Extended Abstract Track

# Composed-Program Induction with 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 requires learning structure, inferring programs, and discovering refined primitives. Existing approaches either lack explicit decomposition mechanisms or rely on hand-crafted primitives. We propose the Program Lattice Auto Encoder (PLAE), which learns compositional transformations in a structured latent program space. PLAE trains an encoder where program effects correspond to integer linear combinations of program bases, forming a discrete program lattice. Program induction reduces to solving the Closest Vector Problem (CVP), enabling two complementary inference modes: fast System-1 reasoning via CVP and deliberate System-2 reasoning through stepwise lattice walks with intermediate verification. The framework supports abstraction discovery through lattice reduction, which refines primitive bases to uncover more fundamental components. This work connects neural and symbolic reasoning by providing a mathematically principled framework for 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.

Building on this view, we propose the Program Lattice Auto Encoder(PLAE). 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) which

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

© 2025 J. Park, J. Park, J. Shim, S. Kim, P. Vennemann & S. Kim.

Extended Abstract Track

find a vector $v \in L$ such that minimize the distance $\|v - t\|$ where $t \in \mathbb{R}^n$ is arbitrary target vector in lattice $L$. 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, PLAE 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, PLAE 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, PLAE integrates learning, inference, and abstraction into a unified framework for compositional reasoning.

## 2. Program Lattice Auto Encoder(PLAE)

**Task** Consider a set of $m$ example input-output pairs $T_i = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ 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 $P = \{p_1, p_2, \dots, p_d\}$.

**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. PLAE models this as latent program effect $\boldsymbol{\Delta}_k$ 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 $\bar{\boldsymbol{\Delta}}_1, \bar{\boldsymbol{\Delta}}_2, \dots, \bar{\boldsymbol{\Delta}}_d$ form a latent program basis matrix $B = \begin{bmatrix} \boldsymbol{p_1} & \boldsymbol{p_2} & \cdots & \boldsymbol{p_d} \end{bmatrix}$ where $\boldsymbol{p}_1 = \bar{\boldsymbol{\Delta}}_1, \boldsymbol{p}_2 = \bar{\boldsymbol{\Delta}}_2, \dots, \boldsymbol{p}_d = \bar{\boldsymbol{\Delta}}_d$.

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, we introduce small perturbations $\varepsilon_{k,t}$ for each basis $p_k$ at position $t$. These perturbations, bounded by the lattice packing radius, preserve CVP solutions while encoding operation order.

Extended Abstract Track

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)** PLAE 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 $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, PLAE 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 $y_{m+1}$:

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

This "one-step" application is highly efficient, but as an approximation, it bypasses intermediate verification. This makes it vulnerable to compounding errors, especially for complex tasks requiring long sequences of composed programs, where the final state p may significantly deviate from the true manifold.

***System-2 Inference*** provides a deliberate, high-fidelity alternative specifically to address the compounding error inherent in System-1's approximation. It 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 $\boldsymbol{p}$ is first decomposed into its integer coordinates $z$ (where $p = Bz$), which specify which primitive programs are used and their counts. Separately, by decoding the deviation ($\boldsymbol{\delta} = \bar{\Delta} - p$) from the lattice point, we recover the crucial composition order of these primitives. This yields an ordered sequence of programs $[p_{i_1}, \ldots, p_{i_n}]$, 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 PLAE is abstraction discovery. Instead of discovering better abstractions through heuristic-driven refactorization from predefined primitives, PLAE 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. Experiment Design

We design our experiments to systematically validate the lattice-based compositional structure across increasing levels of complexity. First, we verify whether the encoder can learn to preserve lattice structure in abelian, commutative domains where program composition naturally aligns with vector addition in the latent space. Second, we test the framework's robustness in non-abelian, non-commutative settings by evaluating whether perturbation encodings can effectively recover operation ordering despite the inherent commutativity of the lattice representation. Finally, we compare System-1 (direct CVP-based) and System-2 (stepwise lattice walk) inference modes to assess whether the learned structure enables accurate program application to novel inputs, with System-2 providing intermediate state verification to validate compositional correctness. This evaluation reveals the trade-offs between computational efficiency and compositional fidelity in different inference strategies.

## 4. Discussion

PLAE 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, PLAE 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.

Extended Abstract Track

## References

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

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

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.

Extended Abstract Track

## Appendix A. Algorithms

### A.1. System-1 Inference

**Algorithm 1:** System-1 Inference: One-step Program Induction

```
Input: Training set of input-output pairs $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, new
       input $x_{m+1}$
Input: Encoder $E$, Decoder $D$, Latent program basis $B = [p_1, p_2, \ldots, p_d]$
Output: Predicted output $\hat{y}_{m+1}$
Step 1: Encode and aggregate latent program effect
$\bar{\Delta} \leftarrow \frac{1}{m} \sum_{k=1}^{m} \{E(y_k) - E(x_k)\}$
Step 2: Solve CVP on lattice spanned by $B$
$p \leftarrow \arg\min_{v \in \mathcal{L}(B)} \|v - \bar{\Delta}\|$ // $\mathcal{L}(B)$ = lattice spanned by $B$
Step 3: Apply induced program and decode
$\hat{y}_{m+1} \leftarrow D(E(x_{m+1}) + p)$
return $\hat{y}_{m+1}$
```

### A.2. System-2 Inference

**Algorithm 2:** System-2 Inference: Step-by-step Lattice Walk

```
Input: Training set $T = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, new input $x_{m+1}$
Input: Encoder $E$, Decoder $D$, Latent program basis $B = [p_1, p_2, \ldots, p_d]$
Output: Predicted output $\hat{y}_{m+1}$
$s_0 \leftarrow E(x_{m+1})$ // Initial state
for $t = 1, \ldots, n$ do
    Step 1: Encode and solve CVP
    $\bar{\Delta} \leftarrow \frac{1}{m} \sum_{k=1}^{m} \{E(y_k) - E(x_k)\}$
    $p \leftarrow \arg\min_{v \in \mathcal{L}(B)} \|v - \bar{\Delta}\|$ // $\mathcal{L}(B)$ = lattice spanned by $B$
    Step 2: Decompose into primitive sequence
    $\boldsymbol{\delta} \leftarrow \bar{\Delta} - p$ // Get perturbation/deviation
    $p = Bz$ for some $z \in \mathbb{Z}^d$ // Get integer coordinates
    $[p_{i_1}, p_{i_2}, \ldots, p_{i_n}] \leftarrow \text{OrderedPrimitives}(\boldsymbol{\delta}, z)$ // Decode order
    Step 3: Execute step with verification
    $s_t \leftarrow s_{t-1} + p_{i_t}$ // Walk on lattice
    $x_{\text{intermediate}} \leftarrow D(s_t)$ // Decode intermediate state
    $s_t \leftarrow E(x_{\text{intermediate}})$ // Re-encode for error correction
end
Step 4: Final decoding
$\hat{y}_{m+1} \leftarrow D(s_n)$
return $\hat{y}_{m+1}$
```

Extended Abstract Track

## Appendix B. Training Loss

### B.1. Consistency Loss

The embedding points of input-output pair, $\{x_i, y_i\}$ on the lattice point that is performed by the encoder is denoted by $E(x_i)$ and $E(y_i)$ respectively. Then the difference vector between input and output embedding points are $\Delta_i = E(y_i) - E(x_i)$, called **program effects** depending on which program are used to obtain output from input. For instance, suppose that encoder embed data randomly, then program effects representing data utilizing same program will be different. However, vector embedding corresponding to input-output pair which utilize same program must be embedded identically based on PLAE mechanism. On the other words, for data $\{x_i, y_i\}$ including same program, all program effects $\Delta_i$ must be aligned in the same direction on the lattice. To achieve this, consistency loss are formed so that all program effects converge in the direction of mean program effects, $\bar{\Delta}$. Consistency loss for input-output pairs $\{x_i, y_i\}$ which can be obtained by appying same program:

$$L_{consist} = \sum_{i=1}^{n} \{(E(y_i) - E(x_i)) - \bar{\Delta}\}^2 = \sum_{i=1}^{n} (\Delta_i - \bar{\Delta})^2$$

### B.2. Composition Loss

In PLAE, compositional properties of programs are conserved via structure of lattice. The most ideal situation is entire embedding points are distributed on lattice points which means that all of programs can be split completely with small primitive programs and finding closest lattice points is not needed. Nevertheless, actual domain of embedding space is continuous real plane,is not discrete lattice points. Therefore, sometimes programs corresponding to closest lattice points is not matched with actual programs used in input-output pair because gaps exists between program effects and lattice point. Then composition loss leads each program effects to be close with correct lattice points. For $\{x_i, y_i\}$ whose the most closest lattice point is $Bz$ where $B = [\boldsymbol{p_1 p_2 ... p_d}]$ represent matrix consist of basis vector $\boldsymbol{p_i}$ matching with primitive program and $\boldsymbol{z} = [z_1, z_2, ..., z_d]^\top, z_i \in \mathbb{Z}$ is program sequence.

$$L_{comp} = \sum_{i=1}^{d} (\Delta_i - \boldsymbol{p^*})^2 = \sum_{i=1}^{d} \{(E(y_i) - E(x_i)) - B\boldsymbol{z}\}^2$$

where $\boldsymbol{p^*}$ is target lattice points corresponding to each programs for input-output pairs.