Extended Abstract Track

Composed-Program Induction on Latent Program Lattice

Jumyung Park JUMYUNG_UG@GM.GIST.AC.KR
Jiwon Park PARKJOHN58@GM.GIST.AC.KR
Jinseo Shim JINSEO5892@GM.GIST.AC.KR
Sejin Kim SEJINKIM@GIST.AC.KR
Paulina Vennemann* PAULINA.VENNMANN@WEB.DE
Sundong Kim SUNDONG@GIST.AC.KR
Gwangju Institute of Science and Technology (GIST)

Abstract

Compositional reasoning involves solving new problems by systematically composing basic primitives into structured transformations through three essential capabilities: learning compositional structure, inferring a program using trained compositional structure, and discovering refined primitives via abstraction of compositional structure. Existing approaches either lack explicit decomposition mechanisms for handling complex compositions or rely on hand-crafted primitives that limit adaptability. To address these limitations, we propose the Program Lattice Transformer (PLT) that learns compositional transformations with a structured latent program space. PLT preserves compositional structure by training an encoder where program effects and their compositions correspond to integer linear combinations of program bases, forming a discrete program lattice that captures the geometric structure of compositional reasoning. Program induction then reduces to solving a Closest Vector Problem (CVP) in this lattice, enabling principled inference through two complementary modes: fast System-1 reasoning via solving CVP to infer a composed program and deliberate System-2 reasoning through stepwise lattice walks with intermediate verification. The framework naturally supports abstraction discovery through lattice reduction, which refines primitive bases to improve efficiency and uncover more fundamental components. This work connects neural and symbolic reasoning by providing a mathematically principled framework for learning and inference in compositional domains.

Keywords: compositional reasoning, program induction, lattice problem, dual-process reasoning, abstraction discovery

1. Introduction

Compositional reasoning is the process of solving new problems by systematically composing primitive operations into complex transformations. This ability hinges on three key components: (i) learning the compositional structure that governs how primitives interact, (ii) inference using this learned structure to generalize from limited examples, and (iii) abstraction discovery that compresses and refines primitives to accelerate future reasoning. As the Kaleidoscope Hypothesis (Chollet, 2024) suggests, we focus on realizing the symmetric structure beneath apparent complexity.

Compositional reasoning tasks, such as those found in the Abstraction and Reasoning Corpus(ARC-AGI) (Chollet, 2019), require identifying specific rules from 3–4 input-output

* MS student at Technische Universität Hamburg. This work is done when she was an intern at GIST.

(IO) pair examples, where each task follows a particular transformation. The apparent diversity of ARC transformations—from geometric rotations to pattern completions—is generated by repeated composition of a compact set of primitives. The term "composition" arises because these rules consist of combinations of basic primitive programs.

Building on this view, we propose the Program Lattice Transformer (PLT). The central idea is to endow the model with a structured latent program space that respects the compositional structure. Drawing on category theory, we design the encoder to be trained to mimic a functor that maps input–output pairs and their transformations into a latent space while preserving their compositional structure. In this space, primitive operations correspond to each latent program basis, and composed programs emerge as integer linear combinations of these bases, forming a latent program lattice. Program induction then reduces to solving a Closest Vector Problem (CVP) (Micciancio and Goldwasser, 2002) in this lattice. Moreover, abstraction discovery is achieved through lattice reduction (Wübben et al., 2011), which refines non-atomic programs into shorter, more orthogonal primitives spanning the same lattice. Lattice program space naturally supports dual-process reasoning (Frankish, 2010). For System-1 inference, PLT encodes observed input–output pairs, takes their difference as a latent program effect, adds this effect to a new input's embedding, and decodes—fast, intuitive, approximate. For System-2 inference, PLT factorizes the same effect into a sequence of basis and executes them stepwise—repeatedly decoding/encoding in a guided "lattice walk" that verifies and refines intermediate states. Both modes arise from the same latent structure, with System-2 improving the basis via lattice reduction and thereby strengthening future System-1 inferences.

In this way, PLT integrates learning, inference, and abstraction into a unified framework for compositional reasoning. This PLT research addresses the fundamental limitations of existing approaches to compositional reasoning. Neural-based models like ViTARC (Li et al., 2024) and Latent Program Networks (Macfarlane and Bonnet, 2024) infer rules in a single forward pass but struggle with complex compositions due to a lack of explicit decomposition mechanisms. Program synthesis approaches like DreamCoder (Ellis et al., 2021) explicitly model compositional structure but depend on hand-crafted primitives. PLT addresses this by learning primitives from the representation of IO pairs, enabling both compositional program construction and adaptive primitive discovery through lattice reduction.

2. Program Lattice Transformer (PLT)

Task Consider a set of m example input-output pairs $T_i = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ where the output object y is generated by applying a consistent program f_i on the corresponding input object x: $\forall (x, y) \in T_i,\ f_i(x) = y$. For a new input x_{m+1}, the model should predict the corresponding output y_{m+1} that would be generated by applying the same consistent program f_i. The program f_i for each task T_i is a composition of programs from a set of d primitive programs $P = \{p_1, p_2, \ldots, p_d\}$.

Learning the Compositional Structure (Training) From given input-output pairs T_i, the model can only observe the apparent effect of applying the underlying consistent program. PLT models this as latent program effect $\mathbf{\Delta}_k$ by embedding each input and output (x_k, y_k) into vectors $(E(x_k), E(y_k))$ with an encoder E, and averaging (pair permutation

invariant aggregation) the difference vectors.

$$\bar{\boldsymbol{\Delta}}_i = \frac{1}{m}\sum_{k=1}^{m}\boldsymbol{\Delta}_k \;= \frac{1}{m}\sum_{k=1}^{m}\{E(y_k) - E(x_k)\}$$

The underlying programs for the first d tasks are treated as the initial set of primitive programs, and the latent program effects $\bar{\boldsymbol{\Delta}}_1, \bar{\boldsymbol{\Delta}}_2, \ldots, \bar{\boldsymbol{\Delta}}_d$ form a latent program basis matrix $B = \begin{bmatrix}\boldsymbol{p_1} & \boldsymbol{p_2} & \cdots & \boldsymbol{p_d}\end{bmatrix}$ where $\boldsymbol{p}_1 = \bar{\boldsymbol{\Delta}}_1, \boldsymbol{p}_2 = \bar{\boldsymbol{\Delta}}_2, \ldots, \boldsymbol{p}_d = \bar{\boldsymbol{\Delta}}_d$.

Modeling program composition as an integer linear combination of latent program basis, the latent program basis spans a latent program lattice L whose lattice points correspond to the latent program effects of valid discrete composition of primitive programs.

$$L = \{B\boldsymbol{z} : \boldsymbol{z} \in \mathbb{Z}^d\}$$

As a result, an integer vector $\boldsymbol{z}$ corresponding to a lattice point on this latent program lattice represents how many of each primitive program are composed in the corresponding program.

Since vector addition is commutative while program composition is not, modeling program composition as the addition of latent program vectors loses information about the order of composition. To break this commutativity on the lattice, we introduce a small, unique perturbation $\varepsilon_{k,t}$ for the latent program basis p_k being composed as the t-th primitive program in the composed program. As long as the sum of these perturbations is bounded by the packing radius of the lattice, they do not alter the closest lattice point solution but can be decoded to reconstruct the sequence of operations. This approach approximates non-commutative structure while retaining the tractability of the lattice representation, though it limits the maximum reliable program length in practice.

The encoder is parameterized and trained to model the compositional structure as a discrete lattice by minimizing the deviance from the additive structure in latent program space:

$$\forall i \in [1,d], \forall j \in [1,d], \quad p = p_j \circ p_i \implies \boldsymbol{p_i} + \boldsymbol{p_j} \approx \boldsymbol{p}$$

Program Induction on the Learned Structure (Inference) PLT solves the tasks utilizing the learned latent program lattice in two complementary modes: System-1 and System-2 inference.

System-1 Inference is a fast, approximate, one-step program induction. Given a task $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, PLT encodes the given input-output example pairs into the latent program space to obtain the aggregate latent program effect:

$$\bar{\boldsymbol{\Delta}} = \frac{1}{m}\sum_{k=1}^{m}\{E(y_k) - E(x_k)\}$$

The learned latent program lattice L provides a strong inductive bias that only the lattice points $\boldsymbol{v} \in L$ represent a valid discrete composition of primitive programs, which reduces program induction to the Closest Vector Problem (CVP) on the latent program lattice. The latent program $\boldsymbol{p}$ consistent with the given example pairs is induced as the closest lattice point on the L to the aggregate latent program effect $\bar{\boldsymbol{\Delta}}$.

$$\boldsymbol{p} = \arg\min_{\boldsymbol{v}\in L} \|\boldsymbol{v} - \bar{\boldsymbol{\Delta}}\|$$

This induced latent program $\boldsymbol{p}$ can be directly added to the encoding of the new input x_{m+1} to simulate the effect of applying the induced program, and decoded with the decoder D to infer the corresponding new output y_{m+1}:

$$\hat{y_{m+1}} = D(E(x_{m+1}) + \boldsymbol{p})$$

System-2 Inference differs from System-1 by decomposing the induced program into a sequence of primitive programs, then applying them step-by-step. The induced latent program $\boldsymbol{p}$ can be decomposed into an integer linear combination of latent program bases. Decoding the deviation from lattice points into a sum of the perturbations added to encode the order of composition, we can recover not only the number of each primitive but also their composition order. This yields a sequence of primitive programs, which can be simulated by iteratively encoding, adding the corresponding latent program basis, and decoding. Geometrically, this is a walk on the latent program lattice towards the lattice point that represents the inferred latent composed program. In contrast to system-1, this allows the model to tackle the compounding error of composing approximate programs.

Abstraction program discovery (Library Building) A final component of PLT is abstraction discovery. Instead of discovering better abstractions through heuristic-driven refactorization from predefined primitives, PLT refines its library through lattice reduction. Lattice reduction, such as LLL (Nguyen and Vallée, 2010), finds a new integer basis B' spanning the same lattice points as the old basis B but with shorter, and more orthogonal vectors - akin to Gram-Schmidt orthogonalization constrained to integer combinations.

$$L = \{B\boldsymbol{z} | \boldsymbol{z} \in \mathbb{Z}^d\} = \{B'\boldsymbol{z} | \boldsymbol{z} \in \mathbb{Z}^d\}$$

Lattice reduction refines the latent program basis to be shorter and more orthogonal, spanning the same lattice points more efficiently. This leads to improved efficiency of the CVP solver, accelerating further reasoning. Even if initial bases correspond to complex, non-atomic operations, lattice reduction can produce refined bases that improve CVP efficiency and may uncover more fundamental program components.

3. Discussion

PLT suggests how compositional reasoning can be recast into a lattice problem on the latent program lattice. By introducing functor-inspired training, the encoder learns a geometric latent program space, preserving the compositional structure. The latent program lattice provides a natural interface between System-1 and System-2: fast but approximate one-step CVP (System-1) and slower, explicit sequential composition with lattice walk (System-2). Furthermore, the lattice reduction functions as a principled abstraction discovery algorithm.

However, this approach has several components that need more work. The lattice representation is abelian, while many program domains are not; perturbation encodings restore order but only for bounded sequence lengths. Reduced bases are guaranteed to be computationally more efficient, but not always semantically meaningful. Latent program effects are modeled as input-independent translations, which may not hold in all cases. Additionally, the method has not yet been empirically validated. Finally, CVP remains hard in the worst case, so tractability depends on approximate solvers and the quality of the basis.

Despite these limitations, PLT suggests a novel frame bridging neural and symbolic reasoning. By modeling the compositional structure as a lattice, it opens up a mathematical toolbox from lattice theory that can be applied and extend this framework further.

References

François Chollet. On the Measure of Intelligence. *arXiv:1911.01547*, 2019.

François Chollet. Pattern recognition vs true intelligence, 2024. URL https://www.youtube.com/watch?v=JTU8Ha4Jyfc. YouTube video, Machine Learning Street Talk.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pages 835–850, 2021.

Keith Frankish. Dual-process and dual-system theories of reasoning. *Philosophy Compass*, 5(10):914–926, 2010.

Wenhao Li, Yudong Xu, Scott Sanner, and Elias Boutros Khalil. Tackling The Abstraction and Reasoning Corpus with Vision Transformers: The Importance of 2D Representation, Positions, and Objects. *arXiv:2410.06405*, 2024.

Matthew V Macfarlane and Clément Bonnet. Searching latent program spaces. *arXiv preprint arXiv:2411.08706*, 2024.

Daniele Micciancio and Shafi Goldwasser. Closest vector problem. In *Complexity of lattice problems: a cryptographic perspective*, pages 45–68. Springer, 2002.

Phong Q Nguyen and Brigitte Vallée. *The LLL algorithm*. Springer, 2010.

Dirk Wübben, Dominik Seethaler, Joakim Jalden, and Gerald Matz. Lattice reduction. *IEEE Signal Processing Magazine*, 28(3):70–91, 2011.