Formal Semantics in XAI: A Categorical Diagrammatic Framework

Yidan Li[1], Jianying Cui[1(✉)], and Minghui Xiong[2]

[1] Institute of Logic and Cognition, Department of Philosophy, Sun Yat-sen University, Guangzhou, China
liyd66@mail2.sysu.edu.cn, cuijiany@mail.sysu.edu.cn
[2] Guanghua Law School, Zhejiang University, Hangzhou, China.
xiongminghui@zju.edu.cn

Abstract. Machine learning-based AI systems, despite their widespread success, suffer from limited trust and deployment in critical domains due to their black-box nature. In response, this paper introduces a semantic framework of category-theoretic diagrammatic formalism, which focuses on diagrammatization of model components and formal semantic mapping: translating intuition into mathematical derivations. Our analysis proceeds in three stages: (1) structural decomposition, where string diagrams visually represent model components and their interactions; (2) semantic mapping, where semantic interpretations are assigned to each component, particularly, unlike existing DisCoCat frameworks, our work offers a higher-order extension using typed lambda calculus to systematically capture complex semantics like negation and quantification; and (3) compositional reasoning and verification, where categorical morphisms enable traceability and behavioral transparency through diagrammatic transformation rules that ensure formal uniqueness of interpretations. This semantic framework supports interpretable validation of AI models by leveraging string diagrams to intuitively visualize component interactions and compositional information flow.

Keywords: Compositional Reasoning · Formal Semantics · Explainability · Category Theory · String Diagrams · Diagrammatic Semantics

1 Introduction

With the widespread application of machine learning (ML) models in high-risk fields such as healthcare and finance, model interpretability is critical to addressing the core issues of transparency and ethics in decision-making for AI systems. According to the differences in the stages of model intervention, interpretability can be classified into intrinsic and post-hoc interpretability [2]. The rapidly evolving field of explainable artificial intelligence (XAI) primarily focuses on post-hoc explanation techniques which take a trained AI model and aim to give explanations for either its overall behaviour, or individual outputs[14]. Rudin has highlighted that post-hoc explanations may be untrustworthy for two facts. Firstly, they rely on approximations of the original model, which are inherently

unfaithful, and secondly, such explanations are often non-unique and might even seem adjustable at will [7, 6].

Unlike post-hoc interpretability techniques, intrinsic interpretability refers to enable transparency and self-interpretation of the model's decision-making process through structural design. Current research on intrinsic interpretability unfolds along two dimensions. The first is modifying model components, which modifies the 'black-box' components and their associated structures [2]. For example, reasoning transparency is implemented through architectural modifications (e.g., sparse attention mechanisms [17]), the introduction of interpretable components (e.g., dynamic weights [5]), or modifying the middle layer [10]. These works ensure that intermediate states can be interpreted using human language due to the modular design and through component-verifiable arithmetic units, but to some extent increase model complexity at the expense of representational flexibility [8, 12]. The second dimension is mechanistic investigation for intrinsic interpretability. Introducing knowledge distillation mechanisms [11]can partially enhance model interpretability. Especially in handling complex tasks, the distilled knowledge may still be difficult to interpret [15]. The chain-of-thought (CoT) generation methods [19]narrow the performance gap between interpretable and frontier models by relying on heuristic rules in language models. However, these methods lack formal guarantees [1, 9].

To address the limitations of current intrinsic interpretability methods, we introduce a novel paradigm grounded in category theory. This framework provides a mathematically rigorous foundation for modeling how meaning is composed, enabling intrinsic interpretability through formal verification. We leverage string diagrams, a powerful visual calculus from category theory. These diagrams can be seen as a powerful evolution of the intuitive flowcharts common in machine learning papers [16]. However, unlike traditional flowcharts, string diagrams are not mere visual aids; they are a rigorous mathematical language where the visual structure and transformation rules directly correspond to formal, provable derivations of a system's behavior. Specifically, we achieve this "combinatorial interpretability" through the following steps: 1) defining the structure of the Categorical Compositional Distributional models (DisCoCat) with a category-theoretic framework; 2) employing string diagrams to represent the way the model is combined; 3) deriving the overall behavior through the rules of combination; and 4) parsing the contributions of each module to improve the transparency and interpretability of the model.

The rest of this article is organised as follows. Section 2, we introduce the basic mathematical theory, including categories ,string diagrams and the definition of a compositional model. Section 3, taking DisCoCat model as an example, the linguistic structure is mapped to a "computable" category. The functor transfers objects and morphisms between categories while preserving their compositional structures.Section 4 , We combine specific text, which involves complex semantics such as negation, disjunction, and universal quantification, and these are assigned higher-order types; Section 5 presents the future research directions.

2 Formal Semantics for Compositional Models

To formally describe compositionality, we employ the mathematics of category theory and its associated graphical language of string diagrams to examine the composition of systems and morphisms [13]. These diagrams provide a rigorous yet visually intuitive framework, where in structural constraints guide abstract reasoning.

2.1 Categories and String Diagrams

The semantics of diagrams are defined within the cartesian monoidal category [16] whose objects are Euclidean spaces (viewed as real finite-dimensional vector spaces) and whose morphisms are continuous functions between them. The most powerful feature of these categories is that they admit a purely diagrammatic calculus, in which wires denote spaces and boxes denote morphisms. The two ways to compose processes, namely sequential composite($\circ$) and parallel composition ($\otimes$) in a symmetric monoidal category (SMC).[1]For all spaces, the basic diagrammatic vocabulary are given in Fig. 1.

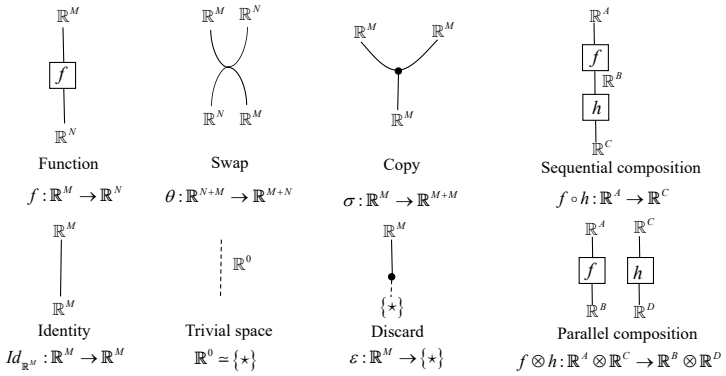


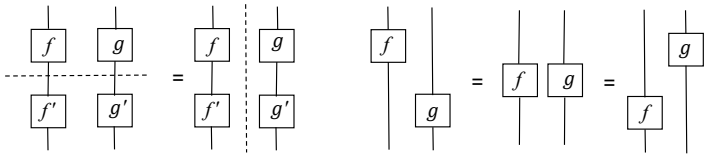
Fig. 1: The Syntax of String Diagrams for Cartesian Categories

String diagrams, grounded in SMCs, offer geometric syntax for model architectures. The visually intuitive equivalences in the information flow of string diagrams correspond directly to equivalent symbolic derivations of system behavior. As an example, functoriality of $\otimes$ means that we always have:

$$(f \otimes g) \circ (f' \otimes g') = (f \circ f') \otimes (g \circ g') \tag{1}$$

In diagrams this is automatic, the 'interchange law' which lets us freely slide boxes along wires. This means that the cumbersome algebraic proofs are naturally handled by diagram isotopies.

[1] An SMC is a category $\mathcal{C}$ equipped with a bifunctor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, a unit object I, and natural isomorphisms satisfying coherence conditions for associativity and symmetry.



String diagrams, grounded in SMCs,represent Euclidean spaces as wires, providing a geometric syntax for tensor operations. For instance, a morphism $f : \mathbb{R}^M \to \mathbb{R}^N$ corresponds to a map between N-dimensional and M-dimensional vector spaces over the scalar field $\mathbb{R}$. To simplify notation, parallel iterations of such maps are indicated by labelled ticks on wires.

Just as $\mathbb{R}^M$ denotes the space of length-M vectors taking values in $\mathbb{R}$, the Cartesian product space $\mathbb{R}^{A\times C}$ corresponds to the space of $\mathbb{R}$-valued A-by-C matrices. Generalizing this formalism, ticks on wires extend to higher-dimensional tensor spaces by indicating indices of arbitrary rank. For example, a tensor in $\mathbb{R}^{A\times B\times C}$ can be decomposed into a sequence of tensor product factors, each represented by a wire with ticks annotating its index structure. This visual encoding not only aligns with algebraic definitions but also enables functions between $\mathbb{R}$-valued tensor spaces to be naturally expressed through diagrammatic compositions. such as in Fig. 2.

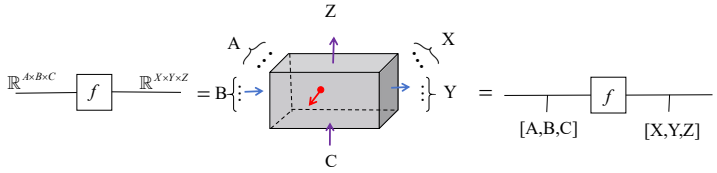


Fig. 2: String diagram representation of tensor operations in SMCs

2.2 Compositional Models based on String Diagrams

In this section, we introduce the formal definitions of two core concepts: compositional models and their interpretability[16].These definitions are grounded in categorical structures, including object categories, morphisms, string diagrams, and representation functor. Specifically, a categorical model is characterized as a functor mapping from a structural category to a semantic category.

Definition 1. *A (monoidal) signature G consists of:*

- *a set G_{ob} of 'objects', which are called variables.*
- *a set G_{mor} of 'morphisms' called generators, whose inputs and outputs are lists of objects*[2].
- *a set G_{eq} of equations of the form $D_1 = D_2$,where D_1, D_2 are string diagrams constructed from the generators.*

[2] Formally, these come with functions $\mathrm{in}, \mathrm{out} : G_{mor} \to G_{ob}^*$ that send each generator to a list of input variables and output variables, respectively. All possible morphisms can be generated by composing generators and tensor products. For example, if $\mathrm{in}(f) = [A, B]$ and $\mathrm{out}(f) = [C]$, then f represents the morphism from $A \otimes B$ to C.

Typically, a signature G consists of basic objects and processes, which combine to generate a category. Given a class of string diagrams (e.g., monoidal diagrams), we define Free(G): objects are lists of variables $(X_i)_{i=1}^{n}$ from G_{ob} , and a morphism D constructed from G_{mor} with inputs and outputs as corresponding lists. Two morphisms $f \cong g$ are deemed equivalent in Free(G)if and only if there exists a finite sequence of diagrammatic isotopies (such as sliding boxes along wires, bending wires downward, and performing swap operations) and equational rewrites, such that homotopy chain preserves topological equivalence of the string diagrams.

A combinatorial model integrates basic variables and generators through composition rules (e.g., sequential or parallel), thereby characterizing complex processes.

Definition 2. *A compositional model M=(S,C,F) consists of:*

- *a structure category S=Free(G);*
- *a semantics category C;*
- *a representation functor:*

$$F : S \to C$$

A *compositional model* is a triple $M = (S, C, F)$, where $S = \text{Free}(G)$ is a structure category freely generated from a graph G, C is a semantics category, and $F : S \to C$ is a functor assigning semantic interpretations to objects and morphisms of S. String diagrams serve as natural representations of such models, with wires and boxes corresponding to variables and generators, respectively. Such as in Fig. 3 The pair (D, M), where D is a diagram of a specific shape and M provides its interpretation, defines a diagram in C[3].

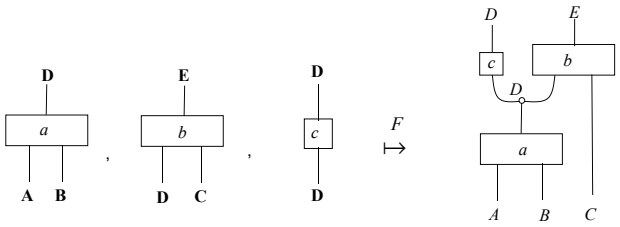


Fig. 3: Compositional model architecture and string diagram semantics

This means that the signature G consists of variables **A**, **B** , **C**, **D**, **E** and generators a ,b ,c. The structure category $S = Free(G)$ is generated by all possible cd-diagrams constructible from the given variables and generators in Fig. 4.

[3] we distinguish objects and morphisms in S from their representation in C by using different fonts.

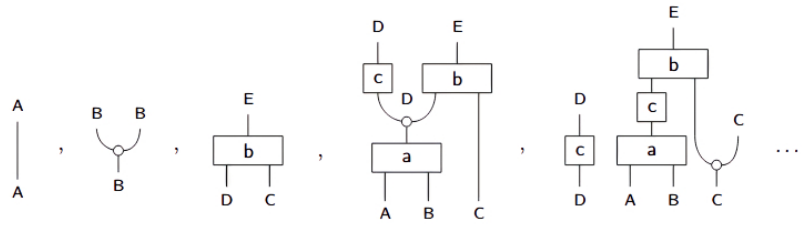


Fig. 4: Signature G and the construction of the free category S

3 CCG as a Pipeline for DisCoCat

The compositionality of language systems manifests as the ability of discrete symbols to form "multi-word constructions". As a paradigm of compositional reasoning, finds a powerful mathematical realization in the DisCoCat model. Different from the traditional version of DisCoCat that is bound to pregroups, we offer a version of DisCoCat whose domain is Combinatory Categorial Grammar (CCG)[20]. This reformulation offers two key advantages. First, as CCG is a mildly context-sensitive grammar[18], it enhances the generative power of DisCoCat beyond that of pregroup-based systems; and second, the availability of robust CCG parsers —such as the one developed by Clark and Curran[3]—enables the automatic derivation of syntactic structures from large-scale corpora. This section details the formal machinery for this CCG-based DisCoCat model, establishing a clear and theoretically sound pathway from syntactic structure to semantic representation.

3.1 CCG as a Biclosed Category

This paper illustrates, with CCG as an example, how categorical migration enables linguistically meaningful compositionality in structured language representations. DisCoCat models are structure-preserving maps, which can be formalised as functors $F : \mathbf{G} \to \mathbf{FdVect}$ from the category generated by a formal grammar $\mathbf{G}$ to the monoidal category $\mathbf{FdVect}$ of finite-dimensional vector spaces and linear maps equipped with the tensor product[4].

CCG is a syntactic framework based on typed functional application. Its structure encompasses two-layered syntax: one is a rule set for constructing complex types from primitive types and the other is a rule set for combining terms according to these types.

Definition 3. *A categorial grammar is formally defined as a quadruple* $G = (V, X, \Sigma, s)$*, where:*

- V *is the set of lexical items.*
- X *is the set of basic type.Type is the set of type expressions generated from* X*,* $T(X)$ *is the set of all formal expressions, recursively generated from* X *using the following operations:* $x \otimes y$*,* $x \backslash y$*,* $x/y \in T(X)$*, for all* $x, y \in T(X)$
- Σ *is a lexicon ,defined as a set of pairs* $[w : t]$*, where* $w \in V$ *and* $t \in T(X)$*.*
- $s \in X$ *represents the sentence type.*

Formalizing the term-combination rules mathematically, CCG can be modeled as a proof system that naturally forms a *biclosed category* $\mathcal{B}$. In this categorical structure:

- Objects of $\mathcal{B}$ correspond to CCG types in T(X).
- Morphisms of $\mathcal{B}$ represent valid syntactic derivations or proofs: a morphism $f : X \to Y$ encodes a proof that type X can be combined (via CCG rules) to yield type Y. Function application rules determine the number, type, and direction of arguments, as well as the new types generated by applying these rules. The CCG rules such as in Tab.1

Table 1: Category Composition Rules in CCG

Category Composition Rule (Rule B)		**Category Lifting Rule (T-Rule)**	
FA	$X/Y \quad Y \to X$	BA	$Y \quad X\backslash Y \to X$
$FC(> B)$	$X/Y \quad Y/Z \to X/Z$	$FTR(> T)$	$X \to T/(T/X)$
$BC(< B)$	$Z\backslash Y \quad Y\backslash X \to Z\backslash X$	$BTR(< T)$	$X \to T\backslash(T/X)$

The key structural property is the existence of two natural isomorphisms for left/right-currying (for any objects A, B, C):

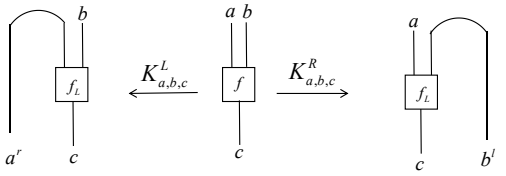
$$\kappa^L_{A,B,C} : \mathcal{B}(A \otimes B, C) \cong \mathcal{B}(B, A\backslash C)$$

$$\kappa^R_{A,B,C} : \mathcal{B}(A \otimes B, C) \cong \mathcal{B}(A, C/B)$$

where:

- $A \otimes B$ denotes the sequential composition of types.
- $A\backslash C$ (left-accepting) and C/B (right-accepting) are standard CCG function types

The CCG combination rules are converted into string diagrams through currying/uncurrying.



This structure provides formal foundations for *type-term-rule* relationships. The fundamental insight is that CCG's application rules are not ad hoc, but emerge naturally from these isomorphisms via (un)currying of identity morphisms[4].

[4] The backward application rule $A \otimes (A\backslash B) \to B$ arises from uncurrying $Id_{A\backslash B} : A\backslash B \to A\backslash B$. Likewise, forward application $((C/B) \otimes B \to C)$ is the uncurrying of $Id_{C\backslash B}$.

3.2 The Functorial Passage to Semantic Tensors

To now give semantics to this syntactic structure, we must define a **closed monoidal functor** $F\colon \mathcal{B} \to \mathcal{C}$, as follows.

Definition 4. *Let Σ be a CCG lexicon. a DisCoCat model based on CCG consists of:*

- *S=FreeBi(Σ) is a free biclosed category,which generated from Σ;*
- *a semantic biclosed categorie $\mathcal{B}$;*
- *a biclosed monoidal functor F.*

where $\mathcal{C}$ is a compact-closed category (e.g., FdVect) that serves as the semantic domain. This functor F preserves the compositional structure of $\mathcal{B}$ while mapping syntactic types and derivations to semantic objects and operations. Let $\{N, S\}$ denote the set of CCG atomic categories representing a noun phrase and a sentence, respectively, and T denote a word type. $\mathcal{C}$ like FdVect, the functor maps:

- Basic type mapping:
$$F(NP) = \mathbf{n},\ F(S) = \mathbf{s}$$
where $\mathbf{n}, \mathbf{s}, \mathbf{p}$ are vector spaces for noun phrases, sentences, and prepositional phrases respectively.
- Lexical mapping: A word morphism $w\colon I \to t$ in $\mathcal{B}$ is mapped to $F(w)\colon I \to F(t)$ in $\mathcal{C}$, e.g.,
$$F(\text{Alice})\colon I \to \mathbf{n} \text{ (a vector in } \mathbf{n}),$$
$$F(\text{likes})\colon I \to \mathbf{n}^* \otimes \mathbf{s} \otimes \mathbf{n}^* \text{ (a 3rd-order tensor).}$$
- Slash operator preservation:
$$F(X\backslash Y) = F(X)^* \otimes F(Y), \quad F(X/Y) = F(Y) \otimes F(X)^*$$
- Functorial laws:
$$\begin{aligned} &F(f \circ g) = F(f) \circ F(g) \\ &F(\mathrm{Id}_X) = \mathrm{Id}_{F(X)} \\ &F(X \otimes Y) = F(X) \otimes F(Y) \\ &F(I_{\mathcal{B}}) = I_{\mathcal{C}} \end{aligned}$$

In that case, any CCG lexicon Σ generates a free bi-closed category $\mathbf{FreeBi}(\boldsymbol{\Sigma})$ and the DisCoCat models $F : \mathbf{FreeBi}(\Sigma) \to \mathbf{FdVect}$ map grammatical structures to the closed structure of $Vect$ in a canonical way.

Encoding CCG in this way makes every order-preserving grammatical rule appear as a canonical biclosed diagram, while rules that relax word order or introduce cross-serial dependencies are realised by additional morphisms. Because the target category **FdVect** is symmetric monoidal, these permutation morphisms can be implemented graphically by simply swapping wires.

Overall, viewing CCG as a biclosed category $\mathcal{B}$ and interpreting it through the functor F into a compact-closed semantic category $\mathcal{C}$ provides a mathematically rigorous account of linguistic compositionality: syntactic combinations correspond exactly to semantic operations such as tensor contraction, allowing sentence meaning to emerge systematically and transparently from word meaning and syntactic structure.

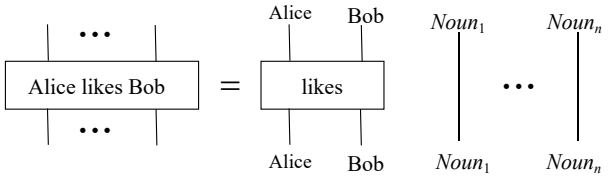
4 Explanations from diagrams

To summarize the current situation, on the one hand, DisCoCat models defined in terms of tensor representations can be trained from data and allow for efficient semantic comparison (with e.g. inner products), The meaning of a sentence is given by composing these terms together to get a diagram which can be interpreted in Vect, as in a DisCoCat model, or in fact in any monoidal category C.DisCoPy [5] is a string diagram modelling tool, but it is very powerful in one way - each diagram can be mapped to a specific tensor representation using a functor.

A commonly used semantics category for DisCoCat models is the category $C = FVec$ of finite-dimensional real vector spaces and linear maps $F : V \to W$. This category is compact, meaning that it is symmetric monoidal and that its left and right adjoints coincide. For a space V we have that $V^l = V^r = FVec(V, \mathbb{R})$. In such a DisCoCat model, a word such as *likes* corresponds to a vector of the tensor product $n \otimes s \otimes n$, also simply called a tensor. For example, the sentence *Alice likes Bob* is mapped to a vector of s given by the tensor contraction:

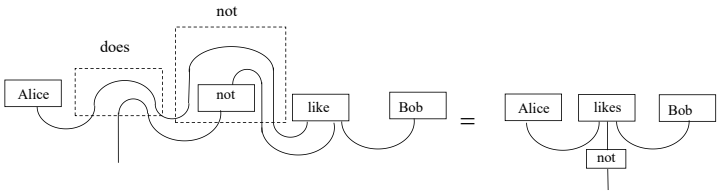
$$|\text{Alice likes Bob}\rangle_k = \sum_{i=1}^{n}\sum_{j=1}^{n} |\text{Alice}\rangle_i |\text{likes}\rangle_{i,k,j} |\text{Bob}\rangle_j \tag{2}$$

Our analysis of the DisCoCat compositional model and its interpretability raises a key question: How can the model's internal diagrammatic structure explain its specific outputs? To address this, we demonstrate how interpreted diagrams enable reasoning about the model's behavior. Consider a DisCoCat model where a verb, such as *likes* in the sentence *Alice likes Bob*, is represented as a channel. The word *likes* acts only on the wires corresponding to *Alice* and *Bob*, while behaving as the identity on all other discourse referents. Consequently, the resulting channel permits signaling exclusively between the *Alice* and *Bob* wires, with all other wires signaling only to themselves.



[5] https://github.com/oxford-quantum-group/discopy

The string diagram representation of the logical structure of the sentence *Alice does not like Bob*. Among them, *like* is regarded as a functional word. In the string diagram, *does* being entirely made up of wires, while *not* also involving a ¬-labeled box which represents negation of the meaning. The sentence's logical structure through diagrammatic rewriting becomes clear when we yank them:



In standard DisCoCat models, the meaning of a sentence is constructed by mapping the meanings of individual words to tensors, which are then composed according to the grammatical structure of the sentence. This framework allows meanings to be compared efficiently via operations such as inner products and supports data-driven learning. However, when dealing with sentences involving complex semantic phenomena—such as negation and quantification, as illustrated by the example *"no fish can swim here, but some are swimming"*—the type structures assigned to words become significantly more complicated. As a result, the number of parameters that must be learned increases rapidly, leading to scalability issues in practical applications.Higher-Order DisCoCat improves the efficiency of reduces parameter count, as shown in Tab.2.

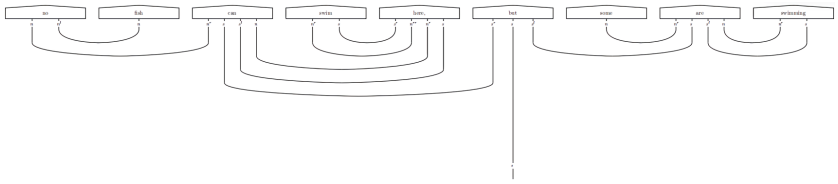
Table 2: Word Types and Their Categorical Semantics

Word	**Type**	**Meaning** $F(w \to t)$
no	$n \leftarrow n$	$\lambda f.\ \mathrm{cut}(f)$
fish	n	$(\mathrm{fish} : 1 \to N) \in \Sigma$
can	$(n \to s) \leftarrow (n \to s)$	$\lambda P.\ P$
swim	$n \to s$	$\lambda x.\ \mathrm{swim}(x)$
here	$(n \to s) \to (n \to s)$	$\lambda P x.\ \mathrm{here}(P(x))$
some	$n \leftarrow n$	$\lambda PQ.\ \exists x.\ P(x) \wedge Q(x)$
are	$(n \to s) \leftarrow n$	$\lambda PQ.Q(P(\mathrm{id}_N)^T)$

Thus, higher-order DisCoCat model[6] is needed to extend the composition functionality of the original framework. This approach combines the strengths of both the DisCoCat and Montague semantics paradigms. Specifically, we assign to each word a simply-typed lambda term, capturing higher-order semantic phenomena in a systematic way. Unlike in traditional Montague semantics, where the resulting logical forms are computationally intractable for machine learning (e.g., equivalence of first-order logic formulas is undecidable), our lambda terms operate over diagrammatic primitives. The meaning of a sentence is obtained by composing these lambda terms into a diagram, which can be interpreted in a vector space, as in a standard DisCoCat model, or more generally in any monoidal

[6] https://docs.discopy.org/en/main/notebooks/higher-order-discocat.html

category. Of course, when using the Web tool-DisCoCat diagram generator[7] to generate DisCoCat diagrams, we find that due to the NP-ellipsis on the right side of the quantifier *some* in the sentence, its type is inconsistent with that of the quantifier *no*.



We implement a formal categorical framework for logical semantic derivation, with complete computational details provided in Appendix A. Large Language Models (e.g., GPT) spontaneously acquire combinatorial rules and categorical structures within their high-dimensional latent spaces during training. Analyzing diagram representations aims to identify specific regions or directions corresponding to linguistic features, including syntactic categories, logical operations, and semantic roles. The DisCoCat framework enables parallel syntactic and semantic inference by mapping formal linguistic structures to the model's implicit representation space.

5 Conclusion and Future Work

This study addresses the interpretability challenges of deploying machine learning models in high-risk domains by proposing a category-theoretic diagrammatic semantic framework. Through a three-stage methodology of structural decomposition, semantic mapping, and compositional verification, our framework tries to explain AI models' generated behaviours. Unlike existing post-hoc interpretability methods that rely on model approximations and lack credibility, our intrinsic interpretability approach embeds verifiable semantic traceability during model design through rigorous preservation of compositional structures via categorical morphisms.

At the theoretical level, we demonstrate the potential of integrating formal semantic structures with contemporary model architectures to enhance interpretability and compositional reasoning, providing novel formal tools for analyzing the emergence mechanisms of hierarchical semantic structures. The visual encoding of model components and information flow through string diagrams not only enables interpretable validation of compositional reasoning processes, but also establishes strict mathematical correspondences at the syntax-semantics interface. While the current study focuses on some tractable subsets of grammatical phenomena, it lays the groundwork for more comprehensive formal-semantic integration in language understanding systems. This formal framework can be

[7] https://qnlp.cambridgequantum.com/generate.html

effectively applied to quantum natural language processing (QNLP),which employs categorical compositional semantics. The lambeq toolkit[8] orchestrates this workflow, integrating readers for multimodal input processing while supporting both knowledge-driven parameter initialization and data-driven optimization. This unified architecture bridges formal linguistics with quantum-native representations, enabling systematic evaluation of compositional models' expressivity across classical and quantum paradigms.

Future efforts will aim to extend the framework along several dimensions. One avenue involves refining hierarchical diagrammatic representations to retain expressivity in large-scale architectures such as Transformers while improving their structural transparency. Another direction explores compositional semantics in multimodal settings, with a focus on constructing unified semantic representations across modalities like text-image pairs. Finally, expanding the formal implementation to cover more complex syntactic phenomena—such as nested clauses and ellipsis—will be crucial to improving both the coverage and generalization of the approach, bringing formal theoretical insights closer to practical language understanding.

Acknowledgement. The paper is supported by Major Project of the National Social Science Foundation of China (No.19ZDA042) and Philosophy and Social Science Planning Projects of Guangdong Province(No.GD24CZX02)

References

[1] Alvarez Melis, D., Jaakkola, T.: Towards robust interpretability with self-explaining neural networks. In: Advances in Neural Information Processing Systems. vol. 31 (2018)
[2] Carvalho, D.V., Pereira, E.M., Cardoso, J.S.: Machine learning interpretability: A survey on methods and metrics. Electronics **8**(8), 832 (2019)
[3] Clark, S., Curran, J.R.: Wide-coverage efficient statistical parsing with CCG and log-linear models. Computational Linguistics **33**(4), 493–552 (2007)
[4] de Felice, G.: Categorical Tools for Natural Language Processing. Ph.D. thesis, University of Oxford (2022)
[5] Foerster, J.N., Gilmer, J., Sohl-Dickstein, J., Chorowski, J., Sussillo, D.: Input switched affine networks: An RNN architecture designed for interpretability. In: International Conference on Machine Learning. pp. 1136–1145 (2017)
[6] Freiesleben, T., König, G.: Dear XAI community, we need to talk! Fundamental misconceptions in current XAI research (2023)
[7] Graziani, M., Dutkiewicz, L., Calvaresi, D., Amorim, J.P., Yordanova, K., Vered, M., Nair, R., Abreu, P.H., Blanke, T., Pulignano, V., et al.: A global taxonomy of interpretable AI: unifying the terminology for the technical and social sciences. Artificial Intelligence Review **56**(4), 3473–3504 (2023)

[8] https://github.com/CQCL/lambeq

[8] Hudson, D.A., Manning, C.D.: Compositional attention networks for machine reasoning. In: International Conference on Learning Representations (2018)
[9] Ji, J., Qiu, T., Chen, B., Zhang, B., Lou, H., Wang, K., Duan, Y., He, Z., Zhou, J., Zhang, Z., Zeng, F., Ng, K.Y., Dai, J., Pan, X., O'Gara, A., Lei, Y., Xu, H., Tse, B., Fu, J., McAleer, S., Yang, Y., Wang, Y., Zhu, S.C., Guo, Y., Gao, W.: AI alignment: A comprehensive survey (2024)
[10] Li, C., Yao, K., Wang, J., Diao, B., Xu, Y., Zhang, Q.: Interpretable generative adversarial networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 1280–1288 (2022a)
[11] Li, J., Li, Y., Xiang, X., Xia, S.T., Dong, S., Cai, Y.: TNT: An interpretable Tree-Network-Tree Learning Framework using Knowledge Distillation. Entropy **22**(11), 1203 (2020)
[12] Mascharka, D., Tran, P., Soklaski, R., Majumdar, A.: Transparency by design: Closing the gap between performance and interpretability in visual reasoning. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4942–4950 (2018)
[13] Piedeleu, R., Zanasi, F.: An introduction to string diagrams for computer scientists (2023)
[14] Räuker, T., Ho, A., Casper, S., et al.: Toward transparent AI: A survey on interpreting the inner structures of deep neural networks. In: IEEE Conference on Secure and Trustworthy Machine Learning (SaTML) (2023)
[15] Sachdeva, N., McAuley, J.: Data distillation: A survey. Transactions on Machine Learning Research (2023)
[16] Selinger, P.: A survey of graphical languages for monoidal categories. In: New structures for physics, pp. 289–355 (2010)
[17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. vol. 30 (2017)
[18] Vijay-Shanker, K., Weir, D.J.: The equivalence of four extensions of contextfree grammars. Mathematical systems theory **27**(6), 511–546 (1994)
[19] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems **35**, 24824–24837 (2022)
[20] Yeung, R., Kartsaklis, D.: A CCG-based version of the DisCoCat framework. In: Proceedings of the Workshop on Semantic Spaces at the Intersection of NLP, Physics, and Cognitive Science (SemSpace). pp. 20–31. Association for Computational Linguistics, Groningen, The Netherlands (2021)

A Appendix

Listing 1.1: DisCoPy Implementation

```
# Step 1: Define Formula as a subclass of frobenius.Diagram
from discopy import frobenius
from discopy.balanced import Diagram
from discopy.tensor import Dim, Tensor
from discopy.cat import Category, factory

@factory
class Formula(frobenius.Diagram):
    ty_factory = frobenius.PRO # Natural numbers as objects

    def eval(self, size):
        return frobenius.Functor(
            ob=lambda _: Dim(size),
            ar=lambda box: box.data,
            cod=Category(Dim, Tensor[bool]))(self)

class Cut(frobenius.Bubble, Formula): pass
class Ligature(frobenius.Spider, Formula): pass
class Predicate(frobenius.Box, Formula): pass

Id, Formula.bubble_factory = Formula.id, Cut
Tensor[bool].bubble = lambda self, **_: self.map(lambda x: not x)

# Step 2: Parse natural language sentences
from discopy.grammar.categorial import Ty, Word, BA, FA
from discopy import Cup, Id
n, s = Ty('n'), Ty('s') # Noun and sentence types

# Lexicon definitions
no = Word("no", n << n)
fish = Word("fish", n)
can = Word("can", (n >> s) << (n >> s))
swim = Word("swim", n >> s)
here = Word("here", (n >> s) >> (n >> s))
Some = Word("Some", n << n)
are = Word("are", (n >> s) << n)
swimming = Word("swimming", n >> s)
but = Word("but", (s << s) >> s)

# Sentence composition
no_fish = no @ fish >> BA(n)
can_swim_here = can @ (swim @ here) >> FA((n >> s), (n >> s))
first_clause = no_fish @ can_swim_here >> FA(n, n >> s) >> Cup(n, s)

some_fish = Some @ fish >> BA(n)
are_swimming = are @ swimming >> FA(n, n >> s)
second_clause = some_fish @ are_swimming >> FA(n, n >> s) >> Cup(n, s)

full_sentence = but @ (first_clause @ second_clause) >> FA(s, s << s) >>
    Cup(s, s)
```

```
# Step 3: Random interpretations
from random import choice
size = 42
random_bits = lambda n=size: [choice([True, False]) for _ in range(n)]

# Predicate definitions
is_fish = random_bits()
is_swimming = random_bits()
can_swim_here_pred = [[choice([True, False]) for _ in range(size)]
                      for _ in range(size)]

# Step 4: Higher-order DisCoCat functor
from discopy import closed
from discopy.python import Function

F = closed.Functor(
    cod=Category(tuple[type, ...], Function),
    ob={s: Diagram, n: Diagram},
    ar={
        fish: lambda: frobenius.Box('fish', Ty(), n),
        swim: lambda: frobenius.Box('swim', n, s),
        no: lambda P: (P >> frobenius.Cup(n, n.r)).bubble(),
        Some: lambda P: P,
        but: lambda x, y: x @ y >> frobenius.Swap(s, s)
    })

evaluate = lambda sentence: bool(F(sentence)().eval(size))
assert evaluate(full_sentence) == (
    all(not is_fish[x] or not can_swim_here_pred[x][x]
        for x in range(size)) # "no fish can swim here"
    and any(is_fish[x] and is_swimming[x]
          for x in range(size)) # "some are swimming"
)
```