2 \begin{bmatrix} 3 \\ -4 \\ -2 \end{bmatrix} + \alpha_3 \begin{bmatrix} 5 \\ -10 \\ -8 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 5 \\ -1 & -4 & -10 \\ -1 & -2 & -8 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}. \tag{10}$$

This homogeneous system of linear equations can be transformed into its row-reduced form using Gaussian elimination as follows:

$$\left[\begin{array}{ccc|c} 2 & 3 & 5 & 0 \\ -1 & -4 & -10 & 0 \\ -1 & -2 & -8 & 0 \end{array}\right] \xrightarrow{\text{row-reduction}} \left[\begin{array}{ccc|c} 2 & 3 & 5 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]. \tag{11}$$

From this reduction, we observe that: (1) the first and second columns have pivots, indicating that vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are linearly independent, and (2) the third column has no pivot, indicating that vector $\mathbf{x}_3$ can be linearly represented by $\mathbf{x}_1$ and $\mathbf{x}_2$. Furthermore, we can deduce that $2\alpha_1 + 3\alpha_2 + 5\alpha_3 = 0$ and $\alpha_2 + 3\alpha_3 = 0$, implying $\alpha_2 = -3\alpha_3$ and $\alpha_1 = 2\alpha_3$. Thus:

$$\mathbf{0} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \alpha_3 \mathbf{x}_3 = 2\alpha_3 \mathbf{x}_1 - 3\alpha_3 \mathbf{x}_2 + \alpha_3 \mathbf{x}_3, \tag{12}$$

or

$$\mathbf{x}_3 = -2\mathbf{x}_1 + 3\mathbf{x}_2. \tag{13}$$

However, the row-reduced matrix indicates relationships between coefficients and cannot directly define interdependence relationships of data vectors. The row-reduction operation typically has a time complexity of $\mathcal{O}(d^3)$, which can be computationally expensive for high-dimensional data vectors. Moreover, such linear interdependence among instances or attributes is rare in real-world data batches. Instead of directly using linear relationship analysis, we propose to compute interdependence metrics based on other operators, such as *inner product* and *bilinear form*.

Inner Product based Interdependence Metric: The inner product, a generalization of the dot product, multiplies vectors to produce a scalar. It quantifies the relationship between two vectors (or variables) by calculating the angle between them in the vector space, capturing both vector magnitudes and relative orientation. Formally, for two variables $X_i = \mathbf{x}_i$ and $X_j = \mathbf{x}_j$ taking vectors of the same dimension, their inner product is calculated as:

$$I(X_i, X_j) = \mathbf{x}_i \mathbf{x}_j^\top \in \mathbb{R}. \tag{14}$$

Bilinear Form based Interdependence Metric: A bilinear form is a function linear in both arguments that maps two vectors to a scalar. It measures interdependence by quantifying vector interactions or correlations, with properties like symmetry and orthogonality providing additional insights. The inner product metric can also be viewed as a special case of bilinear forms.

Formally, for variables X_i and X_j taking value vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, their bilinear form-based interdependence metric can be calculated as follows:

$$B(X_i, X_j) = \mathbf{x}_i \mathbf{W} \mathbf{x}_j^\top \in \mathbb{R}. \tag{15}$$

The square matrix $\mathbf{W} \in \mathbb{R}^{d \times d}$ is normally a constant, with elements defined by $\mathbf{W}(p, q) = B(\mathbf{e}_p, \mathbf{e}_q)$, is called the *matrix of the bilinear form* on the d basis vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d \in \mathbb{R}^d$ of the vector space. Additionally, there exist other ways of defining the matrix. Specifically, when matrix $\mathbf{W}$ is the identity matrix, this bilinear form reduces to the inner product. In practice, matrix $\mathbf{W}$ can also be defined as a learnable parameter, and its low-rank representation helps define the interdependence matrix for the Transformer model, which will be introduced later in Section 5.1.6.

3.3 Data Interdependence Examples

In this section, we present examples of real-world data to illustrate the interdependence present in data collected from various sources. These examples span different modalities, including images, language, graphs, and time series. As previously discussed, interdependence relationships can exist at both instance and attribute granularities, with the categorization largely dependent on the data representation format and specific function learning tasks. It is important to note that for the examples discussed below, we present just one potential approach to defining data interdependence relationships. Depending on the specific problem and learning settings, other valid methods for defining interdependence relationships within the data batch may exist. Readers are encouraged to select the most appropriate approach for modeling such dependence relationships in their particular contexts.

3.3.1 Image Data Interdependence

Images can be viewed as a sequence of pixels organized into a square or rectangular shape with specific height and width. In the right plot, we illustrate a colored image of a hummingbird in a square shape with 20 × 20 pixels, *i.e.,* both the image height and width are 20. For colored images, depending on the encoding method used (such as RGB, YCbCr, and CMYK), each pixel is represented by multiple integers. We show the RGB codes of three randomly picked pixels on the left-hand side of the plot. As demonstrated, it is difficult to interpret the physical meanings of these individual pixel RGB values or their potential contribution to identifying objects in the image.

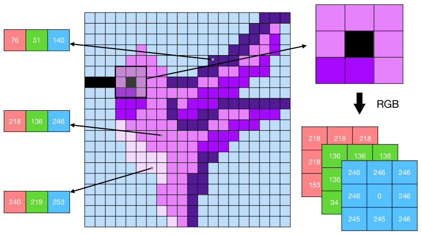


Figure 2: Interdependence in Images.

The significance of individual image pixels in addressing the target learning task is profoundly influenced by their surrounding context, which are normally their nearby pixels. The collective variation patterns of these adjacent pixels provide crucial information about the objects present within the images. On the right-hand side of the illustration, we present a 3 × 3 pixel segment along with its corresponding RGB color codes. This small-scale representation, when juxtaposed with individual pixels shown on the left, offers a more nuanced perspective. The significant fluctuations in the central pixel's values compared to its surrounding pixels within this segment indicate its position at the boundary, conveying substantially more information than isolated pixels.

3.3.2 Language Data Interdependence

Language data, as an important carrier of information, typically appears in an ordered sequence structure, which may include natural language, programming language, and mathematical language. People read and write language data sequentially, which can convey rich semantic information. In the right plot, we illustrate an example sentence "This section is long and boring.", and provide its dependency parsing tree. In natural language processing, sentences

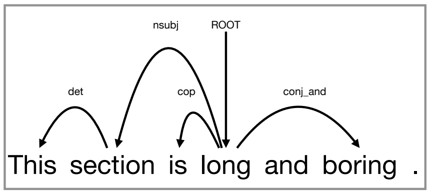


Figure 3: Interdependence in Language.

can typically be decomposed into a sequential list of tokens via a tokenizer based on a pre-defined vocabulary set. For this discussion, we will consider tokens as the smallest units to introduce the interdependence of language data.

In language data, extensive dependence relationships exist among tokens within the same sentence or paragraph (and even across documents). For instance, “this section” in the above example determines the use of “is” rather than “are”, and the word “is” constrains the following words to be adjectives, *e.g.,* “long”. Furthermore, the conjunction “and” indicates that the two adjectives should have close semantic meanings, *i.e.,* “long” and “boring”. The semantic meaning of each word depends on its sentence context, which is a crucial factor that should be incorporated into model design and learning.

3.3.3 Time-Series Data Interdependence

Time series data, such as daily temperature readings, stock market prices, and annual GDP growth, provide another representative example of interdependence in data. In the right plot, we show the stock price curves of six biotechnology companies - AMGN, BIIB, CERN, ILMN, LULU, and MNST - over a four-year period. In time series data, data points at later timestamps typically depend on those at previous timestamps. Moreover, for some time series data exhibiting long-term periodic patterns, this interdependence may span a much longer time period, *e.g.,* a month or even several years.

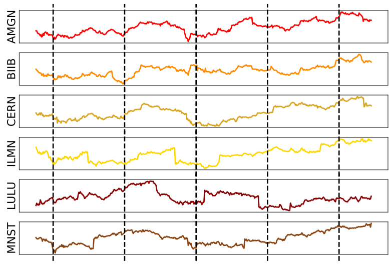


Figure 4: Interdependency in Time-Series.

For the stock price data illustrated in the right plot, these selected stocks belong to the same sector, and the price of one stock may also depend on other correlated stocks. In such stock price time series data, individual data points alone can hardly reveal any information about the underlying price changing patterns. To extract useful features and signals, it may be necessary to include other data points spanning both temporal and sector dimensions.

3.3.4 Graph Data Interdependence

In addition to images, language and time-series data, graphs are another representative example of data structures with extensive dependence relationships among nodes. Graphs can be represented as a set of nodes connected by links. In the right plot, we illustrate an example of the Myricetin molecule (*i.e.,* $C_{15}H_{10}O_8$), a member of the flavonoid class of polyphenolic compounds with antioxidant properties. By treating atoms as nodes and atomic bonds as links, the Myricetin molecule can be represented as a molecular graph structure. Single and double bonds can be represented as different types of links in the graph, rendering it heterogeneous. (Note: The distinction between homogeneous and heterogeneous graphs is slightly out of the scope of this paper and will not be discussed further.)

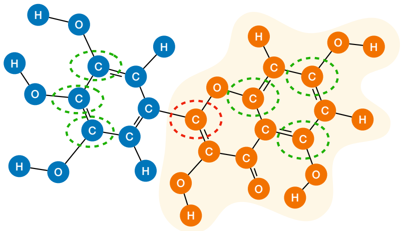


Figure 5: Interdependence in Graph.

In the molecular graph, it is difficult to infer the roles or functions of individual nodes, such as the central carbon atom in the red dashed circle, based solely on the node itself. We must consider its surrounding nodes on which it depends. Unlike images, nodes’ dependence relationships in graphs can span across the entire structure to distant nodes, and local neighbors may not provide sufficient information for inferring their functions in the molecule. For instance, in the plot, we highlight several other carbon nodes with identical surrounding neighbors connected by the same types of links. To infer the functions of the central carbon node, we may also need to consider the functional groups it is involved in, *e.g.,* the one highlighted in the orange background color in the plot.

3.4 Data Interdependence Handling

The diverse data interdependence relationships illustrated in the above examples play a critical role in the function learning tasks studied in this paper. In this section, we introduce two different approaches for handling such data interdependence relationships: *interdependence padding* and *interdependence aggregation*. Furthermore, we demonstrate that these two approaches can be unified under a shared representation, expressed as the multiplication of the data batch with an *interdependence matrix* defined based on the input data.

3.4.1 Interdependence Padding

Formally, given two variables Z and Y representing the input and output of a data instance in a function learning task, and a set of attribute variables $A_1, A_2, \ldots, A_d$ that Z depends on, we can represent the dependence relationships among these variables with the top plot shown on the right, with notations borrowed from Bayesian networks. The notation $Z \rightarrow Y$ denotes the direct dependence relationship of variable Y on variable Z; and $(A_1, A_2, \ldots, A_d) \rightarrow Z$ denotes the direct dependence of Z on variables $A_1, A_2, \ldots, A_d$.

The *interdependence padding* approach proposes to introduce one extra new variable Z' to model the information from variables $A_1, A_2, \ldots, A_d$ that Z depends on. The new variable Z' acts as an intermediate bridge between $A_1, A_2, \ldots, A_d$ and Z. Also, according to the Bayesian network, the newly created variable Z' renders Z and $A_1, A_2, \ldots, A_d$ to be conditionally independent given Z', *i.e.,* $Z \perp\!\!\!\perp (A_1, A_2, \ldots, A_d) | Z'$.

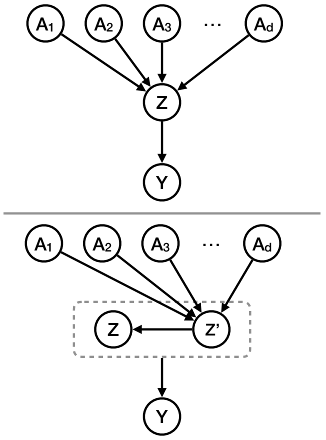


Figure 6: An Illustration of Variable Dependence Padding.

There may exist different ways to define the new variable Z'. In this paper, we will define Z' as a concatenation of the dependent variables $A_1, A_2, \ldots, A_d$, *i.e.,* $Z' = [A_1, A_2, \ldots, A_d]$. This new variable Z' and the input variable Z together will define the interdependence padding operator as

$$\text{padding}(Z | A_1, A_2, \cdots, A_d) = [Z, Z']. \tag{16}$$

For a single variable, the above interdependence padding-based approach may work well, as it includes all the dependent information into the padded new variable, which will be used for inferring the desired variable Y. However, in practice, there may exist multiple variables, such as $Z_1, Z_2, \ldots, Z_k$, which may all depend on $A_1, A_2, \ldots, A_d$, as illustrated in the right plot. To make the variables $Z_1, Z_2, \ldots, Z_k$ conditionally independent from those in $A_1, A_2, \ldots, A_d$, redundant interdependence padding can be applied to all the variables $Z_1, Z_2, \ldots, Z_k$ as follows:

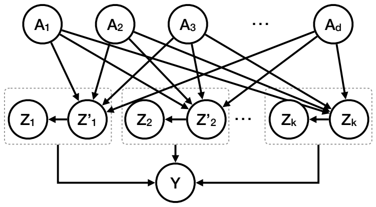


Figure 7: Redundancy in the Variable Interdependence Paddings.

$$\begin{cases} \text{padding}(Z_1 | A_1, A_2, \cdots, A_d) = [Z_1, Z'_1], \\ \text{padding}(Z_2 | A_1, A_2, \cdots, A_d) = [Z_2, Z'_2], \\ \cdots \\ \text{padding}(Z_k | A_1, A_2, \cdots, A_d) = [Z_k, Z'_k], \end{cases} \tag{17}$$

where the newly created padding variables $Z'_1, Z'_2, \cdots, Z'_k$ are all concatenations of the dependent variables $A_1, A_2, \cdots, A_d$. In other words, multiple duplicated copies of data vectors represented by variables $A_1, A_2, \cdots, A_d$ will be concatenated to those of $Z_1, Z_2, \cdots, Z_k$, which is actually how the existing models, like convolutional neural networks, handle the data interdependence.

Discussions: By default, we will only use the above interdependence padding for handling attribute interdependence relationships. Such duplicated and interdependence padding should not be a significant problem when the size or dimension of data that $A_1, A_2, \cdots, A_d$ represent is small. For example, if each variable represents a feature (such as a single pixel in images), duplicating these redundant features—similar to the approach in convolutional neural networks—does not pose significant learning challenges for small-sized input data. Additionally, the parameters used to handle these features can be shared across the newly created padding variables, thereby reducing the learning cost in terms of the number of parameters.

However, in real-world practice, such variables may also represent vectors of feature segments or data instances with high dimensions. Applying the above redundant interdependence padding-based approach will not only introduce much higher storage consumption but also require more learnable parameters to handle the longer variable list after padding. To address the problem, below, we will introduce an alternative data interdependence modeling method via the aggregation operator instead.

3.4.2 Interdependence Aggregation

Besides interdependence padding, another alternative approach to modeling such data interdependence relationships is via the interdependence aggregation operator, which incorporates information from all the variables via aggregation operators, such as weighted summation.

Formally, given the variable Z and other variables $A_1, A_2, \dots, A_d$ that it depends on, as illustrated by the upper plot of Figure 6, the *interdependence aggregation* approach proposes to integrate those variables as follows:

$$\text{aggregation}(Z|A_1, A_2, \cdots, A_d) = \alpha_0 \cdot Z + \alpha_1 \cdot A_1 + \alpha_2 \cdot A_2 + \cdots + \alpha_d \cdot A_d, \tag{18}$$

where the scalar weights $\alpha_1, \dots, \alpha_d \in \mathbb{R}$ denote the interdependence strength of Z on the corresponding variables in $A_1, A_2, \dots, A_d$. Specifically, α_0 denotes the interdependence of Z on itself, which can also be referred to as self-dependence.

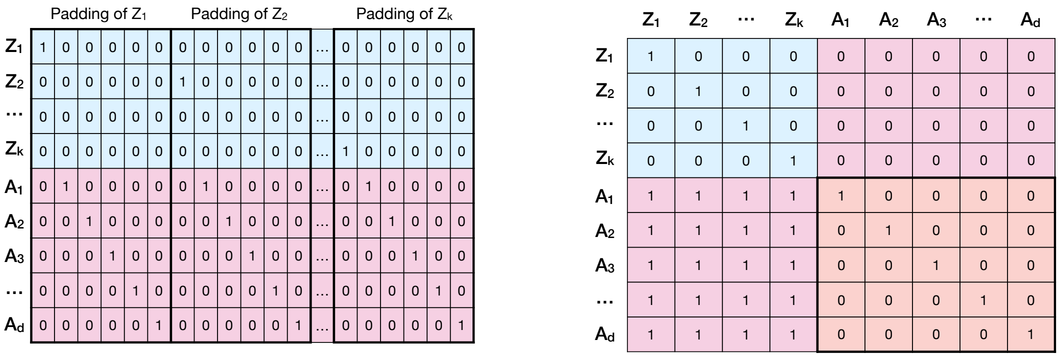
When it comes to the multi-variate cases as shown in Figure 7, similar operators can also be applied for the other variables $Z_1, Z_2, \dots, Z_k$ which also depend on $A_1, A_2, \dots, A_d$ as follows:

$$\begin{cases} \text{aggregation}(Z_1|A_1, A_2, \cdots, A_d) = \alpha_0^1 \cdot Z_1 + \alpha_1^1 \cdot A_1 + \alpha_2^1 \cdot A_2 + \cdots + \alpha_d^1 \cdot A_d, \\ \text{aggregation}(Z_2|A_1, A_2, \cdots, A_d) = \alpha_0^2 \cdot Z_2 + \alpha_1^2 \cdot A_1 + \alpha_2^2 \cdot A_2 + \cdots + \alpha_d^2 \cdot A_d, \\ \cdots \\ \text{aggregation}(Z_k|A_1, A_2, \cdots, A_d) = \alpha_0^k \cdot Z_k + \alpha_1^k \cdot A_1 + \alpha_2^k \cdot A_2 + \cdots + \alpha_d^k \cdot A_d. \end{cases} \tag{19}$$

There exist different ways to define the above scalars $\alpha_0, \alpha_1, \dots, \alpha_d$ (with different superscripts). The simplest way is to assign them equal constants of 1 (or $\frac{1}{d+1}$), which renders the above aggregation operator to a summation (or averaging) operator. To further distinguish and model the different roles of variables $A_1, A_2, \dots, A_d$ on different variables $Z_1, Z_2, \dots, Z_k$, we will introduce several different approaches to define the dependence strength weight parameters.

Discussions: Furthermore, besides weighted aggregation, some extra transformation operators (such as linear transformation, expansion, compression, and other more complex ones with learnable parameters) can also be applied to the variables prior to or after the aggregation. This provides the interdependence aggregation with greater modeling capacities for complex function learning problems. Some of these will be briefly discussed in the following part in this section, and more will be introduced in detail in Section 5.

Compared with the above interdependence padding method, the interdependence aggregation will consume less computational space and time, and provide greater learning capacities for modeling the interdependent data. However, there is no free lunch; interdependence aggregation also creates extra parameters (or hyper-parameters) which may need to be learned (or manually defined). Below,



$$\begin{array}{c|cccccc|cccccc|c|cccccc} & \multicolumn{6}{c|}{\text{Padding of } Z_1} & \multicolumn{6}{c|}{\text{Padding of } Z_2} & & \multicolumn{6}{c}{\text{Padding of } Z_k} \\ \hline Z_1 & 1&0&0&0&0&0 & 0&0&0&0&0&0 & \cdots & 0&0&0&0&0&0 \\ Z_2 & 0&0&0&0&0&0 & 1&0&0&0&0&0 & \cdots & 0&0&0&0&0&0 \\ \cdots & 0&0&0&0&0&0 & 0&0&0&0&0&0 & \cdots & 0&0&0&0&0&0 \\ Z_k & 0&0&0&0&0&0 & 0&0&0&0&0&0 & \cdots & 1&0&0&0&0&0 \\ A_1 & 0&1&0&0&0&0 & 0&1&0&0&0&0 & \cdots & 0&1&0&0&0&0 \\ A_2 & 0&0&1&0&0&0 & 0&0&1&0&0&0 & \cdots & 0&0&1&0&0&0 \\ A_3 & 0&0&0&1&0&0 & 0&0&0&1&0&0 & \cdots & 0&0&0&1&0&0 \\ \cdots & 0&0&0&0&1&0 & 0&0&0&0&1&0 & \cdots & 0&0&0&0&1&0 \\ A_d & 0&0&0&0&0&1 & 0&0&0&0&0&1 & \cdots & 0&0&0&0&0&1 \end{array}$$

(a) Matrix $\mathbf{A}_a$ for Interdependence Padding.

$$\begin{array}{c|cccc|ccccc} & Z_1 & Z_2 & \cdots & Z_k & A_1 & A_2 & A_3 & \cdots & A_d \\ \hline Z_1 & 1&0&0&0 & 0&0&0&0&0 \\ Z_2 & 0&1&0&0 & 0&0&0&0&0 \\ \cdots & 0&0&1&0 & 0&0&0&0&0 \\ Z_k & 0&0&0&1 & 0&0&0&0&0 \\ \hline A_1 & 1&1&1&1 & 1&0&0&0&0 \\ A_2 & 1&1&1&1 & 0&1&0&0&0 \\ A_3 & 1&1&1&1 & 0&0&1&0&0 \\ \cdots & 1&1&1&1 & 0&0&0&1&0 \\ A_d & 1&1&1&1 & 0&0&0&0&1 \end{array}$$

(b) Matrix $\mathbf{A}'_a$ for Interdependence Aggregation.

Figure 8: An Illustration of Interdependence Matrix for Interdependence Padding and Interdependence Aggregation.

we will illustrate that these two different interdependence modeling approaches can be unified with the interdependence matrix.

3.4.3 Data Interdependence Matrix

Based on the above discussion, we introduce the concept of the *interdependence matrix* in this paper, which can model both the attribute and instance interdependence relationships of the input data batch. Moreover, as mentioned above, the interdependence matrix can also unify the previously discussed interdependence padding and interdependence aggregation-based modeling approaches into one shared representation, which will be discussed as follows.

Attribute Interdependence Matrix: Formally, given the variables $Z_1, Z_2, \ldots, Z_k$ shown in Figure 7, which depend on the attribute variables $A_1, A_2, \ldots, A_d$, we can group these variables and represent them as the data instance variable $X = [Z_1, Z_2, \ldots, Z_k, A_1, A_2, \ldots, A_d]$. To simplify the descriptions, we introduce a data instance vector $\mathbf{x} \in \mathbb{R}^m$, where the dimension $m = k + d$ and the vector elements denote the values of variables $Z_1, Z_2, \ldots, Z_k$ and $A_1, A_2, \ldots, A_d$, respectively.

Based on the above notations, we can rewrite Equation (17) for interdependence padding by multiplying the input data instance vector $\mathbf{x} \in \mathbb{R}^m$ with the attribute interdependence matrix $\mathbf{A}_a \in \mathbb{R}^{m \times (k \times (d+1))}$ shown in Figure 8a as follows:

$$\mathbf{x} = \mathbf{x}\mathbf{A}_a \in \mathbb{R}^{k \times (d+1)}. \tag{20}$$

The output vector is composed of the values corresponding to the variables $Z_1, Z_2, \ldots, Z_k$ with the paddings sequentially attached to them, whose length will be $k \times (d + 1)$.

For the interdependence aggregation approach, with the interdependence matrix $\mathbf{A}'_a \in \mathbb{R}^{m \times m}$ shown in Figure 8b, we can also update the involved attributes data instance vector $\mathbf{x}$ by aggregating all the dependent conditions as follows:

$$\mathbf{x} = \mathbf{x}\mathbf{A}'_a \in \mathbb{R}^{m}. \tag{21}$$

The output vector has a length of m. For the entries corresponding to variables $Z_1, Z_2, \ldots, Z_k$ in the vector, they will be summed with their conditions; while for the entries corresponding to variables $A_1, A_2, \ldots, A_d$, they will remain unchanged in the output vector.

The above matrix $\mathbf{A}_a \in \mathbb{R}^{m \times m'}$ (and $\mathbf{A}'_a$) describes the interdependence relationships among the attributes, which is named the attribute interdependence matrix. Depending on the choice of the interdependence matrix, the dimension term m' denoting the output dimension of the data instance can take different values, which will be specified in the matrix definition.

Instance Interdependence Matrix: Similarly, for describing the data instance interdependence relationships within the data batch, we can also define the instance interdependence matrix $\mathbf{A}_i \in \mathbb{R}^{b' \times b}$, where b denotes the batch size and b' is the output batch size after considering the instance interdependence relationships. For most cases, the term b' will be equal to b, but we also provide flexibility and allow b' to be a parameter determined by the matrix definition. Unlike $\mathbf{A}_a$ for attribute interdependence modeling (*i.e.,* the columns of data batch), the matrix $\mathbf{A}_i$ will operate on the data instances instead (*i.e.,* the rows).

Formally, given the input data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$ involving b data instances, based on the data interdependence matrix $\mathbf{A}_i \in \mathbb{R}^{b' \times b}$, we can represent the updated data batch that incorporates the data instance interdependence as follows:

$$\mathbf{X} = \mathbf{A}_i \mathbf{X} \in \mathbb{R}^{b' \times m}. \tag{22}$$

To consider both the attribute interdependence and instance interdependence relationships, we can transform the data batch $\mathbf{X}$ with both matrices $\mathbf{A}_i \in \mathbb{R}^{b' \times b}$ and $\mathbf{A}_a \in \mathbb{R}^{m \times m'}$ as follows:

$$\mathbf{X} = \mathbf{A}_i \mathbf{X} \mathbf{A}_a \in \mathbb{R}^{b' \times m'}. \tag{23}$$

How to Define Data Interdependence Matrix: In this section, we have demonstrated the use of data interdependence matrices for modeling attribute and instance dependence relationships within input data batches. The examples presented in Figure 8 and the quantitative metrics discussed in Section 3.2 have illustrated various approaches to defining these interdependence matrices, which are applicable to a wide range of function learning tasks.

Meanwhile, it is important to note that these examples and metrics represent only specific instances of interdependence matrices. To enhance model flexibility and learning capacity in real-world applications, instead of manually pre-defining interdependence matrices, we introduce a new family of component functions called *data interdependence functions* to define and learn such matrices instead. These functions can automatically compute and fine-tune data interdependence matrices based on the input data batch, additional contextual information, and optional learnable parameters.

Furthermore, the data interdependence functions in our newly developed **TINYBIG V0.2.0** are designed to compute both attribute and instance interdependence matrices. These computations are guided by specified target dimension hyper-parameters (*i.e.,* "attribute" or "instance"), allowing a single implementation to calculate interdependence matrices along any dimension of the data batch. The following sections will delve into these data interdependence functions and their integration into the redesigned RPN 2 model.

4 RPN 2: Enhanced RPN with Data Interdependence Functions

Building upon the previously discussed background knowledge of function learning tasks and the RPN model, as well as the elucidated concepts of data interdependence, this section introduces the new RPN 2 (Reconciled Polynomial Network, version 2) model.

This enhanced RPN 2 model incorporates data interdependence modeling capabilities through a suite of innovative component functions, collectively termed **data interdependence functions**. These incorporated functions enable RPN 2 to effectively address function learning tasks on diverse multi-modal data characterized by complex interdependence relationships.

4.1 RPN 2: Reconciled Polynomial Network 2

Formally, given the underlying data distribution function $f : \mathbb{R}^m \to \mathbb{R}^n$ to be inferred, we can represent the new RPN 2 model with data interdependence modeling capabilities to approximate the underlying function f as follows:

$$g(\mathbf{x}|\mathbf{w}) = \langle \kappa_{\xi}(\mathbf{x}), \psi(\mathbf{w}) \rangle + \pi(\mathbf{x}), \tag{24}$$

where

- $\kappa_{\xi} : \mathbb{R}^m \to \mathbb{R}^D$ is named as the **data interdependent transformation function**. It is a composite function of the *data transformation function* κ and the *data interdependence function* ξ. Notation D denotes the target space dimension.
- $\psi : \mathbb{R}^l \to \mathbb{R}^{n \times D}$ is named as the **parameter reconciliation function** defined on the parameters only. Notation l denotes the learnable parameter vector space dimension.
- $\pi : \mathbb{R}^m \to \mathbb{R}^n$ is named as the **remainder function**.

Moreover, the data transformation function serves as a general term, encompassing both data expansion functions where $D \geq m$, and data compression functions where $D \leq m$ (which will be introduced in the following Section 6.1). The data interdependence functions can model the relationships among both attributes and instances of the input data batch, whose detailed representation will be provided in the following subsection. The parameter reconciliation function fabricates a low-dimensional parameter vector of length l to a high-dimensional parameter matrix of dimensions $n \times D$, where $l \ll n \times D$. It may also be referred to by its general name, the parameter fabrication function; these terms will be used interchangeably throughout this paper. The remainder function potentially provides complementary information and helps reduce the approximation errors. Figure 9 illustrates the RPN 2 model architecture, with its modules and components to be introduced in the following section.

4.2 Data Interdependent Transformation Function

The data interdependence transformation function κ_{ξ}, as introduced above, is a composition of the data transformation function κ and the data interdependence functions ξ. As proposed in the previous RPN paper [89], the data transformation function κ efficiently projects input data into an intermediate vector space characterized by novel basis vectors, which is subsequently mapped to the output space through inner products with reconciled parameters. Meanwhile, the data interdependence functions ξ, newly introduced in this paper, capture the intricate interdependence relationships among data instances and attributes. These functions will extract nuanced information from the input data batch, operating both prior to and following the data projection facilitated by function κ.

DEFINITION 3 ***(Data Interdependence Function)****: Formally, given an input data batch* $\mathbf{X} \in \mathbb{R}^{b \times m}$ *(with* b *instances and each instance with* m *attributes), the attribute and instance data interdependence functions are defined as:*

$$\xi_a : \mathbb{R}^{b \times m} \to \mathbb{R}^{m \times m'}, \textit{ and } \xi_i : \mathbb{R}^{b \times m} \to \mathbb{R}^{b \times b'}, \tag{25}$$

where m' *and* b' *denote the output dimensions of their respective interdependence functions, respectively.*

To elucidate the mechanisms of attribute and instance interdependence functions in defining the data interdependence transformation function κ_{ξ}, we shall consider a multi-instance input data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$ as an exemplar. Here, b and m denote the number of instances and attributes, respectively. Given this input data batch $\mathbf{X}$, as shown in the right plot of Figure 9, we can formulate the data interdependence transformation function κ_{ξ} as follows:

$$\kappa_{\xi}(\mathbf{X}) = \mathbf{A}_{\xi_i}^{\top} \kappa(\mathbf{X}\mathbf{A}_{\xi_a}) \in \mathbb{R}^{b' \times D}. \tag{26}$$

These attribute and instance interdependence matrices $\mathbf{A}_{\xi_a} \in \mathbb{R}^{m \times m'}$ and $\mathbf{A}_{\xi_i} \in \mathbb{R}^{b \times b'}$ are computed with the corresponding interdependence functions defined above, *i.e.,*

$$\mathbf{A}_{\xi_a} = \xi_a(\mathbf{X}) \in \mathbb{R}^{m \times m'}, \text{ and } \mathbf{A}_{\xi_i} = \xi_i(\mathbf{X}) \in \mathbb{R}^{b \times b'}. \tag{27}$$

The dimension of the target transformation space, denoted as D, is determined by the codomain dimension m' of the attribute interdependence function. In most cases, the domain and codomain dimensions of the attribute and instance dependence functions analyzed in this paper are identical, *i.e.,* $m' = m$ and $b' = b$. However, for certain interdependence functions, adjustments to the codomain dimension are permitted, such as those incorporating padding modes discussed in Section 3.4. It is critical to note that the codomain dimensions m' and b' must are explicitly specified in the definitions of the functions ξ_a and ξ_i, respectively.

4.3 Versatile Data Interdependence Function

The RPN 2 model features a versatile architecture. For the RPN 2 model implemented within the TINYBIG V0.2.0 toolkit, the default architecture adheres to the structure defined by Equation 26. However, the TINYBIG V0.2.0 toolkit also provides users with the flexibility to modify the architecture to meet specific project requirements.

In addition to the current interdependence matrices involved in Equation 26, as depicted by the dashed lines connecting the attribute and instance interdependence matrices to the inputs and outputs of the data transformation function, the RPN 2 model enables the definition of attribute and instance interdependence matrices both prior and posterior to the data transformation operator. These configurations can be achieved with minor updates to the RPN 2 model architecture, as detailed below:

$$\kappa_{\xi}(\mathbf{X}) = \underbrace{\mathbf{A}^{\top}_{\xi_i^{\text{post}}}}_{\substack{\text{posterior instance}\\ \text{interdependence}}} \kappa\Big(\underbrace{\mathbf{A}^{\top}_{\xi_i^{\text{prior}}}}_{\substack{\text{prior instance}\\ \text{interdependence}}} \mathbf{X} \underbrace{\mathbf{A}_{\xi_a^{\text{prior}}}}_{\substack{\text{prior attribute}\\ \text{interdependence}}} \Big) \underbrace{\mathbf{A}_{\xi_a^{\text{post}}}}_{\substack{\text{posterior attribute}\\ \text{interdependence}}} \in \mathbb{R}^{b' \times D}. \tag{28}$$

Furthermore, in practical implementations, all interdependence functions currently incorporated in the TINYBIG V0.2.0 toolkit establish interdependence relationships along the column dimension. With minor reshaping of inputs, these implementations can be applied across any dimension of the input data batch. In some implementations of interdependence functions, we allow for the inclusion of optional learnable parameters, denoted as $\mathbf{w}_{\xi_a} \in \mathbb{R}^{l_{\xi_a}}$ and $\mathbf{w}_{\xi_i} \in \mathbb{R}^{l_{\xi_i}}$, which slightly alter the function representations as follows:

$$\mathbf{A}_{\xi_a} = \xi_a(\mathbf{X} | \mathbf{w}_{\xi_a}) \in \mathbb{R}^{m \times m'}, \text{ and } \mathbf{A}_{\xi_i} = \xi_i(\mathbf{X}^{\top} | \mathbf{w}_{\xi_i}) \in \mathbb{R}^{b \times b'}. \tag{29}$$

By using $\mathbf{X}^{\top}$ as input to ξ_i, the column-based interdependence function implementation can also be leveraged to model interdependence relationships along the row dimension of the input data batch $\mathbf{X}$. By default, the interdependence matrix outputs incorporate optional pre- and post-processing operations, including but not limited to input data batch normalization, as well as the row and column normalizations of the output matrix.

4.4 Wide and Deep Model Architectures

Analogous to the previous RPN model introduced in [89], the new RPN 2 model, enhanced with data interdependence functions, can also adopt a wide and deep architecture incorporating multi-head, multi-channel and multi-layer structures. These expansive architectures endow RPN 2 with greater model capacities for addressing function learning tasks on complex interdependent data.

Wide Architecture: As illustrated in the left plot of Figure 9, RPN 2 concurrently feeds the input batch to multiple heads, with each head possessing unique component functions. Simultaneously, as depicted in the right plot, each head within RPN 2 employs a multi-channel architecture. This architecture fabricates multiple copies of reconciled parameters and interdependence matrices to compute the desired output, specifically:

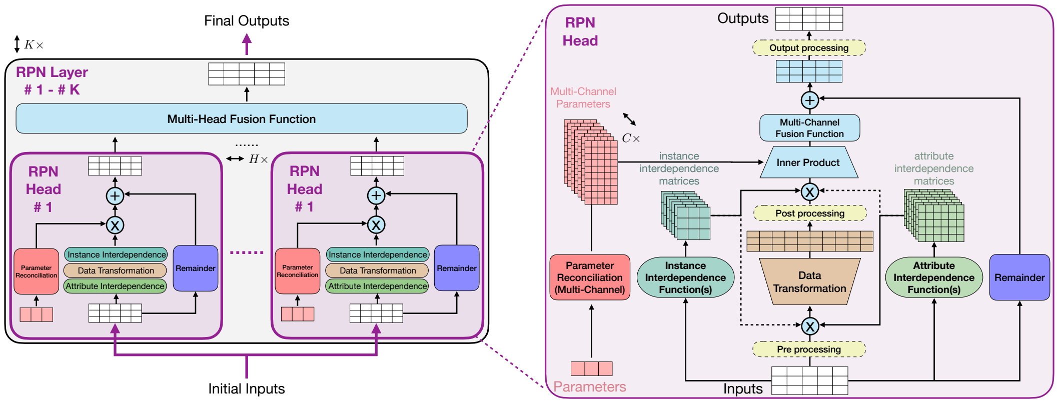


Figure 9: An illustration of the RPN 2 framework. The left plot illustrates the multi-layer (K-layer) architecture of RPN 2. Each layer involves multi-head for function learning, whose outputs will be fused together. The right plot illustrates the detailed architecture of the RPN 2 head, involving data transformation, multi-channel parameter reconciliation, remainder functions, and their internal operations. The attribute and instance interdependence functions calculate the interdependence matrices, which will be applied to the input data batch either prior or posterior to the data transformation function. The components in the rounded rectangle with yellow color in dashed lines denote the optional data processing functions (*e.g.,* activation functions and norm functions) for the inputs, expansions and outputs.

$$g(\mathbf{X}|\mathbf{w}, H, C) = \text{Fusion}\left(\left\{ \left\langle \kappa^{(h)}_{\xi^{(h)},c}\left(\mathbf{X}\right), \psi^{(h)}\left(\mathbf{w}^{(h),c}_{\psi}\right) \right\rangle + \pi^{(h)}\left(\mathbf{X}\right) \right\}_{h,c=1}^{H,C} \right), \tag{30}$$

where "Fusion$(\cdot)$" denotes the multi-head and multi-channel fusion functions (we will discuss it in the following subsection). Specifically, the data independent transformation function $\kappa^{(h)}_{\xi^{(h)},c}\left(\mathbf{X}\right)$ at the h_{th}-head and c_{th}-channel can be represented as

$$\kappa^{(h)}_{\xi^{(h)},c}\left(\mathbf{X}\right) = \left(\mathbf{A}^{(h),c}_{\xi_i}\right)^{\top} \kappa^{(h)}\left(\mathbf{X}\mathbf{A}^{(h),c}_{\xi_a}\right). \tag{31}$$

The terms $\mathbf{A}^{(h),c}_{\xi_a}$ and $\mathbf{A}^{(h),c}_{\xi_i}$ represent the attribute and instance interdependence matrices for the h_{th} head and c_{th} channel of the model, respectively. Furthermore, in the above model notation $g(\mathbf{X}|\mathbf{w}, H, C)$, the learnable parameter vector $\mathbf{w}$ encompasses both the input parameters to the reconciliation functions, *i.e.,* $\left\{\mathbf{w}^{(h),c}_{\psi}\right\}_{h,c}$, and the (optional) parameters of the data interdependence functions, *i.e.,* $\left\{\mathbf{w}^{(h),c}_{\xi_a}, \mathbf{w}^{(h),c}_{\xi_i}\right\}_{h,c}$. For simplicity, we can represent the dimensions of these learnable parameter vectors $\mathbf{w}^{(h),c}_{\psi}$, $\mathbf{w}^{(h),c}_{\xi_a}$ and $\mathbf{w}^{(h),c}_{\xi_i}$ at different heads and channels to be l_{ψ}, l_{ξ_a} and l_{ξ_i}, respectively, without indicating their head and channel indices.

The parameters of the data interdependence functions are, in fact, optional and not required for many of the interdependence functions to be introduced in the subsequent section. Throughout this paper, when discussing learnable parameters, we primarily refer to the parameters of the reconciliation functions, unless otherwise specified.

Deep Architecture: Similarly, by stacking multiple RPN 2 layers on top of each other, we may build a deep RPN 2 involving a deeper interdependence relationships among the features and data instances spanning across multiple layers:

$$\begin{cases} \text{Input:} & \mathbf{h}_0 = \mathbf{X}, \\ \text{Layer 1:} & \mathbf{h}_1 = \left\langle \kappa_{\xi,1}\left(\mathbf{X}\right), \psi_1(\mathbf{w}_{\psi,1}) \right\rangle + \pi_1(\mathbf{X}), \\ \text{Layer 2:} & \mathbf{h}_2 = \left\langle \kappa_{\xi,2}\left(\mathbf{X}\right), \psi_2(\mathbf{w}_{\psi,2}) \right\rangle + \pi_2(\mathbf{X}), \\ \cdots & \cdots\ \cdots \\ \text{Layer K:} & \mathbf{h}_K = \left\langle \kappa_{\xi,K}\left(\mathbf{X}\right), \psi_K(\mathbf{w}_{\psi,K}) \right\rangle + \pi_K(\mathbf{X}), \\ \text{Output:} & \hat{\mathbf{y}} = \mathbf{h}_K. \end{cases} \tag{32}$$

In the aforementioned equations, we intentionally omitted the head and channel indices, as well as the fusion function, to streamline the notations. By default, each layer of RPN 2 may incorporate interdependence functions with multi-head and multi-channel architectures. Furthermore, each layer in the RPN 2 architecture can possess unique data interdependence functions, affording greater flexibility in model architecture design.

4.5 Output Fusion Strategies

To aggregate outputs from the wide architecture introduced above, RPN 2 introduces the "Fusion(·)" function as indicated in the above Equation (30), which combines the learning results across multiple heads and channels. The **TINYBIG V0.2.0** toolkit implements several different fusion strategies, which can be used within the RPN 2 model to consolidate outputs learned by this multi-head and multi-channel wide architecture.

Some of the fusion strategies implemented in the **TINYBIG V0.2.0** toolkit are briefly described as follows:

- **Summation**: This strategy directly aggregates the outputs learned by the multi-head and multi-channel model architecture via summation. It significantly enhances RPN 2's learning capacity without introducing substantial computational overhead.
- **Average**: A variant of the summation fusion strategy, this approach calculates the average of outputs learned across different heads and channels of the RPN 2 model. Similar as the summation strategy, it treats those heads and channels with equal importance.
- **Parameterized Fusion**: The parameterized fusion function strategy computes a weighted fusion (*e.g.,* summation, average) of the outputs from different heads and channels. The outputs from different heads and channels are assigned with different weights, which are defined as learnable parameters to be learned together with the model.
- **Metric-based Fusion**: The metric-based fusion strategy aggregates outputs from different heads and channels using predefined numeric and statistical metrics, applied to elements at corresponding positions of the outputs across heads and channels. Examples of such metrics include, but are not limited to, maximum, minimum, product, median, and vector norms.
- **Concatenation**: This method concatenates the learning results from the multi-head and multi-channel architecture for each instance along the column dimension, yielding an extended vector output for each data instance.
- **Parameterized Concatenation**: To address potential space constraints arising from increased data batch sizes due to the concatenation fusion method, the parameterized concatenation fusion strategy incorporates a linear transformation function with learnable parameters. This function projects the concatenated outputs of the wide architecture into a dense vector for each data instance. To minimize the number of learnable parameters in the transformation, parameter reconciliation techniques from the previous study [89], such as low-rank approximations and dual low-rank hypercomplex multiplication, can be integrated into the fusion function.

4.6 Computational Costs

We previously have introduced the learning costs of the old RPN without interdependence modeling capabilities in [89] already. Here, we focus on analyzing the additional costs introduced by the data interdependence functions incorporated into the new RPN 2 model architecture. We assume the RPN 2 model has K layers, H heads, where each head involves C channels. Each channel learns the attribute and instance interdependence matrices of dimensions $m \times m'$ and $b \times b'$ based on the input data batch of size $b \times m$ and (optional) learnable parameters of length l_{ξ_a} and l_{ξ_i}. The learning cost of the attribute and instance interdependence functions are denoted as $t_a(m, m')$ and $t_i(b, b')$, respectively.

- **Space Cost**: The newly introduced storage requirements for the attribute and instance interdependence matrices, along with the (optional) extra learnable parameters, can be represented as $\mathcal{O}(KH(\underbrace{mm' + bb'}_{\text{space for matrices}} + \underbrace{C(l_{\xi_a} + l_{\xi_i})}_{\text{(optional) space for param.}}))$. In real practice, the interdependence matrices are typically sparse, resulting in substantially lower storage costs than the above notations $b'b$ and $m'm$ denoted above actually.

- **Time Cost**: The additional computational cost for learning the attribute and instance interdependence matrices with the corresponding interdependence functions and multiplying them with the data batch can be represented as $\mathcal{O}(KH(\underbrace{t_a(m, m')}_{\substack{\text{attribute matrix}\\\text{computing}}} + \underbrace{t_i(b, b')}_{\substack{\text{instance matrix}\\\text{computing}}} + \underbrace{bmm'}_{\substack{\text{attribute matrix}\\\text{multiplication}}} + \underbrace{b'Db}_{\substack{\text{instance matrix}\\\text{multiplication}}}))$.

- **Learnable Parameters**: Depending on the specific definitions of attribute and instance interdependence functions, some may involve additional learnable parameters. The number of involved learnable parameters can be represented as $\mathcal{O}(KHC(l_{\xi_a} + l_{\xi_i}))$, which is optional in real practice.

5 Data Interdependence Functions

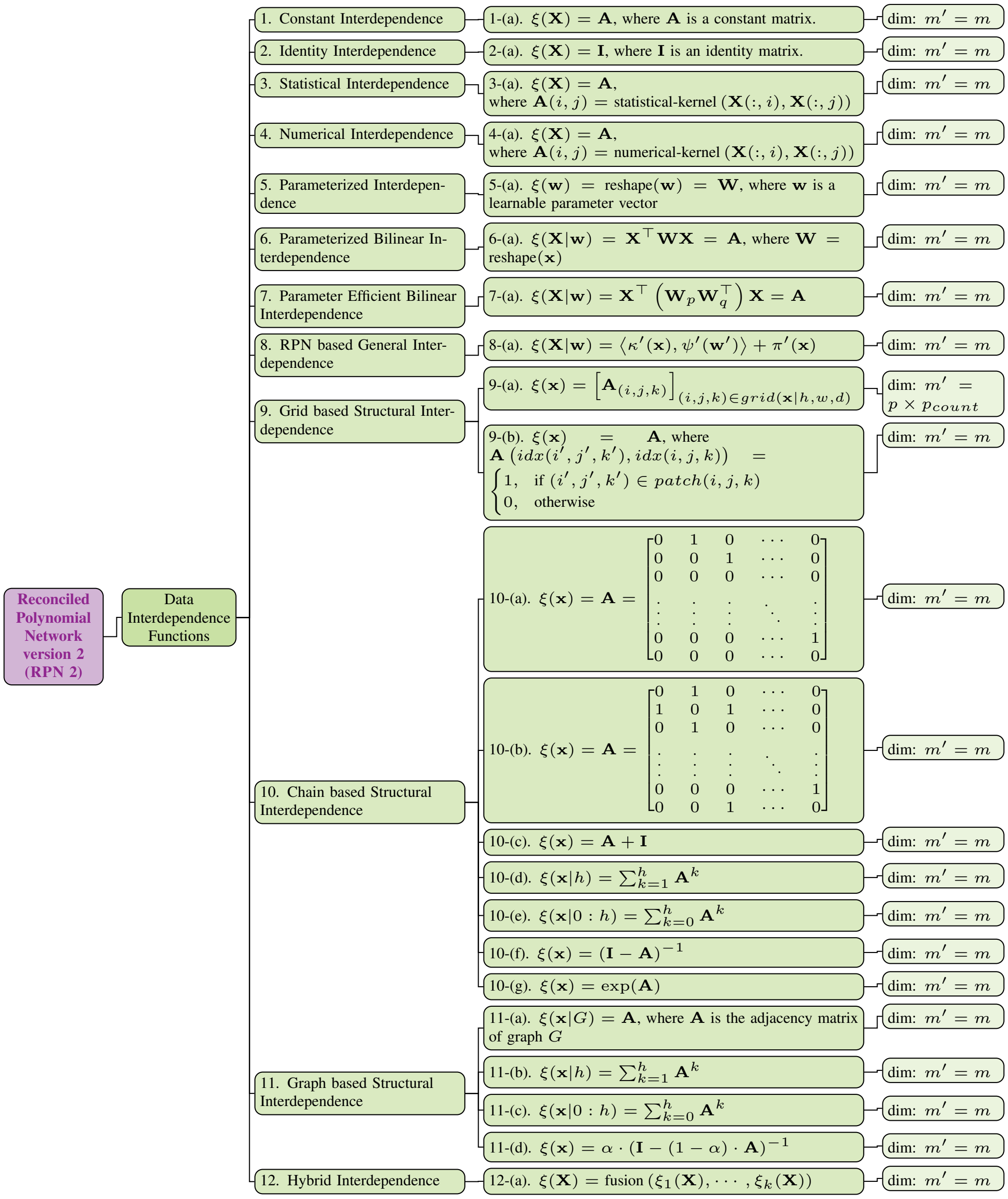
This section introduces a set of novel component functions for constructing the RPN 2 model, designed to address function learning tasks involving complex and interdependent data. We assume readers are familiar with the previous RPN paper [89] and the expansion, reconciliation, and remainder functions that have been already implemented in the TINYBIG toolkit, which will not be reiterated here. Readers seeking a concise overview of this section can also refer to Figure 10, which summarize the lists of interdependence functions to be introduced in this section.

Specifically, we introduce a new family of interdependence functions capable of modeling a wide range of interdependence relationships among both attributes and instances. These functions can be defined using input data batches, underlying geometric and topological structures, optional learnable parameters, or a hybrid combination of these elements.

5.1 Data Interdependence Functions

The data interdependence functions are designed to model the complex relationships among attributes and data instances, corresponding to the columns and rows of the input data batch, respectively. This section presents several categories of interdependence functions, each firmly grounded in robust theoretical foundations, including linear algebra and statistical theory. The interdependence functions defined using computational geometry and topology approaches, which are based on input data batch side information such as shapes, structures, and spatial interconnections, will be introduced in the subsequent subsection instead.

Many of the interdependence functions presented herein are versatile, capable of computing interdependence matrices for both columns and rows of the input data batch, which correspond to the



Reconciled Polynomial Network version 2 (RPN 2)
Data Interdependence Functions
1. Constant Interdependence
1-(a). $\xi(\mathbf{X}) = \mathbf{A}$, where $\mathbf{A}$ is a constant matrix.
dim: $m' = m$
2. Identity Interdependence
2-(a). $\xi(\mathbf{X}) = \mathbf{I}$, where $\mathbf{I}$ is an identity matrix.
dim: $m' = m$
3. Statistical Interdependence
3-(a). $\xi(\mathbf{X}) = \mathbf{A}$, where $\mathbf{A}(i, j) =$ statistical-kernel $(\mathbf{X}(:, i), \mathbf{X}(:, j))$
dim: $m' = m$
4. Numerical Interdependence
4-(a). $\xi(\mathbf{X}) = \mathbf{A}$, where $\mathbf{A}(i, j) =$ numerical-kernel $(\mathbf{X}(:, i), \mathbf{X}(:, j))$
dim: $m' = m$
5. Parameterized Interdependence
5-(a). $\xi(\mathbf{w}) =$ reshape$(\mathbf{w}) = \mathbf{W}$, where $\mathbf{w}$ is a learnable parameter vector
dim: $m' = m$
6. Parameterized Bilinear Interdependence
6-(a). $\xi(\mathbf{X}|\mathbf{w}) = \mathbf{X}^\top \mathbf{W} \mathbf{X} = \mathbf{A}$, where $\mathbf{W} =$ reshape$(\mathbf{x})$
dim: $m' = m$
7. Parameter Efficient Bilinear Interdependence
7-(a). $\xi(\mathbf{X}|\mathbf{w}) = \mathbf{X}^\top \left(\mathbf{W}_p \mathbf{W}_q^\top\right) \mathbf{X} = \mathbf{A}$
dim: $m' = m$
8. RPN based General Interdependence
8-(a). $\xi(\mathbf{X}|\mathbf{w}) = \langle \kappa'(\mathbf{x}), \psi'(\mathbf{w}') \rangle + \pi'(\mathbf{x})$
dim: $m' = m$
9. Grid based Structural Interdependence
9-(a). $\xi(\mathbf{x}) = \left[\mathbf{A}_{(i,j,k)}\right]_{(i,j,k) \in grid(\mathbf{x}|h,w,d)}$
dim: $m' = p \times p_{count}$
9-(b). $\xi(\mathbf{x}) = \mathbf{A}$, where $\mathbf{A}\left(idx(i',j',k'), idx(i,j,k)\right) = \begin{cases} 1, & \text{if } (i',j',k') \in patch(i,j,k) \\ 0, & \text{otherwise} \end{cases}$
dim: $m' = m$
10. Chain based Structural Interdependence
10-(a). $\xi(\mathbf{x}) = \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$
dim: $m' = m$
10-(b). $\xi(\mathbf{x}) = \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 1 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 1 & \cdots & 0 \end{bmatrix}$
dim: $m' = m$
10-(c). $\xi(\mathbf{x}) = \mathbf{A} + \mathbf{I}$
dim: $m' = m$
10-(d). $\xi(\mathbf{x}|h) = \sum_{k=1}^{h} \mathbf{A}^k$
dim: $m' = m$
10-(e). $\xi(\mathbf{x}|0:h) = \sum_{k=0}^{h} \mathbf{A}^k$
dim: $m' = m$
10-(f). $\xi(\mathbf{x}) = (\mathbf{I} - \mathbf{A})^{-1}$
dim: $m' = m$
10-(g). $\xi(\mathbf{x}) = \exp(\mathbf{A})$
dim: $m' = m$
11. Graph based Structural Interdependence
11-(a). $\xi(\mathbf{x}|G) = \mathbf{A}$, where $\mathbf{A}$ is the adjacency matrix of graph G
dim: $m' = m$
11-(b). $\xi(\mathbf{x}|h) = \sum_{k=1}^{h} \mathbf{A}^k$
dim: $m' = m$
11-(c). $\xi(\mathbf{x}|0:h) = \sum_{k=0}^{h} \mathbf{A}^k$
dim: $m' = m$
11-(d). $\xi(\mathbf{x}) = \alpha \cdot (\mathbf{I} - (1-\alpha) \cdot \mathbf{A})^{-1}$
dim: $m' = m$
12. Hybrid Interdependence
12-(a). $\xi(\mathbf{X}) =$ fusion $(\xi_1(\mathbf{X}), \cdots, \xi_k(\mathbf{X}))$
dim: $m' = m$

Figure 10: An overview of data interdependence, fusion, and data compression functions implemented in the TINYBIG v0.2.0 toolkit for constructing the RPN 2 model architecture.

attribute and instances, respectively. However, to simplify the presentations below, we will focus on computing the interdependence matrices for the columns of the input data batch, *i.e.,* the attribute interdependence. It is worth noting that, as briefly discussed in the previous section, these function implementations can also be readily adapted to the alternative mode (*e.g.,* row-based analysis or instance interdependence) through the simple transposition of the input data batch.

5.1.1 Constant Interdependence Function

Among the array of interdependence functions to be introduced in this section, as one of the most basic function, the constant interdependence function generates an output interdependence matrix in the form of a constant matrix with a specified shape.

Formally, based on the (optional) input data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$, we define the constant interdependence function as:

$$\xi(\mathbf{X}) = \mathbf{A} \in \mathbb{R}^{m \times m'}. \tag{33}$$

This function facilitates the definition of customized constant interdependence matrices, allowing for a manually defined matrix $\mathbf{A}$ to be provided as a hyper-parameter during function initialization. Two special cases warrant particular attention: when $\mathbf{A}_c$ consists entirely of zeros, it is designated as the *zero interdependence matrix*, whereas a matrix of all ones is termed the *one interdependence matrix*. Moreover, it is noteworthy that the constant interdependence function exclusively utilizes the shape information of the input data batch, specifically the dimension m. It does not incorporate any additional information from the data batch matrix $\mathbf{X}$ in constructing the output interdependence matrices. The second dimension, m', can be manually specified when defining the interdependence matrix $\mathbf{A}$. In the absence of an explicit specification, by default, we will assign $m' = m$. More importantly, this function operates without learnable parameters, thereby incurring no additional learning costs during the model's training process.

As a standard feature shared with subsequent interdependence functions, this function accommodates the integration of pre- and post-processing operations, such as input batch normalization and activation, and the output matrix row and column normalizations. For brevity, this default capability will not be reiterated in the descriptions of subsequent functions.

5.1.2 Identity Interdependence Function

A notable special case of the aforementioned constant interdependence functions is the identity interdependence function. This function outputs the interdependence matrix as a diagonal constant identity (or eye) matrix, formally represented as:

$$\xi(\mathbf{X}) = \mathbf{I} \in \mathbb{R}^{m \times m'}, \tag{34}$$

where the output interdependence matrix is, by default, a square matrix with $m' = m$.

The identity interdependence function proves particularly useful for modeling the *independence* of attributes and data instances within the input data batch. Notably, all models developed based on the previous version of RPN, as introduced in [89], can be precisely reduced to a special case of RPN 2 with identity interdependence functions for both attributes and instances, *i.e.,* both the instances and attributes are independent.

In practical applications, leveraging sparse matrix representation and multiplication techniques results in minimal additional storage and time costs introduced by the identity interdependence function, rendering these costs nearly negligible. For scenarios where even these minor costs are undesirable in modeling independence relationships, an alternative approach is to define the interdependence functions as "None" in the implementation, which will be properly handled by the TINYBIG v0.2.0 toolkit without incurring any extra space or time costs.

5.1.3 Statistical Kernel Based Interdependence Function

The previous Section 3.2.1 has introduced several statistical kernels for quantifying interdependence relationships among vectors. Building upon this foundation, we now present an expansive family of interdependence functions derived from these kernels. Rooted in probability theory and statistical inference, these kernels offer the advantage of explicitly modeling and quantifying uncertainty in the interdependence calculations derived from the input data batch.

Formally, given a data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$, we can define the statistical kernel-based interdependence function as:

$$\xi(\mathbf{X}) = \mathbf{A} \in \mathbb{R}^{m \times m'}, \text{ where } \mathbf{A}(i, j) = \text{kernel}\left(\mathbf{X}(:, i), \mathbf{X}(:, j)\right), \tag{35}$$

We now present a concise overview of several frequently employed statistical kernels that can be utilized to define the aforementioned interdependence function:

(a) KL Divergence:

$$\text{kernel}(\mathbf{x}, \mathbf{y}) = \sum_i \mathbf{x}(i) \log \left(\frac{\mathbf{x}(i)}{\mathbf{y}(i)} \right), \tag{36}$$

where $\mathbf{x}$ and $\mathbf{y}$ have been normalized.

(b) Pearson Correlation:

$$\text{kernel}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{b} (\frac{\mathbf{x}(i) - \mu_x}{\sigma_x})(\frac{\mathbf{y}(i) - \mu_y}{\sigma_y})}{b}, \tag{37}$$

where $\mu_x, \mu_y, \sigma_x, \sigma_y$ are the mean and std.

(c) RV Coefficient:

$$\text{kernel}(\mathbf{x}, \mathbf{y}) = \frac{tr(\mathbf{\Sigma}_{x,y} \mathbf{\Sigma}_{y,x})}{\sqrt{tr(\mathbf{\Sigma}_x^2) tr(\mathbf{\Sigma}_y^2)}}. \tag{38}$$

(d) Mutual Information:

$$\text{kernel}(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \log \left(\frac{det(\mathbf{\Sigma}_x) det(\mathbf{\Sigma}_y)}{det\,(\mathbf{\Sigma})} \right). \tag{39}$$

We have previously provided detailed descriptions of the RV coefficient and mutual information kernels in Section 3.2.1. To further elucidate these concepts, let us consider an input data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$. We can compute the column-wise mean values as vector $\boldsymbol{\mu} \in \mathbb{R}^m$, which allows us to define the centered data batch $\mathbf{X}'$ as follows:

$$\mathbf{X}' = \mathbf{X} - \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu} \\ \cdots \\ \boldsymbol{\mu} \end{bmatrix} \in \mathbb{R}^{b \times m}. \tag{40}$$

Subsequently, we can define the column-wise covariance matrix $\mathbf{\Sigma}$ based on this centered data matrix:

$$\mathbf{\Sigma} = \frac{1}{b-1} \left(\mathbf{X}' \right)^\top \mathbf{X}' \in \mathbb{R}^{m \times m}. \tag{41}$$

This covariance matrix serves as a crucial component in calculating the aforementioned RV coefficient and mutual information kernels for computing the corresponding interdependence matrices.

5.1.4 Numerical Kernel Based Interdependence Function

In addition to the statistical metrics discussed above, we can also the define interdependence functions based on numerical metrics, some of which have been briefly introduced in Section 3.2.2. Similar as the above statistical kernel-based functions, these numerical kernel-based interdependence functions compute the pairwise numerical scores for column vectors within the input data batch, thereby constructing a comprehensive interdependence matrix.

Formally, given a data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$, we define the numerical metric-based interdependence function as:

$$\xi(\mathbf{X}) = \mathbf{A} \in \mathbb{R}^{m \times m'}, \text{ where } \mathbf{A}(i,j) = \text{kernel}\left(\mathbf{X}(:,i), \mathbf{X}(:,j) \right). \tag{42}$$

By convention, the resulting matrix $\mathbf{A}$ is square, with dimensions $m' = m$. This section offers a diverse array of approaches for defining numerical metrics on vectors from the input data batch. We elucidate below a curated selection of these methods, encompassing those previously introduced in Section 3.2.2, as well as additional noteworthy new kernels:

(a) Linear (Inner-Product) Kernel:

$$\text{kernel}(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle, \tag{43}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the column vectors from the input data batch.

(b) Polynomial Kernel:

$$\text{kernel}(\mathbf{x}, \mathbf{y} | c, d) = \left(\langle \mathbf{x}, \mathbf{y} \rangle + c \right)^d, \tag{44}$$

where c and d are the hyper-parameters of the kernel function.

(c) Hyperbolic Tangent Kernel:

$$\text{kernel}(\mathbf{x}, \mathbf{y}|\alpha, c) = \tanh\left(\alpha \left\langle \mathbf{x}, \mathbf{y} \right\rangle + c\right). \tag{45}$$

(d) Exponential Kernel:

$$\text{kernel}(\mathbf{x}, \mathbf{y}|\gamma) = \exp\left(-\gamma \left\| \mathbf{x} - \mathbf{y} \right\|_1\right). \tag{46}$$

(e) Cosine Similarity based Kernel:

$$\text{kernel}(\mathbf{x}, \mathbf{y}) = \frac{\left\langle \mathbf{x}, \mathbf{y} \right\rangle}{\left\| \mathbf{x} \right\| \cdot \left\| \mathbf{y} \right\|}, \tag{47}$$

(f) Minkowski Distance based Kernel:

$$\text{kernel}(\mathbf{x}, \mathbf{y}) = 1 - \left\| \mathbf{x} - \mathbf{y} \right\|_p, \tag{48}$$

where $p \in \{1, 2, \cdots, \infty\}$.

(g) Gaussian RBF Kernel:

$$\text{kernel}(\mathbf{x}, \mathbf{y}|\sigma) = \exp\left(-\frac{\left\| \mathbf{x} - \mathbf{y} \right\|_2^2}{2\sigma^2}\right), \tag{49}$$

(h) Laplacian Distance:

$$\text{kernel}(\mathbf{x}, \mathbf{y}|\sigma) = \exp\left(-\frac{\left\| \mathbf{x} - \mathbf{y} \right\|_1}{\sigma}\right), \tag{50}$$

(i) Anisotropic RBF Kernel:

$$\text{kernel}(\mathbf{x}, \mathbf{y}) = \exp\left(-(\mathbf{x} - \mathbf{y})\mathbf{A}(\mathbf{x} - \mathbf{y})^\top\right),$$

where $\mathbf{A} = \text{diag}(\mathbf{a})$ is a diagonal matrix modeling vector element-specific scaling factors. (51)

(j) Custom Hybrid Kernels:

$$\text{kernel}(\mathbf{x}, \mathbf{y}|\alpha, \beta) = \alpha k_1(\mathbf{x}, \mathbf{y}) + \beta k_2(\mathbf{x}, \mathbf{y}),$$

where k_1 and k_2 are custom designed kernel functions and α, β are their weights. (52)

The aforementioned Minkowski distance-based kernel metric serves as a generalized representation of several frequently employed distance metrics. Its versatility is evident in its ability to reduce to specific, well-known distances depending on the value of the parameter p in the vector norm. Notable special cases include the Manhattan distance ($p = 1$), Euclidean distance ($p = 2$), and Chebyshev distance ($p = \infty$), which can be expressed as follows:

$$\begin{aligned}
\text{kernel}(\mathbf{x}, \mathbf{y}) &= 1 - \left\| \mathbf{x} - \mathbf{y} \right\|_1 = 1 - \sum_{i=1}^{d} \left| \mathbf{x}(i) - \mathbf{y}(i) \right|, \\
\text{kernel}(\mathbf{x}, \mathbf{y}) &= 1 - \left\| \mathbf{x} - \mathbf{y} \right\|_2 = 1 - \sqrt{\sum_{i=1}^{d} \left(\mathbf{x}(i) - \mathbf{y}(i) \right)^2}, \\
\text{kernel}(\mathbf{x}, \mathbf{y}) &= 1 - \left\| \mathbf{x} - \mathbf{y} \right\|_\infty = 1 - \max\left(\{ \left| \mathbf{x}(i) - \mathbf{y}(i) \right| \}_{i=1}^{d} \right).
\end{aligned} \tag{53}$$

5.1.5 Parameterized Interdependence Function

In addition to the above interdependence function solely defined based on the input data batch, another category of fundamental interdependence functions is the parameterized interdependence function, which constructs the interdependence matrix exclusively from learnable parameters.

Formally, given a learnable parameter vector $\mathbf{w} \in \mathbb{R}^{l_\xi}$, the parameterized interdependence function transforms it into a matrix of desired dimensions $m \times m'$ as follows:

$$\xi(\mathbf{w}) = \text{reshape}(\mathbf{w}) = \mathbf{W} \in \mathbb{R}^{m \times m'}. \tag{54}$$

This parameterized interdependence function operates independently of any data batch, deriving the output interdependence matrix solely from the learnable parameter vector $\mathbf{w}$, whose requisite length of vector $\mathbf{w}$ is $l_\xi = m \times m'$.

In addition to the above function, several parameter reconciliation techniques, as introduced in the preceding RPN paper [89], can also be utilized to generate the interdependence matrix $\mathbf{W}$ of the desired shape from the input parameter vector $\mathbf{w}$. These methods, including low-rank parameter reconciliation (LoRR), hypercomplex multiplication (HM), low-rank parameterized HM (LPHM), and dual LPHM, also offer the advantages of reducing the number of required learnable parameters. For a comprehensive exposition of these techniques, readers are recommended to refer to the previous RPN paper [89]; we shall not revisit them in detail here.

5.1.6 Parameterized Bilinear Interdependence Function

In addition to the numerical metrics discussed above, we have previously introduced another quantitative measure, namely the *bilinear form*, in Section 3.2.2. This measure enumerates all potential interactions between vector elements to compute interdependence scores. In this section, we propose defining the interdependence function based on the bilinear form with learnable parameters to model these element interactions.

Formally, given a data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$, we can represent the parameterized bilinear form-based interdependence function as follows:

$$\xi(\mathbf{X}|\mathbf{w}) = \mathbf{X}^\top \mathbf{W} \mathbf{X} = \mathbf{A} \in \mathbb{R}^{m \times m}, \tag{55}$$

where $\mathbf{W} = \text{reshape}(\mathbf{w}) \in \mathbb{R}^{b \times b}$ denotes the parameter matrix reshaped from the learnable parameter vector $\mathbf{w} \in \mathbb{R}^{l_\xi}$ with length $l_\xi = b^2$.

5.1.7 Parameter Efficient Bilinear Interdependence Function

For input data batches with a large number of instances (*i.e.,* when b is very large), the introduced parameter matrix $\mathbf{W}$ for the above parameterized bilinear interdependence function will also have a large dimension. To avoid introducing a large number of parameters for the interdependence function definition, we can employ parameter reconciliation techniques similar to those introduced in [89]. These techniques, such as low-rank approximation, HM, LPHM, and Dual LPHM, can all be used to reduce the number of required learnable parameters. These methods have been implemented in the TINYBIG V0.2.0 toolkit and are all ready for use.

Low-Rank Parameterized Bilinear Interdependence Function: To illustrate, we present the bilinear interdependence function with low-rank parameter reconciliation, which will also be used to construct the Transformer model with the RPN 2 model architecture. More discussions on that will be provided in the following Section 8.

Similar to the low-rank parameter reconciliation function, the parameter matrix $\mathbf{W} \in \mathbb{R}^{b \times b}$ used in Equation (55) can be represented as the product of two low-rank sub-matrices $\mathbf{W}_p \in \mathbb{R}^{b \times r}$ and $\mathbf{W}_q \in \mathbb{R}^{b \times r}$, both of rank r (where $r \ll b$). This will transform the equation to:

$$\xi(\mathbf{X}|\mathbf{w}) = \mathbf{X}^\top \left(\mathbf{W}_p \mathbf{W}_q^\top \right) \mathbf{X} = (\mathbf{X}^\top \mathbf{W}_p)(\mathbf{X}^\top \mathbf{W}_q)^\top = \mathbf{A} \in \mathbb{R}^{m \times m}. \tag{56}$$

In the implementation, the parameter vector $\mathbf{w}$ is first partitioned into two sub-vectors and subsequently reshaped into two matrices $\mathbf{W}_p, \mathbf{W}_q \in \mathbb{R}^{b \times r}$. This approach will reduce the number of the required learnable parameter vector $\mathbf{w}$ to $l_\xi = 2br$.

5.1.8 RPN based General Data Interdependence Function

While it is beyond the scope of this paper to enumerate all the parameter-efficient reconciliation techniques introduced in previous work, it is worth noting that many of the data expansion techniques previously discussed in [89] can also be potentially utilized in defining interdependence functions. For example, the inner-product and bilinear form based interdependence functions defined above can both be implemented based on Taylor's expansion of the input data batch.

To illustrate, consider the output interdependence matrix $\mathbf{A}$ generated by the bilinear interdependence function in Equation (55), its $(i, i)_{th}$ element models the interactions of the i_{th} column with itself:

$$\begin{aligned}\mathbf{A}(i,i) = \mathbf{X}(:,i)^\top \mathbf{W} \mathbf{X}(:,i) &= \sum_{p=1}^{b} \sum_{q=1}^{b} \mathbf{X}(p,i)\mathbf{X}(q,i)\mathbf{W}(p,q), \\ &= \left\langle \mathbf{X}(:,i) \otimes \mathbf{X}(:,i), \text{flatten}(\mathbf{W}) \right\rangle ,\end{aligned} \tag{57}$$

where $\otimes$ denotes the Kronecker product of the vectors and the flatten$(\cdot)$ operator will flatten the matrix to a vector for computing the inner product. In essence, the calculated $\mathbf{A}(i,i)$ contains the weighted summation of the second-order Taylor's expansion of the input data batch column vector $\mathbf{X}(:,i)$. Similar representations also hold for other entries of the interdependence matrix.

To generalize this function definition, we propose to disentangle the interdependence function as a RPN-head proposed in the previous paper [89] composed of three components: the data expansion function, parameter reconciliation function, and remainder function (without further considering any interdependence relationships to avoid nested modeling). Formally, the interdependence function $\xi : \mathbb{R}^{b \times m} \to \mathbb{R}^{m \times m'}$ can be viewed as a mapping between the input vector space of dimension $(b \times m)$ and the output vector space of dimension $(m \times m')$. We propose to represent it as follows:

$$\xi(\mathbf{X}|\mathbf{w}) = \langle \kappa'(\mathbf{x}), \psi'(\mathbf{w}') \rangle + \pi'(\mathbf{x}), \tag{58}$$

where $\mathbf{x} = \text{flatten}(\mathbf{X})$ is the flattened vector of length $(b \times m)$ from the input data batch matrix $\mathbf{X} \in \mathbb{R}^{b \times m}$. This is viewed as a single (independent) pseudo "data instance" for the above RPN-layer. The notations are defined as:

$$\kappa' : \mathbb{R}^{(b \times m)} \to \mathbb{R}^{D}, \psi' : \mathbb{R}^{l} \to \mathbb{R}^{(m \times m') \times D}, \text{ and } \pi_{\xi} : \mathbb{R}^{(b \times m)} \to \mathbb{R}^{(m \times m')}, \tag{59}$$

which denote the data expansion function, parameter reconciliation function, and remainder function, respectively, as specifically defined for the interdependence function. The prime symbols attached to the function name notations are added to differentiate them from the component functions used for building the RPN 2 architecture.

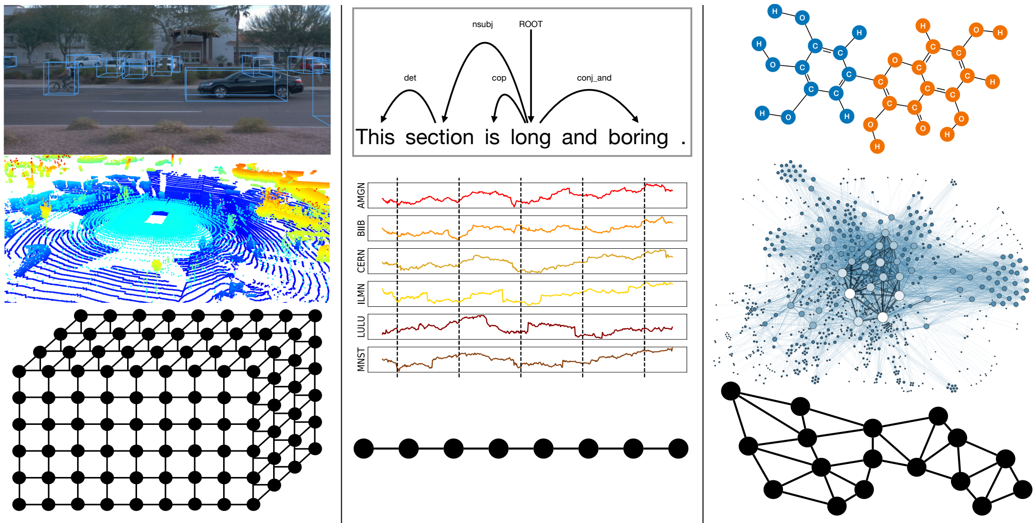
By default, the component functions for data expansion, parameter reconciliation, and remainder introduced in previous work, as well as those to be introduced in this paper, can all be used to define the interdependence function illustrated above. To avoid over-complicating the interdependence function definition, we will only use the single-layer, single-head, and single-channel RPN-layer here. Additionally, to prevent infinitely nested interdependence function definitions, we assume all attributes in this vector "$\mathbf{x} = \text{flatten}(\mathbf{X})$" to be independent.

The RPN-layer based interdependence function described above provides readers with considerable flexibility in defining customized interdependence functions for diverse input data. These functions have been implemented in the tinyBIG v0.2.0 toolkit and are now ready for use.

5.2 Structural Interdependence Functions

This section delves into the definition of data interdependence functions based on the geometric and topological structural information inherent in the data batch. We examine diverse structural interdependence relationships among attributes and instances from computational geometry and topology perspectives, taking into account structural factors, such as spatial configurations, sequential orders, and interconnections, about the input data batch.

Note: This subsection contains numerous technical details accompanied by complex notations denoting grids, patches, and matrices. Despite its challenging nature, we strongly encourage readers to thoroughly engage with this material. The interdependence functions defined herein are crucial and will be instrumental in unifying Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Graph Neural Networks (GNNs) into RPN 2's canonical representation. Omitting this subsection may impede readers' comprehension of the subsequent model unification and new technique designs.



(a) grid (b) chain (c) graph

Figure 11: An illustration of the underlying structures of multi-modal data. (a) Images and point clouds represented as grid-structured data, with nodes depicting pixels and voxels, and links representing spatial relationships; (b) Language and time series data represented as chain-structured data, with nodes depicting tokens and numerical values, and links representing sequential orders; (c) Molecule compounds and online social networks represented as graph-structured data, with nodes depicting atoms and users, and links representing atomic bonds and social connections.

5.2.1 Modality Specific Structural Interdependence Relationships

Computational geometry and topology are fields within computer science that focus on the study of algorithms expressible in terms of data geometry and topology structures. These disciplines involve the design, analysis, and implementation of algorithms to solve geometric and topological problems, often dealing with objects such as points, lines, and polygons, as well as concepts of connectedness, compactness, and continuity. Both computational geometry and topology play fundamental roles in numerous areas of computer science and machine learning, with their algorithms being crucial for processing, analyzing, and interpreting data in geometric and topological forms.

Figure 11 illustrates examples of real-world data and their corresponding topological structures. Images and point clouds can be represented as grids in 2D and 3D space; language and time series data exhibit chain structures with sequential connections; and molecule compounds and social networks can be represented as graphs. These diverse data types demonstrate various structural interdependence relationships among instances and attributes in terms of spatial distributions, sequential orders, and extensive interconnections, respectively.

Such geometric and topological structure information of input data plays a crucial role in elucidating underlying data distribution patterns. However, when converting these data into mathematical representations (*e.g.,* vectors or matrices) and building models to fit them, much of this geometric and topological structure information is either lost or not fully utilized. In the following subsections, we will define diverse data interdependence functions to explicitly model these varied data interdependence relationships.

Note: In addition to the grid, chain, and graph structures discussed here, there exist other data types (such as graphics mesh, hierarchical ontology, climate and space satellite sensor data) that can be represented with alternative or more complex structures. We plan to study these in future papers addressing concrete real-world application problems and will incorporate their corresponding data interdependence functions into the tinyBIG v0.2.0 toolkit as well.

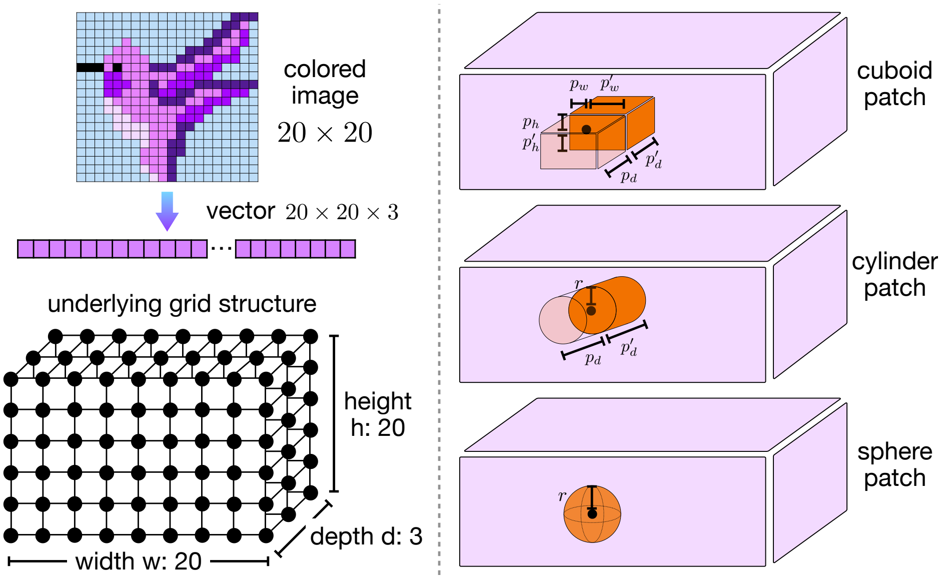


Figure 12: An illustration of data grid structure and diverse patch shapes. (a) Left: Representation of an input data instance vector and its corresponding 3D grid structure. (b) Right: Illustration of three patch types defined on the grid: cuboid, cylinder, and sphere.

5.2.2 Topological Grid Structure and Geometric Patch Shapes

This section introduces several structural interdependence functions based on geometric patches present in input data batches with grid structures. These geometric patch-based data structural interdependence functions are primarily applied to model the interdependence relationships among attributes, such as pixels and voxels in images and point clouds. Additionally, in practical applications, they can also be utilized to model relationships among data instances with fixed grid-structured interdependence relationships.

Geometric Grid: Formally, given a data instance vector $\mathbf{x} \in \mathbb{R}^m$, as illustrated in Figure 12, we can represent its underlying 3D grid structure as an ordered list of coordinates denoting the attributes' locations in the grid:

$$grid(\mathbf{x}|h, w, d) = \left[(i, j, k)\right]_{i \in \{0,1,\cdots,h-1\}, j \in \{0,1,\cdots,w-1\}, k \in \{0,1,\cdots,d-1\}}. \tag{60}$$

This paper focuses on cuboid-structured grids, which are prevalent as underlying structures of data instances in real-world applications, such as the aforementioned images and point clouds. The grid size is represented as $|grid(\mathbf{x}|h, w, d)| = h \times w \times d$, where h, w, d denote the height, width, and depth dimensions of the grid, respectively. The grid size should equal the length of the data vector, *i.e.*, $m = h \times w \times d$, where $\mathbf{x} \in \mathbb{R}^m$. By default, we can also use the input data instance vector dimension m to represent the grid size.

Attribute Index - Grid Coordinate Bijection: A bijective mapping exists between the coordinates of the data instance vector $\mathbf{x}$ and its corresponding grid structure $grid(\mathbf{x}|h, w, d)$. For each attribute in vector $\mathbf{x}$, we can identify its corresponding coordinates in its underlying grid; conversely, for each coordinate tuple from the grid, we can precisely locate the attribute from vector $\mathbf{x}$ by its index.

Formally, given the coordinate index tuple (i, j, k) from $grid(\mathbf{x}|h, w, d)$, its corresponding attribute index in vector $\mathbf{x}$ can be represented as:

$$idx(i, j, k) = i \cdot w \cdot d + j \cdot d + k, \forall i \in \{0, 1, \cdots, h-1\}, j \in \{0, 1, \cdots, w-1\}, k \in \{0, 1, \cdots, d-1\}. \tag{61}$$

Conversely, given an attribute with index idx from vector $\mathbf{x}$, we can calculate its corresponding coordinate tuples (i, j, k) in $grid(\mathbf{x}|h, w, d)$ as follows:

$$i = \left\lfloor \frac{idx}{w \cdot d} \right\rfloor, j = \left\lfloor \frac{idx\%(w \cdot d)}{d} \right\rfloor, k = idx\%d, \tag{62}$$

where notation $\%$ denotes the modulus operator and $\lfloor \cdot \rfloor$ is the floor operator.

These bijection mappings enable direct access to the attributes in the data instance $\mathbf{x}$ and the coordinate tuples in its underlying grid structure $grid(\mathbf{x}|h, w, d)$. For simplicity, in subsequent descriptions, we may treat these attributes and grid coordinates equivalently without distinguishing their differences.

Geometric Cuboid Patch: A geometric patch denotes a small-sized, localized region in the input data instance's underlying grid structure, facilitating the analysis of local data structures and details by breaking down complex data into manageable local parts. As illustrated in Figure 12, the RPN 2 model allows patches of different shapes, such as *cuboid*, *cylinder*, and *sphere*, each capturing distinct types of local structural interdependence relationships.

As shown in Figure 12, these geometric patches of different shapes can be described with a set of hyper-parameters denoting the sizes, such as $(p_h, p'_h; p_w, p'_w; p_d, p'_d)$ for cuboid patch, $(r; p_d, p'_d)$ for cylinder patch, and (r) for sphere patch. For cases where cuboid and cylinder patches have symmetric shapes along dimensions (*i.e.,* $p'_h = p_h$, $p'_w = p_w$, $p'_d = p_d$ for cuboid patch and $p'_d = p_d$ for cylinder patch), these shape hyper-parameters can be simplified to $(p_h; p_w; p_d)$ for cuboid patch and $(r; p_d)$ for cylinder patch.

Given a coordinate tuple (i, j, k) in the grid, we can represent the patch, *e.g.,* a cuboid with shape $(p_h, p'_h; p_w, p'_w; p_d, p'_d)$, centered at (i, j, k) as an ordered list of coordinate tuples:

$$patch(i, j, k) = \left[(i + \nabla i, j + \nabla j, k + \nabla k)\right]_{\nabla i \in [-p_h, p'_h], \nabla j \in [-p_w, p'_w], \nabla k \in [-p_d, p'_d]}, \tag{63}$$

Its size is represented as $|patch(i, j, k)| = (p_h + p'_h + 1) \times (p_w + p'_w + 1) \times (p_d + p'_d + 1)$, denoting the number of coordinate tuples covered by the patch. For simplicity, we introduce the term p to represent the patch size, *i.e.,*

$$p = |patch(i, j, k)|, \tag{64}$$

which is determined by the hyper-parameters and will be used frequently in the follow descriptions.

Cylinder and Sphere Patches: Similarly, we can represent cylinder patches of shape $(r; p_d, p'_d)$ and sphere patches of shape (r) centered at coordinates (i, j, k) as follows:

$$\begin{aligned} patch(i, j, k) &= \left[(i + \nabla i, j + \nabla j, k + \nabla k)\right]_{\nabla i, \nabla j \in [-r, r] \wedge \nabla i^2 + \nabla j^2 \leq r^2, \nabla k \in [-p_d, p'_d]}, \\ patch(i, j, k) &= \left[(i + \nabla i, j + \nabla j, k + \nabla k)\right]_{\nabla i, \nabla j, \nabla k \in [-r, r] \wedge \nabla i^2 + \nabla j^2 + \nabla k^2 \leq r^2}, \end{aligned} \tag{65}$$

whose size is also represented by the term $p = |patch(i, j, k)|$ by default.

The main motivation for proposing the cylinder and sphere patch shapes in this paper is due to their advantages in maintaining their rotational invariances. Rotational transformations frequently occur in various types of input data during data sensing, collection, storage, clean and processing, particularly in images and point clouds. When considering cuboid patch shapes, although the data instance itself remains unaltered under rotation (*e.g.,* about an axis parallel to the depth), the grid nodes encompassed by these patches can undergo significant variations.

In contrast, cylinder and sphere patches exhibit superior rotational invariance characteristics:

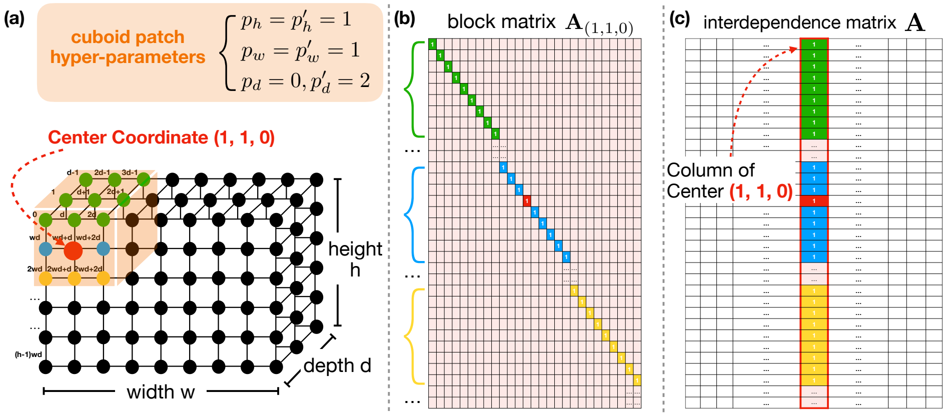

(a)
cuboid patch hyper-parameters
$p_h = p'_h = 1$
$p_w = p'_w = 1$
$p_d = 0, p'_d = 2$
Center Coordinate (1, 1, 0)
d-1
2d-1
3d-1
1
d+1
2d+1
0
d
2d
wd
wd+d
wd+2d
2wd
2wd+d
2wd+2d
(h-1)wd
height h
depth d
width w
(b)
block matrix $\mathbf{A}_{(1,1,0)}$
(c)
interdependence matrix $\mathbf{A}$
Column of Center (1, 1, 0)

Figure 13: An illustration of cuboid patch based structural interdependence matrices. (a) Underlying grid of input data instance and cuboid patch centered as $(1, 1, 0)$ with pre-defined sizes. (b) Block interdependence matrix $\mathbf{A}_{(1,1,0)}$ in the padding mode of center coordinate $(1, 1, 0)$. (c) Interdependence matrix $\mathbf{A}$ in the aggregation mode and the column corresponding to coordinate $(1, 1, 0)$.

- **Cylinder patches**: These maintain consistent attribute coverage when the data undergoes rotation around an axis parallel to the depth dimension.
- **Sphere patches**: These demonstrate even greater invariance properties. The grid nodes encompassed by a sphere patch remain constant irrespective of the orientation of the rotation axis in three-dimensional space.

These inherent properties of cylinder and sphere patches provide robust alternatives to cuboid shapes, especially in scenarios where rotational invariance is crucial for data analysis and processing.

5.2.3 Geometric Patch based Structural Interdependence Function

Based on the preceding descriptions, we introduce the geometric patch-based structural interdependence function defined for input data instance $\mathbf{x} \in \mathbb{R}^m$ as follows:

$$\xi(\mathbf{x}) = \mathbf{A} \in \mathbb{R}^{m \times m'}. \tag{66}$$

As depicted in Plot (a) of Figure 13, the composition of the interdependence matrix $\mathbf{A}$ can be conceptualized as the systematic placement and translation of pre-defined patches along the dimensions of the data grid structure. In this representation, the surrounding nodes within each patch define the dependent conditions for the central node (highlighted in red) in the grid. This structural arrangement aligns with the two distinct operational modes of the interdependence function, namely the interdependence padding and interdependence aggregation modes, as elaborated in Section 3.4. These modes, while utilizing the same underlying patch structure, result in different compositions of the interdependence matrices $\mathbf{A}$, each capturing unique aspects of the data's structural interdependencies and leading to different learning performance.

Padding Mode: In the interdependence padding mode, the function composes matrix $\mathbf{A}$ as the concatenation of a sequence of block matrices:

$$\xi(\mathbf{x}) = \mathbf{A} = \left[\mathbf{A}_{(i,j,k)}\right]_{(i,j,k) \in grid(\mathbf{x}|h,w,d)} \in \mathbb{R}^{m \times m'}. \tag{67}$$

For each coordinate tuple $(i, j, k) \in grid(\mathbf{x}|h, w, d)$ in the underlying grid structure of instance vector $\mathbf{x}$, a block sub-matrix $\mathbf{A}_{(i,j,k)} \in \mathbb{R}^{m \times p}$ is defined. Specifically, the block sub-matrix $\mathbf{A}_{(i,j,k)}$ has p columns, each corresponding to one of the coordinate tuples in the $patch(i, j, k)$ centered at coordinate (i, j, k). Moreover, for the column of coordinate $(i', j', k') \in patch(i, j, k)$, all entries are filled with zeros except the entry with index $idx(i', j', k')$, which is filled with value 1.

Plot (b) in Figure 13 provides a visual representation of this structure. It illustrates the sparse block sub-matrix $\mathbf{A}_{(1,1,0)}$, corresponding to the patch centered at coordinate tuple $(1, 1, 0)$. In this visualization, entries with value 1 are prominently highlighted, while the remaining entries, all zeros, form the background of the matrix.

The concatenation of these sub-matrices along the column dimension composes the sparse matrix $\mathbf{A} \in \mathbb{R}^{m \times m'}$. The matrix dimension m' can be represented as $m' = (h \times w \times d) \times p = m \times p$, which is proportional to the sizes of both the grid and patches.

Aggregation Mode: In contrast, the interdependence matrix defined in the aggregation mode is considerably denser:

$$\xi(\mathbf{x}) = \mathbf{A} \in \mathbb{R}^{m \times m'}. \tag{68}$$

In the underlying grid structure of instance vector $\mathbf{x}$, each coordinate tuple $(i, j, k) \in grid(\mathbf{x}|h, w, d)$ corresponds to a specific column in matrix $\mathbf{A}$. This column is uniquely identified by the index $idx(i, j, k)$. Plot (c) of Figure 13 provides a visual representation of how the entries in this column are populated. The filling pattern is determined by the coordinates encompassed by the patch centered at (i, j, k), as follows:

$$\mathbf{A}\left(idx(i', j', k'), idx(i, j, k)\right) = \begin{cases} 1, & \text{if } (i', j', k') \in patch(i, j, k) \\ 0, & \text{otherwise} \end{cases}, \tag{69}$$

for all $(i, j, k) \in grid(\mathbf{x}|h, w, d)$. Furthermore, in this mode, the matrix dimension m' equals the size of the grid, *i.e.,* $m' = m$, and is independent of the patch size.

The interdependence function exhibits remarkable versatility, accommodating patches of diverse geometric shapes through its two distinct compositional modes for interdependence matrices. When applied to a given input data instance, variations in patch shape yield different manifestations of the $patch(i, j, k)$ notation, as formally expressed in Equations (63) and (65). These shape-induced variations subsequently engender distinct interdependence matrices, each capturing unique structural relationships within the data. The efficacy of different patch shapes, as well as the comparative performance of the function's two operational modes, will be rigorously assessed through empirical evaluation in the following Section 9.

5.2.4 Cuboid Geometric Patch Packing based Structural Interdependence Function

In geometry, *packing* and *covering* problems constitute a specialized category of optimization challenges pertaining to geometric objects within a defined space. These problems typically involve the arrangement of identical geometric objects to maximize density without overlap, while ensuring the largest possible spatial coverage. The structural interdependence functions introduced in this study, however, deviate from conventional packing problems. Our approach permits the overlapping of patches when positioning them within the data's underlying grid structure, while simultaneously aiming to achieve the largest possible comprehensive grid coverage to minimize information loss.

The geometric patch-based structural interdependence functions defined earlier enumerate all nodes in the grid as patch centers for composing the interdependence matrices, resulting in the densest possible packing. This configuration is illustrated in Plot (d) of Figure 14, which demonstrates the concept in a 2D grid. From a computational cost perspective, such dense packing significantly impacts the output dimension m' of these matrices, particularly in the padding mode where $m' = m \times p$, leading to considerably high dimensionality.

In this study, we introduce the concept of patch center distance hyper-parameters as a means to control the packing of patches that cover the grid structure. For cuboid patches, these parameters are

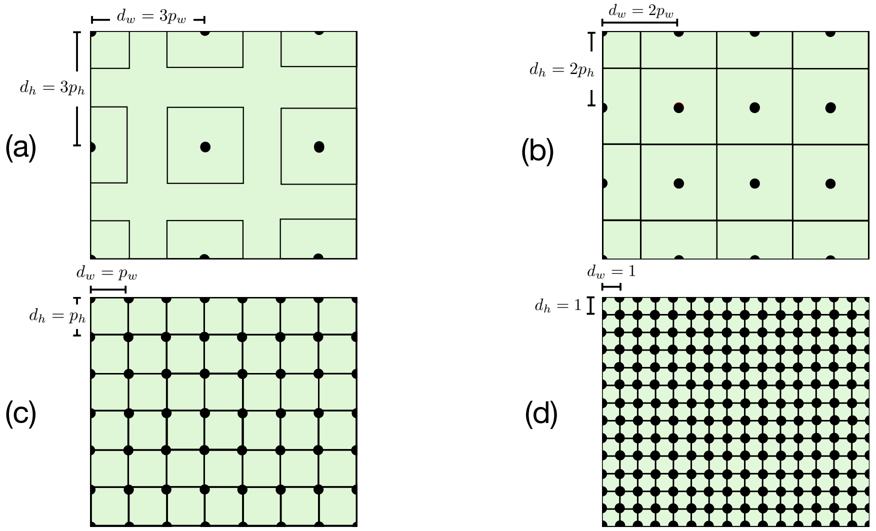


Figure 14: An illustration of cuboid geometric patch with shape parameters $p'_h = p_h$, $p'_w = p_w$ (and $p'_d = p_d$) packing with varying patch center distance parameters d_h, d_w (and d_d) in a 2D grid plane. (a) $d_h = 3p_h$, $d_w = 3p_w$: sparse packing with gaps between patches; (b) $d_h = 2p_h$, $d_w = 2p_w$: complete grid full coverage without patch overlaps; (c) $d_h = p_h$, $d_w = p_w$: complete grid coverage with full patch overlap between adjacent patches; and (d) $d_h = 1$, $d_w = 1$: densest packing configuration, where each grid node serves as the center of a patch. For the depth dimension that is not shown, the patch packing can be analyzed in a similar way.

denoted as d_h, d_w, and d_d, corresponding to the height, width, and depth dimensions, respectively. Figure 14, specifically Plots (a)-(c), provides a visual representation of cuboid patch packing in a two-dimensional example grid plane. These illustrations demonstrate the effect of varying center distance hyper-parameters on patches of identical shape, defined by patch sizes p_h and p_w. By modulating these center distance parameters, we can precisely control two critical aspects of patch arrangement: the extent of grid coverage and the degree of overlap between adjacent patches, which also determine the information coverage and redundancy of the packing results.

Formally, given a grid of shape (h, w, d) and patch center hyper-parameters d_h, d_w, and d_d, we can select the coordinates (i, j, k) as patch centers from the following set, initializing from the coordinate $(0, 0, 0)$:

$$\begin{aligned} i &\in \left\{0, d_h, 2 \cdot d_h, \cdots, \left\lfloor \frac{h}{d_h} \right\rfloor \cdot d_h \right\}, \\ j &\in \left\{0, d_w, 2 \cdot d_w, \cdots, \left\lfloor \frac{w}{d_w} \right\rfloor \cdot d_w \right\}, \\ k &\in \left\{0, d_d, 2 \cdot d_d, \cdots, \left\lfloor \frac{d}{d_d} \right\rfloor \cdot d_d \right\}. \end{aligned} \tag{70}$$

This selection method facilitates the packing of a set of patches within the grid, whose size can be represented as

$$p_{count} = \left(1 + \left\lfloor \frac{h}{d_h} \right\rfloor\right) \left(1 + \left\lfloor \frac{w}{d_w} \right\rfloor\right) \left(1 + \left\lfloor \frac{d}{d_d} \right\rfloor\right) \tag{71}$$

For patches that extend beyond the grid boundaries, a default value padding approach is employed. Specifically, dummy attribute values (such as 0) are used to pad these boundary-exceeding patches, ensuring uniformity in patch sizes across the entire grid structure.

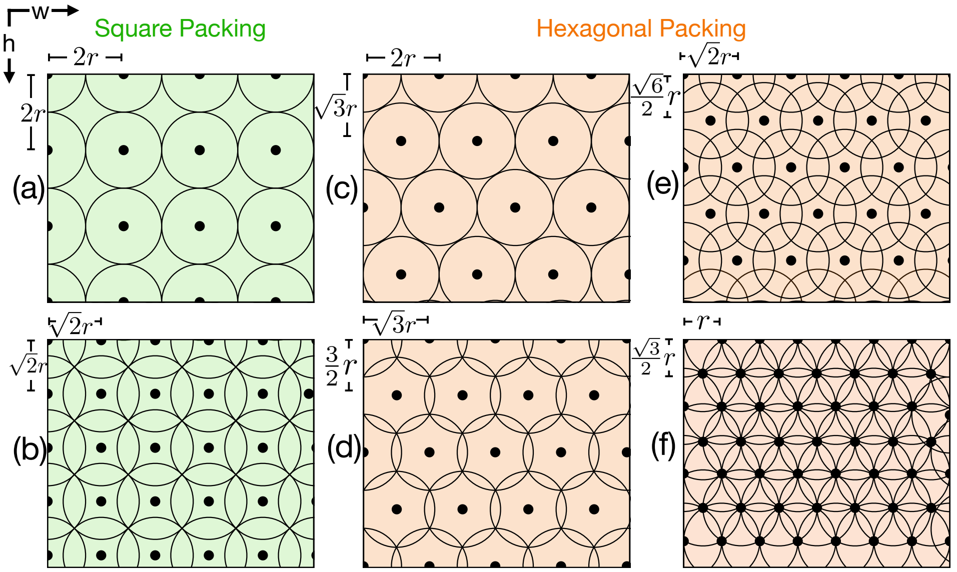


Figure 15: An illustration of of cylinder geometric patch packing with patch shape parameters r (and $p'_d = p_d$) and varying patch center distance parameters d_h, d_w (and d_d) in a 2D grid plane. The depth dimension is omitted for clarity. (a) $d_h = 2r$, $d_w = 2r$: sparse square packing with inter-patch gaps; (b) $d_h = \sqrt{2}r$, $d_w = \sqrt{2}r$: complete square packing; (c) $d_h = \sqrt{3}r$, $d_w = 2r$: sparse hexagonal packing with inter-patch gaps; (d) $d_h = \frac{3}{2}r$, $d_w = \sqrt{3}r$: complete hexagonal packing with minimal overlap; (e) $d_h = \frac{\sqrt{6}}{2}r$, $d_w = \sqrt{2}r$: complete hexagonal packing with increased overlap; (f) $d_h = \frac{\sqrt{2}}{2}r$, $d_w = r$: complete hexagonal packing with full overlap.

Using these selected patch centers, we can define the interdependence function as follows:

$$\xi(\mathbf{x}) = \mathbf{A} \in \mathbb{R}^{m \times m'}. \tag{72}$$

Depending on the working modes, the output dimension m' will be

$$m' = \begin{cases} p \times p_{count}, & \text{for the padding mode,} \\ p_{count}, & \text{for the aggregation mode.} \end{cases} \tag{73}$$

The structural interdependence function defined above is adaptable to patches of various shapes, including cylinder and sphere patches. In the following sections, we will examine the packing strategies for cylinder and sphere patches, with a focus on determining feasible patch center distance hyper-parameters d_h, d_w, and d_d. Both the packing strategy and the selected hyper-parameters play crucial roles in determining the patch center coordinates, the definition of the interdependence function, and the output dimension, as indicated by Equations (70)-(73).

5.2.5 Cylinder Geometric Patch Packing based Structural Interdependence Function

The packing of cylinder-shaped geometric patches within a cuboid grid presents greater challenges compared to the cuboid patches discussed previously. While the packing along the depth dimension remains relatively straightforward, adhering to the principles established for cuboid patches, the arrangement of the circular surface on the plane composed of the height and width dimensions proves more complex.

Figure 15 illustrates various packing strategies for these circular surfaces. These strategies can be categorized based on two primary criteria: the organization of patch centers and the coverage of the grid. Regarding the organization of patch centers, we observe square packings, depicted in Plots (a)-(b) in light green color, and hexagonal packings, shown in Plots (c)-(f) in light orange color. In terms of grid coverage, the strategies can be classified as sparse packings, exemplified in Plots (a) and (c), and complete packings, illustrated in Plots (b), (d)-(f). Each of these packing strategies offers unique characteristics and trade-offs between information coverage and redundancy.

Sparse Packing Strategies: The sparse packing strategies aim to maximize the coverage of the grid with packed patches, treating the patches as rigid bodies with no overlapping.

- **Square packing**: This strategy positions patch centers in rows and columns parallel to the height and width dimensions of the grid structure. As illustrated in Plot (a) of Figure 15, a square packing configuration with center distance hyper-parameters $d_h = 2r$ and $d_w = 2r$ (where r is the circular surface radius of the cylinder patch) results in adjacent circular surfaces of the patches. However, this arrangement cannot achieve complete surface coverage, with a packing coverage rate of approximately $\frac{\pi}{4} = 0.785$. This indicates that a significant number of data elements are neither used as central nor dependent attributes.

- **Hexagonal packing**: This approach employs a "zig-zag" arrangement of patch centers to minimize uncovered gaps and reduce information loss. Plot (c) of Figure 15 demonstrates a hexagonal packing configuration with stride parameters $d_h = \sqrt{3}r$ and $d_w = 2r$, which also results in adjacent patch placement. Notably, this method increases the packing coverage rate to $\frac{\pi}{2\sqrt{3}} = 0.907$, offering a substantial improvement over the square packing strategy and more effectively mitigating information loss during the packing process.

Complete Packing Strategies: The complete packing strategies aim to entirely cover the grid with packed patches, allowing for overlap among patches.

- **Square packing**: As illustrated in Plot (b) of Figure 15, reducing the patch center hyper-parameters to $d_h = \sqrt{2}r$ and $d_w = \sqrt{2}r$ results in overlapping circular surfaces that completely cover the grid surface. For each circle, the overlapping ratio with other circles is approximately $\frac{2\pi-4}{\pi} = 0.726$, indicating that 72.6% of the attributes covered by each cylinder patch are also involved in adjacent patches.

- **Hexagonal packing**: Plot (d) of Figure 15 demonstrates that reducing the center distance hyper-parameters to $d_h = \frac{3}{2}r$ and $d_w = \sqrt{3}r$ also achieves complete grid coverage. Compared to square packing, this configuration yields a lower overlapping ratio of approximately $\frac{2\pi-3\sqrt{3}}{\pi} = 0.345$ per circle. Plots (e) and (f) illustrate further reductions in parameters to $d_h = \frac{\sqrt{6}}{2}r$, $d_w = \sqrt{2}r$ and $d_h = \frac{\sqrt{2}}{2}r$, $d_w = r$, respectively, both resulting in substantially higher overlapping ratios among patches.

All the cylinder patch packing strategies discussed above have been implemented in the **TINYBIG v0.2.0** toolkit. Given the shape hyper-parameters r and p_d of the cylinder patch, the **TINYBIG v0.2.0** toolkit can automatically compute the feasible patch center coordinates from the data grid structure for composing the interdependence matrices.

5.2.6 Sphere Geometric Patch Packing based Structural Interdependence Function

Sphere patches offer superior rotational invariance compared to cylinder patches, maintaining this property for rotation axes in three-dimensional space. This characteristic is crucial for defining interdependence functions for data that require preservation of rotational invariance properties. However, the packing of sphere-shaped patches within a cuboid grid presents significantly greater challenges than cylinder patch packing. As illustrated in Figure 16, sphere patch packing strategies can be categorized into sparse and complete packing configurations, differentiated by their grid space coverage and patch overlapping ratios. These distinctions parallel those observed in cylinder patch packing,

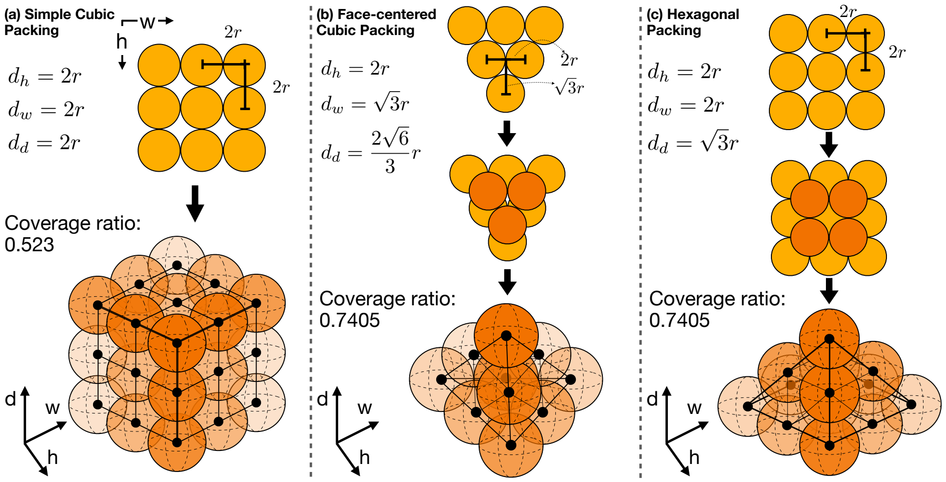


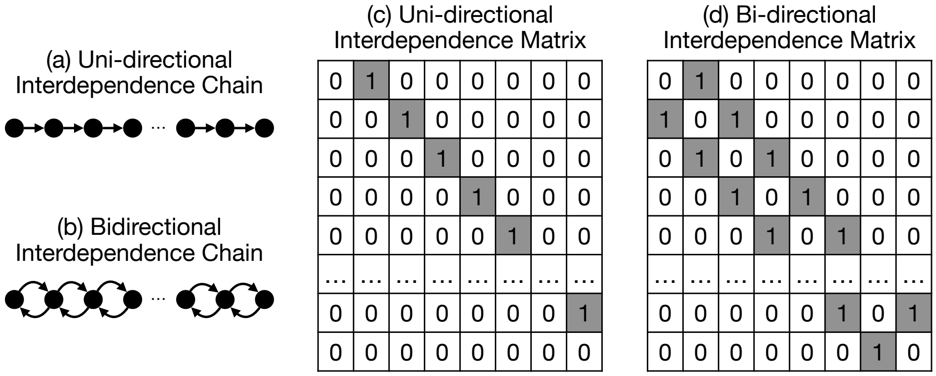
Figure 16: An illustration of sphere geometric patch packing with radius r in both 2D and 3D grid spaces, demonstrating varying patch center distance parameters d_h, d_w, and d_d. (a) $d_h = 2r, d_w = 2r, d_d = 2r$: Simple cubic sphere packing. (b) $d_h = \sqrt{3}r, d_w = 2r, d_d = \frac{2\sqrt{6}}{3}r$: Face-centered cubic sphere packing. (c) $d_h = 2r, d_w = 2r, d_d = \sqrt{3}r$: Hexagonal sphere packing.

yet the three-dimensional nature of spheres introduces additional complexities in achieving optimal spatial arrangements.

Sparse packing: Formally, given a sphere patch with radius r, the sparse patch packing strategy positions sphere patches adjacent to each other without overlapping, inevitably resulting in gaps between patches. Figure 16 illustrates three distinct sphere patch packing strategies examined in this paper:

- **Simple cubic packing**: This straightforward approach to packing sphere patches within a cuboid grid is illustrated in Plot (a) of Figure 16. With patch center distance hyper-parameters $d_h = 2r, d_w = 2r, d_d = 2r$, each sphere is stacked directly atop another, touching 6 other patches. The grid space coverage ratio for this simple cubic packing is approximately $\frac{\pi}{6} = 0.523$.
- **Face-centered cubic packing**: As shown in Plot (b) of Figure 16, this strategy places spheres diagonally adjacent to each other within each layer. Subsequent layers of spheres are positioned in the interstices between spheres of the underlying layer, with every third layer directly overlying its counterpart. This configuration, utilizing patch center distance hyper-parameters $d_h = \sqrt{3}r, d_w = 2r, d_d = \frac{2\sqrt{6}}{3}r$, enables each patch to contact 12 adjacent patches, yielding a coverage ratio of $\frac{\pi}{3\sqrt{2}} \approx 0.7405$.
- **Hexagonal packing**: Plot (c) of Figure 16 depicts this packing strategy, where the bottom layer consists of spheres in direct contact. Spheres in subsequent layers are situated in the interstices of the underlying layer. With hyper-parameters $d_h = 2r, d_w = 2r, d_d = \sqrt{3}r$, alternate layers directly overlie each other. Each patch contacts 12 neighboring patches, achieving a coverage ratio of approximately $\frac{\pi}{3\sqrt{2}} \approx 0.7405$, identical to that of face-centered cubic packing.

Complete packing: The sphere patch packing strategies introduced earlier can be adapted to achieve complete grid space coverage by reducing the patch center distance hyper-parameters d_h, d_w, d_d and allowing patch overlap. This approach gradually eliminates gaps between patches, ensuring comprehensive coverage of the grid space. Based on the sphere patch organizations illustrated in Figure 16, we propose the following potential stride parameters d_h, d_w, d_d to eliminate gaps and avoid information loss:


(a) Uni-directional Interdependence Chain
···
(b) Bidirectional Interdependence Chain
···

(c) Uni-directional Interdependence Matrix

0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
...	...	...	...	...	...	...	...
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

(d) Bi-directional Interdependence Matrix

0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0
0	0	1	0	1	0	0	0
0	0	0	1	0	1	0	0
...	...	...	...	...	...	...	...
0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0

Figure 17: An illustration of uni- and bi-directional chain-structured topological interdependence relationships. (a) Uni-directional chain-structured interdependence relationship. (b) Bi-directional chain-structured interdependence relationship. (c) Interdependence matrix of the uni-directional chain. (d) Interdependence matrix of the bi-directional chain.

- **Simple cubic packing**: Complete coverage is achieved by setting the patch center distance hyper-parameters to $d_h = d_w = d_d = \frac{2\sqrt{3}}{3}r$. This configuration ensures that all attributes are encompassed within sphere patches.
- **Face-centered cubic packing**: For this packing strategy, setting patch center distance parameters to $d_h = \sqrt{2}r$, $d_w = \frac{2\sqrt{6}}{3}r$, and $d_d = \frac{4}{3}r$ effectively eliminates space gaps between spheres.
- **Hexagonal packing**: Complete coverage in hexagonal packing is attained with patch center distance parameters $d_h = d_w = \frac{2\sqrt{3}}{3}r$ and $d_d = r$. This configuration ensures that every attribute in the data instance is included within at least one sphere patch.

All these sphere packing strategies have been implemented within the new **TINYBIG V0.2.0** toolkit, enabling direct application with the sphere patch packing-based structural interdependence function. This implementation facilitates the practical use of these advanced packing strategies in various data analysis tasks.

5.2.7 Chain based Structural Interdependence Function

In addition to the grid structures discussed earlier, as shown in Figure 11, many real-world datasets can be represented as chains, reflecting their underlying topological structure and sequential interdependence relationships. Chain-structured interdependence refers to a series of interconnected dependencies where each element in the chain relies on the previous one (or the later one), creating a linear sequence of relationships. Examples of data with chain-structured interdependence relationships include, but are not limited to natural languages, gene sequences, audio recordings and stock prices. The chain interdependence function can define the interdependence matrix for both features and data instances. We will use data attribute interdependence relationships to illustrate this concept.

Formally, given a data batch $\mathbf{x} \in \mathbb{R}^m$ with sequential chain-structured interdependence relationships among the attributes as illustrated in Plot (a) of Figure 17, we can define the corresponding unidirectional chain interdependence function as follows:

$$\xi(\mathbf{x}) = \mathbf{A} \in \mathbb{R}^{m \times m'}, \tag{74}$$

where $\mathbf{A}$ is the composed attribute interdependence matrix illustrated in Plot (c) of Figure 17. By default, the output dimension m' equals the input instance dimension, *i.e.*, $m' = m$.

In many cases, we sum this interdependence matrix with an identity matrix to denote self-dependency:

$$\xi(\mathbf{x}) = \mathbf{A} + \mathbf{I} \in \mathbb{R}^{m \times m}. \tag{75}$$

Here, $\mathbf{I} \in \mathbb{R}^{m \times m}$ is a square diagonal identity matrix of size $m \times m$, allowing the function to model both interdependence and self-dependence with a single dependency function. This self-dependence can also be defined using the linear remainder term in RPN 2, both of which contribute to defining sequential interdependence relationships.

For some input data batches, bidirectional chain-structured interdependence relationships may exist among the inputs. In this case, as illustrated in Plot (b) of Figure 17, each element in the chain relies on both the previous and subsequent elements. We can define a bidirectional chain interdependence function to compose the interdependence matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$.

As with the unidirectional case, this interdependence matrix can be summed with an identity matrix to represent self-dependency. The bidirectional interdependence matrix is particularly useful for modeling dependency relationships in sequence inputs, such as language data with complete context both before and after each token. This contextual information provides more comprehensive data for learning useful features of each token in the input language, potentially leading to more accurate learning results.

5.2.8 Multi-Hop Chain based Structural Interdependence Function

According to the interdependence matrix (both unidirectional and bidirectional), for the last attribute in the input data instance $\mathbf{x}$ to access information from the beginning ones it depends on, it may require the multiplication with the data instance vector $\mathbf{x}$ with the interdependence matrix $\mathbf{A}$ at least $m-1$ times:

$$\underbrace{\mathbf{xI}}_{\text{0-hop}}, \underbrace{\mathbf{xA}}_{\text{1-hop}}, \underbrace{\mathbf{xAA}}_{\text{2-hop}}, \cdots, \underbrace{\mathbf{xA}^{m-1}}_{\text{(m-1)-hop}}, \tag{76}$$

Such a step-wise information propagation along the chains denotes how current sequential models operate. However, this approach can be computationally expensive, especially for input data with extremely long chain-structured interdependence relationships. To reduce computational time and space costs, we introduce the multi-hop chain-based structural interdependence function:

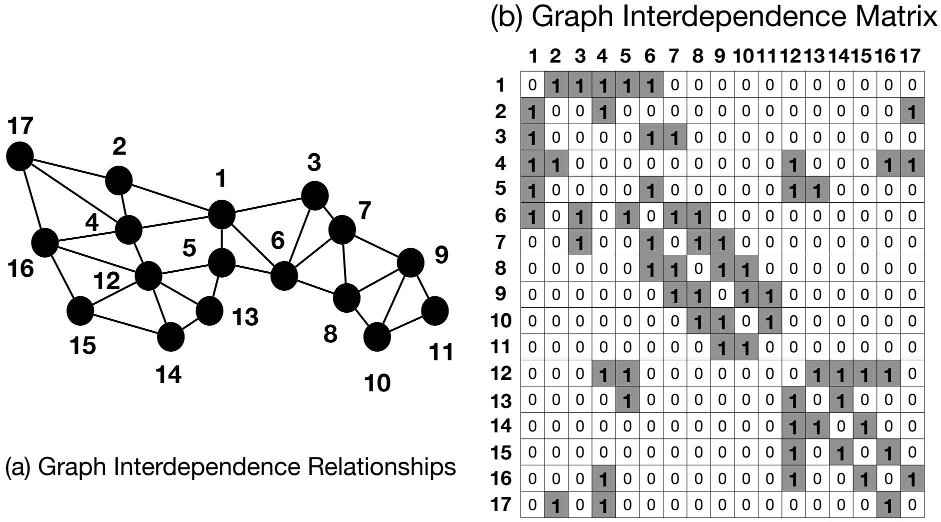
$$\xi(\mathbf{x}|h) = \mathbf{A}^h \in \mathbb{R}^{m \times m}, \text{for } h \in \{0, 1, 2, \cdots, m-1\}, \tag{77}$$

where h is the hop parameter for modeling multi-hop interdependence relationships of the input data instance along the chain.

With this interdependence matrix, each attribute in the data instance can directly access information from those h-hops away along the chain structure. To accumulate all data instances within h-hops, we introduce the accumulative multi-hop chain-based structural interdependence function as follows:

$$\xi(\mathbf{x}|0:h) = \mathbf{I} + \mathbf{A} + \mathbf{A}^2 + \mathbf{A}^3 + \cdots + \mathbf{A}^h = \sum_{i=0}^{h} \mathbf{A}^i \in \mathbb{R}^{m \times m}. \tag{78}$$

To further optimize the computations, we propose approximating the accumulative interdependence function using Taylor's polynomial expansion series. Considering the Taylor's expansions of the reciprocal function $\frac{1}{1-x}$ and the exponential function $\exp(x)$:



(a) Graph Interdependence Relationships

(b) Graph Interdependence Matrix

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
3	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
4	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1
5	1	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0
6	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0
7	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0
10	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
12	0	0	0	1	1	0	0	0	0	0	0	0	1	1	1	1	0
13	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0
15	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0
16	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1
17	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 18: An illustration of interdependence relationships in the graph topological structure and its corresponding structural interdependence matrix.

$$
\begin{aligned}
\frac{1}{1-x} &= 1 + x + x^2 + x^3 + \cdots = \sum_{h=0}^{\infty} x^h. \\
\exp(x) &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{h=0}^{\infty} \frac{x^h}{h!}.
\end{aligned}
\tag{79}
$$

Based on them, we define the reciprocal structural interdependence function and exponential structural interdependence function for approximating the above multi-hop chain-structured topological interdependence relationships as:

$$
\xi(\mathbf{x}) = (\mathbf{I} - \mathbf{A})^{-1} \in \mathbb{R}^{m \times m}, \text{ and } \xi(\mathbf{x}) = \exp(\mathbf{A}) \in \mathbb{R}^{m \times m}. \tag{80}
$$

The matrix $\mathbf{A}$, as defined above, is a nilpotent matrix, with its power becoming zero for exponents greater than $m - 1$ (*i.e.,* $\mathbf{A}^h = \mathbf{0}, \forall h > m - 1$). Consequently, the functions described do not introduce interdependence relationships beyond m hops. These calculations offer superior time and space efficiency compared to multi-hop based matrix power computations. The resulting interdependence matrix from these functions incorporates information from data instances within the input data batch. Notably, in the exponential function, data instances separated by h hops are subject to a penalization factor of $\frac{1}{h!}$, effectively weighting the influence of more distant relationships.

Meanwhile, for the reciprocal interdependence function, the matrix "$\mathbf{I} - \mathbf{A}$" will be singular when the chain is bi-directional, which can be calculated with the above accumulative interdependence Equation (78) with $h = m - 1$ instead. In the implementation of these interdependence functions in the toolkit, the chain interdependence matrix can be optionally normalized in the columns (or row for instance interdependence) to avoid dramatically high values for matrix entries in computation.

5.2.9 Graph based Structural Interdependence Function

In addition to chain based structural interdependence functions, some data structures exhibit more extensive interdependence relationships, such as graphs. Graph-structured interdependence relationships model complex dependencies between different features or data instances, where each element may depend on multiple other elements.

Given a data batch $\mathbf{x} \in \mathbb{R}^m$ with extensively interdependent attributes, we can represent the interdependence relationships as a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{0, 1, \cdots, m - 1\}$ is the node set and

$\mathcal{E} = \{(i,j)\}_{i,j \in \{0,1,\cdots,m-1\}}$ is the link set. For simplicity, we use the data instance's row index $i \in \{0, 1, \cdots, m-1\}$ to represent its corresponding node in the graph structure. The Plot (a) of Figure 18 illustrates an example of such graph-structured interdependence relationships among the attributes. In this example, these attributes are represented by the graph nodes, while their interdependence relationships are represented by the links connecting them.

Based on the graph structure, we define the graph interdependence function as:

$$\xi(\mathbf{x}|G) = \mathbf{A} \in \mathbb{R}^{m \times m'}, \tag{81}$$

where the output dimension $m' = m$ by default.

In the output binary interdependence matrix $\mathbf{A}$, the entry $\mathbf{A}(i,j)$ is filled with value 1 if and only if the node pair $(i,j) \in \mathcal{E}$ is a link in the graph. In other words, the interdependence matrix $\mathbf{A}$ represents the adjacency matrix of the graph G. For the graph example discussed above, we illustrate its interdependence matrix at Plot (b) of Figure 18.

5.2.10 Graph PageRank based Structural Interdependence Function

Similar to chain-structured interdependence relationships, multiplying the graph interdependence matrix with the data batch allows each instance to access information from its immediate neighbors (1-hop away). To retrieve information from neighbors that are h-hops away, we may need to multiply the interdependence matrix h times with the data batch:

$$\underbrace{\mathbf{xI}}_{\text{0-hop}}, \underbrace{\mathbf{xA}}_{\text{1-hop}}, \underbrace{\mathbf{xAA}}_{\text{2-hop}}, \cdots, \underbrace{\mathbf{xA}^h}_{\text{h-hop}}. \tag{82}$$

To model multi-hop dependency relationships among data instances, we introduce the multi-hop graph interdependence function and the accumulative multi-hop graph interdependence function as follows:

$$\xi(\mathbf{x}|h) = \mathbf{A}^h \in \mathbb{R}^{m \times m}, \text{ and } \xi(\mathbf{x}|0:h) = \sum_{i=0}^{h} \mathbf{A}^i \in \mathbb{R}^{m \times m}. \tag{83}$$

In addition to these formulas that calculate powers of matrix $\mathbf{A}$, existing graph studies also offer other approaches to calculate long-distance dependency relationships among data instances, such as the PageRank algorithm. Without delving into the step-wise derivation of PageRank updating equations, we define the PageRank multi-hop graph interdependence function using the convergence matrix representation from [91].

$$\xi(\mathbf{x}) = \alpha \cdot \left(\mathbf{I} - (1-\alpha) \cdot \mathbf{A}\right)^{-1} \in \mathbb{R}^{m \times m}. \tag{84}$$

Here, $\alpha \in [0,1]$ is a hyper-parameter of the function, typically set to 0.15 by default. Usually, matrix $\mathbf{A}$ is normalized before being used in this formula. Several frequently used matrix normalization approaches (*e.g., degree-based normalization*, *mean-std-based normalization*, and *softmax-based normalization*) have been implemented in the **TINYBIG V0.2.0** toolkit, which can be directly applied in the above interdependence function definition for input graph structures.

5.3 Hybrid Interdependence Functions

For the interdependence functions defined above, we also allow users to fuse them together to define more complex interdependence functions, which may be required for some special learning needs. For instance, for some input data, both the instance attributes and the underlying modality specific structures play a critical role in defining the interdependence relationships among the attributes (or instances), a fusion of the interdependence functions defined based on them will become necessary.

In this part, we will discuss the hybrid interdependence functions that can integrate the above interdependence functions defined based on different information of the input data batch together for modeling the diverse interdependence relationships. Moreover, compared with the multi-channel and multi-head architecture discussed before in Section 4.4, the hybrid interdependence function fusion provides a lightweight and more flexible implementation to achieve similar purposes. Also, even within the RPN 2 model with one single head and channel, the hybrid interdependence function still allows the model to define and model complex interdependence relationships.

5.3.1 Hybrid Interdependence Function Representation

Formally, given the input data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$, we can define a set of data and structural interdependence functions $\xi_1, \xi_2, \cdots, \xi_k : \mathbb{R}^{b \times m} \to \mathbb{R}^{m \times m'}$ to measure the interdependence relationships among the attributes. These functions can be effectively fused together as follows:

$$\begin{aligned} \xi(\mathbf{X}) &= \text{fusion}\left(\xi_1(\mathbf{X}), \xi_2(\mathbf{X}), \cdots, \xi_k(\mathbf{X})\right) \\ &= \text{fusion}\left(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_k\right) \\ &= \mathbf{A} \in \mathbb{R}^{m \times m'}, \end{aligned} \tag{85}$$

where $\mathbf{A}_i = \xi_i(\mathbf{X})$ denotes the interdependence matrix obtained by function $\xi_i, \forall i \in \{1, 2, \cdots, k\}$. Different fusion strategies can be used to define the fusion$(\cdot)$ operator used above, which will be introduced in the following subsection specifically.

The hybrid interdependence function defined above is particularly valuable for modeling complex relationships that incorporate information from multiple sources, such as data batches and underlying topological structures. To demonstrate the application of hybrid interdependence functions, we will use graph data as an example below, explaining how to define an interdependence function for data batches with underlying graph structures. Similar hybrid functions can also be applied to data batches with underlying chain and grid structures.

5.3.2 An Example: Hybrid Graph Interdependence Function

Formally, given an input data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$ and its corresponding graph structure G, we define a hybrid interdependence function by combining the parameterized bilinear interdependence function with a one-hop graph interdependence function, as follows:

$$\xi(\mathbf{X}, G) = \text{fusion}(\mathbf{A}_g, \mathbf{A}_b) = \mathbf{A}_g \circ \mathbf{A}_b \in \mathbb{R}^{m \times m}, \tag{86}$$

where the notations

$$\begin{aligned} \text{Graph Interdependence: } \mathbf{A}_g &= \xi_g(\mathbf{X}|G) \in \mathbb{R}^{m \times m}, \\ \text{Bilinear Interdependence: } \mathbf{A}_b &= \xi_b(\mathbf{X}|\mathbf{w}) = \mathbf{X}^\top \mathbf{W} \mathbf{X} \in \mathbb{R}^{m \times m}, \end{aligned} \tag{87}$$

denote the attribute interdependence matrices derived from the graph structure and data batch, respectively. If we use the one-hop graph interdependence function described in Section 5.2.9, the resulting graph interdependence matrix $\mathbf{A}_g$ serves as a binary mask, preserving the bilinear interdependence scores between nodes with direct connections. Assisted with the optional post-processing functions (such as softmax based interdependence matrix column normalizations), this hybrid interdependence function endows the RPN 2 model with capabilities akin to those of the Graph Attention Network (GAT) [76], which uses a linear attention mechanism, and Graph-BERT [91], which utilizes a transformer-based attention mechanism.

The hybrid interdependence function offers significant flexibility in defining interdependence relationships between attributes and instances, using either identical or distinct input information. For complex datasets with diverse interdependence patterns, this approach enhances RPN 2's modeling capacity and performance robustness. The effectiveness of this hybrid interdependence will be further examined through experiments on real-world benchmark datasets in Section 9.

The fusion function, denoted as $\text{fusion}(\cdot)$ above, plays a critical role in defining hybrid interdependence by determining how different interdependence strategies are combined. This same function can also be used for effective integration of the outputs from multi-channel and multi-head components in our architecture. We will discuss these fusion functions in detail in the following section.

6 Data Compression Function and Fusion Function

Beyond interdependence functions, we also expand the scope of data transformation capabilities by presenting a new family of data compression functions. Together with previously introduced data expansion functions [89], they will define a comprehensive set of methods for transforming data across vector spaces. Additionally, to integrate outputs learned from different heads and channels, we propose a family of fusion functions based on various numerical and statistical metrics and aggregation techniques, which will be utilized in the wide architecture of the RPN 2 model.

Readers seeking for a concise review of these component functions may also refer to Figure 19, which provides a summary of the functions to be introduced in this and the following sections.

6.1 Data Compression Functions

In addition to the aforementioned data interdependence functions, which are defined based on either the input data batch or the underlying structural information, we introduce a new family of component functions in this paper: the *data compression function*. This new data compression function, together with the data expansion functions introduced in our previous work [89], will comprise the family of data transformation functions.

As briefly mentioned in Section 4.1, the data transformation function $\kappa : \mathbb{R}^m \to \mathbb{R}^D$ projects data instances from a space of dimension m to a target space of dimension D. Depending on the relative sizes of m and D, this function can be further categorized into two types: *data expansion functions* ($D \geq m$) and *data compression functions* ($D \leq m$). We have introduced various data expansion functions in the previous article [89], and readers are encouraged to refer to that paper for more information before proceeding with this subsection. Several new data expansion functions will also be introduced in the following subsection, which expands data instances with orthogonal polynomials and wavelets.

In this subsection, we focus on introducing the data compression functions that can be incorporated into the new RPN 2 model. Since the data interdependence relationships have been adequately addressed by the previously introduced data interdependence functions, we will treat all data instances and attributes as independent by default for the compression functions discussed below.

6.1.1 Identity and Reciprocal Compression Function

The identity and reciprocal functions, previously introduced in [89], can serve as both expansion and compression functions. For a given data instance $\mathbf{x} \in \mathbb{R}^m$, we can represent the identity and reciprocal data compression functions as follows:

$$\kappa(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^d, \text{ and } \kappa(\mathbf{x}) = \frac{1}{\mathbf{x}} \in \mathbb{R}^d. \tag{88}$$

To distinguish these from data expansion functions, we use the lower-case notation d to represent the compression space dimension. As with expansion functions, optional pre- and post-processing functions (*e.g.,* activation or normalization functions) can be applied to the data instance before and

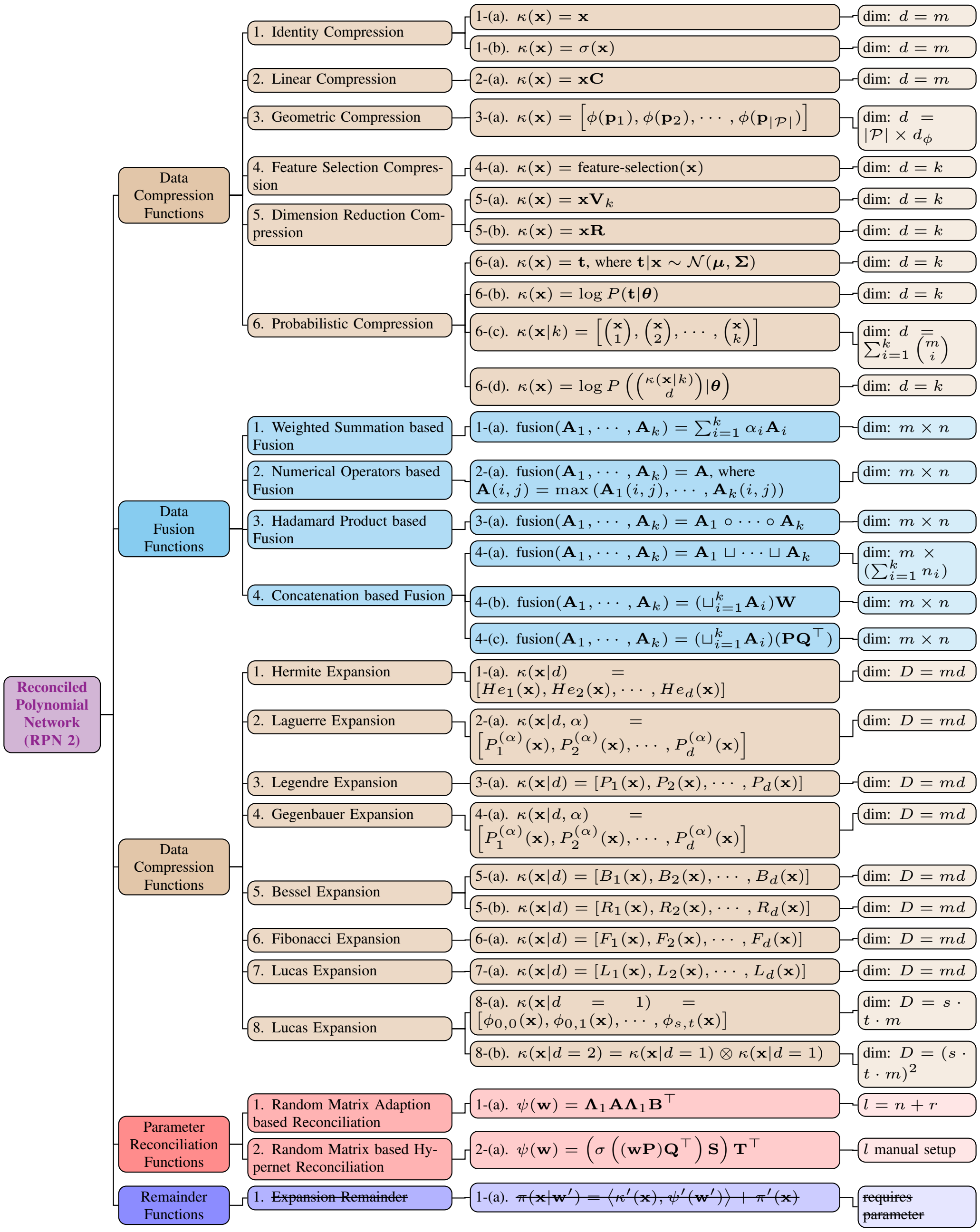

Reconciled Polynomial Network (RPN 2)
Data Compression Functions
1. Identity Compression
1-(a). $\kappa(\mathbf{x}) = \mathbf{x}$
dim: $d = m$
1-(b). $\kappa(\mathbf{x}) = \sigma(\mathbf{x})$
dim: $d = m$
2. Linear Compression
2-(a). $\kappa(\mathbf{x}) = \mathbf{x}\mathbf{C}$
dim: $d = m$
3. Geometric Compression
3-(a). $\kappa(\mathbf{x}) = \left[\phi(\mathbf{p}_1), \phi(\mathbf{p}_2), \cdots, \phi(\mathbf{p}_{|\mathcal{P}|})\right]$
dim: $d = |\mathcal{P}| \times d_{\phi}$
4. Feature Selection Compression
4-(a). $\kappa(\mathbf{x}) =$ feature-selection$(\mathbf{x})$
dim: $d = k$
5. Dimension Reduction Compression
5-(a). $\kappa(\mathbf{x}) = \mathbf{x}\mathbf{V}_k$
dim: $d = k$
5-(b). $\kappa(\mathbf{x}) = \mathbf{x}\mathbf{R}$
dim: $d = k$
6. Probabilistic Compression
6-(a). $\kappa(\mathbf{x}) = \mathbf{t}$, where $\mathbf{t}|\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
dim: $d = k$
6-(b). $\kappa(\mathbf{x}) = \log P(\mathbf{t}|\boldsymbol{\theta})$
dim: $d = k$
6-(c). $\kappa(\mathbf{x}|k) = \left[\binom{\mathbf{x}}{1}, \binom{\mathbf{x}}{2}, \cdots, \binom{\mathbf{x}}{k}\right]$
dim: $d = \sum_{i=1}^{k} \binom{m}{i}$
6-(d). $\kappa(\mathbf{x}) = \log P\left(\binom{\kappa(\mathbf{x}|k)}{d} |\boldsymbol{\theta}\right)$
dim: $d = k$
Data Fusion Functions
1. Weighted Summation based Fusion
1-(a). fusion$(\mathbf{A}_1, \cdots, \mathbf{A}_k) = \sum_{i=1}^{k} \alpha_i \mathbf{A}_i$
dim: $m \times n$
2. Numerical Operators based Fusion
2-(a). fusion$(\mathbf{A}_1, \cdots, \mathbf{A}_k) = \mathbf{A}$, where $\mathbf{A}(i,j) = \max(\mathbf{A}_1(i,j), \cdots, \mathbf{A}_k(i,j))$
dim: $m \times n$
3. Hadamard Product based Fusion
3-(a). fusion$(\mathbf{A}_1, \cdots, \mathbf{A}_k) = \mathbf{A}_1 \circ \cdots \circ \mathbf{A}_k$
dim: $m \times n$
4. Concatenation based Fusion
4-(a). fusion$(\mathbf{A}_1, \cdots, \mathbf{A}_k) = \mathbf{A}_1 \sqcup \cdots \sqcup \mathbf{A}_k$
dim: $m \times (\sum_{i=1}^{k} n_i)$
4-(b). fusion$(\mathbf{A}_1, \cdots, \mathbf{A}_k) = (\sqcup_{i=1}^{k} \mathbf{A}_i)\mathbf{W}$
dim: $m \times n$
4-(c). fusion$(\mathbf{A}_1, \cdots, \mathbf{A}_k) = (\sqcup_{i=1}^{k} \mathbf{A}_i)(\mathbf{P}\mathbf{Q}^{\top})$
dim: $m \times n$
Data Compression Functions
1. Hermite Expansion
1-(a). $\kappa(\mathbf{x}|d) = [He_1(\mathbf{x}), He_2(\mathbf{x}), \cdots, He_d(\mathbf{x})]$
dim: $D = md$
2. Laguerre Expansion
2-(a). $\kappa(\mathbf{x}|d, \alpha) = \left[P_1^{(\alpha)}(\mathbf{x}), P_2^{(\alpha)}(\mathbf{x}), \cdots, P_d^{(\alpha)}(\mathbf{x})\right]$
dim: $D = md$
3. Legendre Expansion
3-(a). $\kappa(\mathbf{x}|d) = [P_1(\mathbf{x}), P_2(\mathbf{x}), \cdots, P_d(\mathbf{x})]$
dim: $D = md$
4. Gegenbauer Expansion
4-(a). $\kappa(\mathbf{x}|d, \alpha) = \left[P_1^{(\alpha)}(\mathbf{x}), P_2^{(\alpha)}(\mathbf{x}), \cdots, P_d^{(\alpha)}(\mathbf{x})\right]$
dim: $D = md$
5. Bessel Expansion
5-(a). $\kappa(\mathbf{x}|d) = [B_1(\mathbf{x}), B_2(\mathbf{x}), \cdots, B_d(\mathbf{x})]$
dim: $D = md$
5-(b). $\kappa(\mathbf{x}|d) = [R_1(\mathbf{x}), R_2(\mathbf{x}), \cdots, R_d(\mathbf{x})]$
dim: $D = md$
6. Fibonacci Expansion
6-(a). $\kappa(\mathbf{x}|d) = [F_1(\mathbf{x}), F_2(\mathbf{x}), \cdots, F_d(\mathbf{x})]$
dim: $D = md$
7. Lucas Expansion
7-(a). $\kappa(\mathbf{x}|d) = [L_1(\mathbf{x}), L_2(\mathbf{x}), \cdots, L_d(\mathbf{x})]$
dim: $D = md$
8. Lucas Expansion
8-(a). $\kappa(\mathbf{x}|d = 1) = [\phi_{0,0}(\mathbf{x}), \phi_{0,1}(\mathbf{x}), \cdots, \phi_{s,t}(\mathbf{x})]$
dim: $D = s \cdot t \cdot m$
8-(b). $\kappa(\mathbf{x}|d = 2) = \kappa(\mathbf{x}|d = 1) \otimes \kappa(\mathbf{x}|d = 1)$
dim: $D = (s \cdot t \cdot m)^2$
Parameter Reconciliation Functions
1. Random Matrix Adaption based Reconciliation
1-(a). $\psi(\mathbf{w}) = \boldsymbol{\Lambda}_1 \mathbf{A} \boldsymbol{\Lambda}_1 \mathbf{B}^{\top}$
$l = n + r$
2. Random Matrix based Hypernet Reconciliation
2-(a). $\psi(\mathbf{w}) = \left(\sigma\left((\mathbf{w}\mathbf{P})\mathbf{Q}^{\top}\right)\mathbf{S}\right)\mathbf{T}^{\top}$
l manual setup
Remainder Functions
1. ~~Expansion Remainder~~
1-(a). ~~$\pi(\mathbf{x}|\mathbf{w}') = \langle \kappa'(\mathbf{x}), \psi'(\mathbf{w}') \rangle + \pi'(\mathbf{x})$~~
~~requires parameter~~

Figure 19: An overview of the newly added data compression and fusion functions, and the updated data expansion, parameter reconciliation, and remainder functions implemented in the **TINYBIG v0.2.0** toolkit for constructing the RPN 2 model architecture. The cross-line added to the "Expansion Remainder" denotes the function has been deleted from the toolkit.

after the compression function. It's worth noting that in these cases, the output dimension equals the input dimension, *i.e.*, $d = m$.

6.1.2 Linear Compression Function

We can also define a linear data compression function based on a pre-defined constant matrix $\mathbf{C} \in \mathbb{R}^{m \times d}$. This function compresses a data instance from the input space of dimension m to the compression space of dimension d as follows:

$$\kappa(\mathbf{x}) = \mathbf{x}\mathbf{C} \in \mathbb{R}^{d}. \tag{89}$$

It's worth noting that this function was also defined as a data expansion function in our previous work [89]. The linear transformation matrix $\mathbf{C}$ is pre-computed and provided as an input at the function definition. Additionally, the compression space dimension d is a manually set hyper-parameter.

This linear compression function can be particularly effective when dealing with sparse input data. By reducing the vector dimensions, it helps decrease both the model space, time and learning costs, often with minimal information loss.

6.1.3 Geometric Patch based Compression Function

In Section 5.2, we introduced several approaches to obtain geometric patches from the underlying grid structures of the input data batch. These methods can also be utilized to define data compression functions.

Formally, given a data instance $\mathbf{x}$ and its underlying grid structure, we can extract a set of patches denoted as $\mathcal{P} = \{p_1, p_2, \cdots p_{|\mathcal{P}|}\}$. The size of this patch set (*i.e.,* $|\mathcal{P}|$) is determined by three factors: the original size of $\mathbf{x}$, the patch shapes, and the patch center distance hyper-parameters used in the selected packing strategies. Each patch $p_i \in \mathcal{P}$ is represented as an ordered list of coordinates covered by the patch in the data grid structure. These coordinates can be used to retrieve the corresponding attribute values from the data instance vector $\mathbf{x}$ via the index-coordinate bijection introduced in the previous Section 5.2.2.

For simplicity, we use the notation $\mathbf{p}_i = \mathbf{x}(p_i) \in \mathbb{R}^p$ to represent the attribute elements covered by patch $p_i \in \mathcal{P}$ from the input data instance vector $\mathbf{x}$, where p denotes the patch size as introduced in Section 5.2.2. In this paper, we propose to compress the patch vector $\mathbf{p}_i$ using a mapping $\phi : \mathbb{R}^p \to \mathbb{R}^{d_\phi}$, which transforms it into a dense representation of length d_ϕ. This mapping defines our patch-based data compression function as follows:

$$\kappa(\mathbf{x}) = \left[\phi(\mathbf{p}_1), \phi(\mathbf{p}_2), \cdots, \phi(\mathbf{p}_{|\mathcal{P}|})\right] \in \mathbb{R}^{d}. \tag{90}$$

In the above compression function, its compression output vector dimension is $d = |\mathcal{P}| \times d_\phi$. The dimension parameter d_ϕ must be manually specified when defining the patch vector compression mapping ϕ. For the majority of mappings ϕ studied in this paper, the output is typically a scalar, *i.e.,* the dimension $d_\phi = 1$.

In practice, there are various ways to define the patch vector compression mapping ϕ. We illustrate several of these approaches as follows:

(a) Vector Norms:

$$\phi(\mathbf{p}) = \|\mathbf{p}\|_p \in \mathbb{R}, \tag{91}$$

where $p \in \{1, 2, \cdots, \infty\}$.

(b) Entropy:

$$\phi(\mathbf{p}) = -\sum_i (\mathbf{p}(i) \log \mathbf{p}(i)) \in \mathbb{R}, \tag{92}$$

where vector $\mathbf{p}$ needs to be positive.

(c) Statistical Metrics:

$$\phi(\mathbf{p}) = \text{metric}(\mathbf{p}) \in \mathbb{R}. \tag{93}$$

(d) Numerical Operator:

$$\phi(\mathbf{p}) = \text{operator}(\mathbf{p}) \in \mathbb{R}. \tag{94}$$

For the vector norm-based patch compression mapping, the L_p norm can be selected with p values from the set $\{1, 2, \cdots, \infty\}$. In the case of statistical metric-based patch compression functions, we employ the general function notation "metric($\cdot$)" to represent various statistical measures, including but not limited to *variance*, *standard deviation*, and *skewness*. For numerical operator-based patch compression functions, the "operator($\cdot$)" notation encompasses different numerical operators, such as *maximum*, *minimum*, *sum*, and *product*, as well as various averaging functions like *arithmetic mean*, *geometric mean*, *harmonic mean*, *median*, and *mode*. All of these diverse patch compression mappings have been implemented in the TINYBIG V0.2.0 toolkit.

6.1.4 Feature Selection based Compression Function

Several conventional machine learning techniques can also be employed to define data compression functions, notably *feature selection* and *dimension reduction*. *Feature selection* is a process in machine learning and data analysis that aims to choose a subset of relevant features from a larger set of available features for model construction. *Dimensionality reduction* techniques, on the other hand, seek to reduce the number of features or variables in a dataset while preserving as much important information as possible. In classic machine learning models, both feature selection and dimension reduction offer numerous advantages, such as improving model performance, reducing overfitting, simplifying models for easier interpretation, and decreasing training time.

Unlike the aforementioned compression functions, which can be applied directly to any input data instances or mini-batches, most existing feature selection and dimensionality reduction methods are primarily designed as pre-processing steps for static inputs on complete datasets. However, the multi-layered RPN 2 model involves dynamic inputs at each layer, where input and output values continuously change as new model parameters are learned. Consequently, most of these conventional static feature selection and dimensionality reduction methods can hardly be incorporated into the layers or adapted to such dynamic learning settings.

In this paper, we propose utilizing several incremental feature selection and dimension reduction approaches for defining the data compression functions at each layer. Simultaneously, we call for the development of novel online (or incremental) feature selection and dimension reduction methods from the research community, specifically those that can be adapted into the layers for deep function learning tasks.

Formally, given an input data instance $\mathbf{x} \in \mathbb{R}^m$, we can represent the feature selection-based data compression function as follows:

$$\kappa(\mathbf{x}) = \text{feature-selection}(\mathbf{x}) \in \mathbb{R}^d. \tag{95}$$

Here, the function "feature-selection($\cdot$)" represents an incremental feature selection approach capable of processing any input data instance $\mathbf{x}$ during the learning process. The output dimension d must be manually specified when defining this function. Several examples of such approaches, including the *incremental variance threshold* method and *incremental feature clustering* method, will be introduced in detail in the following sections.

Incremental Variance Threshold Method: Slightly different from the notations used in the above compression functions, the incremental variance threshold method needs to calculate the variance for each attribute based on the inputs, which can only operate on the data batches instead of the individual instances discussed above.

Formally, given an input data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$, we calculate its attribute variance vector as follows:

$$var(\mathbf{X}) = \mathbf{v} \in \mathbb{R}^m. \tag{96}$$

The i_{th} entry of vector $\mathbf{v}$ denotes the variance calculated for the i_{th} attribute of data batch $\mathbf{X}$, *i.e.,* the column vector $\mathbf{X}(:, i)$.

For small-sized input data batches, the calculated variance vector may change dramatically between batches, potentially leading to unstable performance. To address this issue, the incremental variance threshold method maintains a record of all historical variance vectors calculated since the first batch. This record is incrementally updated with new variance vectors as follows:

$$\bar{\mathbf{v}} = \frac{(t-1) \cdot \bar{\mathbf{v}} + \mathbf{v}}{t} \in \mathbb{R}^m, \tag{97}$$

where t denotes the counter index of the current batch and $\bar{\mathbf{v}}$ represents the historical variance record vector for all attributes. Furthermore, during both training and inference, attributes with variance values greater than a specified threshold are selected for the output:

$$\text{feature-selection}(\mathbf{X}) = \left[\mathbf{X}(:, i)\right]_{i \in \{1, 2, \cdots, m\} \wedge \bar{\mathbf{v}}(i) \geq p}, \tag{98}$$

where p denotes the provided variance threshold hyper-parameter.

In practice, rather than setting the variance threshold hyper-parameter p (which may lead to varying numbers of selected attributes and cause dimension inconsistency between different data batches and training epochs), it is more common to select the top-k attributes with the largest variances. This approach precisely determines the output dimension as $d = k$.

Incremental Feature Clustering Method: For the incremental feature clustering-based method, given an input data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$, the algorithm partitions the m attributes into k non-overlapping clusters based on pairwise similarities (*e.g.,* cosine similarity) calculated from the input data batches. Formally, we can represent these attribute clusters as:

$$\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}, \text{ where } \bigcup_i \mathcal{C}_i = \{1, 2, \cdots, m\} \wedge \mathcal{C}_i \cap \mathcal{C}_j = \emptyset, \forall i, j \in \{1, 2, \cdots, k\}. \tag{99}$$

As new data batches arrive, this method incrementally updates the similarity scores among the attributes (similar to Equation (97)), creating new partitions of the attributes. Within each identified attribute cluster, the most representative features are selected, typically those with the highest variance scores (or based on other selection criteria). This process can be represented as:

$$\text{feature-selection}(\mathbf{X}) = \left[\mathbf{X}(:, i)\right]_{i \in \mathcal{C}_j \wedge \mathcal{C}_j \in \mathcal{C} \wedge \bar{\mathbf{v}}(i) \geq p}, \tag{100}$$

where p is a threshold parameter and it can be set as the highest variance scores for each cluster (*e.g.,* the top-1 attribute with the largest variance in each cluster). Consequently, for the incremental feature clustering-based compression function, the output compression space dimension equals the provided cluster number hyper-parameter, *i.e.,* $d = k$.

For most input data batches, the number of attributes is typically not very large, especially compared to the number of instances. Therefore, recording attribute variances and learning attribute clusters does not incur significant space or time costs. Moreover, as the RPN 2 model's performance converges during the learning process, changes in the input data batches for each layer between sequential epochs become minor, resulting in minimal changes to attribute variances or clusters.

To further optimize the feature selection-based compression function, we can implement an early-stopping parameter. This parameter halts the updating of variance metrics and clustering results after a specified number of training epochs, helping to reduce the computational costs associated with tuning these incremental feature selection methods.

6.1.5 Dimension Reduction based Compression Function

Similar to feature selection, dimension reduction is another frequently used technique in classic machine learning for transforming high-dimensional data into a lower-dimensional space. In this paper, we propose the use of two dimensional reduction methods to define the data compression function, including the *incremental principal component analysis* and *random projection*.

Incremental PCA: Incremental Principal Component Analysis (PCA) [66] is a technique used for dimensionality reduction when dealing with large or continuously growing datasets. It extends traditional PCA to handle data incrementally by updating the principal components as new data arrives, rather than recomputing them from scratch.

Formally, given the input data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$, its singular value decomposition (SVD) can be represented as:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top}, \tag{101}$$

where $\mathbf{U} \in \mathbb{R}^{b \times b}$ and $\mathbf{V} \in \mathbb{R}^{m \times m}$ are both orthogonal matrices. The matrix $\mathbf{\Sigma} = diag(\sigma_1, \sigma_2, \cdots, \sigma_r, 0, \cdots, 0) \in \mathbb{R}^{b \times m}$ is a rectangular diagonal matrix with r singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ and zeros on the diagonal. The number of non-zero singular values on the diagonal of matrix $\mathbf{\Sigma}$ also defines the rank of matrix $\mathbf{X}$.

The column vectors of matrix $\mathbf{U}$ are orthogonal and called the *left singular vectors*, while the orthogonal column vectors of matrix $\mathbf{V}$ are called the *right singular vectors*. In the right singular matrix $\mathbf{V}$, we denote the columns corresponding to the k largest singular values as $\mathbf{V}_k \in \mathbb{R}^{m \times k}$. This helps calculate the principal components for each data instance $\mathbf{x} \in \mathbb{R}^m$ in the data batch as:

$$\kappa(\mathbf{x}) = \mathbf{x}\mathbf{V}_k \in \mathbb{R}^d, \tag{102}$$

where the instance output dimension $d = k$.

As a new data batch $\mathbf{X}' \in \mathbb{R}^{b \times m}$ arrives, we need to efficiently calculate the SVD of the concatenation of $\mathbf{X}$ and $\mathbf{X}'$. The resulting $\mathbf{V}'$ matrix will help update the principal components from the new data batch $\mathbf{X}'$. In [66], the authors introduce an efficient approach to update the previous SVD decomposition results (*i.e.,* $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$ of $\mathbf{X}$) to calculate the new SVD decomposition matrices $\mathbf{U}'$, $\mathbf{\Sigma}'$, $\mathbf{V}'$ by incorporating the new data batch $\mathbf{X}'$. In this paper, we will use this method of incremental data batch dimension reduction for defining the data compression function.

Random Projection: Besides incremental PCA, random projection is another computationally efficient method for dimensionality reduction that can be used incrementally. While not as commonly used as PCA for incremental learning, it has unique advantages, especially for very high-dimensional data. The random projection method was proposed based on the *Johnson-Lindenstrauss lemma* [34], which states that "*a set of points in a high-dimensional space can be projected onto a lower-dimensional space while preserving pairwise distances*".

Formally, given the input data instance $\mathbf{x} \in \mathbb{R}^m$, random projection proposes to generate a random matrix $\mathbf{R} \in \mathbb{R}^{m \times k}$ of size $m \times k$ to project the input data instance into a lower-dimensional space as follows:

$$\kappa(\mathbf{x}) = \mathbf{x}\mathbf{R} \in \mathbb{R}^k. \tag{103}$$

There exist different approaches to generate the random matrix $\mathbf{R}$, such as sparse random projection and Gaussian random projection.

For the *sparse random projection* method, the random matrix $\mathbf{R}$ elements are generated subject to the density parameter $s \in [0, 1]$. For instance, each matrix element $\mathbf{R}(i, j)$ may take values:

$$\mathbf{R}(i,j) = \begin{cases} -\sqrt{\frac{1}{s \cdot k}} & \text{with probability } \frac{s}{2}, \\ 0 & \text{with probability } 1 - s, \\ +\sqrt{\frac{1}{s \cdot k}} & \text{with probability } \frac{s}{2}. \end{cases} \tag{104}$$

Meanwhile, for the *Gaussian random projection*, the random matrix elements, *e.g.,* $\mathbf{R}(i, j)$, are randomly drawn from the Gaussian distribution as follows:

$$\mathbf{R}(i,j) \sim \mathcal{N}\left(0, \frac{1}{k}\right). \tag{105}$$

Similar to the feature selection-based compression functions, an early-stop parameter can also be applied to these dimension reduction-based compression functions. This parameter will halt the updating and tuning of the function's internal components as the RPN 2 model performance stabilizes with training epochs.

In addition to the dimension reduction methods based on *incremental PCA* and *random projection*, we have implemented several manifold-based techniques, including Isomap, t-SNE, Locally Linear Embedding, MDS, Spectral Embedding, and SMACOF. While we will not delve into the specifics of these manifold-based compression functions here, readers are encouraged to select the most suitable methods for their particular project and function learning tasks.

6.1.6 Probabilistic Compression Function

Building upon our previous work [89], which introduced probabilistic expansion functions for expanding input data instance vectors into their log-likelihood in naive or combinatorial modes, we now introduce a novel category of data compression functions based on probability distributions, termed probabilistic compression functions. These functions aim to compress data instances into vectors using probabilistic sampling methods.

Formally, given a data instance $\mathbf{x} \in \mathbb{R}^m$, we define the probabilistic compression function based on probabilistic sampling as:

$$\kappa(\mathbf{x}) = \mathbf{t} \in \mathbb{R}^d, \tag{106}$$

where the output vector $\mathbf{t}$ is conditionally dependent on $\mathbf{x}$ following certain distributions. For example, using a Gaussian distribution:

$$\mathbf{t}|\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{107}$$

The dimension d of the output vector $\mathbf{t}$ is a hyper-parameter requiring manual setup. While we use the Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ as an example, $\mathbf{t}|\mathbf{x}$ can follow other distributions, such as Cauchy or Laplace distributions, which have been employed in defining probabilistic expansion functions in our previous work [89].

Similar to the previously introduced expansion functions, probabilistic compression functions can operate in both the naive and combinatorial modes.

Naive Probabilistic Compression Function: This naive probabilistic compression function assumes independence among attributes in the input data instance vector. It performs sequential random sampling of attributes without replacement from the data instance to compose the output vector of the desired length. Beyond simple random sampling with uniform distributions (implemented in the TINYBIG V0.2.0 toolkit as well), we can also sample attributes based on instance attribute values as follows.

Formally, given a data instance vector $\mathbf{x} \in \mathbb{R}^m$, the sampling probabilities of all available attributes in the first sampling step can be represented as:

$$P(f(\mathbf{x}_i)|\theta_i), \forall i \in \{1, 2, \cdots, m\}, \tag{108}$$

where θ_i denotes the hyper-parameters of the distribution corresponding to the i_{th} attribute. For Gaussian distributions, it can represent the mean μ_i and standard deviation σ_i of the i_{th} attribute, as indicated in Equation (107). In practice, we may set equal-valued hyper-parameters θ for all attributes, *i.e.,* $\theta_i = \theta, \forall i \in 1, 2, \cdots, m$.

The mapping $f(\cdot)$ used above projects the attribute value $\mathbf{x}_i$ to a scalar compatible with the distribution. For normalized attribute values, we can simply define $f(\mathbf{x}_i) = \mathbf{x}_i$; otherwise, we can define it as a normalization mapping, *i.e.,* $f(\mathbf{x}_i) = \text{normalize}(\mathbf{x}_i|\theta_i)$. Additionally, we can define the mapping to extract insightful numerical or statistical metrics about the attributes, similar to those introduced in Section 6.1.3.

Furthermore, to maintain consistency with probabilistic expansion function outputs in certain learning scenarios, we can project the sampled attributes to their log-likelihoods, redefining the naive probabilistic compression function as:

$$\kappa(\mathbf{x}) = \log P(\mathbf{t}|\boldsymbol{\theta}) \in \mathbb{R}^d. \tag{109}$$

Combinatorial Probabilistic Compression Function: Building on the combinatorial expansions introduced in our previous work [89], we now present the combinatorial probabilistic compression function. Unlike naive probabilistic compression functions, this approach considers relationships among variables in multivariate distributions, enabling better modeling of complex data distributions.

Formally, given a data instance vector $\mathbf{x} \in \mathbb{R}^m$, we can represent the combination of k selected attributes from $\mathbf{x}$ as $\binom{\mathbf{x}}{k}$, for $k \in \{1, 2, \cdots, m\}$. For an input vector $\mathbf{x}$ of length m, the notation $\binom{\mathbf{x}}{k}$ represents a set containing $\binom{m}{k}$ attribute combination tuples, *e.g.,*

$$\begin{aligned}
k = 1 &: \binom{\mathbf{x}}{1} = \left\{(\mathbf{x}_1), (\mathbf{x}_2), \cdots, (\mathbf{x}_m)\right\}, \\
k = 2 &: \binom{\mathbf{x}}{2} = \left\{(\mathbf{x}_1, \mathbf{x}_2), (\mathbf{x}_1, \mathbf{x}_3), \cdots, (\mathbf{x}_{m-1}, \mathbf{x}_m)\right\}, \\
k = 3 &: \binom{\mathbf{x}}{2} = \left\{(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3), (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_4), \cdots, (\mathbf{x}_{m-2}, \mathbf{x}_{m-1}\mathbf{x}_m)\right\}.
\end{aligned} \tag{110}$$

For all attribute combination tuples in $\binom{\mathbf{x}}{d}$, using multivariate distributions (*e.g.,* $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ defined on d variables), one approach to compose the output attribute $\mathbf{t} \in \mathbb{R}^d$ is to randomly select one combination tuple from $\binom{\mathbf{x}}{d}$ according to tuple-wise probability scores, similar to Equation (108).

However, this sampling-based method can be challenging to implement due to the exponential size growth of $\binom{\mathbf{x}}{d}$ as d increases. For large d values, both enumerating d-sized attribute combinations in $\binom{\mathbf{x}}{d}$ and computing multivariate distributions with d variables become computationally expensive.

To address this, we can utilize combinatorial expansion functions, consistent with the probabilistic expansion functions introduced in [89]:

$$\kappa(\mathbf{x}|k) = \left[\binom{\mathbf{x}}{1}, \binom{\mathbf{x}}{2}, \cdots, \binom{\mathbf{x}}{k}\right]. \tag{111}$$

This expansion output contains $\sum_{i=1}^{k} \binom{m}{i}$ tuples of varying lengths. Based on this expansion, we can define the combinatorial probabilistic compression function by sampling d tuples from $\kappa(\mathbf{x}|1:k)$, treating the tuples as independent "items". Depending on tuple length, a corresponding multivariate distribution can be applied to compute the log-likelihood of the tuples:

$$\kappa(\mathbf{x}) = \log P\left(\binom{\kappa(\mathbf{x}|k)}{d}|\boldsymbol{\theta}\right) \in \mathbb{R}^d. \tag{112}$$

In practice, due to the exponential growth of output dimensions in the combinatorial expansion function denoted by Equation (111), the hyper-parameter k is typically set to a small value, *e.g.,* $k = 2$ or $k = 3$. This ensures both the efficiency of this probabilistic compression function and also the output value consistency with the previous probabilistic expansion function outputs.

6.2 Fusion Functions

The wide architecture with multi-head and multi-channel design endows RPN 2 with enhanced learning capacities. In the previous RPN model, we used a summation-based fusion strategy that assigned equal importance to each head and channel. However, for more complex function learning tasks, this simple summation approach may no longer be adequate. In this section, we introduce several advanced fusion strategies that can more effectively aggregate the outputs from the wide architectures. Moreover, these fusion functions are versatile, supporting not only the construction of hybrid interdependence functions but also other essential functions, thus offering RPN 2 significant flexibility in architectural design.

To simplify the notations, we represent the inputs to the fusion function as matrices $\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_k$, with the fusion output denoted by

$$\mathbf{A} = \text{fusion}(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_k). \tag{113}$$

The dimensions of the input matrices $\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_k$ may be identical or vary, depending on the specific definition of the fusion function. We will specify their dimensions in detail when introducing the concrete functions below.

6.2.1 Weighted Summation based Fusion Function

The weighted summation-based fusion function requires that all input matrices have identical dimensions. Formally, given interdependence matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k \in \mathbb{R}^{m \times n}$ of dimension $m \times n$, we can combine them through a weighted summation as follows:

$$\text{fusion}(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_k) = \sum_{i=1}^{k} \alpha_i \mathbf{A}_i \in \mathbb{R}^{m \times n}, \tag{114}$$

where α_i represents the weight assigned to matrix $\mathbf{A}_i$ for each $i \in \{1, 2, \cdots, k\}$.

These weights can either be provided manually, leveraging domain expertise, or initialized as learnable parameters. Alternatively, we may set them to fixed values, such as 1 or $\frac{1}{k}$, simplifying the fusion function to a straightforward summation or averaging of the input matrices, respectively.

6.2.2 Numerical Operators based Fusion Function

In addition to weighted summation, the fusion operator can be defined using various numerical operators introduced in Section 6.1.3, such as maximum, minimum, and different averaging functions. Like the summation-based fusion function, these numerical operator-based fusion functions also require that the input matrices have identical dimensions.

For example, we can define the fusion operator using the maximum operator, combining the input interdependence matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k \in \mathbb{R}^{m \times n}$ as follows:

$$\text{fusion}(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_k) = \mathbf{A} \in \mathbb{R}^{m \times n}, \tag{115}$$

and the entry $\mathbf{A}(i, j)$ (for $i \in \{1, 2, \cdots, m\}$ and $j \in \{1, 2, \cdots, n\}$) can be represented as

$$\mathbf{A}(i, j) = \max\left(\mathbf{A}_1(i, j), \mathbf{A}_2(i, j), \cdots, \mathbf{A}_k(i, j)\right). \tag{116}$$

6.2.3 Hadamard Product based Fusion Function

In this paper, we extend the Hadamard product, originally defined for two matrices, to handle k matrices, $\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_k \in \mathbb{R}^{m \times n}$, enabling element-wise fusion across multiple inputs:

$$\mathbf{A} = \text{fusion}(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_k) = \mathbf{A}_1 \circ \mathbf{A}_2 \circ \cdots \circ \mathbf{A}_k \in \mathbb{R}^{m \times n}, \tag{117}$$

where for the $(i, j)_{th}$ element is calculated as

$$\mathbf{A}(i, j) = \mathbf{A}_1(i, j) \times \mathbf{A}_2(i, j) \times \cdots \times \mathbf{A}_k(i, j). \tag{118}$$

The Hadamard product-based fusion function is particularly beneficial for hybrid interdependence functions that incorporate both structural and data batch information. Here, structural information can serve as a filtering mask, selectively extracting relevant information from data batches. We provide an example of defining the hybrid interdependence function for graph-structured data in Section 5.3.2.

6.2.4 Concatenation and Linear Transformation based Fusion Function

Similar to the wide architectures discussed earlier, we can concatenate the input interdependence matrices and then reduce them to the desired dimensions using a linear transformation. This fusion function requires only that the input matrices have the same number of rows, allowing for differing numbers of columns.

Formally, given input interdependence matrices $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_k$, where each matrix $\mathbf{A}_i \in \mathbb{R}^{m \times n_i}$ has m rows and n_i columns, we define the fusion operator as follows:

$$\begin{aligned} \mathbf{A} &= \text{fusion}(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_k) \\ &= (\mathbf{A}_1 \sqcup \mathbf{A}_2 \sqcup \cdots \sqcup \mathbf{A}_k)\, \mathbf{W} \in \mathbb{R}^{m \times n} \end{aligned} \tag{119}$$

where $\sqcup$ denotes the row-wise concatenation of the matrices. The term $\mathbf{W} \in \mathbb{R}^{(\sum_{i=1}^{k} n_i) \times n}$ represents a learnable parameter matrix that projects the concatenated matrix to a dimension of n.

The concatenation of these interdependence matrices results in a relatively large dimension, specifically $\sqcup_{i=1}^{k} \mathbf{A} i \in \mathbb{R}^{m \times (\sum i=1^{k} n_i)}$. To reduce the number of learnable parameters, we can also apply low-rank parameter reconciliation, allowing us to rewrite the fusion function as follows:

$$\begin{aligned} \mathbf{A} &= \text{fusion}(\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_k) \\ &= \left(\sqcup_{i=1}^{k} \mathbf{A}_i\right) \left(\mathbf{P}\mathbf{Q}^\top\right) \in \mathbb{R}^{m \times n}, \end{aligned} \tag{120}$$

where the parameter matrix $\mathbf{W}$ is factorized into the inner product of two sub-matrices, specifically $\mathbf{W} = \mathbf{P}\mathbf{Q}^\top \in \mathbb{R}^{(\sum_{i=1}^{k} n_i) \times n}$, where $\mathbf{P} \in \mathbb{R}^{(\sum_{i=1}^{k} n_i) \times r}$ and $\mathbf{Q} \in \mathbb{R}^{n \times r}$ represent the low-rank factorization sub-matrices of $\mathbf{W}$.

Beyond the operators discussed above, several other fusion operators can also be employed to define fusion strategies for the hybrid interdependence function. While we illustrated this with attribute interdependence functions as an example of hybrid interdependence, similar functions can also be defined and fused for instance interdependence relationships, though these will not be covered in this section.

7 Other Component Function Updates in the Enhanced RPN 2 Model

Moreover, to enhance the existing suite of data expansion and parameter reconciliation functions, we introduce several new implementations designed to augment these components. These functions, in combination with others mentioned earlier, facilitate the creation of more versatile, efficient, and effective architectures within RPN 2. In refining the tinyBIG v0.2.0 toolkit, we deliberately deleted the complementary expansion-based remainder function proposed in our previous work [89]. This decision was made to eliminate redundancy in RPN 2, as such complementary expansion-based remainders can be equivalently implemented using the multi-head and multi-channel mechanisms, which are now the default strategy in building RPN 2. These updated component functions introduced in this section have also been summarized in the previous Figure 19.

7.1 Data Expansion Functions

In addition to the aforementioned interdependence and compression functions, this paper also expands the existing list of data expansion functions proposed in our previous work [89]. In our previous RPN paper [89], we introduced several orthogonal polynomials, such as the Chebyshev and Jacobi polynomials. In mathematics, an orthogonal polynomial sequence is a family of polynomials where any two distinct polynomials in the sequence are orthogonal to each other under some inner product. Building on our previous work, we will define several new orthogonal polynomials that can perform different data expansions for extracting useful information from the input data batch.

In addition to these new orthogonal polynomials, we also introduce a novel data expansion function based on wavelets, designed to expand multi-modal data such as audio signals and images into new spaces. In signal processing, a wavelet is formally defined as a wave-like oscillation with an amplitude that begins at zero, increases or decreases, and then returns to zero one or more times. Compared to the transformations introduced in [89], such as Fourier transformation, wavelet transformation replaces trigonometric basis functions with wavelet-based basis functions, which offers greater advantages in terms of multi-resolution analysis, adaptability, and computational efficiency.

In this section, we will introduce additional orthogonal polynomials and new wavelet transformation techniques that can be utilized to define data expansion functions. These newly added component functions significantly enrich the foundational building blocks available for RPN 2 model design, potentially enhancing its learning performance across diverse function learning tasks.

7.1.1 Hermite Polynomials based Expansion Function

Hermite polynomials, first defined by Pierre-Simon Laplace in 1810 and studied in detail by Pafnuty Chebyshev in 1859, were later named after Charles Hermite, who published work on these polynomials in 1864. The Hermite polynomials can be defined in various forms:

Probabilist's Hermite polynomials:

$$He_n(x) = (-1)^n \exp\left(\frac{x^2}{2}\right) \frac{\mathrm{d}^n}{\mathrm{d}x^n} \exp\left(-\frac{x^2}{2}\right). \tag{121}$$

Physicist's Hermite polynomials:

$$H_n(x) = (-1)^n \exp\left(x^2\right) \frac{\mathrm{d}^n}{\mathrm{d}x^n} \exp\left(-x^2\right). \tag{122}$$

These two forms are not identical but can be reduced to each via rescaling:

$$H_n(x) = 2^{\frac{n}{2}} He_n(\sqrt{2}x), \text{ and } He_n(x) = 2^{-\frac{n}{2}} H_n\left(\frac{x}{\sqrt{2}}\right). \tag{123}$$

In this paper, we will use the Probabilist's Hermite polynomials for to define the data expansion function by default, which can be formally defined as the following recursive representations:

$$He_{n+1}(x) = xHe_n(x) - nHe_{n-1}(x), \forall n \geq 1. \tag{124}$$

Some examples of the Probabilist's Hermite polynomials are also illustrated as follows:

$$\begin{aligned}
He_0(x) &= 1; \\
He_1(x) &= x; \\
He_2(x) &= x^2 - 1; \\
He_3(x) &= x^3 - 3x; \\
He_4(x) &= x^4 - 6x^2 + 3.
\end{aligned} \tag{125}$$

Based on the Probabilist's Hermite polynomials, we can define the data expansion function with order d as follows:

$$\kappa(\mathbf{x}|d) = [He_1(\mathbf{x}), He_2(\mathbf{x}), \cdots, He_d(\mathbf{x})] \in \mathbb{R}^D, \tag{126}$$

where d is the order hyper-parameter and the output dimension $D = md$. Similar as the data expansion functions introduced in the previous paper [89], the constant term $He_0(x)$ is not included in the expansion outputs.

7.1.2 Laguerre Polynomials based Expansion Function

In mathematics, the Laguerre polynomials, named after Edmond Laguerre, are the nontrivial solutions of Laguerre's differential equation:

$$xy'' + (\alpha + 1 - x)y' + dy = 0, \tag{127}$$

where $y = y(x)$ is a function of variable x. Notations y' and y'' denote first- and second-order derivatives of function y with respect to variable x. Term $d \in \mathbb{N}$ is a non-negative integer and $\alpha \in \mathbb{R}$ is a hyper-parameter.

The closed-form of the Laguerre polynomials can be represented as follows:

$$P_n^{(\alpha)}(x) = \frac{e^x}{n!}\frac{\mathrm{d}^n}{\mathrm{d}x^n}(e^{-x}x^n) = \frac{x^{-\alpha}}{n!}\left(\frac{\mathrm{d}}{\mathrm{d}x} - 1\right)^n x^{n+\alpha}, \tag{128}$$

where $\frac{\mathrm{d}}{\mathrm{d}x}$ denotes the derivative operator.

In practice, the Laguerre polynomials can be recursively defined as follows, which will be used for defining the data expansion function below. Specifically, when $\alpha = 0$, the above Laguerre polynomials are also known as simple Laguerre polynomials.

Base cases $n = 0$ and $n = 1$:

$$P_0^{(\alpha)}(x) = 1, \text{ and } P_1^{(\alpha)}(x) = 1 + \alpha - x. \tag{129}$$

High-order cases with degree $n \geq 2$:

$$P_n^{(\alpha)}(x) = \frac{(2n - 1 + \alpha - x)P_{n-1}^{(\alpha)}(x) - (n - 1 + \alpha)P_{n-2}^{(\alpha)}(x)}{n} \tag{130}$$

The recursive-form representations of the Laguerre polynomials can be used to define the data expansion function as follows:

$$\kappa(\mathbf{x}|d, \alpha) = \left[P_1^{(\alpha)}(\mathbf{x}), P_2^{(\alpha)}(\mathbf{x}), \cdots, P_d^{(\alpha)}(\mathbf{x})\right] \in \mathbb{R}^D, \tag{131}$$

where d and α are the function hyper-parameters and the output dimension $D = md$.

7.1.3 Legendre Polynomials based Expansion Function

The Legendre polynomials, named after mathematician Adrien-Marie Legendre, are defined as an orthogonal system over the interval $[-1, 1]$, where the polynomial term $P_n(x)$ of degree n satisfies the following equation:

$$\int_{-1}^{+1} P_m(x) P_n(x) dx = 0, \text{ if } m \neq n. \tag{132}$$

Specifically, according to Bonnet's formula, the Legendre polynomials can be recursively represented as follows:

Base cases $n = 0$ and $n = 1$:

$$P_0(x) = 1, \text{and } P_1(x) = x. \tag{133}$$

High-order cases with degree $n \geq 2$:

$$P_n(x) = \frac{x(2n-1)P_{n-1}(x) - (n-1)P_{n-2}(x)}{n} \tag{134}$$

The Legendre polynomials help define the data expansion function as follows:

$$\kappa(\mathbf{x}|d) = [P_1(\mathbf{x}), P_2(\mathbf{x}), \cdots, P_d(\mathbf{x})] \in \mathbb{R}^D, \tag{135}$$

where the output dimension $D = md$.

7.1.4 Gegenbauer Polynomials based Expansion Function

The Gegenbauer polynomials, named after mathematician Leopold Gegenbauer, are orthogonal polynomials that generalize both the Legendre and Chebyshev polynomials, and are special cases of Jacobi polynomials.

Formally, the Gegenbauer polynomials are particular solutions of the Gegenbauer differential equation:

$$(1-x^2)y'' - (2\alpha+1)xy' + d(d+2\alpha)y = 0, \tag{136}$$

where $y = y(x)$ is a function of variable x and $d \in \mathbb{N}$ is a non-negative integer.

When $\alpha = \frac{1}{2}$, the Gegenbauer polynomials reduce to the Legendre polynomials introduced earlier; when $\alpha = 1$, they reduce to the Chebyshev polynomials of the second kind.

The Gegenbauer polynomials can be recursively defined as follows:

Base cases $n = 0$ and $n = 1$:

$$P_0^{(\alpha)}(x) = 1, \text{and } P_1^{(\alpha)}(x) = 2\alpha x. \tag{137}$$

High-order cases with degree $n \geq 2$:

$$P_n^{(\alpha)}(x) = \frac{2x(n-1+\alpha)P_{n-1}^{(\alpha)}(x) - (n+2\alpha-2)P_{n-2}^{(\alpha)}(x)}{n} \tag{138}$$

Based on the Gegenbauer polynomials, we can define the expansion function as follows:

$$\kappa(\mathbf{x}|d,\alpha) = \left[P_1^{(\alpha)}(\mathbf{x}), P_2^{(\alpha)}(\mathbf{x}), \cdots, P_d^{(\alpha)}(\mathbf{x})\right] \in \mathbb{R}^D, \tag{139}$$

where the output dimension $D = md$.

7.1.5 Bessel and Reverse Bessel Polynomials based Expansion Functions

Formally, the Bessel polynomials are an orthogonal sequence of polynomials with the following closed-form representation:

$$B_n(x) = \sum_{k=0}^{n} \frac{(n+k)!}{(n-k)!k!} \left(\frac{x}{2} \right)^k . \tag{140}$$

Another definition, favored by electrical engineers, is sometimes known as the reverse Bessel polynomials, with the following closed-form representation:

$$R_n(x) = x^n B_n \left(\frac{1}{x} \right) = \sum_{k=0}^{n} \frac{(n+k)!}{(n-k)!k!} \frac{x^{n-k}}{2^k}. \tag{141}$$

Both the Bessel and reverse Bessel polynomials can be recursively defined as follows:

Base cases $n = 0$ and $n = 1$:

$$\begin{aligned} \textit{Bessel: } & B_0(x) = 1, \text{ and } B_1(x) = x + 1; \\ \textit{Reverse Bessel: } & R_0(x) = 1, \text{ and } R_1(x) = x + 1. \end{aligned} \tag{142}$$

High-order cases with degree $n \geq 2$:

$$\begin{aligned} \textit{Bessel: } & B_n(x) = (2n-1)xB_{n-1}(x) + B_{n-2}(x); \\ \textit{Reverse Bessel: } & R_n(x) = (2n-1)B_{n-1}(x) + x^2 B_{n-2}(x). \end{aligned} \tag{143}$$

Both the Bessel and reverse Bessel polynomials can be used to define the data expansion functions as follows:

$$\begin{aligned} \textit{Bessel: } & \kappa(\mathbf{x}|d) = [B_1(\mathbf{x}), B_2(\mathbf{x}), \cdots, B_d(\mathbf{x})] \in \mathbb{R}^D, \\ \textit{Reverse Bessel: } & \kappa(\mathbf{x}|d) = [R_1(\mathbf{x}), R_2(\mathbf{x}), \cdots, R_d(\mathbf{x})] \in \mathbb{R}^D, \end{aligned} \tag{144}$$

where the output dimension $D = md$.

7.1.6 Fibonacci and Lucas Polynomials based Expansion Functions

Formally, the Fibonacci polynomials are a polynomial sequence that can be considered a generalization of the Fibonacci numbers. Similarly, Lucas polynomials are generated from the Lucas numbers in an analogous manner.

Both Fibonacci and Lucas polynomials can be defined recursively. The Lucas polynomials can be viewed as identical to the Fibonacci polynomials but with different base case representations:

Base cases $n = 0$ and $n = 1$:

$$\begin{aligned} \textit{Fibonacci: } & F_0(x) = 0, \text{ and } F_1(x) = 1; \\ \textit{Lucas: } & L_0(x) = 2, \text{ and } L_1(x) = x. \end{aligned} \tag{145}$$

High-order cases with degree $n \geq 2$:

$$\begin{aligned} \textit{Fibonacci: } & F_n(x) = xF_{n-1}(x) + F_{n-2}(x); \\ \textit{Lucas: } & L_n(x) = xL_{n-1}(x) + L_{n-2}(x). \end{aligned} \tag{146}$$

Example 1 *Based on these recursive representations, we can illustrate some examples of the Fibonacci and Lucas polynomials as follows:*

Fibonacci Polynomials:

$$\begin{aligned} F_0(x) &= 0 \\ F_1(x) &= 1 \\ F_2(x) &= x \\ F_3(x) &= x^2 + 1 \\ F_4(x) &= x^3 + 2x \\ F_5(x) &= x^4 + 3x^2 + 1 \end{aligned} \tag{147}$$

Lucas Polynomials:

$$\begin{aligned} L_0(x) &= 2 \\ L_1(x) &= x \\ L_2(x) &= x^2 + 2 \\ L_3(x) &= x^3 + 3x \\ L_4(x) &= x^4 + 4x^2 + 2 \\ L_5(x) &= x^5 + 5x^3 + 5x \end{aligned} \tag{148}$$

Both the Fibonacci and Lucas polynomials can be used to define the data expansion functions as follows:

$$\begin{aligned} \textit{Fibonacci: } \kappa(\mathbf{x}|d) &= [F_1(\mathbf{x}), F_2(\mathbf{x}), \cdots, F_d(\mathbf{x})] \in \mathbb{R}^D, \\ \textit{Lucas: } \kappa(\mathbf{x}|d) &= [L_1(\mathbf{x}), L_2(\mathbf{x}), \cdots, L_d(\mathbf{x})] \in \mathbb{R}^D, \end{aligned} \tag{149}$$

where the output dimension $D = md$.

7.1.7 Wavelet based Expansion Functions

In our previous paper [89], we introduced the Fourier series for expanding data instances into a sequence of trigonometric functions. Fourier series are closely related to the Fourier transform, which can be used to find frequency information for non-periodic functions. In fact, the Fourier transform can be viewed as a special case of the continuous wavelet transform. While the standard Fourier transform is only localized in frequency, wavelets are localized in both time and frequency, making them particularly useful for non-stationary signals where frequency components change over time.

Wavelet transform represents the input data as a summation of basis functions, known as wavelets. Specifically, the basis functions in wavelet transformation can be categorized into the *mother wavelet* and *father wavelet*, which are orthogonal and can both be derived from the *child wavelet* via rescaling and translation operators.

Formally, given the input variable $\mathbf{x} \in \mathbb{R}^m$, to approximate the underlying mapping $f : \mathbb{R}^m \to \mathbb{R}^n$ with wavelet analysis, we can define the approximated output as

$$f(\mathbf{x}) \approx \sum_{s,t} \langle f(\mathbf{x}), \phi_{s,t}(\mathbf{x}|a, b) \rangle \cdot \phi_{s,t}(\mathbf{x}|a, b), \tag{150}$$

where $\phi_{s,t}(\cdot|a, b)$ denotes the child wavelet defined by hyper-parameters $a > 1$ and $b > 0$:

$$\phi_{s,t}(x|a, b) = \frac{1}{\sqrt{a^s}} \phi \left(\frac{x - t \cdot b \cdot a^s}{a^s} \right). \tag{151}$$

Specifically, the functions $\{\phi_{s,t}\}_{s,t \in \mathbb{Z}}$ defines the orthonormal basis of the space and the mapping $\phi(\cdot)$ used in the child wavelet may have different representations:

(a) Haar wavelet:

$$\phi(\tau) = \begin{cases} 1, & 0 \le \tau < \frac{1}{2}, \\ -1, & \frac{1}{2} \le \tau < 1, , \\ 0, & \text{otherwise}. \end{cases} \tag{152}$$

(b) Beta wavelet:

$$\phi(\tau|\alpha, \beta) = \frac{1}{B(\alpha, \beta)} \tau^{\alpha-1} (1 - \tau)^{\beta-1},$$

where $\alpha, \beta \in [1, \infty]$.

(153)

(c) Ricker wavelet:

$$\phi(\tau) = \frac{2\left(1 - \left(\frac{\tau}{\sigma}\right)^2\right)}{\sqrt{3\sigma}\pi^{\frac{1}{4}}} \exp\left(-\frac{\tau^2}{2\sigma^2}\right). \tag{154}$$

(e) Difference of Gaussians:

$$\phi(\tau|\sigma_1, \sigma_2) = P(\tau|0, \sigma_1) - P(\tau|0, \sigma_2), \tag{156}$$

where $P(\cdot|0, \sigma_1)$ denotes the PDF of the Gaussian distribution.

(d) Shannon wavelet:

$$\phi(\tau) = \frac{\sin(2\pi\tau) - \sin(\pi\tau)}{\pi\tau}. \tag{155}$$

(f) Meyer wavelet:

$$\phi(\tau) = \begin{cases} \frac{2}{3} + \frac{4}{3\pi} & \tau = 0, \\ \frac{\sin(\frac{2\pi}{3}\tau) + \frac{4}{3}\tau\cos(\frac{4\pi}{3}\tau)}{\pi\tau - \frac{16\pi}{9}\tau^3} & \text{otherwise.} \end{cases} \tag{157}$$

To apply the aforementioned wavelets for data expansion, we need to re-examine Equation (150) introduced earlier. This equation can be interpreted in various ways within the context of RPN 2:

$$\sum_{s,t} \underbrace{\langle f(\mathbf{x}), \phi_{s,t}(\mathbf{x}|a,b) \rangle}_{\text{coefficients}} \cdot \underbrace{\phi_{s,t}(\mathbf{x}|a,b)}_{\text{the expansion}}, \tag{158}$$

and

$$\sum_{s,t} \langle \underbrace{f(\mathbf{x})}_{\text{coefficients}}, \underbrace{\phi_{s,t}(\mathbf{x}|a,b) \rangle \cdot \phi_{s,t}(\mathbf{x}|a,b)}_{\text{the expansion}}. \tag{159}$$

Based on these above two representations, we can introduce the 1_{st}-order and 2_{nd}-order wavelet data expansion functions as follows:

$$\kappa(\mathbf{x}|d=1) = [\phi_{0,0}(\mathbf{x}), \phi_{0,1}(\mathbf{x}), \cdots, \phi_{s,t}(\mathbf{x})] \in \mathbb{R}^{D_1}. \tag{160}$$

and

$$\kappa(\mathbf{x}|d=2) = \kappa(\mathbf{x}|d=1) \otimes \kappa(\mathbf{x}|d=1) \in \mathbb{R}^{D_2}. \tag{161}$$

The output dimensions of the order-1 and order-2 wavelet expansions are $D_1 = s \cdot t \cdot m$ and $D_2 = (s \cdot t \cdot m)^2$, respectively.

7.2 Parameter Reconciliation Function

We have introduced several different categories of parameter reconciliation functions in the previous paper [89] already. In this part, we will introduce a new category of parameter reconciliation functions defined based on the random matrices.

7.2.1 Random Matrix Adaption based Parameter Reconciliation Function

According to the low-rank reconciliation (LoRR) function introduced in our previous paper [89], given a parameter vector $\mathbf{w} \in \mathbb{R}^l$ of length l, we can fabricate it into a parameter matrix of shape $n \times D$ as follows:

$$\psi(\mathbf{w}) = \mathbf{A}\mathbf{B}^\top \in \mathbb{R}^{n \times D}, \tag{162}$$

where $\mathbf{A} \in \mathbb{R}^{n \times r}$ and $\mathbf{B} \in \mathbb{R}^{D \times r}$ are the low-rank parameter sub-matrices of rank r reshaped from the input parameter vector $\mathbf{w}$. The length of the input parameter is determined by the rank hyper-parameter r, *i.e.,* $l = (n + D) \cdot r$.

The recent VeRA paper [39] proposes to freeze the sub-matrices $\mathbf{A}$ and $\mathbf{B}$ as random constants, *e.g.,*

$$\mathbf{A}, \mathbf{B} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{163}$$

where the sub-matrix elements are randomly sampled from the Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

The learnable parameters can be added as the diagonal matrices $\mathbf{\Lambda}_1 = diag(\boldsymbol{\lambda}_1) \in \mathbb{R}^{n \times n}$ and $\mathbf{\Lambda}_2 = diag(\boldsymbol{\lambda}_2) \in \mathbb{R}^{r \times r}$, where vectors $\boldsymbol{\lambda}_1 \in \mathbb{R}^n$ and $\boldsymbol{\lambda}_2 \in \mathbb{R}^r$ are split from the input parameter $\mathbf{w}$. This defines the random matrix adaptation-based parameter reconciliation function as follows:

$$\psi(\mathbf{w}) = \mathbf{\Lambda}_1 \mathbf{A} \mathbf{\Lambda}_1 \mathbf{B}^\top \in \mathbb{R}^{n \times D}, \tag{164}$$

where the required number of learnable parameters is $l = n + r$.

7.2.2 Random Matrix based Hypernet Parameter Reconciliation Function

We introduced the Hypernet parameter reconciliation function in our previous paper [89]. Given the input parameter vector $\mathbf{w} \in \mathbb{R}^l$, the function can be represented as:

$$\psi(\mathbf{w}) = \text{Hypernet}(\mathbf{w}) \in \mathbb{R}^{n \times D}, \tag{165}$$

where "Hypernet$(\cdot)$" can be defined with different models, such as MLP, with randomized and frozen parameters. In the previous paper [89], we implemented this function with a 2-layered MLP model of dimensions $(l, d, n \cdot D)$, *i.e.,*

$$\text{Hypernet}(\mathbf{w}) = \sigma(\mathbf{w}\mathbf{H}_1)\mathbf{H}_2 \in \mathbb{R}^{n \times D}, \tag{166}$$

where $\mathbf{H}_1 \in \mathbb{R}^{l \times d}$ and $\mathbf{H}_2 \in \mathbb{R}^{d \times (n \cdot D)}$ are the randomly initialized frozen parameters of the MLP. Notation d denotes the middle hidden layer dimension and $\sigma(\cdot)$ denotes the sigmoid function. By default, we have the middle dimension $l < d < n \cdot D$.

In experimental testing, we encountered implementation challenges with this reconciliation function. For some data expansion functions, the expansion space D can be very large, and the hypernet initialization may consume space of $\mathcal{O}\left(d \cdot (l + n \cdot D)\right)$.

In this paper, based on the aforementioned random matrix adaptation techniques, we propose replacing the two large-sized frozen parameter matrices $\mathbf{H}_1$ and $\mathbf{H}_2$ with their low-rank representations:

$$\begin{aligned} \text{Hypernet}(\mathbf{w}) &= \sigma\left(\mathbf{w}(\mathbf{P}\mathbf{Q}^\top)\right)\left(\mathbf{S}\mathbf{T}^\top\right) \\ &= \left(\sigma\left((\mathbf{w}\mathbf{P})\mathbf{Q}^\top\right)\mathbf{S}\right)\mathbf{T}^\top \in \mathbb{R}^{n \times D}, \end{aligned} \tag{167}$$

where $\mathbf{P} \in \mathbb{R}^{l \times r}$, $\mathbf{Q} \in \mathbb{R}^{d \times r}$, $\mathbf{S} \in \mathbb{R}^{d \times r}$ and $\mathbf{T} \in \mathbb{R}^{(n \times D) \times r}$ are the low-rank random and frozen sub-matrices that can compose the matrices $\mathbf{H}_1$ and $\mathbf{H}_2$ of the hypernet. Moreover, by leveraging the associative law of matrix multiplication, we can avoid explicitly calculating and storing $\mathbf{H}_1$ and $\mathbf{H}_2$ as indicated by the above equation. These low-rank random matrix representations reduce the space consumption of this function to $\mathcal{O}\left(r \cdot (l + 2d + n \cdot D)\right)$.

8 Unifying Existing Backbones with RPN 2

The incorporation of new interdependence functions and compression functions significantly enhances RPN 2's capabilities, offering both improved modeling power and increased learning efficiency when dealing with complex data characterized by diverse underlying interdependence relationships. By strategically selecting these component functions based on the specific input data, we can construct highly versatile model architectures using RPN 2. Furthermore, RPN 2 provides a unified framework capable of representing the many influential contemporary backbone architectures, including but not limited to Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Graph Neural Networks (GNNs), and Transformers.

Motivations of Backbone Unification: While unifying existing backbone models is not the primary focus of this paper, we propose this unification with several key goals. First, we aim to uncover the foundational architectural similarities and highlight the critical differences among current backbone models (to be discussed in this section). Second, a unified architectural representation enables the theoretical analyses and comparisons of learning performance for these existing structures (to be discussed in the following Section 10). Finally, and most importantly, this unification allows us to pinpoint the core weaknesses within these models, opening up pathways for potential enhancements or the creation of novel "Transformer-Next" architectures—an exploration we will pursue in our future papers.

The illustrations of the unified representation of CNN, RNN, GNN and Transformer with RPN 2 are also provided in the following Figure 22 and Figure 24, respectively.

8.1 Unifying CNNs with RPN 2

Convolutional Neural Networks (CNNs) [43] have long served as the backbone model for image processing tasks. Over time, their application has expanded beyond image data, encompassing a diverse range of modalities. With appropriate model extensions, CNNs have been successfully adapted to process data in other modalities, such as point clouds [58], textual data [36], time series [44], and graph structures [14]. In this section, we will introduce the Convolutional Neural Networks (CNNs) initially designed for images with the convolutional and pooling operators, and discuss how to represent CNNs into the unified representation of the RPN 2 model based on the component functions introduced above.

8.1.1 Convolutional Neural Network (CNN)

In this part, we will examine the architecture of CNN model, with particular emphasis on two crucial components in the model, *i.e.,* the convolutional operator and the pooling operator. These operators play pivotal roles in the CNN's ability to effectively extract features from input data, forming the foundation of the model's success in various tasks, particularly in image processing.

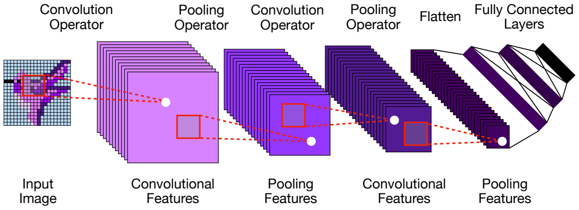


Figure 20: An illustration of CNN model.

CNN Model Architecture: The right plot illustrates the architecture of a classic Convolutional Neural Network (CNN) model, comprising convolutional operators, pooling operators, and fully connected layers. Given an input image of a "hummingbird", the CNN processes it through a series of layers of different operators. Initially, the convolutional layers shift convolutional kernels (*i.e.,* parameter matrices or tensors) across the image to compute convolutional feature maps. These features then pass through pooling layers, where pooling kernels extract salient features and compress the maps through operations such as max-pooling, resulting in compressed feature maps. CNNs often employ a deep architecture by stacking multiple convolutional and pooling layers sequentially, allowing for the extraction of increasingly abstract features. After the final convolutional or pooling layer, the learned feature maps are flattened into a dense feature vector. This flattened vector is then processed by fully connected layers, which perform the final classification of the input image into the appropriate label category, in this case, "Hummingbird".

Convolutional Operator: Convolution has been a cornerstone in conventional image processing tasks, employed for tasks such as blurring, sharpening, embossing, and edge detection. While conventional convolution operators often rely on manually defined kernel matrices, Convolutional Neural Networks (CNNs) innovate by learning these kernels as parameters. This approach endows CNNs with superior modeling capacity and flexibility compared to traditional image processing techniques, as the learned kernels can capture diverse useful image feature patterns.

Formally, given an input image $\mathbf{X} \in \mathbb{R}^{h \times w \times d}$ with height h, width w, and depth d, the convolutional operator shifts a kernel parameter matrix $\mathbf{W} \in \mathbb{R}^{(p_h+p'_h+1)\times(p_w+p'_w+1)\times(p_d+p'_d+1)}$ of sizes $p_h + p'_h + 1$, $p_w + p'_w + 1$, and $p_d + p'_d + 1$ along the rows, columns, and depth dimensions. For a sub-image centered at pixel coordinates (i, j, k), we can represent it as $\bar{\mathbf{X}} = \mathbf{X}(i - p_h : i + p'_h, j - p_w : j + p'_w, k - p_d : k + p'_d)$, whose convolution with the kernel is calculated as:

$$\bar{\mathbf{X}} * \mathbf{W} = \sum_{r=0}^{p_h+p'_h} \sum_{s=0}^{p_w+p'_w} \sum_{t=0}^{p_d+p'_d} \bar{\mathbf{X}}(r, s, t) \cdot \mathbf{W}(p_h + p'_h - r, p_w + p'_w - s, p_d + p'_d - t), \tag{168}$$

where $*$ denotes the convolutional operator. In practice, many algorithms and toolkits substitute this convolutional operator with the cross-correlation operator instead:

$$\bar{\mathbf{X}} \circ \mathbf{W} = \sum_{r=0}^{p_h+p'_h} \sum_{s=0}^{p_w+p'_w} \sum_{t=0}^{p_d+p'_d} \bar{\mathbf{X}}(r, s, t) \cdot \mathbf{W}(r, s, t). \tag{169}$$

The convolutional operator $\bar{\mathbf{X}} * \mathbf{W}$ and the cross-correlation operator $\bar{\mathbf{X}} \circ \mathbf{W}$ are mathematically different but equivalent in practical model learning, with the distinction that the kernel matrix $\mathbf{W}$ is flipped in all dimensions for convolution. In practice, this substitution of convolution with cross-correlation does not impact the model's learning performance, as $\mathbf{W}$ is a learned parameter. Moreover, using cross-correlation can enhance computational efficiency on backend hardware without the redundant costs for the tensor reshaping and flipping at the memory.

Beyond the kernel sizes (or the patch shape and size in RPN 2), CNNs incorporate additional hyper-parameters that significantly influence their behavior and learning performance. A key example is the *stride*, which defines the steps for shifting the kernel across the input. The choice of stride directly affects the spatial dimensions of the resulting feature maps, thereby impacting the model's receptive field and the level of detail preserved in the output. The stride parameters used in CNNs can be precisely determined by the patch packing strategies in RPN 2 as discussed in the previous Sections 5.2.4-5.2.6.

Pooling Operator: The pooling operator, devoid of learnable parameters, serves to compress the convolution feature map by extracting salient features. Various pooling methods can be employed based on the learning context, including max-pooling, mean-pooling, and min-pooling, which extract the maximum, average, and minimum values from feature map regions, respectively.

Similar to the convolutional operator, the pooling operator utilizes kernels of dimensions p_h, p_w, and p_d (and also p'_h, p'_w, p'_d), which traverse the convolutional feature along its row, column, and depth dimensions with specified stride lengths. However, unlike convolution kernels, pooling kernels contain no learnable parameters and merely delineate the input regions for pooling operations. As depicted in Figure 21, a 4×4 feature map subjected to a 2×2 max-pooling kernel with stride $(2, 2)$ yields a condensed 2×2 feature map, where each value represents the maximum from the corresponding input region.

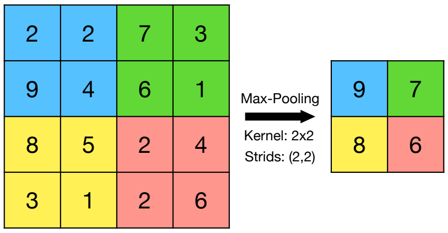
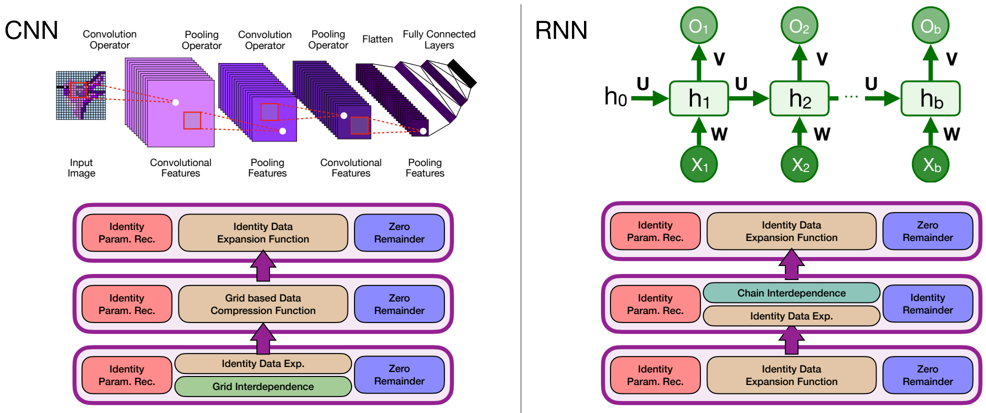


Figure 21: An illustration of pooling operator.

8.1.2 Representing CNN with RPN 2

Before formalizing the representation of CNN architecture within RPN 2, we will first discuss how to represent the convolutional and pooling operators using the component functions introduced in previous sections.

Representing Convolution with Grid Structural Interdependence Function: The convolutional operator in CNNs utilizes learnable kernel parameters to extract information from input images.



(a) Representing CNN with RPN 2

(b) Representing RNN with RPN 2

Figure 22: An illustration of representing CNN and RNN with RPN 2.

These kernels shift along the dimensions of the input image tensor, operating on sub-image patches of equal size to the kernels. The feature extraction process can be represented as the inner product of flattened sub-image patches with the kernel parameter vector, where identical kernel parameters are shared across all sub-image patches.

Formally, for an input image $\mathbf{X} \in \mathbb{R}^{h \times w \times d}$ with height h, width w, and depth d, we can represent its flattened form as vector $\mathbf{x} = \text{reshape}(\mathbf{X}) \in \mathbb{R}^{(h \times w \times d)}$. The original image modality specific topological structure can be represented as $grid(\mathbf{x}|h, w, d)$, as described in Section 5.2.2. The sub-image matrix $\bar{\mathbf{X}}$ from Equation (168) actually corresponds to a cuboid patch $patch(i, j, k)$ centered at coordinate (i, j, k) in the grid, with patch shape hyper-parameters $p_h, p'_h, p_w, p'_w, p_d$, and p'_d defined in Section 5.2.2.

The cross-correlation (or convolution) based feature extraction defined in above Equation (169) can be equivalently represented as:

$$\bar{\mathbf{X}} \circ \mathbf{W} = \langle \mathbf{p}, \mathbf{w} \rangle = \sum_{i=0}^{p-1} \mathbf{p}(i) \cdot \mathbf{w}(i), \tag{170}$$

where $\mathbf{p} = \mathbf{x}(patch(i, j, k)) = \text{reshape}(\bar{\mathbf{X}}) \in \mathbb{R}^p$ is the flattened patch vector representation of the sub-image, with $p = (p_h + p'_h + 1) \times (p_w + p'_w + 1) \times (p_d + p'_d + 1)$ denoting the patch size. Notation $\mathbf{w} = \text{reshape}(\mathbf{X})$ denotes the flattened kernel parameter vector of equal length p.

The feature extraction for the entire input image can be calculated concurrently using patch packing strategies discussed in the previous Sections 5.2.4-5.2.6. Using the structural interdependence function from Section 5.2.3 with patch center distance hyper-parameters d_h, d_w, d_d, we can represent the extracted feature map as:

$$\langle \kappa_\xi(\mathbf{x}), \psi(\mathbf{w}) \rangle = \langle \kappa\left(\mathbf{x}\xi(\mathbf{x})\right), \psi(\mathbf{w}) \rangle = \langle \mathbf{x}\mathbf{A}, \mathbf{c} \otimes \mathbf{w} \rangle, \tag{171}$$

where

- $\xi(\mathbf{x}) = \mathbf{A} \in \mathbb{R}^{m \times m'}$: This denotes the structural interdependence function with the padding mode.
- $\kappa(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^m$: This denotes the identity data expansion function.
- $\psi(\mathbf{w}) = \mathbf{I} \otimes \mathbf{w} = \mathbf{W} \in \mathbb{R}^{(p \times p_{count}) \times p_{count}}$: This represents the duplicated padding-based parameter reconciliation function with diagonal block matrix outputs.

The structural interdependence function has been introduced in the previous Equation (66), while the identity data expansion and duplicated padding-based parameter reconciliation function were introduced in the previous paper [89]. Specificall, the matrix dimension term m' is defined as $m' = p \times p_{count}$, as indicated in Equation (73), where p_{count} represents the number of patches in the grid, as defined in the previous Equation (71). The term $\mathbf{I}$ is a constant identity matrix defined as $\mathbf{I} = diag([1, 1, \cdots, 1]) \in \mathbb{R}^{p_{count} \times p_{count}}$, and $\mathbf{I} \otimes \mathbf{w}$ creates p_{count} duplicated paddings of parameter vector $\mathbf{w}$ along the diagonal, which can be extremely sparse.

In practice, an alternative, more flexible, and efficient representation of the convolution operator with RPN 2 involves applying a "*reshape*$(\cdot)$" post-processing operation to the data expansion function. This operation reshapes each expanded instance vector $\kappa\left(\mathbf{x}\xi(\mathbf{x})\right) = \mathbf{x}\mathbf{A}$, of length m', into a two-dimensional matrix of shape $p_{count} \times p$. Under this representation, the parameter reconciliation function can be simplified to an identity function, *i.e.,* $\psi(\mathbf{w}) = \mathbf{w} \in \mathbb{R}^p$, where the inner product with the expanded data matrix is subsequently processed by another "*reshape*$(\cdot)$" output-processing operator, converting the result back into vectors. This method significantly reduces storage and computational costs compared to the previously discussed representation.

Representing Pooling with Compression Function: The pooling operator in CNNs can be precisely represented using the geometric patch-based compression functions introduced in Section 6.1.3. In CNNs, these geometric patches typically have a cuboid shape, with pooling strides corresponding to the patch center distance hyper-parameters. Different pooling approaches can be implemented using various patch compression mappings defined in the previous Section 6.1.3.

Representing CNN with RPN 2: Based on the above analyses, as illustrated in the Plot (a) of Figure 22, we can represent the CNN model within RPN 2 using the following layers. The representation of the fully connected layers (*i.e.,* MLP) has been introduced in the previous paper [89] already, which will not repeated here.

- **Convolutional Layer**: Represented as a single-head layer in RPN 2 with: (1) identity data expansion function; (2) cuboid patch-based structural interdependence function (in padding mode); (3) identity parameter reconciliation function; (4) zero remainder function; and (5) reshape functions for both expansion post-processing and output-processing. For multi-channel CNNs, corresponding channel numbers can be used to define parameters in RPN 2. For models with skip-layer residual connections (*e.g.,* ResNet [26]), a linear remainder function can be used instead of the zero remainder function.
- **Pooling Layer**: Represented as an RPN 2 layer with: (1) cuboid patch-based data compression function (using numerical operator-based patch compression mappings); (2) identity interdependence function; (3) constant parameter reconciliation function; and (4) zero remainder function.

An example of the CNN's representation using RPN 2 is shown in Plot (a) of Figure 22. This configuration includes one convolutional layer, one pooling layer, and one feed-forward layer. A deeper CNN architecture can be similarly represented by adding multiple layers to this structure.

8.2 Unifying RNN with RPN 2

Recurrent neural networks (RNNs) denote a family of deep models that capture the internal transitional states of data sequences, which have been extensively used for the modeling of language [88], time series [10], and video sequences [77]. In this section, we will investigate to unify RNN with RPN 2's representation based on the component functions introduced in the previous sections.

8.2.1 Recurrent Neural Network (RNN)

In this part, we will examine the architecture of the RNN model, with a particular focus on the crucial recurrent state updating operator. This operator forms the cornerstone of RNNs, enabling them to process sequential data by maintaining and updating internal states across time steps.

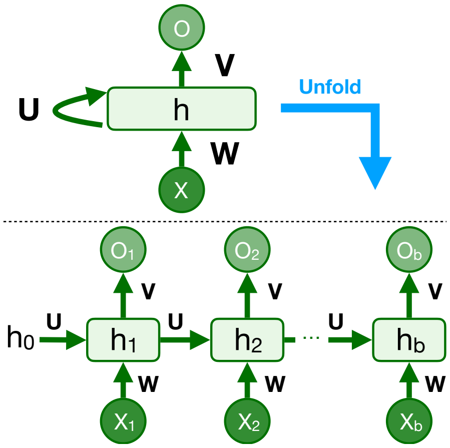


Figure 23: An illustration of RNN model.

RNN Model Architecture: As depicted in the right plot, the Recurrent Neural Network (RNN) model can be represented as a multi-layer architecture with full connections. In this structure, $\mathbf{x}_i \in \mathbb{R}^m$ with different subscripts represent the sequence inputs, $\mathbf{h}_i \in \mathbb{R}^{d_h}$ denotes the learned embeddings of the inputs, and $\mathbf{o}_i \in \mathbb{R}^n$ represents the corresponding outputs. The model's parameters are defined by three key matrices: $\mathbf{W} \in \mathbb{R}^{m \times d_h}$, which represents the parameters of the fully connected layer between input and hidden layers; $\mathbf{V} \in \mathbb{R}^{d_h \times n}$, which denotes the parameters of the fully connected layer between hidden and output layers; and $\mathbf{U} \in \mathbb{R}^{d_h \times d_h}$, which is the state transitional parameter unique to RNNs, allowing embedding state vectors to transition along the sequence. This recurrent structure, particularly the state transition facilitated by parameter $\mathbf{U}$, distinguishes RNNs from traditional feedforward models, enabling them to capture sequential dependencies in the data. By incorporating this feedback loop, RNNs can maintain and utilize information from previous time steps, making them particularly well-suited for processing sequential data such as time series or natural language.

Recurrent State Updating Operator: The RNN model architecture depicted in Figure 23 can be unfolded into a sequential structure. In this unfolded representation, $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_b \in \mathbb{R}^m$ denote the sequence inputs, while $\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_b \in \mathbb{R}^{d_h}$ represent the learned hidden state vectors connected via fully connected layers. These hidden state vectors are then projected to the corresponding outputs $\mathbf{o}_1, \mathbf{o}_2, \cdots, \mathbf{o}_b \in \mathbb{R}^n$. Notably, the parameters $\mathbf{W} \in \mathbb{R}^{m \times d_h}$, $\mathbf{U} \in \mathbb{R}^{d_h \times d_h}$, and $\mathbf{V} \in \mathbb{R}^{d_h \times n}$ are shared across all time steps, maintaining consistency in the network's behavior throughout the sequence.

Formally, for the i_{th} input vector $\mathbf{x}_i$, we can express its learned embedding vector $\mathbf{h}_i$ based on input $\mathbf{x}_i$ as follows:

$$\mathbf{h}_i = \mathbf{x}_i \mathbf{W} \in \mathbb{R}^{d_h}, \forall i \in \{1, 2, \cdots, b\}. \tag{172}$$

The primary distinction between Recurrent Neural Networks (RNNs) and other models discussed in both our previous work [89] and the current paper lies in the sequential dependence among inputs. In RNNs, the state of a later input instance is not solely dependent on itself, but also on the states of preceding inputs. This characteristic can be formally represented as follows:

$$\mathbf{h}'_i = \sigma\left(\mathbf{h}_{i-1}\mathbf{U} + \mathbf{h}_i\right), \forall i \in \{1, 2, \cdots, b\}. \tag{173}$$

We use the notations with the prime symbols, *i.e.*, $\mathbf{h}'_i$, to represent the input state vectors updated with their dependent conditions. For the initial input of the sequence, the embedding input vector $\mathbf{h}_0$ is typically assigned a dummy vector, such as a zero vector or a vector with random values. Subsequently, based on the learned embedding vector $\mathbf{h}_i$, the corresponding output vector $\mathbf{o}_i$ (for $\forall i \in 1, 2, \cdots, b$) is computed as:

$$\mathbf{o}_i = \text{softmax}\left(\mathbf{h}'_i \mathbf{V}\right) \in \mathbb{R}^n. \tag{174}$$

This formulation elegantly captures the essence of RNNs: their ability to process sequential data by maintaining a state that gets updated at each time step, allowing the network to retain information from previous inputs.

8.2.2 Representing RNN with RPN 2

In this part, we will explore how to represent the RNN model within the RPN 2 framework. Our primary focus will be on the hidden state recurrent updating process, as described in Equation (173).

We will investigate how to represent this crucial step using the component functions introduced in previous sections, thereby demonstrating how the unique sequential nature of RNNs can be captured within our unified framework.

Representing Recurrent Operator with Chain Structural Interdependence Function: In the recurrent state update equation (*i.e.,* Equation (173)), updating each state vector $\mathbf{h}_i'$ (for $\forall i \in 1, 2, \cdots, b$) requires both $\mathbf{h}_{i-1}$ and $\mathbf{h}_i$ as inputs. The state vectors across the data batch can be arranged into a matrix $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_b] \in \mathbb{R}^{b \times m}$, enabling concurrent updates as follows:

$$\mathbf{H}' = \left\langle \xi_i(\mathbf{H})^\top \kappa(\mathbf{H}), \psi(\mathbf{w}) \right\rangle + \pi(\mathbf{h}_i) = \sigma(\mathbf{A}_{\xi_i}^\top \mathbf{H}\mathbf{U} + \mathbf{H}), \tag{175}$$

where

- $\kappa(\mathbf{H}) = \mathbf{H}$: This denotes the identity data transformation function.
- $\xi_i(\mathbf{H}) = \mathbf{A}_{\xi_i} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$: This denotes the uni-directional chain structural interdependence function.
- $\psi(\mathbf{w}) = \mathbf{U} \in \mathbb{R}^{d_h \times d_h}$: This denotes the identity parameter reconciliation function that reshapes the input parameter vector into a matrix.
- $\pi(\mathbf{H}) = \mathbf{H}$: This denotes the linear remainder function (without dimension adjustment), and can be viewed as an identity function.

The output is typically processed through an activation function, such as $\sigma(\cdot)$ shown above. In this representation, the interdependence matrix models the uni-directional dependencies. For a bi-directional RNN, the interdependence matrix will also include ones in the lower off-diagonal entries, capturing the reverse directional relationships.

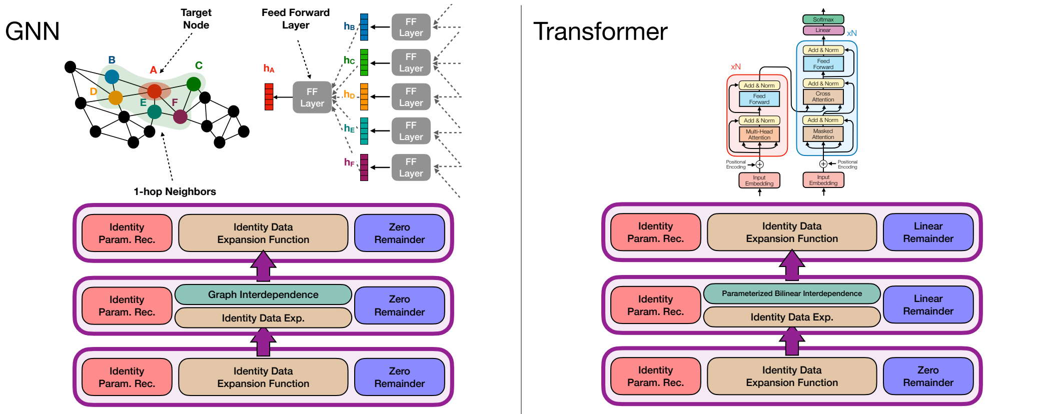
Representing RNN with RPN 2: Based on this analysis, as illustrated in Plot (b) of Figure 22, we propose the following representation of the RNN model using RPN 2. For an input batch with b sequential instances, RPN 2 defines the chain structural interdependence matrix of dimensions $d \times d$ to model their interdependence relationships:

- **Recurrent Layer**: Represented as a single-head and single-channel layer in RPN 2 with: (1) identity data expansion function; (2) chain structural interdependence function; (3) identity parameter reconciliation function; and (4) identity remainder function. The output of each layer is processed with an activation function.

The unified representation of RNN with RPN 2 is also illustrated in the Plot (b) of Figure 22, which involves one recurrent state updating layer. For the RNN with multi-layers, we can stack the above recurrent layers on top of each with perceptron layers inserted between them. Also both the input and output processing layers can be represented by the perceptron layer, which has been introduced in the previous paper [89] and will not be detailed again here.

8.3 Unifying GNN with RPN 2

In addition to the image and sequence data discussed previously, graph-structured data are prevalent in the real world, with notable examples including molecular graphs, online social networks, and interconnected websites. To address the unique challenges posed by such data, Graph Neural



(c) Representing GNN with RPN 2

(d) Representing Transformer with RPN 2

Figure 24: An illustration of representing GNN and Transformer with RPN 2.

Networks (GNNs) have emerged as a specialized family of deep learning models designed to handle graph-structured data with extensive connections. In this section, we will introduce GNN models and investigate how to unify them within RPN 2 canonical representation framework.

8.3.1 Graph Neural Network (GNN)

In this part, we will explore the architecture of the GNN models and delve into the spectral graph convolutional operator. This operator plays a crucial role in updating node representations by aggregating information from neighboring nodes within graph-structured data.

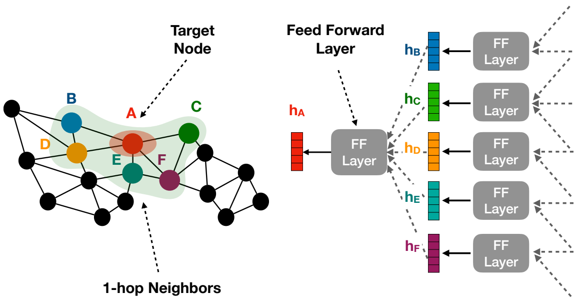


Figure 25: An illustration of GNN model.

GNN Model Architecture: Graph Neural Networks (GNNs) can be conceptualized as a generalization of Recurrent Neural Networks (RNNs), where the input structure evolves from a linear chain to an extensively interconnected graph. The right plot illustrates this concept with an example input graph, showcasing multiple nodes and their complex interconnections. Additionally, it depicts a potential deep GNN architecture designed for learning the embeddings of a target node within the graph. The process of learning the embedding vector for a target node in a GNN involves aggregating information from its surrounding neighbors. These neighboring nodes serve as inputs to a fully connected feed-forward layer in the network. Crucially, the embeddings of these neighbor nodes are themselves learned through a similar process, recursively incorporating information from their own neighbors. This recursive nature of information aggregation allows GNNs to capture complex, multi-hop relationships within the graph data.

Spectral Graph Convolution (SGC) Operator: In the aforementioned graph neural network architecture, the neighborhood aggregation operator (illustrated as the fully connected feedforward module in Figure 25) is formally known as the spectral graph convolutional operator.

Formally, given a graph $G = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and link set $\mathcal{E}$, we define $\Gamma(v) = \{u | u \in \mathcal{V} \wedge (u, v) \in \mathcal{E}\}$ as the set of neighbors for a target node $v \in \mathcal{V}$. Each node $v \in \mathcal{V}$ is initially represented by an input vector $\mathbf{x}_v \in \mathbb{R}^m$. The Graph Convolutional Network (GCN) model learns node embeddings by aggregating neighbor information through multiple layers of the spectral graph convolutional (SGC) operator as follows:

$$\mathbf{h}_v = \text{SGC}(\mathbf{x}_v | \Gamma(u)) = \sigma \left(\sum_{u \in \Gamma(v)} \frac{1}{|\Gamma(u)|} \mathbf{x}_u \mathbf{W} + \mathbf{x}_v \mathbf{W} \right) \in \mathbb{R}^n, \tag{176}$$

where $\mathbf{W} \in \mathbb{R}^{m \times n}$ represents the learnable parameters, and $\sigma(\cdot)$ denotes an activation function, such as sigmoid or ReLU.

Notably, the contribution of each neighbor node $u \in \Gamma(v)$ is weighted by the inverse of its degree, $\frac{1}{|\Gamma(u)|}$. This normalization scheme ensures that nodes with numerous connections (*i.e.,* the neighboring node u with a large node degree $|\Gamma(u)|$) contribute less to the aggregation, preventing highly connected nodes from dominating the information flow. This degree-based weighting mechanism helps balance the influence of nodes with varying connectivity levels in the graph structure.

8.3.2 Representing GNN with RPN 2

In this part, we will demonstrate how to represent the GNN model architecture using RPN 2. Specifically, we will interpret the spectral graph convolutional (SGC) operator in GNNs as a structural interdependence function defined for the graph data modality.

Representing SGC Operator with Graph Structural Interdependence Function: In GNN models, node representations can be learned concurrently in a data batch, which can significantly reduce computational time compared to individual node representation learning.

Formally, given an input graph $G = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$, we can organize all node raw features as a data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$, where $b = |\mathcal{V}|$ if the batch contains all nodes in the graph. The spectral graph convolutional (SGC) operator for all node representations can be represented as:

$$\mathbf{H} = \text{SGC}(\mathbf{X} | G) = \sigma \left(\mathbf{A} \mathbf{X} \mathbf{W} \right) \in \mathbb{R}^{b \times n}, \tag{177}$$

where $\mathbf{A} \in \mathbb{R}^{b \times b}$ is the row-normalized adjacency matrix describing connections among nodes within the graph. It is calculated as $\mathbf{A} = \mathbf{D}^{-1} \hat{\mathbf{A}} + \mathbf{I}$, where $\hat{\mathbf{A}}$ is the graph's adjacency matrix and $\mathbf{D}$ is the diagonal degree matrix with $\mathbf{D}(i,i) = \sum_j \hat{\mathbf{A}}(i,j)$. As noted in Section 5.2.9, matrix $\mathbf{A}$ can be composed with the graph-based structural interdependence function (*i.e.,* Equation (81)) with optional row-normalization, which models the interdependence relationships among instances.

Therefore, we can rewrite the SGC operator-based graph instance batch updating equation as:

$$\text{SGC}(\mathbf{X} | G) = \langle \kappa_{\xi}(\mathbf{X}), \psi(\mathbf{w}) \rangle = \langle \xi(\mathbf{X}) \kappa(\mathbf{X}), \psi(\mathbf{w}) \rangle = \sigma \left(\mathbf{A} \mathbf{X} \mathbf{W}^{\top} \right). \tag{178}$$

where

- $\xi(\mathbf{X} | G) = \mathbf{A} \in \mathbb{R}^{b \times b}$: This denotes the graph-based structural interdependence function.
- $\kappa(\mathbf{X}) = \mathbf{X} \in \mathbb{R}^{b \times m}$: This denotes the identity data expansion function.
- $\psi(\mathbf{w}) = \text{reshape}(\mathbf{w}) = \mathbf{W} \in \mathbb{R}^{n \times m}$: This represents the identity parameter reconciliation function.

For graphs with a large number of nodes and links, the normalized adjacency matrix $\mathbf{A}$ can be extremely large. In such cases, as proposed in the Graph-Bert paper [91], batches of small-sized subgraphs covering only the neighborhood can be sampled and fed into the model for node embedding vector updating, greatly reducing memory consumption.

Representing GNN with RPN 2: Based on the above description, we can represent the GNN model architecture with RPN 2 by selecting the following component functions to compose the model layers:

- **SGC Layer**: The RPN 2 layer with one single head and one channel, comprising: (1) graph-based structural interdependence function with optional row normalizations as the post-processing function, (2) identity data transformation function, (3) identity parameter reconciliation function, and (4) zero remainder function. The layer may also use an optional activation function for output processing. For models with skip-layer residual connections (similar to Graph-Bert [91]), a linear remainder function can be used instead of the zero remainder function.

An example of the GNN's representation using RPN 2 is shown in Plot (a) of Figure 24, which includes a single graph convolutional layer. To construct deeper GNN architectures, additional SGC-based data batch updating layers can be incorporated by adding corresponding layers to RPN 2.

8.4 Unifying Transformer with RPN 2

Since being processed in 2017, Transformer [75] has been the dominant backbone model used in building many AI models. In recent years, Transformer has demonstrated its effectiveness in processing the inputs in different modalities, including but not limited to images [16], point cloud [92] and graphs [91]. Meanwhile, we have also witnessed some criticisms about Transformer in terms of its extremely high time and space costs, which lead to the current huge demands of both computational facilities and energy consumptions. In this section, we will introduce the detailed components used in Transformer, and investigate to unify Transformer within the canonical representation of RPN 2, which may illustrate potential opportunities to address such weakness.

8.4.1 Transformer

In this part, we will first delve into the architecture of the Transformer model, with a particular focus on its pivotal component: the scaled dot-product attention mechanism. This mechanism forms the cornerstone of the Transformer's ability to process sequential data effectively.

Transformer Model Architecture: The right plot illustrates the Transformer model architecture, comprising an encoder (depicted in the left red block) and a decoder (shown in the right blue block). Both the encoder and decoder blocks incorporate several key functional components, including Multi-Head Attention, Feed-Forward layers, and Add & Norm (normalization) layers. The encoder's output serves as input to the decoder, facilitating the generation of the final output. In the decoder block, the Multi-Head Attention component undergoes slight modifications, resulting in Masked Attention and Cross Attention components. The Masked Attention involves adding masks to the attention mechanism, while the Cross Attention accepts inputs from multiple sources. A crucial feature of both encoder and decoder blocks is their incorporation of positional information through positional encoding. This positional data, combined with the input embedding, is fed into the attention component to enhance the learning process. The decoder's output undergoes further processing through Linear and Softmax layers to produce the final output. This sophisticated architecture enables the Transformer to effectively handle sequential data while capturing long-range dependencies, making it particularly effective for tasks such as machine translation and text generation.

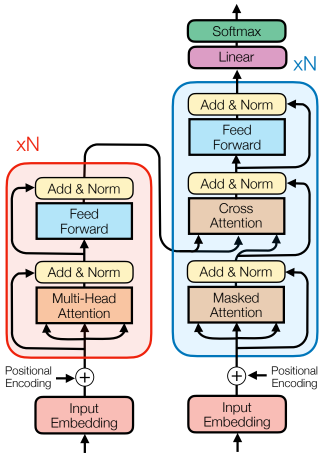


Figure 26: An illustration of the transformer model architecture.

Scaled Dot-Product Attention in Transformer: In our previous RPN paper [89], we have already demonstrated how to represent the MLP model within the RPN framework. Building upon that foundation, we will now focus on examining the attention mechanism integral to the Transformer model.

Formally, consider an input data batch $\mathbf{X} \in \mathbb{R}^{b \times m}$ containing b instances. The Transformer model processes and embeds this input into latent representations, taking into account the element-wise relationships within the input through its attention mechanism. In the case of single-head attention, we can represent the calculated pairwise attention matrix among the input elements as follows:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{r}}\right) \in \mathbb{R}^{b \times b}, \text{ where } \mathbf{Q} = \mathbf{X}\mathbf{W}_q \in \mathbb{R}^{b \times r}, \text{ and } \mathbf{K} = \mathbf{X}\mathbf{W}_k \in \mathbb{R}^{b \times r}. \tag{179}$$

In the attention matrix $\mathbf{A}$, each element $\mathbf{A}(i, j)$ represents the calculated attention score between the i_{th} and j_{th} instances of the input batch. The model employs learnable parameter matrices $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{m \times r}$ to compress the input batch $\mathbf{X}$ into query and key matrices $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{b \times r}$, respectively. This transformation allows the model to project the input into a space where relevant similarities can be more easily computed.

Building upon the attention matrix, we can formally express the learned representations of the input batch as follows:

$$\mathbf{H} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{r}}\right)\mathbf{V} \in \mathbb{R}^{b \times n}, \text{ where } \mathbf{V} = \mathbf{X}\mathbf{W}_v \in \mathbb{R}^{b \times n}. \tag{180}$$

In the above formula, matrix $\mathbf{W}_v \in \mathbb{R}^{m \times n}$ denotes the learnable parameters involved in defining the value matrix $\mathbf{V} \in \mathbb{R}^{b \times n}$ of the input batch.

8.4.2 Representing Transformer with RPN 2

In this part, we will explore how to represent the Transformer architecture using RPN 2 by defining its internal component functions. We will particularly focus on the scaled dot-product attention module, a key component of the Transformer model.

Representing Attention with Parameter Efficient Bilinear Interdependence Function: We can rewrite the attention matrix from Equation (179) by substituting $\mathbf{Q}$ and $\mathbf{K}$ with their detailed representations $\mathbf{X}\mathbf{W}_q$ and $\mathbf{X}\mathbf{W}_k$:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{X}\mathbf{W}_q(\mathbf{X}\mathbf{W}_k)^\top}{\sqrt{r}}\right) = \text{softmax}\left(\frac{\mathbf{X}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{X}^\top}{\sqrt{r}}\right). \tag{181}$$

The term $\mathbf{A} = \mathbf{X}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{X}^\top$ can be represented as a low-rank parameterized bilinear interdependence function on the input data batch $\mathbf{X}$, as introduced in Section 5.1.6. Here, $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{m \times r}$ denote the low-rank parameter matrices, whose product composes the bilinear parameter matrix $\mathbf{W} = \mathbf{W}_q\mathbf{W}_k^\top \in \mathbb{R}^{m \times m}$.

To handle the division by $\sqrt{r}$, we introduce a special normalization called scaled-softmax normalization:

$$\text{scaled-softmax}(\mathbf{A}|r) = \text{softmax}\left(\frac{\mathbf{A}}{\sqrt{r}}\right). \tag{182}$$

Thus, we can represent the input data batch updating in Transformer, *i.e.,* Equation (180), as:

$$\langle \kappa_\xi(\mathbf{X}), \psi(\mathbf{w}) \rangle = \langle \xi(\mathbf{X})\kappa(\mathbf{X}), \psi(\mathbf{w}) \rangle = \mathbf{A}\mathbf{X}\mathbf{W}^\top, \tag{183}$$

where

- $\xi(\mathbf{X}) = \mathbf{A} = \mathbf{X}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{X}^\top \in \mathbb{R}^{b \times b}$: This denotes the low-rank parameterized bilinear interdependence function with scalaed-softmax row normalization.

- $\kappa(\mathbf{X}) = \mathbf{X} \in \mathbb{R}^{b \times m}$: This denotes the identity data transformation function.
- $\psi(\mathbf{w}) = \mathbf{W} \in \mathbb{R}^{n \times m}$: This denotes the identity parameter reconciliation function.

Residual connections in Transformer can be effectively implemented with remainder functions. Other normalization functions in the module can be implemented with pre-, post-, and output-processing functions in RPN 2, as introduced in [89].

Representing Transformer with RPN 2: Based on this analysis, we can represent the Transformer model within RPN 2 by selecting the following component functions to compose the layers:

- **Attention Layer**: The RPN 2 layer with multi-head, comprising: (1) low-rank parameterized bilinear interdependence function with scalaed-softmax row normalization, (2) identity data transformation function, (3) identity parameter reconciliation function, and (4) linear remainder function. The layer may also use an optional normalization function for output processing.

An example of the Transformer's representation using RPN 2 is shown in Plot (b) of Figure 24, which includes a single transformer block. Additional attention layers in the Transformer can be represented by adding corresponding layers to RPN 2. The feed-forward layer in the Transformer is similar to that in an MLP and will not be discussed in detail here.

9 Empirical Evaluations of RPN 2

To evaluate the effectiveness of the proposed RPN 2 model for function learning tasks on different interdependent datasets, this section presents empirical studies conducted on real-world benchmark datasets across various modalities, including images, language, time-series and graphs. The organization of this section is as follows: Section 9.1 explores the RPN 2 model's performance on discrete classification tasks involving vision and language benchmark datasets, which exhibit grid- and chain-structured interdependence relationships, respectively. Moving beyond discrete data, Section 9.2 focuses on the RPN 2 model's application to continuous time-series prediction and graph-structured data classification tasks, both characterized by inherent chain- and graph-structured interdependence.

Beyond the main experimental results, we provide a comprehensive analysis of the method's performance, including learning convergence, parameter sensitivity, ablation studies, interpretability, and visualizations. These investigations offer deeper insights into the method's advantages and robustness when addressing function learning tasks across datasets with diverse interdependence structures.

9.1 Discrete Vision and Language Data Classification

Equipped with the grid and chain-based structural interdependence functions, the proposed RPN 2 model can effectively capture local interdependence relationships among image pixels and sequential interdependence relationships among words or tokens in documents. These capabilities are expected to enhance RPN 2's learning performance for vision and language classification tasks. To validate the effectiveness of the proposed RPN 2 model and its interdependence functions, extensive experiments were conducted on several benchmark image and language datasets.

This subsection is organized as follows. We begin by introducing the datasets and experimental setups. A detailed analysis of the grid-based structural interdependence function is presented, exploring factors such as local patch shapes, sizes, and packing strategies. For the chain-based structural interdependence function, we examine the impact of various chain-based interdependence matrix representations on language classification tasks. In addition to analyzing the interdependence functions, we investigate the influence of other model components and architectural variations on learning performance for vision and language classification. Finally, we report the main results of

Table 1: The table summarizes key statistical information for the discrete image and language datasets. These datasets are pre-partitioned into training and testing sets. For each image and document instance, the table also specifies their input and output sizes.

	Image Datasets		**Language Classification Datasets**		
	MNIST	**CIFAR-10**	**IMDB**	**AGNews**	**SST2**
Input Image Size	28 × 28	32 × 32 × 3	512 × 300	64 × 300	32 × 300
Output Class #	10	10	2	4	2
Train Instance #	60,000	50,000	25,000	120,000	67,349
Test Instance #	10,000	10,000	25,000	7,600	872

RPN 2, comparing its performance to several baseline methods, including MLP, the earlier version RPN (without interdependence functions), CNN, and RNN.

9.1.1 Dataset Description and Experiment Setups

Dataset Descriptions: Following our previous work [89], we evaluate the proposed method on two image benchmark datasets (MNIST and CIFAR-10) and three language datasets (IMDB, AGNews, and SST2). Comparisons are made with MLP, the previous RPN 2 method, CNN, and RNN. The basic statistical details of these datasets are summarized in Table 1.

- **Image Datasets**: For the MNIST and CIFAR-10 datasets, the images are flattened into vectors and normalized using mean-std normalization. No image augmentations (*e.g.,* horizontal/vertical flipping, rotation, or noise addition) were applied in these experiments.
- **Language Datasets**: For IMDB, AGNews, and SST2, we utilize the pre-trained GloVe 6B encoder to convert each token into a 300-dimensional embedding vector. To manage variable-length documents, dataset-specific maximum lengths are employed for truncation or padding: 512 for IMDB, 64 for AGNews, and 32 for SST2.

Experiment Setups: All image and language datasets used in the experiments come with predefined training and testing splits. For consistency and fairness, all baseline methods use the same data partitions. Model performance is evaluated using accuracy as the default metric.

9.1.2 Grid Interdependence Investigation: Patch Shapes, Sizes and Packing Strategies

In the RPN 2 model learning framework, images are represented as grid structures, where each pixel is uniquely identified by a coordinate tuple. The model captures local interdependence relationships between pixels through patches of distinct geometric shapes—including cuboids, cylinders, and spheres, which are defined as an ordered set of pixel coordinates in the underlying grid structure. While cuboid and cylindrical patches are particularly effective for image data analysis, spherical patches are better suited for 3D point-cloud data—a topic beyond the scope of this section but slated for future investigation. Here, we explore how the choice of patch shape (cuboid or cylinder), size, and packing strategy impacts the performance of the RPN 2 model in image classification tasks.

To systematically evaluate these effects, we conduct extensive experiments on the CIFAR-10 dataset using the RPN 2 model with a consistent architecture comprising three main components: (1) *Attribute Interdependence Layers*: Two layers implementing grid-based structural attribute interdependence functions, each utilizing identity data transformation, identity parameter reconciliation, and zero remainder functions, with both layers maintaining 128 output channels; (2) *Compression Layer*: A single layer employing grid-based data compression function, constant eye parameter reconciliation, and zero remainder functions, outputting 128 channels; and (3) *Perceptron Layers*: Three successive layers utilizing identity data transformation, identity parameter reconciliation, and zero remainder functions, with output dimensions progressively decreasing from 1024 to 512, and finally to 10, respectively.

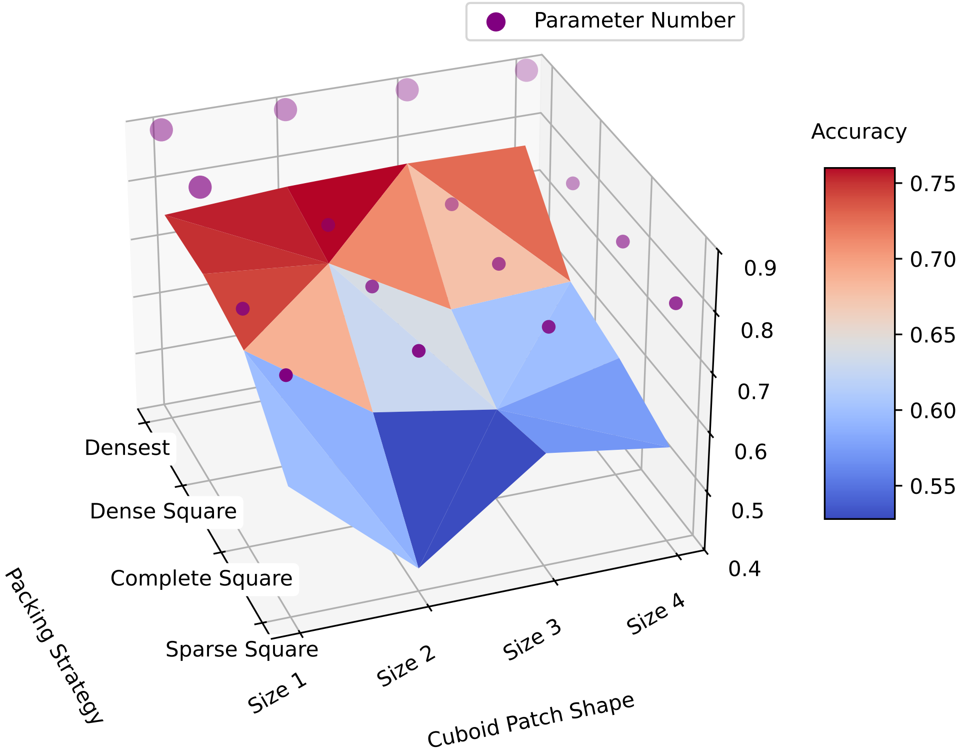


Figure 27: Analysis of the size and packing strategies of the cuboid patch shape. Size 1: $p_h = p'_h = 1$, $p_w = p'_w = 1$; Size 2: $p_h = p'_h = 2$, $p_w = p'_w = 2$; Size 3: $p_h = p'_h = 3$, $p_w = p'_w = 3$; and Size 4: $p_h = p'_h = 4$, $p_w = p'_w = 4$. For all these shapes, we have $p_d = p'_d = 0$ by default.

Our comprehensive evaluation of RPN 2's performance examines variations in both patch shapes and their packing strategies within the underlying grid structure. This analysis focuses particularly on the impact of patch shape modifications in two critical components: the grid-based structural attribute interdependence function and the grid-based data compression function. The comparative performance results for cuboid and cylinder patch shapes are presented in Figures 27 and 28, respectively.

Patch Shapes: Our analysis reveals that patch shape significantly influences both learning performance and the number of learnable parameters. Specifically, a cuboid patch with dimensions $p_h = p'_h = 4, p_w = p'_w = 4$ (and 128 channels) covers $10,368$ feature map elements, while a cylinder patch with radius $p_r = 4$ (and 128 channels) encompasses $6,272$ patch elements. Using the *sparse square* packing strategy for projections between 128 input and output channels, the interdependence function requires $2,021,514$ learnable parameters for cuboid patches versus $1,485,834$ for cylinder patches. Under the *densest packing* strategy, RPN 2 achieves an accuracy of 0.771 with cuboid patches and 0.780 with cylinder patches.

Patch Sizes: The relationship between patch size and RPN 2's learning performance is multifaceted, as it influences not only the patch dimensions but also the center distances in packing and compression functions. For the interdependence function with cuboid patches, we find that minimal dimensions ($p_h = p'_h = 1$ and $p_w = p'_w = 1$) yield superior performance across all packing strategies except *densest packing*. This pattern holds true for cylindrical patches as well. The enhanced perfor-

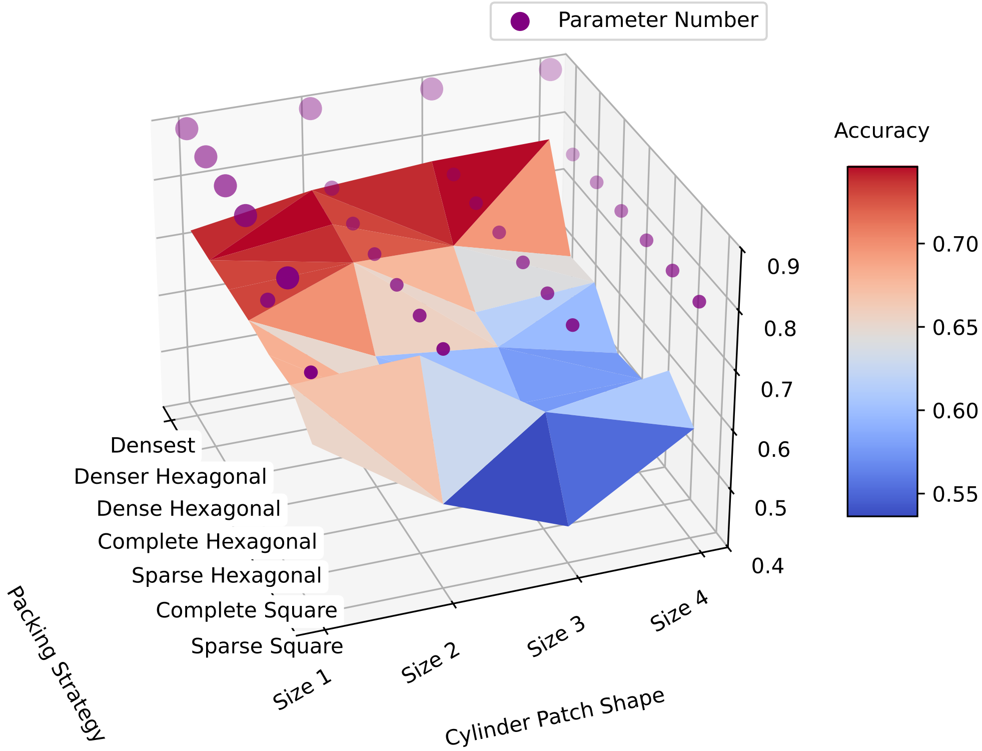


Figure 28: Analysis of the size and packing strategies of the cylinder patch shape. Size 1: $p_r = 1$; Size 2: $p_r = 2$; Size 3: $p_r = 3$; and Size 4: $p_r = 4$. For all these shapes, we have $p_d = p'_d = 0$ by default.

mance likely stems from the smaller packing parameters, which minimize information loss during processing.

Packing Strategies: Among the investigated patch-related parameters, packing strategies demonstrate the most substantial impact on model performance. Our experiments show that the "*densest packing*" and "*dense/denser packing*" strategies consistently outperform sparse or complete packing approaches. These results suggest that maintaining some degree of redundancy is crucial for effectively capturing interdependence relationships and extracting relevant information from image data. The *densest packing* strategy achieves the highest accuracy scores, particularly with larger patch configurations—such as cuboid patches with dimensions $p_h = p'_h = 3$ and $p_w = p'_w = 3$, or cylinder patches with radius $p_r = 4$. However, this superior performance comes at a computational cost, as these configurations require over 135M learnable parameters, significantly more than other packing strategies.

9.1.3 Grid Interdependence Layer Depth and Width Investigation

We further investigate the effects of model depth (*i.e.,* number of interdependence layers) and width (*i.e.,* number of channels) using RPN 2 with a cylinder patch of size $d_r = 4$ and the *densest* packing strategy. Our investigation encompasses architectures with 1, 2, 3, and 4 grid-based interdependence layers, featuring channel configurations of 64, 128, 256, and hybrid combinations. Throughout these variations, we maintain consistent compression and perceptron layers with identical output

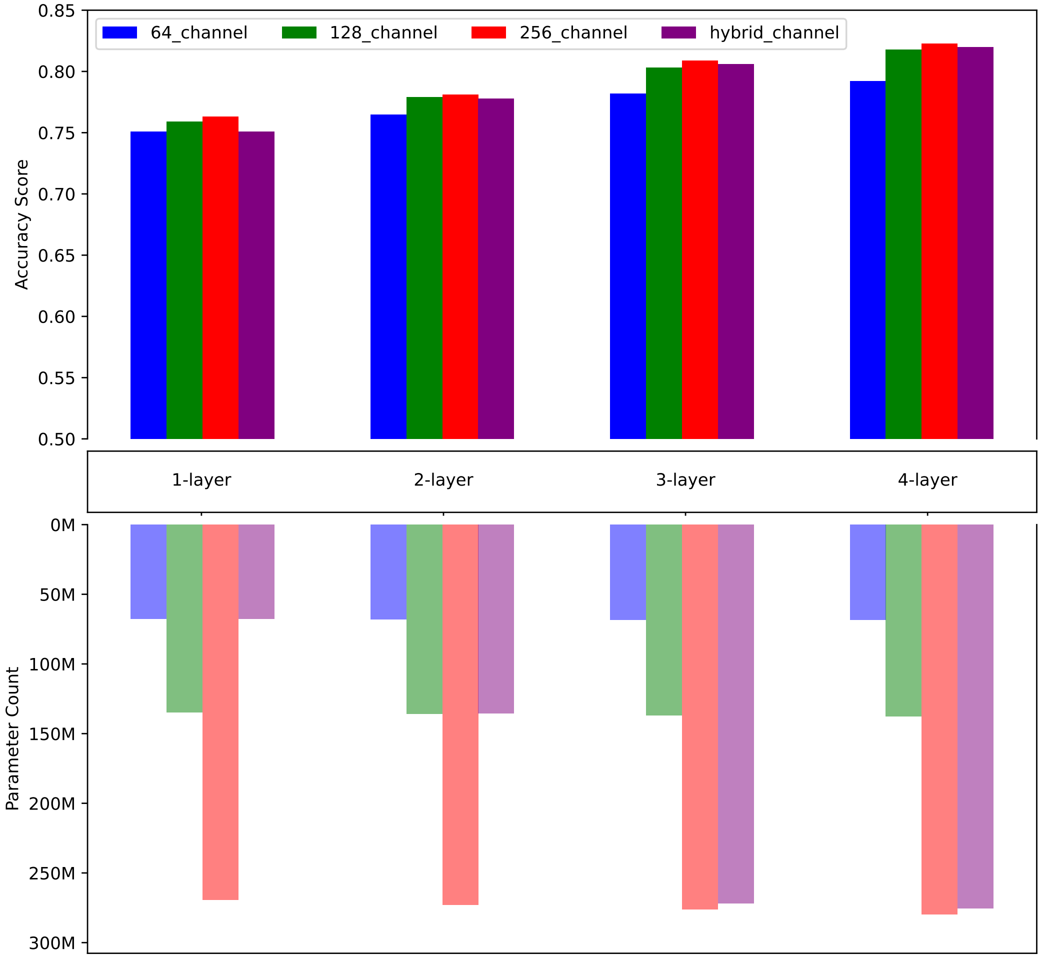


Figure 29: Analysis of RPN 2's performance as a function of model depth (number of interdependence layers) and width (number of channels). The hybrid channel configurations implement progressively increasing channel numbers across layers: (1) one-layer model: $[3, 64]$; (2) two-layer model: $[3, 64, 128]$; (3) three-layer model: $[3, 128, 128, 256]$; and (4) four-layer model: $[3, 128, 128, 256, 256]$.

dimensions. Figure 29 presents both the accuracy scores achieved by these architectural variations and their corresponding numbers of learnable parameters.

The experimental results demonstrate that increasing both depth and width enhances RPN 2's learning capacity, resulting in steady improvements in accuracy scores. Among all tested configurations, the architecture with 4 layers and 256 channels achieves superior performance compared to other variants. However, we observe that increasing model width leads to a dramatic expansion in the number of learnable parameters, with the majority of these additional parameters concentrated in the subsequent perceptron layers.

9.1.4 Grid based Compression Layer Analysis

Our previous analyses employed a single compression layer to process feature maps learned by grid-based interdependence layers before flattening and feeding them to subsequent perceptron layers. Here, we extend our investigation to examine the effects of incorporating multiple compression layers between interdependence layers, enabling progressive compression of learned feature maps throughout the network.

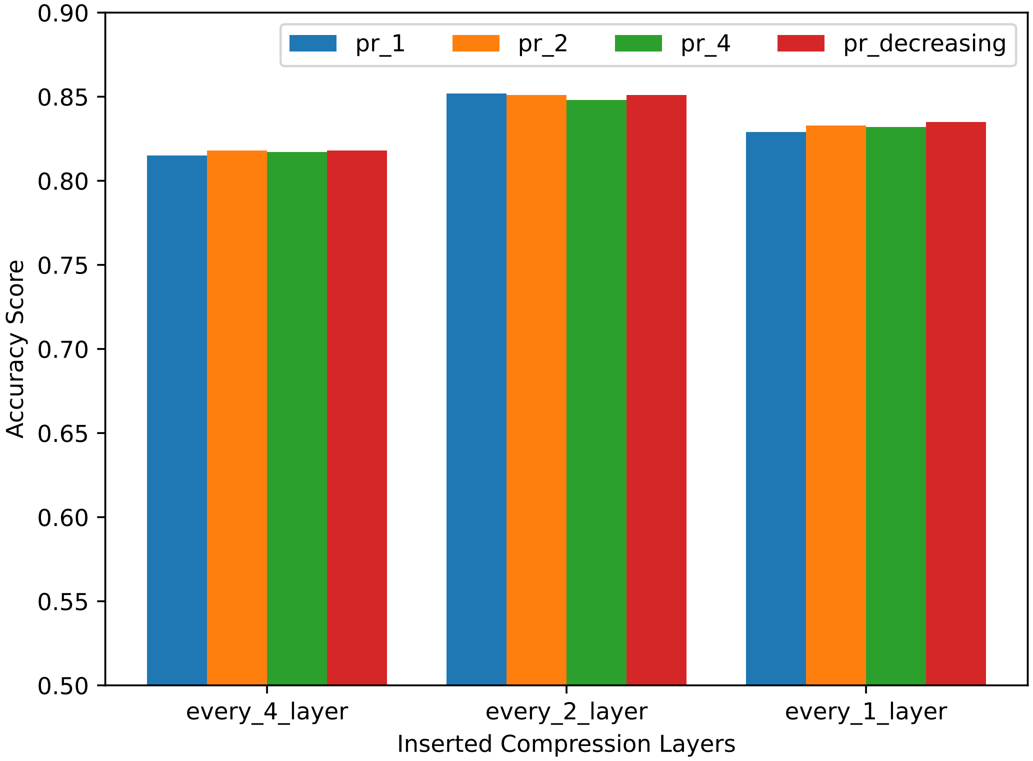


Figure 30: Analysis of compression layer configurations in RPN 2, where compression layers are strategically inserted between interdependence layers for feature map compression. The insertion frequency follows three patterns: “Every 4 layer”, “Every 2 layer”, or “Every 1 layer”, indicating a compression layer follows every 4, 2, or 1 interdependence layers, respectively. The parameter p_r denotes the cylinder patch radius used in compression functions, while pr_decreasing” represents an adaptive configuration starting with $p_r = 4$ and halving at each successive compression layer. All compression layers maintain consistent patch packing strategy parameters with $cd_h = cd_w = 2$.

The analysis provided in this part focuses on the RPN 2 model with 4 layers, where we systematically evaluate compression layer insertion frequencies of every 1, 2, or 4 interdependence layer(s). Additionally, we investigate the impact of varying cylinder patch sizes in compression layers, departing from our previous assumption of identical patch sizes between compression and interdependence layers. Our investigation encompasses compression layers with fixed patch radii ($p_r = 1$, $p_r = 2$, $p_r = 4$) and an adaptive configuration where p_r begins at 4 and halves after each compression layer. For example, in a 4-layer RPN 2 model with compression layers after each interdependence layer, the adaptive configuration yields four compression layers with progressively decreasing patch radii of 4, 2, 1, and 1.

Results shown in Figure 30 reveal three key findings: (1) inserting compression layers every two interdependence layers yields superior performance compared to other frequencies; (2) when compression layers are added every two interdependence layers, cylinder patches with $p_r = 1$ achieve optimal performance; and (3) while large patch sizes ($p_r = 4$) initially degrade model performance, implementing progressive size reduction across compression layers leads to improved results.

9.1.5 Chain Interdependence Investigation: Uni-directional, Bi-directional, Single-Hop, and Multi-Hop

Expanding beyond our analysis of grid-based structural interdependence functions on image data, we evaluate the effectiveness of chain-based structural interdependence functions on language data.

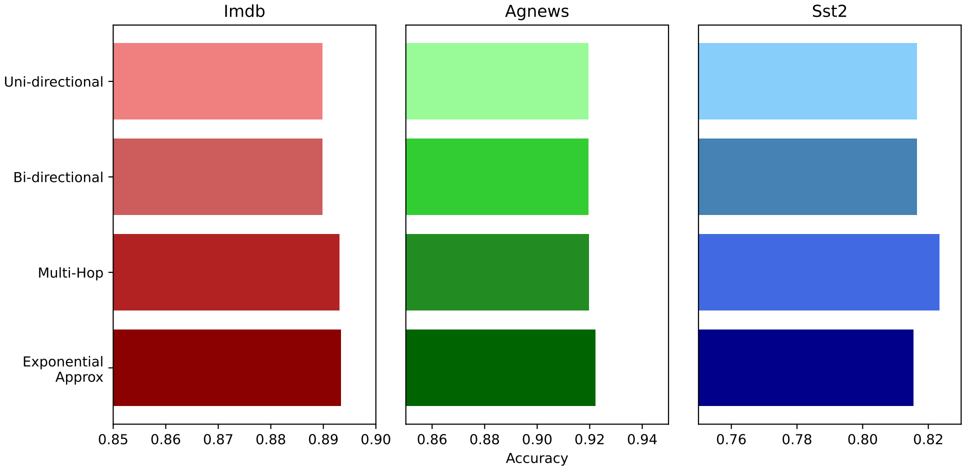


Figure 31: Analysis of chain-based structural interdependence functions' impact on RPN 2's classification performance across IMDB, AGNews, and SST2 datasets. Performance comparison encompasses four distinct interdependence modes: "Uni-directional", "Bi-directional", Multi-Hop", and "Exponential Approx".

Our investigation employs a five-layer RPN 2 architecture comprising: an input perceptron layer, two chain-based structural interdependence function layers with an intervening perceptron layer, and an output perceptron layer. The model processes classification by aggregating learned token representations from the second interdependence layer via summation before feeding them to the output perceptron layer. Figure 31 presents RPN 2's performance results using chain structural interdependence functions across three datasets: IMDB, AGNews, and SST2.

Our analysis reveals several key insights about chain-based structural interdependence functions when their outputs are aggregated through summation. Notably, uni-directional and bi-directional chain structural interdependence functions achieve comparable performance levels. Multi-hop interdependence functions (with hop counts $h = 3$ or $h = 5$) demonstrate significant performance improvements over their single-hop counterparts, with particularly pronounced gains on the SST2 dataset. Furthermore, the exponential approximation of multi-hop interdependence functions achieves comparable—and in some cases slightly superior—performance on IMDB and AGNews datasets, validating the effectiveness of our approximation approach.

9.1.6 The Main Results of RPN 2 on Vision Data Classification

Table 2 presents a comprehensive performance comparison between RPN 2 and several baseline methods. For comparison, we include the standard MLP, our previous RPN model [89], CNN (for image classification tasks), and LSTM (for language classification tasks). While we also evaluated vanilla RNN for language classification, its performance was significantly inferior to LSTM, hence we report LSTM results as the representative sequential model baseline.

As shown in Table 2, the proposed RPN 2, equipped with interdependence functions, significantly outperforms the earlier RPN model introduced in [89], particularly on the CIFAR-10 dataset. These results highlight the effectiveness and importance of the interdependence functions in capturing relationships among attributes and instances. Furthermore, RPN 2 achieves comparable—and occasionally superior—performance to CNN and LSTM models. This demonstrates RPN 2's potential not only to theoretically unify CNN and RNN architectures within its canonical representation but also to empirically deliver performance on par with these established backbone models in practical applications.

Table 2: The learning performance of the comparison methods on discrete image and language datasets is evaluated using Accuracy as the default metric. For each dataset, the best-performing method is highlighted in bold math font.

Models	Image Datasets		Language Datasets		
	MNIST	CIFAR10	IMDB	AGNews	SST2
MLP	9.82×10^{-1}	5.63×10^{-1}	8.85×10^{-1}	9.21×10^{-1}	8.05×10^{-1}
Previous RPN [89]	9.86×10^{-1}	5.61×10^{-1}	8.86×10^{-1}	9.19×10^{-1}	8.07×10^{-1}
CNN (Image) & LSTM (Language)	9.96×10^{-1}	8.47×10^{-1}	$\mathbf{8.93 \times 10^{-1}}$	9.14×10^{-1}	$\mathbf{8.27 \times 10^{-1}}$
RPN 2	$\mathbf{9.98 \times 10^{-1}}$	$\mathbf{8.52 \times 10^{-1}}$	$\mathbf{8.93 \times 10^{-1}}$	$\mathbf{9.22 \times 10^{-1}}$	8.23×10^{-1}

9.1.7 Visualization of Cylinder Patches of Images

In Figure 32, we present several examples of the raw images from the CIFAR-10 dataset and their corresponding expansions using the grid-based interdependence function. The first and third columns contain 10 raw images sampled from the CIFAR-10 dataset, while the second and fourth columns display their respective expanded images.

For each pixel in a raw image, the grid-based structural interdependence function pads the pixels within the cylindrical patch of radius $p_r = 4$ (with a diameter of 9). To visualize these patches, we include some pixels with dummy values to enclose each cylindrical patch within a cuboid of size $9 \times 9 \times 3$. Compared to the raw image pixels, the expanded images still retain the outlines of the objects. Moreover, for each pixel in the raw images, their expansions provide richer contextual information about their local learning neighborhood. This additional information enables the RPN 2 model to utilize a more detailed learning context for classifying these images, leading to improved performance compared to the previous RPN model [89].

9.1.8 Visualization of Chain-based Multi-Hop Interdependence Functions

In Figure 33, we illustrate the chain-based structural interdependence matrices used in RPN 2 for classifying the IMDB dataset. The left plot represents the multi-hop interdependence matrix with hop $h = 5$, while the right plot shows its exponential approximation.

From the visualization, both matrices exhibit non-zero entries concentrated along their diagonal regions. However, as indicated by the diagonal colors, the non-zero entries in the multi-hop interdependence matrix have relatively larger values compared to those in the exponential approximation matrix, which penalizes higher-order matrix powers using constant factors. Additionally, we calculate the percentages of non-zero entries in these matrices. The exponential approximation interdependence matrix contains approximately 7.096% non-zero entries, significantly higher than the sparse multi-hop interdependence matrix, which has only 1.165% non-zero entries.

9.2 Time-Series Data Prediction and Graph Data Classification

In addition to discrete image and language data, the RPN 2 model has also demonstrated effectiveness in learning from continuous data and graph-structured data. In this subsection, we present the experimental evaluation of RPN 2 and several comparison methods on continuous time-series benchmark datasets related to finance and traffic over time, as well as on graph-structured benchmark datasets pertaining to citation networks.

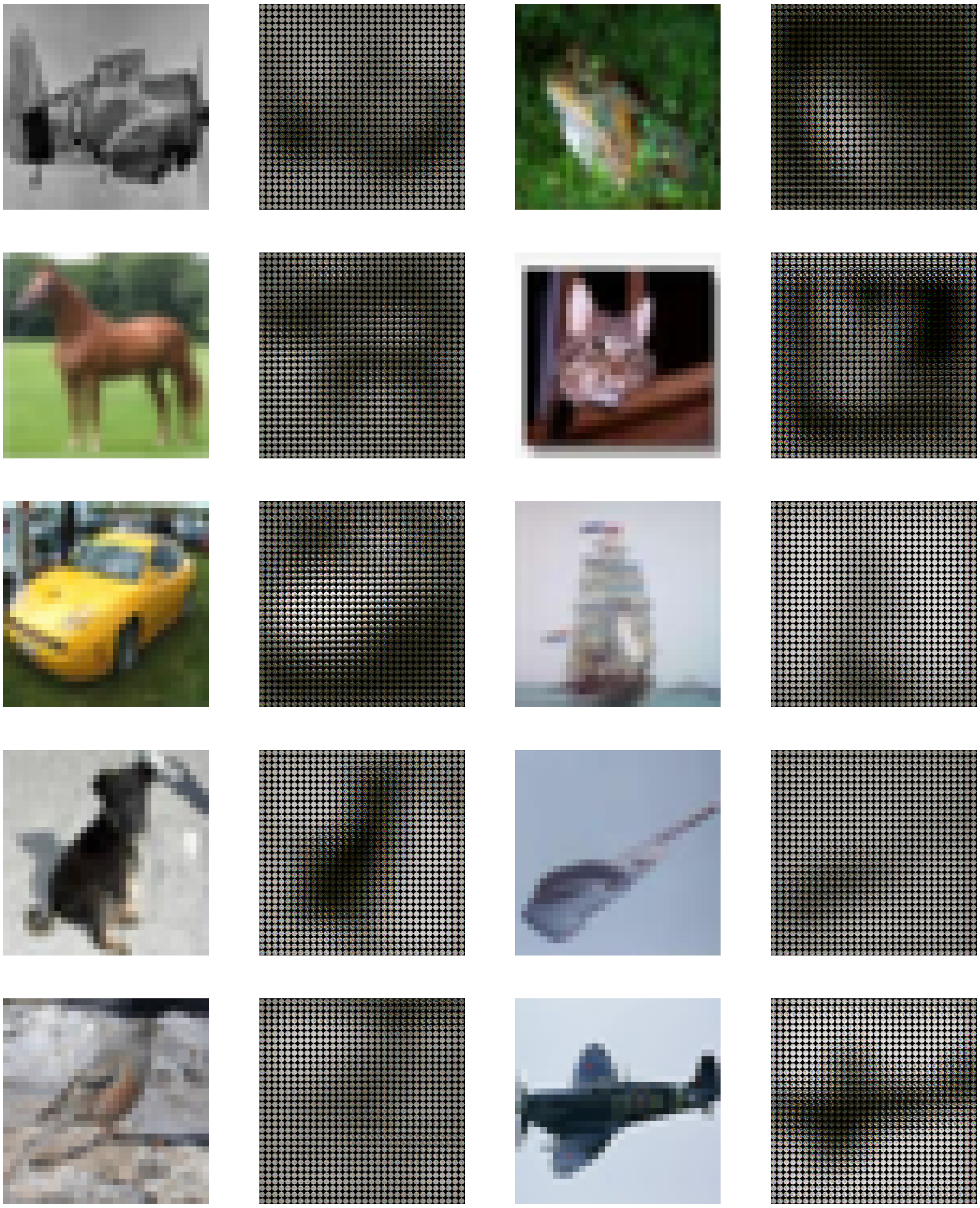
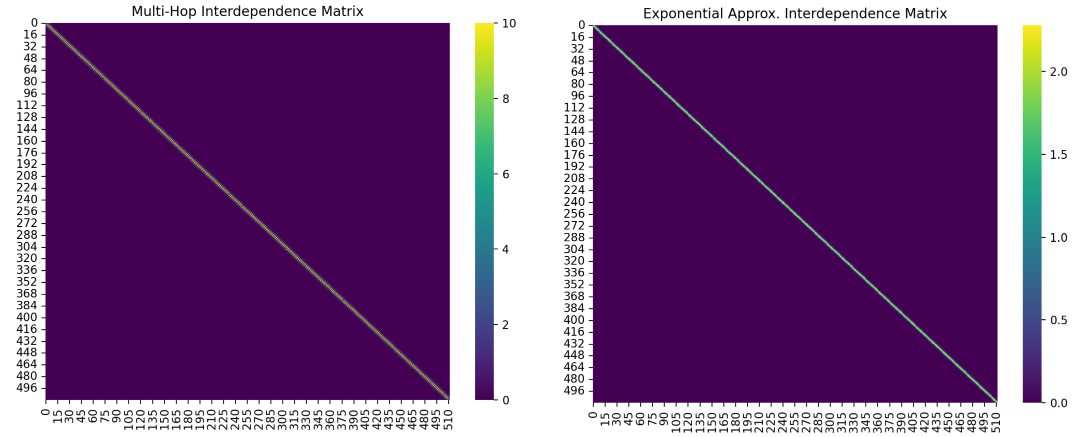


Figure 32: An illustration of raw images (shown in the first and third columns) from the CIFAR-10 dataset and their expanded counterparts (shown in the second and fourth columns), generated using the grid-based structural interdependence functions with a cylindrical patch of radius $p_r = 4$.

We begin by providing a brief overview of these time-series and graph benchmark datasets. Next, we present the experimental results obtained by the comparison methods. Finally, we illustrate some of the learned interdependence matrices specifically for the graph data.



(a) Multi-Hop Interdependence Matrix

(b) Exponential Approx. Interdependence Matrix

Figure 33: An illustration of the chain-based structural interdependence matrices for the IMDB language dataset classification. The percentages of non-zero elements in the two matrices are as follows: (1) multi-hop ($h = 5$) interdependence matrix: 1.165%, and (2) exponential approximation interdependence matrix: 7.096%.

Table 3: The table summarizes the key statistics of the raw time-series benchmark datasets. For the Stock and ETF datasets, instances vary in length, with the mean and standard deviation (mean±std) provided. In contrast, instances in the LA and Bay Area traffic datasets are aligned and have identical lengths. Additionally, the table includes the number of attributes recorded for each data instance at each timestamp.

	Time-Series Datasets			
	Stock	**ETF**	**Traffic-LA**	**Traffic-Bay**
Instance #	7,163	1,344	207	325
Timestamp #	$2,078 \pm 833$	$1,908 \pm 893$	34,272	52,116
Attribute #	6	6	1	1

9.2.1 Time-Series Dataset Description and Experiment Setups

Dataset Descriptions: The time-series data studied in the experiments include stock market and urban traffic datasets. Specifically, the stock market datasets comprise Nasdaq-traded stocks and ETFs, while the urban traffic datasets include data collected from the Los Angeles region and the San Francisco Bay Area. Basic statistical information about these datasets is summarized in Table 3. For the stock and ETF datasets, each instance has 7 attributes at each timestamp, corresponding to "*Open*", "*Close*", "*High*", "*Low*", "*Volume*", and "*OpenInt*". In this project, we specifically predict the "*Open*" attribute, which represents the opening price of the stocks and ETFs for each trading day. Meanwhile, for the traffic data, there is a single attribute representing the accumulated traffic counts within a specific time period, which is used as the target for prediction in the experiments.

Experimental Setups: In the experiments, these datasets were cleaned, pre-processed, and partitioned according to their temporal granularities: per day for stock and ETF data, and per minute for traffic data. For each instance in the finance datasets (*i.e.,* the stock and ETF datasets), we extracted the recent one-year trading records and partitioned them into input-output pairs. The input spans 10 prior trading days, while the output represents the next day's opening price, with a time gap of 1. Similarly, for the traffic data (*i.e.,* the LA and Bay Area datasets), all available records were used to create input-output pairs with the same setup as the finance data. The performance of all comparison methods is evaluated using MSE (Mean Squared Error) as the default metric.

Table 4: The learning results of the comparison methods on the time-series benchmark datasets are presented in the table. The scores represent the MSE achieved by each method. For the best performance on each dataset, the corresponding scores are highlighted in bold math font.

Models	**Time-Series Prediction Datasets**			
	Stock	**ETF**	**Traffic-LA**	**Traffic-Bay**
MLP	1.96×10^{-4}	3.36×10^{-4}	$\mathbf{1.70 \times 10^{-1}}$	8.13×10^{-2}
Previous RPN [89]	1.94×10^{-4}	$\mathbf{3.31 \times 10^{-4}}$	1.71×10^{-1}	8.12×10^{-2}
RNN	$\mathbf{1.85 \times 10^{-4}}$	3.47×10^{-4}	1.74×10^{-1}	8.13×10^{-2}
RPN 2	$\mathbf{1.85 \times 10^{-4}}$	3.35×10^{-4}	1.74×10^{-1}	$\mathbf{7.99 \times 10^{-2}}$

9.2.2 The Main Results of RPN 2 on Time-Series Prediction

In Table 4, we present the forecasting results of RPN 2 on the time-series datasets. For comparison, the table also includes the performance of RNN, the previous RPN, and MLP, where MLP and RPN project the input sequence of length 10 to the desired output. The scores in the table represent the MSE achieved by these methods on the four time-series benchmark datasets.

From the results, we observe that RPN 2 achieves comparable performance to RNN on the Stock, ETF, and Traffic-LA datasets. However, compared to MLP and the previous RPN model, the advantages of RNN and RPN 2 in sequence data modeling are less pronounced. This is likely because MLP and RPN, built with fully connected layers, can effectively utilize the historical record information. Notably, on the Traffic-Bay dataset, RPN 2 outperforms the other methods. This improvement can be attributed to the relatively longer sequences in the Traffic-Bay dataset, which provide more training data samples to enhance the learning process of RPN 2.

9.2.3 Graph Dataset Description and Experiment Setups

Dataset Descriptions: We studied three citation graph benchmark datasets: Cora, Citeseer, and Pubmed. Basic statistical information about these datasets is provided in Table 5. Each dataset consists of a set of nodes representing papers, connected by citation links among them. Specifically, the Cora, Citeseer, and Pubmed datasets contain $2,708$, $3,327$, and $19,717$ nodes, connected by $10,556$, $9,104$, and $88,648$ links, respectively. In addition to the graph structure, each node is associated with a feature vector (a bag-of-words representation of the paper's textual description, weighted by TF-IDF) and a label denoting the paper's topic. Detailed statistical information is also included in Table 5.

Experimental Setups: In the experiments, subsets of nodes were sampled for training and testing the graph learning and node classification models. Specifically, for each class in the three graph datasets, 20 instances were sampled for training, and a set of $1,000$ nodes was randomly sampled for testing. For the graph neural networks and RPN 2, training was conducted using semi-supervised and transductive learning settings, effectively incorporating unlabeled instances into the model training. Conversely, the MLP and the previous RPN model were trained using classic supervised and inductive learning settings, relying solely on the labeled training set.

9.2.4 The Main Results of RPN 2 on Graph Classification

In Table 6, we present the learning performance of RPN 2 and the baseline methods on the three graph benchmark datasets. For each dataset, the table also includes the model architectures, with the numbers inside brackets representing the input, hidden, and output dimensions, respectively.

Table 5: The table provides an overview of the graph benchmark datasets, including the number of nodes and links, as well as the input attributes and output label classes. For the experiments, we randomly sampled 20 node instances per class as the training set for each graph dataset, and 1,000 random nodes as the testing set.

	Graph Datasets		
	Cora	**Citeseer**	**Pubmed**
Node #	2,708	3,327	19,717
Link #	10,556	9,104	88,648
Input Dim.	1,433	3,703	500
Output Dim.	7	6	3
Train #	140	120	60
Test #	1,000	1,000	1,000

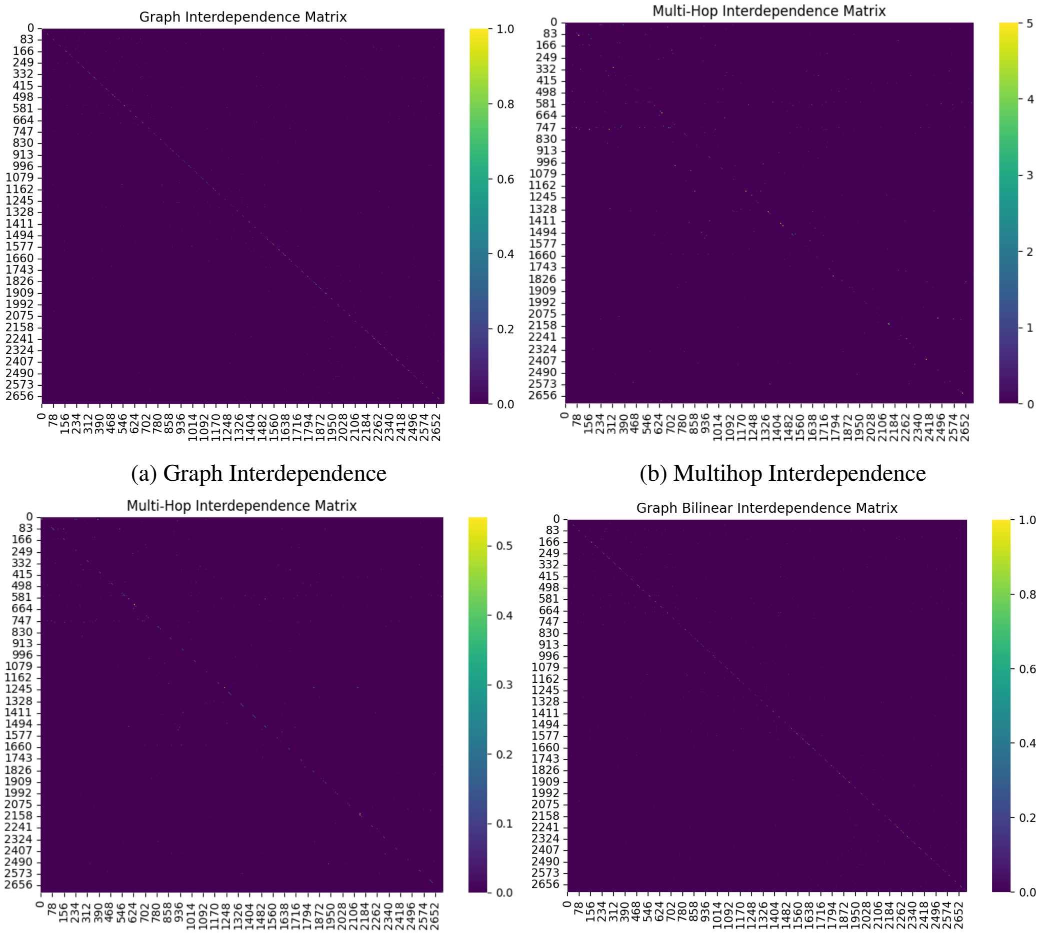
Table 6: The classification performance of the comparison methods on nodes in the graph benchmark datasets is reported. All comparison methods share the same input, hidden, and output layer dimensions, with detailed architectural information provided below the graph datasets.

Models	**Graph Datasets**		
	Cora	**Citeseer**	**Pubmed**
	[1433, 16, 7]	[3703, 16, 6]	[500, 16, 3]
MLP	6.61×10^{-1}	3.05×10^{-1}	3.83×10^{-1}
Previous RPN [89]	6.82×10^{-1}	3.17×10^{-1}	4.06×10^{-1}
GCN	8.15×10^{-1}	7.03×10^{-1}	7.90×10^{-1}
RPN 2 (graph interdependence)	8.22×10^{-1}	$\mathbf{7.09 \times 10^{-1}}$	7.96×10^{-1}
RPN 2 (hybrid interdependence)	$\mathbf{8.30 \times 10^{-1}}$	7.05×10^{-1}	$\mathbf{8.02 \times 10^{-1}}$

The reported results show that the RPN 2 model, equipped with graph-based interdependence functions, achieves a significant performance improvement compared to both the previous RPN model and the MLP model. Additionally, RPN 2 achieves comparable performance to the GCN model across these datasets. Furthermore, the table includes the learning performance of RPN 2 using a hybrid interdependence function, which combines the graph interdependence function with a bilinear interdependence function, using the product as the fusion function. The results indicate that the hybrid interdependence function slightly enhances the performance of RPN 2 on the Cora and Pubmed datasets, demonstrating the effectiveness of bilinear functions in capturing interdependence relationships within input data batches.

9.2.5 Graph Interdependence Matrix Visualization

In Figure 34, we display the graph-based interdependence matrices calculated using various graph structural interdependence functions. Specifically, Plot (a) is based on the graph adjacency matrix, Plots (b) and (c) represent the multi-hop and PageRank graph interdependence matrices, and Plots (d) and (e) correspond to the bilinear and hybrid graph interdependence matrices.



(a) Graph Interdependence

(b) Multihop Interdependence

(c) Pagerank Interdependence

(d) Hybrid Interdependence

Figure 34: An illustration of the graph, multi-hop graph, PageRank, and hybrid interdependence matrices (combining graph and bilinear methods) computed on the Cora dataset is provided. The non-zero entry ratios of these matrices are 0.110%, 0.454%, 0.783%, and 0.110%, respectively. (For better visibility of the non-zero matrix entry values, please zoom in on the figure.)

From the visualizations, the value scales of the entries in the multi-hop interdependence matrix are relatively larger compared to the other matrices. In terms of sparsity, the multi-hop and PageRank interdependence matrices have more non-zero entries than the one-hop and hybrid interdependence matrices. The hybrid interdependence matrix is constructed by masking the bilinear interdependence matrix with a binary graph interdependence matrix, where non-zero entries are column-normalized. Some of these matrices are highly sparse. For better visibility of the matrix entry values, it is recommended to zoom in on the figure to examine the sparsely distributed non-zero entries.

10 Interpretation of RPN 2 with Interdependence Functions

Empirical evaluations provided in the previous section show that incorporating data interdependence functions significantly enhances the learning performance of RPN 2, especially when compared to the previous RPN model introduced in [89]. In this section, we discuss interpretations of RPN 2 with data and structural interdependence functions from both theoretical machine learning and biological neuroscience perspectives. These interpretations also highlight the motivations and advantages of integrating interdependence functions within the RPN 2 model architecture.

10.1 Theoretic Machine Learning Interpretations

The RPN 2 model introduced in previous sections effectively captures interdependent relationships among both instances and attributes, making it adaptable to both *inductive* and *transductive* learning settings. In the following, we use the *transductive* learning setting as an example to examine the theoretical learning performance of RPN 2 with instance interdependence functions; similar results also apply to RPN 2 with attribute interdependence functions in other learning settings.

Formally, in the transductive learning setting, let the dataset be $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, consisting of n data instances, with a subset of these instances having known labels for model training, denoted as $\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m\} \subset \mathcal{D}$. For each instance $\mathbf{x}_i \in \mathcal{T}$, its known label vector is represented by $f(\mathbf{x}_i)$. As introduced in [89], the learning loss incurred by the model $g(\cdot|\mathbf{w})$ on the dataset $\mathcal{D}$ can be formally represented as follows:

$$\underbrace{\int_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}) \left\| g(\mathbf{x}|\mathbf{w}) - f(\mathbf{x}) \right\| \mathrm{d}\mathbf{x}}_{\text{overall error } \mathcal{L}} = \underbrace{\int_{\mathbf{x} \in \mathcal{T}} p(\mathbf{x}) \left\| g(\mathbf{x}|\mathbf{w}) - f(\mathbf{x}) \right\| \mathrm{d}\mathbf{x}}_{\text{empirical error } \mathcal{L}_{em}} + \underbrace{\int_{\mathbf{x} \in \mathcal{D} \setminus \mathcal{T}} p(\mathbf{x}) \left\| g(\mathbf{x}|\mathbf{w}) - f(\mathbf{x}) \right\| \mathrm{d}\mathbf{x}}_{\text{expected error } \mathcal{L}_{exp}}, \tag{184}$$

where $p(\mathbf{x})$ denoting the sampling probability for instance $\mathbf{x} \in \mathcal{D}$.

Motivated by recent work [19], we aim to derive the generalization error bound of RPN 2 with interdependence functions to interpret its learning performance as follows:

$$\mathcal{L}_{exp} \leq \mathcal{L}_{em} + \text{generalization error bound}. \tag{185}$$

The term "*generalization error bound*" in the above equation can be derived based on various learning theories, such as VC Dimension [74, 8] and Rademacher Complexity [4]. These theories consider different factors—such as model architecture, component modules, and input data—in defining generalization error bounds, which will be discussed in detail below.

10.1.1 Generalization Error-Bound based on VC Dimension

To streamline the analysis, we consider a binary classification transductive learning problem using the RPN 2 model, which includes identity expansion, instance interdependence, identity reconciliation, zero remainder, and K layers as an example.

Formally, given a data batch $\mathbf{X} \in \mathbb{R}^{b \times d_0}$, where $d_0 = m$ represents the input dimension, we can represent the RPN 2 model as follows:

$$g_K \circ g_{K-1} \circ \cdots \circ g_1 : \mathbb{R}^{b \times d_0} \to \{+1, -1\}^b, \tag{186}$$

where the k_{th}-layer defines a mapping $g_k : \mathbb{R}^{b \times d_{k-1}} \to \mathbb{R}^{b \times d_k}$, *i.e.,*

$$g_k(\mathbf{X}|\mathbf{w}_{\psi,k}) = \langle \kappa_{\xi,k}(\mathbf{X}), \psi_k(\mathbf{w}_{\psi,k}) \rangle + \pi_k(\mathbf{X}). \tag{187}$$

The corresponding hypothesis class of the above model can be represented as

$$\mathcal{H} = \left\{ g(\mathbf{X}|\mathbf{w}, K) = g_K \circ g_{K-1} \circ \cdots \circ g_1 : \mathbb{R}^{b \times m} \to \{+1, -1\}^b \right\}, \tag{188}$$

which together with the output label "sign" function actually define the VC-dimension of the model:

$$\begin{aligned} \text{VC-dimension}(\text{sign} \circ \mathcal{H}) &= \min \left\{ m, \text{rank}(\mathbf{A}_{\xi_i}), \{d_k\}_{k \in \{1, \cdots, K\}} \right\} \\ &\leq \min \left\{ \text{rank}(\mathbf{A}_{\xi_i}), d_{K-1} \right\}. \end{aligned} \tag{189}$$

Based on the above bound, as indicated in [19], for any $\delta \in (0, 1)$, with probability $1 - \delta$, the expected generalization error for any model $g \in \text{sign} \circ \mathcal{H}$ satisfies

$$\mathcal{L}_{exp}(g) - \mathcal{L}_{em}(g) \leq \sqrt{\frac{8}{m}\left(\min\{\text{rank}(\mathbf{A}_{\xi_i}), d_{K-1}\} \cdot \ln(em) + \ln\left(\frac{4}{\delta}\right)\right)}. \tag{190}$$

The error bound derived above offers valuable insights into the RPN 2 model:

- On rank($\mathbf{A}_{\xi_i}$) $\leq b$: The previous RPN model [89], which lacks the capability to model interdependence relationships, can be viewed as a special case of RPN 2 with an identity interdependence matrix, *i.e.,* $\mathbf{A}_{\xi_i} = \mathbf{I} \in \mathbb{R}^{b \times b}$, yielding a default rank of b. This suggests that the new RPN 2 model—and all the backbone models discussed in previous sections that can be represented as RPN 2—achieves a tighter error bound by incorporating interdependence than the previous RPN model, along with all models previously discussed in [89] that can be represented by RPN.
- On rank($\mathbf{A}_{\xi_i}$) v.s. d_{K-1}: If $d_{K-1} <$ rank($\mathbf{A}_{\xi_i}$), the introduced interdependence function becomes redundant; however, if $d_{K-1} \geq$ rank($\mathbf{A}_{\xi_i}$), the interdependence function reduces the model's error bound. In other words, the model architecture itself influences the effectiveness of the interdependence function.

10.1.2 Generalization Error-Bound based on Rademacher Complexity

Compared to the classic VC dimension theory, Rademacher complexity analysis provides a data-dependent bound for broader hypothesis classes. Building on the RPN 2 model discussed above, we derive its generalization error bound based on Rademacher complexity.

Formally, consider the constrained hypothesis classes $\mathcal{H}^{\beta,\omega} \subset \mathcal{H}$, where model parameters satisfy $\|\mathbf{w}k\| \infty \leq \omega$ and the optional bias term satisfies $\|\mathbf{b}_k\|_1 \leq \beta$ for each layer $k \in 1, 2, \cdots, K$. As analyzed in [19], the Rademacher complexity of this restricted hypothesis class can be represented as follows:

$$\mathcal{R}_{m,n}(\mathcal{H}^{\beta,\omega}) \leq \frac{c_1 n^2}{m(n-m)} \left(\sum_{k=0}^{K-1} c_2^k \|\mathbf{A}_{\xi_i}\|_{\infty}^k\right) + c_3 c_2^K \|\mathbf{A}_{\xi_i}\|_{\infty}^K \|\mathbf{A}_{\xi_i}\mathbf{X}\|_{2\to\infty} \sqrt{\log(n)}, \tag{191}$$

where $c_1 = 2L\beta$, $c_2 = 2L\omega$, $c_3 = L\omega\sqrt{\frac{2}{d}}$, and L is the Lipschitz constant of the layer's optional activation function. Furthermore, following the derivations in [17], for any $\delta \in (0, 1)$, with probability $1 - \delta$, the generalization error for any model $g \in \mathcal{H}^{\beta,\omega}$ satisfies the following:

$$\mathcal{L}_{exp}(g) - \mathcal{L}_{em}(g) \leq \mathcal{R}_{m,n}(\mathcal{H}^{\beta,\omega}) + c_4 \frac{n\sqrt{\min\{m, n-m\}}}{m(n-m)} + c_5 \sqrt{\frac{n}{m(n-m)} \ln\left(\frac{1}{\delta}\right)}, \tag{192}$$

where c_4 and c_5 are constants with the value bounds: $c_4 < 5.05$ and $c_5 < 0.8$.

Based on the generalization error bound above, we gain insights, particularly from Equation (191), where both $\|\mathbf{A}_{\xi_i}\|_{\infty}$ and $\|\mathbf{A}_{\xi_i}\mathbf{X}\|_{2\to\infty}$ influence the tightness of the error bound:

- On the norm term $\|\mathbf{A}_{\xi_i}\|_{\infty}$: This norm of the interdependence matrix $\mathbf{A}_{\xi_i}$ reflects the largest absolute row sums of the matrix. Therefore, to tighten the error bound, normalizing the matrix $\mathbf{A}_{\xi_i}$ may be necessary in practical applications.
- On the norm term $\|\mathbf{A}_{\xi_i}\mathbf{X}\|_{2\to\infty}$: This norm of the interdependence matrix represents $\sup_{\|\mathbf{z}\|_2=1} \|\mathbf{A}_{\xi_i}\mathbf{X}\mathbf{z}\|_{\infty}$, which can be relaxed as follows:

$$\|\mathbf{A}_{\xi_i}\mathbf{X}\|_{2\to\infty} = \max_j \|(\mathbf{A}_{\xi_i}\mathbf{X})_{\cdot,j}\|_2 \leq \|\mathbf{A}_{\xi_i}\|_2 \max_j \|\mathbf{X}_{\cdot,j}\|_2 \leq \|\mathbf{A}_{\xi_i}\|_2 \|\mathbf{X}\|_{2\to\infty}. \tag{193}$$

In other words, both the matrix $\mathbf{A}_{\xi_i}$ and the input data batch $\mathbf{X}$ affect the error bound. Normalizing $\mathbf{A}_{\xi_i}$ can effectively reduce its 2-norm term $\|\mathbf{A}_{\xi_i}\|_2$. As for $\|\mathbf{X}\|_{2\to\infty}$, data batches with smaller row norms, balanced row magnitudes, or properties such as sparsity, low rank, and orthogonality result in smaller norm values.

10.2 Biological Neuroscience Interpretations

In addition to theoretical machine learning interpretations, the interdependence function components in the RPN 2 model architecture may also simulate certain compensatory functions of biological nervous systems, including (but not limited to) sensory integration in the brain cortex, relational memory in the hippocampus, and working attention mechanisms.

10.2.1 Understanding Interdependence from Sensory Integration at Brain Cortex

The brain integrates interdependent information from multiple sensory inputs (such as vision, hearing, and touch) in multimodal areas of the cortex, including the parietal and prefrontal cortices. These regions serve as hubs where sensory inputs from diverse sources converge, enabling the brain to form unified representations of objects and events. During sensory information integration in the cortex, the brain naturally captures relationships among these multimodal sensory sources.

As shown in Figure 35, we illustrate a lateral view of the human brain, highlighting its four main lobes: the frontal, parietal, temporal, and occipital lobes (also illustrated in Figure 37). The frontal and parietal lobes are separated by the central sulcus, a groove between tissues, where the sensory cortex (in pink) and motor cortex (in purple) are located adjacent to each other on either side of the sulcus. In the right panel of Figure 35, we see the primary sensory cortex with an orderly (inverted) tactile representation, extending from the toe (at the top left of the cerebral hemisphere) to the mouth (at the bottom right). Each cerebral hemisphere in the primary somatosensory cortex represents tactile sensations from the opposite (contralateral) side of the body.

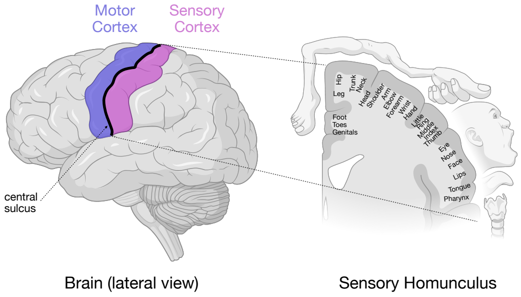


Figure 35: Brain Sensory Cortex.

The amount of primary somatosensory cortex dedicated to a body part is not proportional to the size of the body surface but, rather, to the relative density of tactile receptors. Body parts with a higher density of tactile receptors, such as the lips and hands, occupy larger areas in the somatosensory cortex, reflecting their heightened sensitivity to tactile stimulation.

With this diverse and extensive sensory information, the brain rarely processes inputs independently. Instead, it fuses them, captures interdependent relationships, and generates an optimal response for the motor systems. This integration and processing of multimodal sensory inputs mirror how structured models in machine learning capture interdependencies across different data attributes and instances.

10.2.2 Understanding Interdependence from from Hippocampus and Relational Memory

The hippocampus plays a crucial role in relational memory, enabling the brain to encode relationships between objects, contexts, and spatial locations. This function is especially important for integrating interdependencies across time and space, allowing the brain to learn sequences, temporal patterns, and associations between events.

As illustrated in Figure 36, the hippocampus is located within the temporal lobe (see also Figure 37). Key structures within the temporal lobe that support long-term memory include the hippocampus and surrounding regions, such as the perirhinal, parahippocampal, and entorhinal cortices. The hippocampus, along with the dentate gyrus, has a curved shape reminiscent of a seahorse. Humans and other mammals possess two hippocampi, one on each side of the brain. As part of the limbic system, the hippocampus plays a central role in consolidating information from short-term to long-term memory and in spatial memory, which supports navigation. In addition to the hippocampus, regions like the prefrontal and visual cortices also contribute to explicit memory; however, these will not be discussed in this paper.

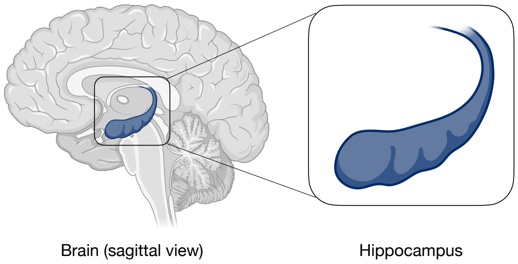


Figure 36: Brain Hippocampus.

The memory mechanism, supported by the hippocampus and other cortical areas, enables the brain to encode, store, retain, and retrieve information and past experiences, capturing interdependencies both explicitly and implicitly among instances and attributes. For example, during spatial navigation, the hippocampus integrates environmental information (attributes) with specific locations or landmarks (instances), allowing for memory retrieval based on complex interdependencies between these elements. This is analogous to how the RPN 2 model captures interdependence in structured data through the data and structural interdependence functions introduced in this paper.

10.2.3 Understanding Interdependence from Attention Mechanisms

In addition to sensory information integration and memory, attention mechanisms in the brain—particularly involving the prefrontal cortex and posterior parietal cortex—enable selective processing of interdependent information by dynamically modulating neural activity. By focusing on specific attributes or instances while filtering out irrelevant details, the brain can prioritize important interdependent relationships, such as the correlation between auditory and visual stimuli within a given context.

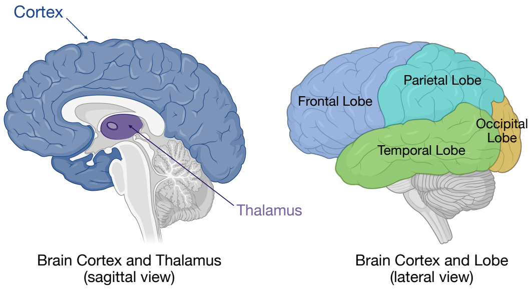

Cortex
Thalamus
Frontal Lobe
Parietal Lobe
Occipital Lobe
Temporal Lobe
Brain Cortex and Thalamus (sagittal view)
Brain Cortex and Lobe (lateral view)

Figure 37: Brain Cortex and Thalamus.

Attention is the behavioral and cognitive process of selectively focusing on specific aspects of information—whether subjective or objective—while disregarding other perceivable information. This process involves a complex network of brain regions that work together to regulate and sustain focus. Key areas include the prefrontal cortex, parietal cortex, anterior cingulate cortex, thalamus, and basal ganglia, some of which are shown in Figure 37. In particular, the thalamus, the purple region depicted in the figure, consists of two oval collections of nuclei that make up most of the diencephalon's mass. Often described as a relay station, the thalamus directs nearly all sensory information (except for olfactory signals) to the cortex, making an initial stop in the thalamus before reaching its cortical destination. The thalamus is subdivided into specialized nuclei, each handling specific types of sensory information, which it routes to the appropriate area in the cortex for further processing.

Supported by the thalamus and other cortical regions, the brain's attention mechanisms are essential for capturing interdependencies among sensory information and sequential events. Attention enables the brain to prioritize relevant information, filter out distractions, and integrate sensory inputs and events over time—key to understanding relationships between different stimuli.

11 Intellectual Merits, Limitations and Future Work Timeline

In this section, we discuss the intellectual merits of the RPN 2 model equipped with interdependence functions. Additionally, we address the current model's limitations, highlighting potential directions for future development and the next phase of this project.

11.1 Intellectual Merits

In this paper, we propose a redesign of the RPN 2 model architecture by incorporating data interdependence functions that model relationships among both attributes and instances. Based on empirical experiments and theoretical analyses provided in previous sections, these interdependence functions significantly enhance the learning performance of the RPN 2 model. Below, we summarize the intellectual merits of the newly proposed RPN 2 model.

Theoretical Merits: Unlike the previous RPN model [89], which assumes attributes and instances are independent and identically distributed, the newly designed RPN 2 model effectively captures interdependent relationships among both attributes and instances through a set of interdependence functions defined using information from the input data batch. These data interdependence functions significantly expand the modeling capabilities of RPN 2 for complex function learning tasks on interdependent data. The theoretical analyses provided in this paper also offer insights into defining optimal interdependence functions that lead to tighter generalization error bounds based on both VC dimension and Rademacher complexity theories. From a biological neuroscience perspective, these interdependence functions emulate certain compensatory functions of the biological nervous system within the RPN 2 model.

Technical Merits: The RPN 2 model introduces a diverse family of interdependence functions that capture relationships among attributes and instances from various perspectives, including the input data batch, underlying topological and geometrical structures, and combinations of these. These interdependence functions enable unification of diverse backbone models, such as CNNs, RNNs, GNNs, and Transformers (and their variants), as discussed in this paper. We demonstrate that these backbone models share similar architectures, with key differences arising from how the interdependence functions are defined. These observations provide valuable insights for designing the future "Transformer-Next" new backbone models.

Computational Merits: The interdependence functions introduced here compute matrices to model relationships among instances and attributes. These computed interdependence matrices are typically small and often sparse across many learning settings, resulting in minimal storage requirements. Additionally, because interdependence functions operate on the data batch as input—applying attribute interdependence prior to data transformation—they save considerable computational time compared to operations performed on transformed data batches. As with the previous paper [89], the RPN 2 model architecture allows for computations of different component functions to be distributed across multiple chips, machines, or cloud platforms, enhancing learning efficiency and safeguarding data privacy and model parameter security.

11.2 Limitations

When implementing the RPN 2 architecture and its component modules, we identified several limitations, particularly in modeling capabilities for dynamic data, learning algorithms, and the potential deployment of large-scale intelligent systems.

Learning Limitations: One challenge encountered in the implementation of the RPN 2 model lies in adapting loss functions, optimizers, and error backpropagation-based learning algorithms. While conventional representation learning-based loss functions, optimizers, and algorithms are applicable to the current function learning task, they exhibit certain inconsistencies with function-oriented learning models. In both this paper and the previous work [89], our function learning models require compressing the input data batch into the output space using perceptron-based layers (involving identity data transformation, identity reconciliation, and zero remainder functions). Although such layers are necessary for traditional representation learning models, they perform minimal data transformation in function learning models, significantly reducing the data batch's representational capabilities as dimensions are compressed.

Modeling Limitations: This paper successfully unifies several dominant backbone models within the canonical RPN 2 representation. However, challenges remain in representing RNN models with

RPN 2, as the current model lacks dynamic processing capabilities, requiring temporal interdependence in RNNs to be converted into spatial interdependence instead. This highlights a limitation of RPN 2 in modeling "temporal dynamics", a key factor in developing future world models with spatial intelligence. To address this, we plan to redesign modules in the current RPN 2 model to enable "temporal dynamics" modeling, which will be a primary focus of our future work.

Large-Scale Intelligent Systems: To demonstrate the effectiveness of our proposed techniques, we aim to build a large-scale intelligent system based on the RPN 2 model, showcasing its multimodal modeling capabilities, learning performance advantages, and inherent parameter efficiency. Creating such systems will require redesigning many component functions and models to ensure learning efficiency when handling large-scale data and vast numbers of parameters. Once the above limitations in learning and modeling are addressed, we will initiate the large-scale intelligent system project, which will be developed in parallel with the current RPN 2-based backbone framework project.

11.3 Future Work Timeline

In the upcoming year of 2025, we plan to address the current limitations of the RPN 2 model and the TINYBIG V0.2.0 library. Our primary focus will be on refining the RPN 2 model framework and developing application projects that leverage the enhanced RPN 2 model and TINYBIG V0.2.0 toolkit.

Framework Enhancement Projects: Based on our current development pace, we estimate to spend another six months on addressing the learning limitations in the current RPN 2 model. Specifically, we aim to explore function learning-oriented loss functions, optimized objective functions, and model learning algorithms. A new version of RPN 2, featuring these learning-related enhancements, is expected for release by mid-2025. Additionally, we plan to incorporate "time" and related functions into the RPN 2 model to enable dynamic learning scenarios. Modeling "temporal dynamics" will require a substantial redesign of the current RPN 2 functions and components, which will be time-intensive. Following the resolution of learning limitations, we anticipate another six months to integrate "temporal dynamics" into the model architecture, with a target release by the end of 2025.

Application Projects: In parallel with addressing the challenges in learning and modeling, we plan to undertake several system-building projects to conduct preliminary testing of RPN 2 on large-scale datasets with one or a few modalities. We expect to initiate a large-scale intelligent system project by the end of 2025, a project that will likely take several years due to its anticipated challenges and complexities. Technical reports on the progress of the RPN 2 and intelligent system projects will be released as new developments become available.

12 Related Work

In this section, we briefly discuss existing work related to this paper, including various backbone models proposed for data across different modalities and recent advancements in multi-modality foundation model learning.

12.1 Related Backbone Models

With the integration of data interdependence functions, the RPN 2 model proposed in this paper has demonstrated the capability to unify several dominant backbone models, including convolutional neural networks, recurrent neural networks, graph neural networks, and Transformers. In this section, we briefly introduce these backbone models and highlight key papers that have contributed critical technical breakthroughs.

12.1.1 Convolutional Neural Networks

Originally defined in the 19th century in Fourier's work [21], convolution has become a widely used mathematical operator in both continuous signal processing and discrete image processing.

In traditional computer vision, convolution is extensively applied in tasks such as image blurring, sharpening, resizing, and edge detection. Manually defined convolution kernels can provide meaningful physical interpretations but may have limited applicability for diverse vision tasks. To address this limitation, Yann LeCun introduced the convolutional neural network (CNN), known as LeNet [43], proposing that convolution kernels be defined as learnable parameters that can be automatically learned from input data. These learnable convolution kernels significantly improve the learning performance of vision models [55] and expand their applicability to various vision-related tasks [83, 2]. Following LeNet, additional CNN variants have been developed, such as AlexNet [40] and VGG16 [69]. AlexNet, with its 5 convolutional layers, achieved 84.6% accuracy on ImageNet, while VGG16, with 13 convolutional layers, further boosted accuracy to 92.7%. However, as model architectures became deeper, performance degradation emerged, where deeper models tended to show higher training and testing errors than shallower ones. To address this issue, Kaiming He introduced ResNet [26], incorporating skip-layer residual connections into the CNN architecture, which increased performance to 96.4% on ImageNet.

12.1.2 Recurrent Neural Networks

The study of recurrent neural networks (RNNs) originated from research in associative memory. Frank Rosenblatt [65] introduced a three-layer perceptron neural network with recurrent connections in the middle layer. Another foundation of associative memory came from statistical mechanics with the Ising model [9], which modeled thermal equilibrium. Later, Roy J. Glauber [23] extended the Ising model by adding a time component, allowing for temporal evolution. Building on the Ising model, Sherrington and Kirkpatrick developed the Sherrington-Kirkpatrick model as an exactly solvable model of spin glass, featuring an energy function with multiple local minima. Based on this model, John Hopfield introduced the Hopfield network with binary activation functions [30], later extended to continuous activation functions [31]. In recent years, Hopfield has further investigated methods to increase memory storage capacity by modifying network dynamics and energy functions [41, 42, 62]. With the resurgence of neural networks in the 1980s, new RNN architectures, such as the Jordan network [35] and Elman network [18], emerged to study cognitive psychology. To address gradient explosion and vanishing issues, Hochreiter and Schmidhuber [29] introduced the Long Short-Term Memory (LSTM) network, which has since become the dominant RNN model. To model bidirectional dependencies in input sequences, the Bidirectional RNN (Bi-RNN) [68] was developed, using two RNNs to process inputs in different directions. For a simplified LSTM structure, the Gated Recurrent Unit (GRU) [12] was introduced as an alternative RNN architecture. More recently, Gu and Dao proposed the Mamba state space model [25, 13], which incorporates recurrent models for sequence data by updating states over time.

12.1.3 Graph Neural Networks

Unlike images and language, graphs, as topological structures, lack a fixed order of nodes [22], necessitating distinct model designs. Before the advent of graph neural networks (GNNs), graph learning algorithms were primarily developed based on topological structures. Yan and Han proposed gSpan [84], a graph-based substructure pattern mining algorithm that extracts frequent subgraphs for feature learning. Sun introduced path-based algorithms [72] to extract features from graphs for various learning tasks. With the rise of neural networks, unsupervised algorithms for learning graph representations emerged, such as DeepWalk [57] and node2vec [24]. To effectively leverage labeled node information, the Graph Convolutional Network (GCN) [37] was developed to learn node representations by aggregating information from neighboring nodes. The Graph Attention Network (GAT) [76] further enhanced GCN by incorporating an attention mechanism based on pairwise node representations. However, as GNN architectures deepen, issues such as the oversmoothing problem [48] and the suspended animation problem [90] can arise. Techniques such as edge dropout [64] and graph residual learning [90] have been proposed to mitigate these issues. Inspired by the effectiveness of Transformer models in language processing, researchers have recently explored Transformer-based GNN models, including Graph-Transformer [87], GPT-GNN [32], and Graph-BERT [91].

12.1.4 Attention and Transformer

The concept of attention was first introduced by Google DeepMind in [52] to model adaptive selection of high-resolution image regions. Around the same time, Bengio and collaborators applied attention mechanisms in machine translation models [3]. Following these initial explorations, attention mechanisms have proven effective across a range of deep learning tasks and are widely implemented in various deep learning models [82, 85, 79, 5]. Based on scaled dot-product attention, researchers at Google introduced the Transformer model [75] in 2017, which represents language data using pairwise self-attention scores derived from inputs. Shortly after, Google launched the pretrained BERT (Bidirectional Transformer) model [15] for language understanding in 2018. Around the same time, other Transformer-based language models were proposed by teams from different organizations, including GPT from OpenAI [60] and BART from Meta [45]. Today, Transformer has become the dominant backbone for large language models, with model sizes expanding significantly in recent years. For example, the initial GPT-1 model had only 117 million parameters, while the recent GPT-4 model has grown to 1.8 trillion parameters. Inspired by the success of Transformer models in natural language processing, researchers have also explored applying Transformer architectures to other fields, including ViT [16] for image processing, Graph-BERT [91] for graph learning, and DiT [56] for image and video generation.

12.2 Multi-Modality Foundation Models

The RPN 2 model introduced in this paper can also serve as a foundation model applicable to learning tasks across datasets with different modalities, aligning closely with current advancements in multi-modality foundation model learning.

12.2.1 Multi-Modality Data Alignment

Multi-modal data representation learning aims to integrate and interpret heterogeneous data types (*e.g.,* images, text, audio, video, and sensor data) through unified representations. OpenAI introduced Contrastive Language-Image Pretraining (CLIP) [59], which correlates images and language and enables zero-shot image classification. Google Research proposed ALIGN [33], which aligns visual and textual features by training on large-scale image-text pairs with contrastive objectives. Another approach focuses on learning hierarchical representations that capture both intra- and intermodal relationships. The VATT model (Video-Audio-Text Transformer) [1] employs self-supervised learning across multiple modalities to capture high-level correlations. Similarly, recent advancements in large-scale foundation models, such as Unified-IO [51], aim to unify inputs from multiple modalities within a common encoder-decoder framework, enhancing generalization across modalities. Additional models, such as LXMERT [73], Unicoder-VL [46], Oscar [49], and ViLBERT [50], have also been proposed for multi-modal representation learning, particularly for vision-language tasks. Recently, Meta introduced MetaCLIP [81], pre-trained on a larger image-text dataset, which outperforms CLIP on zero-shot image classification tasks.

12.2.2 Multi-Modality Tokenizer

Tokenization is essential in multi-modal learning, as it converts raw inputs from different modalities into token representations that machine learning models can process. Extending the concept of tokenization from natural language processing (NLP) to a multi-modal context has driven innovations in representing complex, multi-source input data. A key development is the introduction of multimodal tokenizers that handle diverse inputs, such as text, images, and videos, enabling downstream tasks like multi-modal fusion. Early models, such as VisualBERT [47], combined pre-trained BERT embeddings with visual features by treating images as sequences of region tokens derived from object detection models like Faster R-CNN. More recent models, such as BEiT-3 [80] and FLAVA [71], expand on this concept by introducing universal tokenizers that map images, text, and audio into a unified token space. Other advances in multi-modal tokenization, such as Pix2Seq [11], represent images as sequences of tokens, facilitating the adaptation of NLP-like transformers to the vision domain. With the rise of large language models, recent research has proposed converting video data into discrete tokens [86] to enable reasoning in language models. For instance, VideoPoet [38] employs video tokenizers to transform multi-modal data into discrete tokens for video generation.

12.2.3 Multi-Modality Generative Models

The field of generative modeling has advanced rapidly, with recent models demonstrating impressive capabilities in data synthesis. The denoising diffusion probabilistic model [27] has proven effective in generating image data and serves as a foundational model for the current surge in generative AI. To improve efficiency, the latent diffusion model [63] performs generation within the latent space using cross-attention mechanisms. The DiT model [56] further introduces a scalable diffusion approach with Transformers as the backbone. Building on these diffusion models, multi-modal generative models can produce coherent outputs (e.g., text, images, or video) from multi-modal inputs by learning cross-modal relationships. Models like DALL-E [61] and Imagen [67] leverage text-to-image generation via diffusion, pushing the boundaries of visually rich outputs conditioned on text. GLIDE [54] explores text-guided image generation and editing with diffusion models. Recently, Stability AI released Stable Diffusion 3 [20], which incorporates scaling rectified flow Transformers. These models not only generate high-quality images but also demonstrate creativity and abstraction across modalities. Beyond text-to-image generation, video generation has gained focus in the community. Make-A-Video [70], Dreamix [53], and Video Diffusion [28] extend diffusion concepts to video by leveraging pretrained text-to-image models with additional motion learning. Using the latent alignment algorithm [7], Stability AI has recently expanded their image generation model for video [6] and 3D object generation from image inputs [78].

13 Conclusion

In this paper, we redesigned the Reconciled Polynomial Network and introduced the new RPN 2 model, which incorporates an innovative component—the data interdependence function—into its architecture to explicitly model diverse relationships among both data instances and attributes. These data interdependence functions significantly enhance RPN 2's capabilities for complex function learning tasks on interdependent data, including but not limited to images, language, time series, and graphs. To demonstrate the efficacy of RPN 2's data interdependence modeling, we conducted extensive experiments on various multi-modal benchmark datasets, showing that RPN 2 consistently outperforms existing backbone models in multiple deep function learning tasks.

This enhancement also expands RPN 2's unifying potential, enabling it to encompass a broader range of prevalent backbone architectures within its canonical representation, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), graph neural networks (GNNs), and Transformers. Through these unifications, we illustrate the shared architectures and unique differences of these backbone models. These insights not only open new avenues for designing superior "Transformer-Next" models but also position RPN 2 as a robust framework for advancing function learning architecture design. Beyond empirical experiments, this paper also interprets the RPN 2 model from theoretical machine learning and biological neuroscience perspectives.

To support the adoption, implementation, and experimentation of RPN 2, we have updated our toolkit to the new TINYBIG V0.2.0, which incorporates interdependence modeling capabilities in model design and learning, along with updates to head and layer modules and model architecture. This updated toolkit enables researchers to efficiently design, customize, and deploy new RPN 2 models across a wide range of function learning tasks on diverse interdependent datasets.

Acknowledgments and Disclosure of Funding

This work is partially supported by NSF through grants IIS-1763365 and IIS-2106972.

References

[1] Hassan Akbari, Linagzhe Yuan, Rui Qian, Wei-Hong Chuang, Shih-Fu Chang, Yin Cui, and Boqing Gong. Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text. ArXiv, abs/2104.11178, 2021.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In 2017 International Conference on Engineering and Technology (ICET), pages 1–6, 2017.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.

[4] Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: risk bounds and structural results. J. Mach. Learn. Res., 3(null):463–482, March 2003.

[5] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V. Le. Attention augmented convolutional networks. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 3285–3294, 2019.

[6] A. Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, and Dominik Lorenz. Stable video diffusion: Scaling latent video diffusion models to large datasets. ArXiv, abs/2311.15127, 2023.

[7] A. Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 22563–22575, 2023.

[8] Anselm Blumer, A. Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the vapnik-chervonenkis dimension. J. ACM, 36(4):929–965, oct 1989.

[9] STEPHEN G. BRUSH. History of the lenz-ising model. Rev. Mod. Phys., 39:883–893, Oct 1967.

[10] Zhengping Che, S. Purushotham, Kyunghyun Cho, David A. Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. Scientific Reports, 8, 2016.

[11] Ting Chen, Saurabh Saxena, Lala Li, David J. Fleet, and Geoffrey E. Hinton. Pix2seq: A language modeling framework for object detection. ArXiv, abs/2109.10852, 2021.

[12] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In Conference on Empirical Methods in Natural Language Processing, 2014.

[13] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. ArXiv, abs/2405.21060, 2024.

[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In Neural Information Processing Systems, 2016.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In North American Chapter of the Association for Computational Linguistics, 2019.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. ArXiv, abs/2010.11929, 2020.

[17] Ran El-Yaniv and Dmitry Pechyony. Transductive rademacher complexity and its applications. In Annual Conference Computational Learning Theory, 2007.

[18] Jeffrey L. Elman. Finding structure in time. Cognitive Science, 14(2):179–211, 1990.

[19] Pascal Mattia Esser, Leena Chennuru Vankadara, and Debarghya Ghoshdastidar. Learning theory can (sometimes) explain generalisation in graph neural networks. ArXiv, abs/2112.03968, 2021.

[20] Patrick Esser, Sumith Kulal, A. Blattmann, Rahim Entezari, Jonas Muller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. ArXiv, abs/2403.03206, 2024.

[21] Joseph Fourier. Théorie analytique de la chaleur. F. Didot, Paris, 1822. English translation: The Analytical Theory of Heat.

[22] Vikas K. Garg, Stefanie Jegelka, and T. Jaakkola. Generalization and representational limits of graph neural networks. In International Conference on Machine Learning, 2020.

[23] Roy J. Glauber. Time-Dependent Statistics of the Ising Model. Journal of Mathematical Physics, 4(2):294–307, 02 1963.

[24] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.

[25] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. ArXiv, abs/2312.00752, 2023.

[26] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2015.

[27] Jonathan Ho, Ajay Jain, and P. Abbeel. Denoising diffusion probabilistic models. ArXiv, abs/2006.11239, 2020.

[28] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. ArXiv, abs/2204.03458, 2022.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Comput., 9(8):1735–1780, November 1997.

[30] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences of the United States of America, 79(8):2554–2558, 1982.

[31] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. Proceedings of the National Academy of Sciences of the United States of America, 81(10):3088–3092, 1984.

[32] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020.

[33] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc V. Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In International Conference on Machine Learning, 2021.

[34] William B. Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into hilbert space. Contemporary mathematics, 26:189–206, 1984.

[35] Michael I. Jordan. Chapter 25 - serial order: A parallel distributed processing approach. In John W. Donahoe and Vivian Packard Dorsel, editors, Neural-Network Models of Cognition, volume 121 of Advances in Psychology, pages 471–495. North-Holland, 1997.

[36] Yoon Kim. Convolutional neural networks for sentence classification. In Conference on Empirical Methods in Natural Language Processing, 2014.

[37] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. ArXiv, abs/1609.02907, 2016.

[38] D. Kondratyuk, Lijun Yu, Xiuye Gu, José Lezama, Jonathan Huang, Rachel Hornung, Hartwig Adam, Hassan Akbari, Yair Alon, Vighnesh Birodkar, Yong Cheng, Ming-Chang Chiu, Josh Dillon, Irfan Essa, Agrim Gupta, Meera Hahn, Anja Hauth, David Hendon, Alonso Martinez, David C. Minnen, David A. Ross, Grant Schindler, Mikhail Sirotenko, Kihyuk Sohn, Krishna Somandepalli, Huisheng Wang, Jimmy Yan, Ming Yang, Xuan Yang, Bryan Seybold, and Lu Jiang. Videopoet: A large language model for zero-shot video generation. ArXiv, abs/2312.14125, 2023.

[39] Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. Vera: Vector-based random matrix adaptation. ArXiv, abs/2310.11454, 2023.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems, volume 25. Curran Associates, Inc., 2012.

[41] Dmitry Krotov and John J. Hopfield. Dense associative memory for pattern recognition. ArXiv, abs/1606.01164, 2016.

[42] Dmitry Krotov and John J. Hopfield. Large associative memory problem in neurobiology and machine learning. ArXiv, abs/2008.06996, 2020.

[43] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[44] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series, page 255–258. MIT Press, Cambridge, MA, USA, 1998.

[45] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdel rahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Annual Meeting of the Association for Computational Linguistics, 2019.

[46] Gen Li, Nan Duan, Yuejian Fang, Daxin Jiang, and Ming Zhou. Unicoder-vl: A universal encoder for vision and language by cross-modal pre-training. ArXiv, abs/1908.06066, 2019.

[47] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. Visualbert: A simple and performant baseline for vision and language. ArXiv, abs/1908.03557, 2019.

[48] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In AAAI Conference on Artificial Intelligence, 2018.

[49] Xiujun Li, Xi Yin, Chunyuan Li, Xiaowei Hu, Pengchuan Zhang, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, Yejin Choi, and Jianfeng Gao. Oscar: Object-semantics aligned pre-training for vision-language tasks. ArXiv, abs/2004.06165, 2020.

[50] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In Neural Information Processing Systems, 2019.

[51] Jiasen Lu, Christopher Clark, Rowan Zellers, Roozbeh Mottaghi, and Aniruddha Kembhavi. Unified-io: A unified model for vision, language, and multi-modal tasks. ArXiv, abs/2206.08916, 2022.

[52] Volodymyr Mnih, Nicolas Manfred Otto Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In Neural Information Processing Systems, 2014.

[53] Eyal Molad, Eliahu Horwitz, Dani Valevski, Alex Rav Acha, Yossi Matias, Yael Pritch, Yaniv Leviathan, and Yedid Hoshen. Dreamix: Video diffusion models are general video editors. ArXiv, abs/2302.01329, 2023.

[54] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. In International Conference on Machine Learning, 2021.

[55] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. ArXiv, abs/1511.08458, 2015.

[56] William S. Peebles and Saining Xie. Scalable diffusion models with transformers. 2023 IEEE/CVF International Conference on Computer Vision (ICCV), pages 4172–4182, 2022.

[57] Bryan Perozzi, Rami Al-Rfou, and Steven S. Skiena. Deepwalk: online learning of social representations. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014.

[58] C. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 77–85, 2016.

[59] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In International Conference on Machine Learning, 2021.

[60] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[61] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. ArXiv, abs/2102.12092, 2021.

[62] Hubert Ramsauer, Bernhard Schafl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Milena Pavlovi'c, Geir Kjetil Ferkingstad Sandve, Victor Greiff, David P. Kreil, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. ArXiv, abs/2008.02217, 2020.

[63] Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 10674–10685, 2021.

[64] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In International Conference on Learning Representations, 2019.

[65] Frank Rosenblatt. Perceptual generalization over transformation groups. In Marshall C. Yovitz and Scott Cameron, editors, Self-organizing Systems: Proceedings of an Inter-disciplinary Conference, 5 and 6 May, 1959, pages 63–100. Pergamon Press, London, New York, 1960. ix, 322 p.

[66] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. International Journal of Computer Vision, 77:125–141, 2008.

[67] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, Seyedeh Sara Mahdavi, Raphael Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. ArXiv, abs/2205.11487, 2022.

[68] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11):2673–2681, 1997.

[69] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[70] Uriel Singer, Adam Polyak, Thomas Hayes, Xiaoyue Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, Devi Parikh, Sonal Gupta, and Yaniv Taigman. Make-a-video: Text-to-video generation without text-video data. ArXiv, abs/2209.14792, 2022.

[71] Amanpreet Singh, Ronghang Hu, Vedanuj Goswami, Guillaume Couairon, Wojciech Galuba, Marcus Rohrbach, and Douwe Kiela. Flava: A foundational language and vision alignment model. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 15617–15629, 2021.

[72] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. Pathsim: meta path-based top-k similarity search in heterogeneous information networks. Proc. VLDB Endow., 4(11):992–1003, August 2011.

[73] Hao Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. In Conference on Empirical Methods in Natural Language Processing, 2019.

[74] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability & Its Applications, 16(2):264–280, 1971.

[75] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Neural Information Processing Systems, 2017.

[76] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio', and Yoshua Bengio. Graph attention networks. ArXiv, abs/1710.10903, 2017.

[77] Subhashini Venugopalan, Marcus Rohrbach, Jeff Donahue, Raymond J. Mooney, Trevor Darrell, and Kate Saenko. Sequence to sequence – video to text. 2015 IEEE International Conference on Computer Vision (ICCV), pages 4534–4542, 2015.

[78] Vikram S. Voleti, Chun-Han Yao, Mark Boss, Adam Letts, David Pankratz, Dmitry Tochilkin, Christian Laforte, Robin Rombach, and Varun Jampani. Sv3d: Novel multi-view synthesis and 3d generation from a single image using latent video diffusion. ArXiv, abs/2403.12008, 2024.

[79] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6450–6458, 2017.

[80] Wenhui Wang, Hangbo Bao, Li Dong, Johan Bjorck, Zhiliang Peng, Qiang Liu, Kriti Aggarwal, Owais Khan Mohammed, Saksham Singhal, Subhojit Som, and Furu Wei. Image as a foreign language: Beit pretraining for all vision and vision-language tasks. ArXiv, abs/2208.10442, 2022.

[81] Hu Xu, Saining Xie, Xiaoqing Ellen Tan, Po-Yao (Bernie) Huang, Russell Howes, Vasu Sharma, Shang-Wen Li, Gargi Ghosh, Luke S. Zettlemoyer, and Christoph Feichtenhofer. Demystifying clip data. ArXiv, abs/2309.16671, 2023.

[82] Ke Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning, 2015.

[83] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. Insights into Imaging, 9(4):611–629, 2018.

[84] Xifeng Yan and Jiawei Han. gspan: graph-based substructure pattern mining. In 2002 IEEE International Conference on Data Mining, 2002. Proceedings., pages 721–724, 2002.

[85] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1480–1489, San Diego, California, June 2016. Association for Computational Linguistics.

[86] Lijun Yu, Yong Cheng, Kihyuk Sohn, José Lezama, Han Zhang, Huiwen Chang, Alexander G. Hauptmann, Ming-Hsuan Yang, Yuan Hao, Irfan Essa, and Lu Jiang. Magvit: Masked generative video transformer. 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 10459–10469, 2022.

[87] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks. In Neural Information Processing Systems, 2019.

[88] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. ArXiv, abs/1409.2329, 2014.

[89] Jiawei Zhang. Rpn: Reconciled polynomial network towards unifying pgms, kernel svms, mlp and kan. ArXiv, abs/2407.04819, 2024.

[90] Jiawei Zhang and Lin Meng. Gresnet: Graph residual network for reviving deep gnns from suspended animation. ArXiv, abs/1909.05729, 2019.

[91] Jiawei Zhang, Haopeng Zhang, Li Sun, and Congying Xia. Graph-bert: Only attention is needed for learning graph representations. ArXiv, abs/2001.05140, 2020.

[92] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip H. S. Torr, and Vladlen Koltun. Point transformer. 2021 IEEE/CVF International Conference on Computer Vision (ICCV), pages 16239–16248, 2020.

A Appendix

A.1 Licensing Rights of Using BioRender Created Contents in This Paper

Descriptions: For the biomedical image content presented in Figures 35-37 in Section 10, we have obtained the necessary permissions for their use in publications. Confirmation letters granting licensing rights from BioRender for these generated contents are included in the following pages.

49 Spadina Ave. Suite 200
Toronto ON M5V 2J1 Canada
www.biorender.com

Confirmation of Publication and Licensing Rights - Open Access

October 10th, 2024

Subscription Type:	*Institution - Academic*
Agreement number:	*EB27ERFE3F*
Publisher Name:	*arxiv*
Figure Title:	*sensory_cortex*
Citation to Use:	*Created in BioRender. Zhang, J. (2024) BioRender.com/f39x623*

To whom this may concern,

This document ("Confirmation") hereby confirms that Science Suite Inc. dba BioRender ("BioRender") has granted the following BioRender user: Jiawei Zhang ("User") a BioRender Academic Publication License in accordance with BioRender's Terms of Service and Academic License Terms ("License Terms") to permit such User to do the following on the condition that all requirements in this Confirmation are met:

1) publish their Completed Graphics created in the BioRender Services containing both User Content and BioRender Content (as both are defined in the License Terms) in publications (journals, textbooks, websites, etc.); and
2) sublicense such Completed Graphics under "open access" publication sublicensing models such as CC-BY 4.0 and more restrictive models, so long as the conditions set forth herein are fully met.

Requirements of User:

1) All Completed Graphics to be published in any publication (journals, textbooks, websites, etc.) must be accompanied by the following citation either as a caption, footnote or reference for each figure that includes a Completed Graphic:
   "Created in BioRender. Zhang, J. (2024) BioRender.com/f39x623 ".
2) All terms of the License Terms including all Prohibited Uses are fully complied with. E.g. For Academic License Users, no commercial uses (beyond publication in journals, textbooks or websites) are permitted without obtaining or switching to a BioRender Industry Plan.
3) A Reader (defined below) may request that the User allow their figure to be a public template for Readers to view, copy, and modify the figure. It is up to the User to determine what level of access to grant.

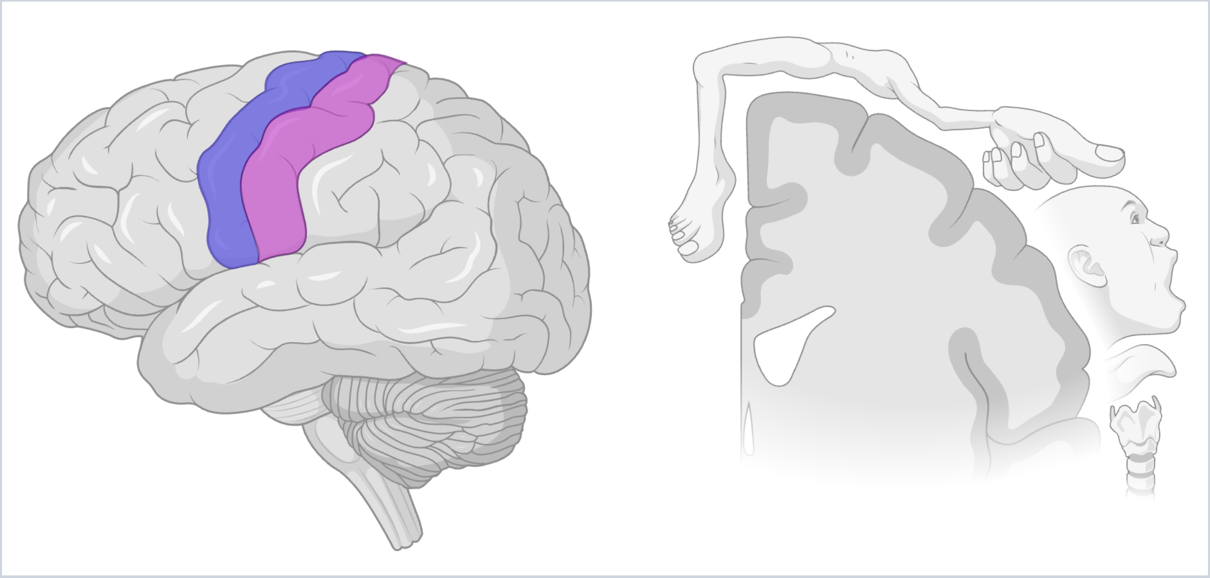
Open-Access Journal Readers:

Open-Access journal readers ("Reader") who wish to view and/or re-use a particular Completed Graphic in an Open-Access journal subject to CC-BY sublicensing may do so by clicking on the URL link in the applicable citation for the subject Completed Graphic.

The re-use/modification options below are available after the Reader requests the User to adapt their

figure as a BioRender template and the User has granted such access.

1) View-Only/Free Plan Use: A Reader who wishes to only view the Completed Graphic may do so in the BioRender Services as either a BioRender Free Plan user or simply as a viewer. By becoming a BioRender Free Plan user, the Reader may view, modify and re-use the Completed Graphic as permitted under BioRender's Basic License Terms (e.g. personal use only, no publishing or commercial use permitted).
2) Re-Use/Publish with No Modifications: For any re-use and re-publication of a Completed Graphic with no modification(s) to the Completed Graphic made by the Reader, a Reader may do so by citing the original author using the citation noted above with the Completed Graphic. The Reader must also comply with the underlying License Terms which apply to the Completed Graphic as noted above (e.g. no commercial use for Academic License).
3) Re-Use/Publish with Modifications: For any re-use and re-publication of a Completed Graphic with a modification(s) made by the Reader, the Reader may do so by becoming a BioRender user themselves under either an Academic or Industry Plan, citing the original author using the citation noted above with the Completed Graphic and complying with the applicable License Terms.



For any questions regarding this document, or other questions about publishing with BioRender, please refer to our BioRender Publication Guide, or contact BioRender Support at support@biorender.com.

49 Spadina Ave. Suite 200
Toronto ON M5V 2J1 Canada
www.biorender.com

Confirmation of Publication and Licensing Rights - Open Access

October 10th, 2024

Subscription Type:	*Institution - Academic*
Agreement number:	*JT27ER7T81*
Publisher Name:	*arxiv*
Figure Title:	*hippocampus*
Citation to Use:	*Created in BioRender. Zhang, J. (2024) BioRender.com/j80y259*

To whom this may concern,

This document ("Confirmation") hereby confirms that Science Suite Inc. dba BioRender ("BioRender") has granted the following BioRender user: Jiawei Zhang ("User") a BioRender Academic Publication License in accordance with BioRender's Terms of Service and Academic License Terms ("License Terms") to permit such User to do the following on the condition that all requirements in this Confirmation are met:

1) publish their Completed Graphics created in the BioRender Services containing both User Content and BioRender Content (as both are defined in the License Terms) in publications (journals, textbooks, websites, etc.); and
2) sublicense such Completed Graphics under "open access" publication sublicensing models such as CC-BY 4.0 and more restrictive models, so long as the conditions set forth herein are fully met.

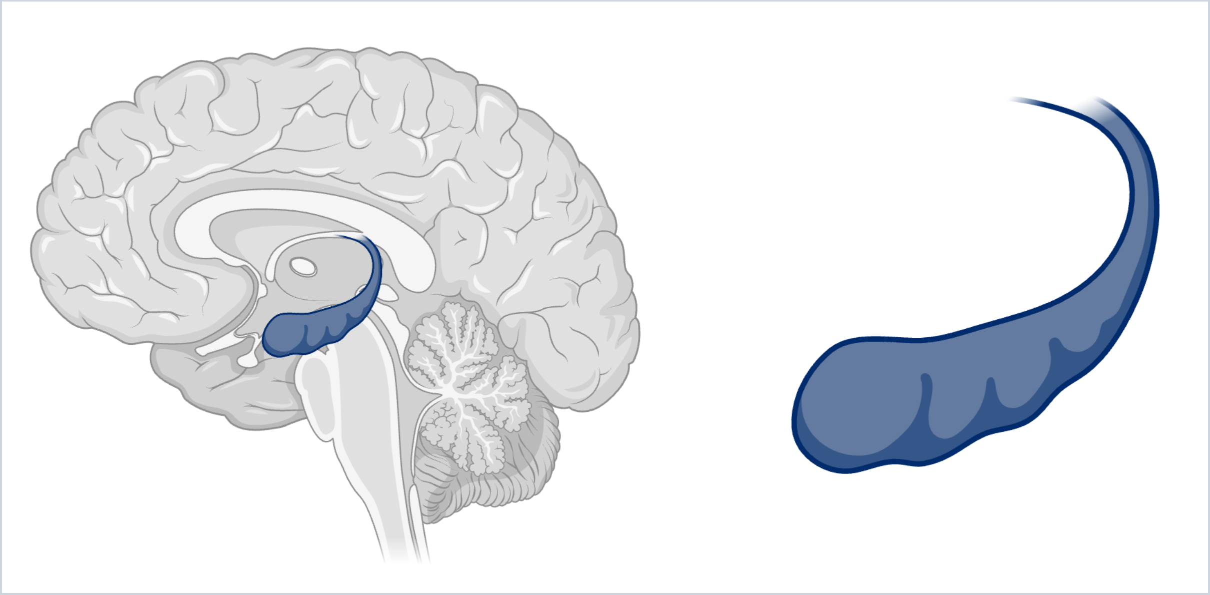
Requirements of User:

1) All Completed Graphics to be published in any publication (journals, textbooks, websites, etc.) must be accompanied by the following citation either as a caption, footnote or reference for each figure that includes a Completed Graphic:
"Created in BioRender. Zhang, J. (2024) BioRender.com/j80y259 ".
2) All terms of the License Terms including all Prohibited Uses are fully complied with. E.g. For Academic License Users, no commercial uses (beyond publication in journals, textbooks or websites) are permitted without obtaining or switching to a BioRender Industry Plan.
3) A Reader (defined below) may request that the User allow their figure to be a public template for Readers to view, copy, and modify the figure. It is up to the User to determine what level of access to grant.

Open-Access Journal Readers:

Open-Access journal readers ("Reader") who wish to view and/or re-use a particular Completed Graphic in an Open-Access journal subject to CC-BY sublicensing may do so by clicking on the URL link in the applicable citation for the subject Completed Graphic.

The re-use/modification options below are available after the Reader requests the User to adapt their



For any questions regarding this document, or other questions about publishing with BioRender, please refer to our BioRender Publication Guide, or contact BioRender Support at support@biorender.com.

49 Spadina Ave. Suite 200
Toronto ON M5V 2J1 Canada
www.biorender.com

Confirmation of Publication and Licensing Rights - Open Access

October 10th, 2024

Subscription Type:	*Institution - Academic*
Agreement number:	*AQ27ERC06B*
Publisher Name:	*arxiv*
Figure Title:	*cortex and thalamus*
Citation to Use:	*Created in BioRender. Zhang, J. (2024) BioRender.com/v74r180*

To whom this may concern,

This document ("Confirmation") hereby confirms that Science Suite Inc. dba BioRender ("BioRender") has granted the following BioRender user: Jiawei Zhang ("User") a BioRender Academic Publication License in accordance with BioRender's Terms of Service and Academic License Terms ("License Terms") to permit such User to do the following on the condition that all requirements in this Confirmation are met:

1) publish their Completed Graphics created in the BioRender Services containing both User Content and BioRender Content (as both are defined in the License Terms) in publications (journals, textbooks, websites, etc.); and
2) sublicense such Completed Graphics under "open access" publication sublicensing models such as CC-BY 4.0 and more restrictive models, so long as the conditions set forth herein are fully met.

Requirements of User:

1) All Completed Graphics to be published in any publication (journals, textbooks, websites, etc.) must be accompanied by the following citation either as a caption, footnote or reference for each figure that includes a Completed Graphic:
"Created in BioRender. Zhang, J. (2024) BioRender.com/v74r180 ".
2) All terms of the License Terms including all Prohibited Uses are fully complied with. E.g. For Academic License Users, no commercial uses (beyond publication in journals, textbooks or websites) are permitted without obtaining or switching to a BioRender Industry Plan.
3) A Reader (defined below) may request that the User allow their figure to be a public template for Readers to view, copy, and modify the figure. It is up to the User to determine what level of access to grant.

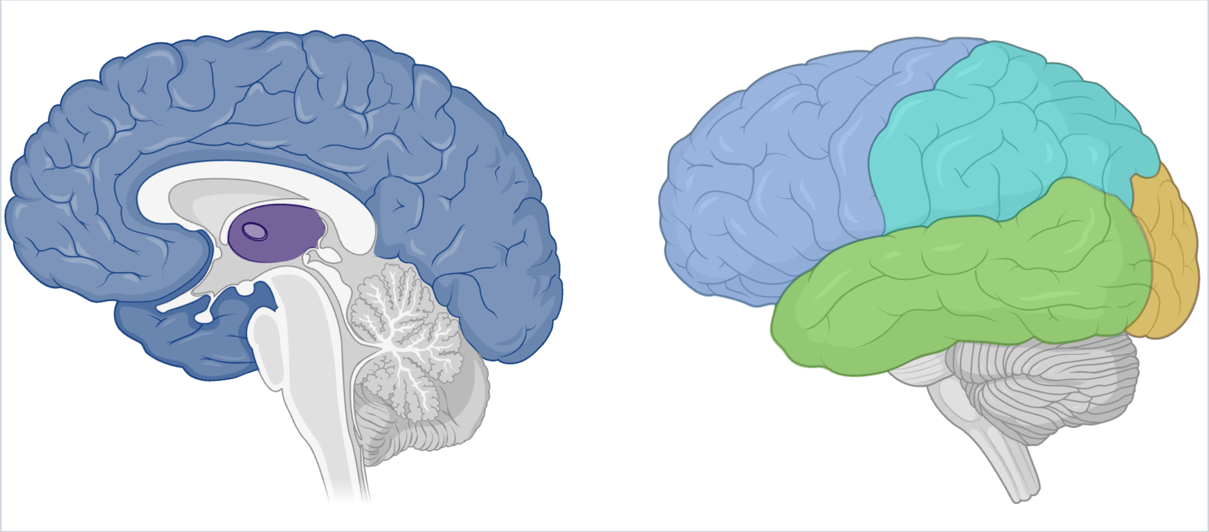
Open-Access Journal Readers:

Open-Access journal readers ("Reader") who wish to view and/or re-use a particular Completed Graphic in an Open-Access journal subject to CC-BY sublicensing may do so by clicking on the URL link in the applicable citation for the subject Completed Graphic.

The re-use/modification options below are available after the Reader requests the User to adapt their

figure as a BioRender template and the User has granted such access.

1) View-Only/Free Plan Use: A Reader who wishes to only view the Completed Graphic may do so in the BioRender Services as either a BioRender Free Plan user or simply as a viewer. By becoming a BioRender Free Plan user, the Reader may view, modify and re-use the Completed Graphic as permitted under BioRender's Basic License Terms (e.g. personal use only, no publishing or commercial use permitted).
2) Re-Use/Publish with No Modifications: For any re-use and re-publication of a Completed Graphic with no modification(s) to the Completed Graphic made by the Reader, a Reader may do so by citing the original author using the citation noted above with the Completed Graphic. The Reader must also comply with the underlying License Terms which apply to the Completed Graphic as noted above (e.g. no commercial use for Academic License).
3) Re-Use/Publish with Modifications: For any re-use and re-publication of a Completed Graphic with a modification(s) made by the Reader, the Reader may do so by becoming a BioRender user themselves under either an Academic or Industry Plan, citing the original author using the citation noted above with the Completed Graphic and complying with the applicable License Terms.



For any questions regarding this document, or other questions about publishing with BioRender, please refer to our BioRender Publication Guide, or contact BioRender Support at support@biorender.com.