Thesis for Bachelor's Degree

Learning Structured Latent Spaces for Compositional Program Induction

Jumyung Park

Department of Electrical Engineering and Computer Science,
College of Information and Computing

Gwangju Institute of Science and Technology

2026

학 사 학 위 논 문

합성프로그램 귀추를 위한 구조화된 잠재 공간 학습

박주명

정보컴퓨팅대학
전기전자컴퓨터공학과

광 주 과 학 기 술 원

2026

Learning Structured Latent Spaces for Compositional Program Induction

Advisor: Sundong Kim

by

Jumyung Park

Department of Electrical Engineering and Computer Science,
College of Information and Computing
Gwangju Institute of Science and Technology

A thesis submitted to the faculty of the Gwangju Institute of Science and Technology in partial fulfillment of the requirements for the degree of Bachelor of Science in the Electrical Engineering and Computer Science, College of Information and Computing Concentration

Gwangju, Republic of Korea
June 15, 2026
Approved by

Professor Sundong Kim
Committee Chair

Learning Structured Latent Spaces for Compositional Program Induction

Jumyung Park

Accepted in partial fulfillment of the requirements for
the degree of Bachelor of Science

June 15, 2026

Committee Chair _____
Prof. Sundong Kim

Committee Member _____
Prof. Kangil Kim

Dedicated to my family.

BS/EC
20235073

Jumyung Park(박주명). Learning Structured Latent Spaces for Compositional Program Induction(합성프로그램 귀추를 위한 구조화된 잠재 공간 학습). Department of Electrical Engineering and Computer Science, College of Information and Computing(정보컴퓨팅대학 전기전자컴퓨터공학과). 2026. 33p. Advisor: Prof. Sundong Kim(김선동) .

Abstract

Program induction from sparse demonstrations is severely underspecified: many distinct programs may agree with the same small set of examples while differing in how they generalize. This thesis argues that the central issue is therefore representational, and proposes that abstraction be understood as organizing a latent space so that latent program induction becomes a structured search problem rather than unconstrained prediction. It develops a first-order structured latent program space and uses its strengths and limitations to clarify what a latent representation must preserve if sparse demonstrations are to become informative.

BS/EC 20235073 박주명. 합성프로그램 귀추를 위한 구조화된 잠재 공간 학습. 정보컴퓨팅대학 전기전자컴퓨터공학과. 2026. 33p. 지도교수: 김선동.

국 문 요 약

희소한 예시로부터 프로그램을 추론하는 문제는 본질적으로 심각하게 과소결정되어 있다. 동일한 소수의 예시에 대해 서로 다른 많은 프로그램들이 모두 일치할 수 있지만, 일반화 방식에서는 서로 크게 달라질 수 있기 때문이다. 본 논문은 따라서 문제의 핵심이 표현에 있다고 주장하며, 추상을 잠재 프로그램 추론이 비제약적 예측이 아니라 구조화된 탐색의 문제가 되도록 잠재 공간을 조직하는 것으로 이해할 것을 제안한다. 이를 위해 본 논문은 1차적 구조를 갖는 구조화된 잠재 프로그램 공간을 개발하고, 그 강점과 한계를 분석함으로써 희소한 예시들이 실제로 정보를 제공하기 위해 잠재 표현이 무엇을 보존해야 하는지를 밝힌다.

Contents

Chapter 1
Introduction

Machine learning is often formulated as the problem of approximating an unknown function from finite observations. Given a limited sample drawn from an unknown distribution, the learner must construct a predictor that generalizes to unseen cases [1]. A central challenge is to make such generalization increasingly data-efficient and robust, so that reliable inference remains possible from only a small number of observations [2]. At the most data-scarce end of this spectrum, a learner must infer and apply a hidden rule from only a handful of demonstrations.

Across the literature, this setting appears under different names and with different emphases: programming-by-example [3, 4, 5], inductive programming [6], program synthesis [7, 8, 9], neural program induction [10], and neural processes [11]. These threads of research do not establish a unified problem definition, but they intersect in the problem of extreme generalization from sparse demonstrations of programs.

Despite their differences, these research threads converge on a common setting. A learner is given a small set of observed input–output examples generated by an unknown program, together with additional inputs whose outputs are withheld. The task is to predict the withheld outputs in a way that is consistent with the observed examples. This need not require explicitly recovering or articulating the underlying program. It is sufficient to infer an internal hypothesis that generalizes correctly.

The difficulty of this setting is not simply that data is scarce. It is that the hidden program is radically underspecified in the raw data space. Many distinct hypotheses may agree with the same sparse observations, while differing substantially in how they extrapolate beyond them. The central scientific question is therefore not only how to infer from demonstrations, but what kind of representation makes such inference well posed enough to support reliable generalization.

The central claim of this thesis is representational. It is that *abstraction can be understood as compiling program algebra into latent geometry.* Under this view, a learned latent space is not merely a feature space for reconstruction. It is a structured space in which hidden transformations can be inferred by search. This is not a claim that arbitrary program algebras admit lossless geometric embeddings, or that a single latent geometry is universally appropriate. It is a claim that, for suitable task families, learning can shape a latent space so that program induction becomes sufficiently selective and efficient to support reliable generalization.

This thesis develops that viewpoint in stages. It first formulates sparse program induction and compositional program induction, and then interprets program induction as search in a latent program space shaped by learning. It next develops a first explicit structured latent program space in which valid programs form a discrete compositional subset of the ambient latent space, making induction a geometric search problem over valid latent hypotheses. It then examines the limitations of this first-order structure and uses those limitations to clarify the representational requirements for richer latent geometry.

Chapter 2
Preliminaries

2.1 Program Induction

We consider a data domain $\mathcal{X}$ and a *program space* $\mathcal{P}$, where each program is a mapping $p : \mathcal{X} \to \mathcal{X}$. A task T is obtained by sampling a program p from a distribution over the program space $\mathcal{P}$ and then sampling a finite set of $(m + k)$ input–output pairs from that program,

$$T = \{(x_i, y_i)\}_{i=1}^{m+k}, \qquad y_i = p(x_i).$$

The task is partitioned into m demonstration pairs and k query pairs. At inference time, a solver receives the demonstration pairs $\{(x_i, y_i)\}_{i=1}^{m}$ together with the query inputs $\{x_i\}_{i=m+1}^{m+k}$, and it must infer the corresponding query outputs $\{y_i\}_{i=m+1}^{m+k}$.

A *dataset* $\mathcal{D}$ is a set of tasks,

$$\mathcal{D} = \{T^{(t)}\}_{t=1}^{N}.$$

We partition $\mathcal{D}$ into a training set and a test set. A solver is trained on the training tasks. On the test tasks, it is evaluated by comparing its predicted query outputs against the ground-truth outputs of the query pairs, which are withheld from the solver at evaluation time.

This formulation is intentionally broad. Many existing formulations in related literatures can be viewed as instances of this setting with additional assumptions about the program space, the observation process, or the inference procedure.

2.2 Compositional Program Induction

The setting considered in this thesis is not program induction in the most unconstrained sense, but *compositional* program induction. The additional assumption is that the program space of interest is generated from a smaller set of primitive programs. If we denote this primitive set by

$$\Pi = \{\pi_1, \ldots, \pi_r\},$$

then the programs of interest are assumed to lie in a subset of the finite compositions of elements of Π. In other words, a program of interest takes the form

$$p = \pi_{i_\ell} \circ \cdots \circ \pi_{i_1}$$

for some finite sequence of primitive programs. The primitive programs need not be given explicitly, and they need not appear in training in isolated form. The assumption is only that the relevant program space has shared compositional structure.

This assumption changes the generalization problem substantially. Without it, the training tasks are merely partial observations of distinct programs, and generalization is largely limited to whatever notion of proximity the model learns around the programs observed during training. Under the compositional assumption, by contrast, observed programs need not be isolated. Distinct training tasks may share latent components even when their full programs differ, making it possible to generalize beyond the specific programs seen in training, including to unseen compositions at test time.

2.3 Latent Program Space Search

Latent Program Networks (LPNs) [12] are important to the present thesis because they make explicit a structural formulation that will be generalized later: program induction may be treated as search in a learned latent program space. The importance of this formulation lies not only in the particular architecture proposed in that work, but in the abstraction it makes available.

At that level of abstraction, the encoder maps a task's demonstrations to a latent program hypothesis, while the decoder produces outputs conditioned on that hypothesis. The hidden program is therefore not represented as an explicit symbolic object, but as a latent variable mediating between demonstrations and predicted outputs. The latent program space is consequently not merely a representational buffer; it is the space in which inference is carried out.

LPN also makes explicit that test-time adaptation may itself be understood as search over this latent space. Starting from the encoder's initial latent hypothesis, one refines that hypothesis by seeking a latent program that better explains the demonstrations under the decoder. Training is therefore not only the fitting of an encoder and decoder; it is also the process by which the latent program space is organized so that the intended test-time search procedure becomes effective.

The broader significance of LPN is thus not restricted to its particular choice of smooth latent space or gradient-based optimization. Its more general contribution is to provide a blueprint in which program induction is framed as search in a learned latent program space, and training is understood as shaping that space for the chosen inference procedure.

2.4 Structured Relations in Learned Representations

The idea that a learned latent space may acquire structured internal dynamics is not unique to program induction. A recurring observation in representation learning is that latent spaces can organize not only objects, but also relations among them, in ways that simplify downstream reasoning. In the case of word embeddings, for example, semantic or syntactic relations may be reflected by relatively simple geometric operations in the learned space. This phenomenon has been widely discussed both in early work on analogical structure in embeddings and in more recent analyses of when and why such linearized relations emerge [13, 14].

The relevance of this observation to the present thesis is limited but important. It suggests that learned latent spaces need not be judged only by how well they separate or

reconstruct data. They may also be judged by whether they organize certain relations of interest into a simpler geometric form. In other words, a representation can be useful because it makes some family of transformations or interactions easier to express and easier to infer.

This thesis adopts that broader lesson, but applies it to a different object. The goal is not to recover semantic analogies among tokens, nor to claim that program spaces should behave exactly like word embedding spaces. Rather, the point is that if representation learning can simplify relational structure in one domain, then it is reasonable to ask whether it can do the same for the relations that matter in program induction. In the present setting, the central relation is composition.

Chapter 3
Learning Abstractions for Program Induction

The previous chapter defined the problem setting and introduced the central assumptions of this thesis. This chapter turns to the representational question that follows from that setup. Given sparse observations of a hidden program, the central issue is what kind of abstraction can make inference feasible. The approach developed here is to learn a representation in which the relevant family of programs admits a simple and searchable latent realization.

3.1 Abstraction as Compilation: From Algebra to Geometry

To exploit compositionality without prescribing a fixed symbolic language, we consider a neural encoder–decoder model that learns a latent representation of states and transformations. The encoder E maps data into a latent space, and the decoder D maps latent states back into data space. A program in data space is then emulated in latent space by a latent transformation acting on encoded states.

Learning therefore plays a role beyond reconstruction. It must shape the latent space so that the program family of interest admits a *simple realization* in that space. Under ordinary reconstruction pressure, the encoder may organize data into latent regions that can be decoded reliably. A data-space program then appears in latent space as movement from one region to another. The relevant abstractions are those in which such movements become sufficiently simple and regular to be inferred from only a few examples.

This yields the central interpretation pursued in the remainder of the thesis. Training can be understood as compiling the compositional structure of the underlying

program family into the geometry of the latent space. Test-time program induction then becomes a search for a latent program that reproduces the demonstrations within that structured space. The architectural and representational choices developed later are motivated by this viewpoint.

3.2 The Limits of Raw Program Space

At the level of the raw data space, the hidden program is severely underdetermined. A few demonstration pairs constrain only a small portion of the behavior of an arbitrary mapping $p : \mathcal{X} \to \mathcal{X}$. If the ambient program space is left unconstrained, the number of hypotheses consistent with the observed demonstrations is enormous. In that setting, correct extrapolation to unseen inputs is difficult to justify. Even if a solver can fit the demonstrations, there is no guarantee that the resulting hypothesis will be useful beyond them.

One classical response is to restrict the hypothesis class directly in the data space. This is reasonable in principle, but it introduces an immediate tradeoff. If the restriction is too loose, the class still contains many irrelevant programs and remains difficult to search. If the restriction is too severe, it may exclude programs that belong to the target family. This is especially problematic when the task family is not naturally expressed by a fixed hand-designed language, or when the relevant structure is only partially visible through sparse examples.

This tradeoff motivates a shift from restricting programs in the raw data space to seeking an abstraction in which the same task family admits a simpler program class. The problem then becomes how to learn a space in which search is informative and tractable.

3.3 Abstraction and Search

The argument of this chapter is algorithmic in character. A search procedure is effective only relative to a data structure that makes the search tractable. The same procedure may succeed or fail depending on how candidate hypotheses are organized.

In the present setting, the latent space serves as the data structure in which program induction takes place.

A useful abstraction, in this sense, is one that makes latent program inference selective, efficient, and reliable enough for the target tasks. Selective means that sparse demonstrations sharply constrain the latent hypothesis. Efficient means that the induced hypothesis can be found by a simple inference procedure. Reliable means that the resulting hypothesis extrapolates correctly on the query inputs often enough to be useful. These requirements are task-dependent. There is no representation that is universally optimal. The representation should therefore be judged by whether it organizes the task family so that the intended inference procedure works well.

Abstraction is therefore evaluated primarily by its role in inference. Compression may be a consequence of this process, but the central objective is to learn a representation in which the hidden transformation is easy to identify from sparse evidence. Equivalently, the target abstraction is one in which latent program induction becomes sharply constrained for the tasks of interest.

3.4 Simulating Programs in Latent Space

This viewpoint leads naturally to an encoder–decoder formulation. Let E map data into a latent space and let D map latent states back into the data space. A hidden data-space program is then represented indirectly by a latent transformation acting on encoded states. The task shifts from searching over arbitrary raw mappings to searching over latent programs in a space shaped by learning.

This construction has a simple geometric interpretation. Under ordinary reconstruction pressure, the encoder may separate different classes or regions of data into latent clusters that can be decoded reliably. A data-space program then appears in latent space as movement from one region to another. The relevant abstractions are those in which such movements become sufficiently simple and regular to be inferred from only a few examples.

The encoder and decoder remain responsible for representation and reconstruction.

The relevant change lies in the geometry they are encouraged to learn. A suitable representation for this problem is one in which transformations induce structured latent motion and avoid arbitrary jumps. In this sense, the training objective asks the model to reconstruct the data in coordinates that are useful for inferring programs.

3.5 Why the Latent Program Class Must Be Restricted

The desired geometry must be encouraged by the structure of the learning problem. If the latent program class is too expressive, the latent space can absorb complexity without acquiring meaningful organization. In that case, the model may solve each task by placing complicated task-specific behavior inside the latent program itself.

Accordingly, the strategy pursued here deliberately restricts the latent program class so that it remains simple, and the model is required to solve the task under that restriction. The restricted class need not be adequate in the raw space. The claim is that learning can organize the latent space so that the restricted class becomes adequate there.

This choice shifts the burden from expressivity to representation learning and optimization. The argument does not require arbitrary program families to be representable by a weak latent class. It requires only that, for some task families, there exist latent organizations under which a weak class is sufficient. Training then becomes the process by which such an organization is found. If the latent program class is intentionally weak, then the latent geometry must become correspondingly informative.

This also clarifies the relation to ordinary reconstruction. If the learned encoder and decoder are frozen and reused as an ordinary encoder–decoder pair, they should still perform reconstruction well. The distinctive feature is the additional pressure to shape the learned representation so that a weak latent program class can solve the intended tasks.

3.6 Compositionality and Extrapolation

The compositional assumption introduced in Chapter 2 is especially consequential here. Without that assumption, the training tasks provide only sparse partial observations of distinct programs. The learner can generally do no more than fit or interpolate around the specific programs that appear in training. If a test task is generated by a genuinely unseen program, there is little reason to expect success, except insofar as the model has learned some incidental notion of proximity among programs.

Compositionality alters this conclusion. If the relevant programs are generated from shared primitive components, then distinct training programs need not be isolated entities. They may overlap structurally even when their full behaviors differ. This opens the possibility that training on one set of programs can support inference on another through components or substructures that were implicitly present across the training tasks.

This changes the relevant notion of generalization. Rather than covering neighborhoods around observed full programs, generalization can target unseen compositions of shared latent structure. If the representation preserves the relevant compositional relations, then sparse observations at test time may be sufficient to identify a plausible latent program even when the full program was never observed during training.

3.7 A Minimal Latent Program Class

Once the latent representation is treated as a search data structure, the next question concerns the choice of latent program class. The first choice explored in this thesis is intentionally minimal. A latent program acts as a simple transport on the latent state space. The use of transport serves as an informative baseline, not as a claim about the true algebra of programs. Failure under this baseline would indicate that the representation is not carrying enough of the relevant structure.

This choice is also motivated by evidence from other domains, where simple geometric operations in latent space have been shown to capture relational structure effectively. This suggests that compositional structure among programs may also be

reflected by simple latent motions. The architecture and objective are therefore chosen to pressure the latent space toward such an organization.

Simple transport should nevertheless be regarded as a first approximation. A single global move may be too crude for many tasks. This motivates an iterative mode, in which a program is approximated as a sequence of small latent steps. The aim is to retain a simple local move while allowing more complex global behavior to emerge through repeated application.

3.8 Cyclic Structure and Minimal Topological Capacity

The minimal transport model also raises a topological requirement. Many transformation families contain finite-order behavior, in which repeated application of a program may return to identity. A flat latent space equipped only with unbounded translation is mismatched to such structure. Repeated application drifts indefinitely, whereas the target behavior closes.

For this reason, the latent space should at least be able to express cycles if the task family requires them. This motivates giving the latent geometry topological capacity for closure, so that transformations are not forced into a globally flat translational model. The toroidal construction later explored in this thesis should be understood in this light. It provides a minimal way to avoid ruling out cyclic behavior in advance, without implying that every problem requires such topology.

3.9 Architectural Consequences

The preceding discussion motivates several architectural choices. If abstraction is to serve program induction, then the model needs a shared latent state space in which transformations act, a restricted latent program class that pressures the geometry to become useful, and an inference procedure matched to that geometry. The resulting system is neither a purely symbolic program synthesizer nor a conventional transductive predictor. It is a learned representational system whose success depends on whether training has compiled enough of the relevant program structure into the latent space.

This chapter motivates the next stage of the thesis. Once the problem is viewed in this way, the natural question concerns the structure a latent program space should have so that valid latent programs can be identified efficiently and applied reliably. The next chapter turns to that question.

Chapter 4
Structured Latent Program Spaces

The previous chapter argued that useful abstraction for this problem should be understood as the construction of a latent space in which program induction becomes selective, efficient, and reliable. The next question concerns the concrete structure such a latent space should have. This chapter develops a first explicit answer by deriving the proposed structure from a small number of representational requirements.

Figure 4.1 summarizes the proposed architecture. During learning, demonstrations induce latent program effects that are used to construct a latent program lattice. During inference, new demonstrations are encoded into the same latent geometry, projected toward valid latent program hypotheses, and applied to query inputs. The abstraction stage refines the latent program basis so that the same valid program set can be searched more effectively.

4.1 Validity in Latent Program Space

If the program space of interest is compositional and discrete, then not every point in a continuous latent space should be interpreted as a plausible program. A latent representation that treats every point as equally admissible implicitly assumes that arbitrary interpolations between program hypotheses are meaningful. This assumption is difficult to justify in the present setting. Programs are composed in discrete steps. Even when their effects on data appear smooth at the output level, the space of valid compositional hypotheses is not naturally identified with the whole ambient continuum.

This observation imposes the first structural constraint on the latent program space. The latent space itself may remain continuous, since it is learned by a neural encoder and decoder and must support gradient-based optimization. The subset of points corresponding to valid program hypotheses, however, should be distinguished from the

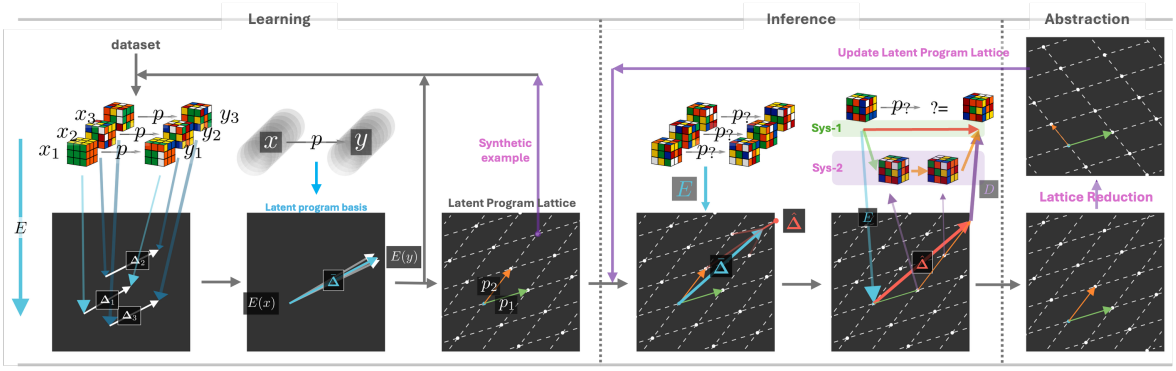

Figure 4.1: Overview of the architecture. Demonstration pairs are encoded into a shared latent state–program geometry, where program effects are represented as displacements. These effects define a latent program lattice used for inference by projection to valid program hypotheses. Basis refinement provides an abstraction mechanism for improving the search geometry without changing the underlying valid program set.

ambient space. Program induction is then formulated as the selection of a valid latent hypothesis that best explains the demonstrations, rather than as the decoding of an arbitrary latent vector.

The distinction between the ambient latent space and the subset of valid latent programs is essential. Without it, there is no principled reason to prefer one region of latent space over another beyond local reconstruction convenience. Once validity is made explicit, the latent space acquires an internal geometry that reflects both state similarity and the structure of admissible transformations.

4.2 A Shared Latent State–Program Geometry

The structured proposal studied in this thesis uses a shared latent space for states and program effects. Let

$$E : \mathcal{X} \to \mathbb{R}^n$$

be an encoder and

$$D : \mathbb{R}^n \to \mathcal{X}$$

be a decoder. For an input–output pair (x, y) generated by a program p, the latent effect of that program on that example is represented by the displacement

$$\Delta(x, y) = E(y) - E(x).$$

Given a task with demonstration pairs $\{(x_i, y_i)\}_{i=1}^{m}$, a natural first estimate of the task-level latent program effect is the aggregate displacement

$$\bar{\Delta} = \frac{1}{m} \sum_{i=1}^{m} \big(E(y_i) - E(x_i)\big).$$

Although this estimate is approximate, it already encodes an important commitment. A program is represented as an effect within the same latent geometry that organizes states, not as an arbitrary code detached from state representation.

This shared-space formulation has several consequences. First, the identity program corresponds to zero effect. Second, applying a latent program to a new input becomes a latent state update followed by decoding. Third, the structure of the state space and the structure of the program space are coupled. Any geometric inadequacy affects both the representation of states and the execution of programs. This coupling is central to the proposal, because it allows the encoder to learn coordinates in which the relevant transformations become simple.

4.3 Discrete Compositional Span

The next requirement is to make compositional validity explicit. Suppose that there is a set of latent primitive program effects

$$\Pi = \{p_1, \ldots, p_d\}, \qquad p_i \in \mathbb{R}^n.$$

If the program space of interest is generated by compositions of these primitive programs, then a first structured approximation represents valid composed latent programs

as integer combinations of these primitive effects. Writing

$$B = [p_1 \; p_2 \; \cdots \; p_d],$$

the corresponding set of valid latent programs is

$$\mathcal{L}(B) = \{Bz : z \in \mathbb{Z}^d\}.$$

This set is the latent program lattice. Its importance does not depend on the claim that all program spaces are literally lattices. It provides a simple explicit structure that captures three desiderata simultaneously. Validity is discrete, composed programs remain within the valid set, and the valid set admits geometric search procedures in the ambient latent space.

This formulation also clarifies the assumptions being made. The primitive effects $p_1, \ldots, p_d$ need not correspond to human-interpretable or directly observed atomic programs. They are generators of a latent compositional span. If the observed programs are themselves already composed, then the latent basis may initially encode non-atomic effects. The resulting structured space can still be useful, provided that it spans the valid latent programs relevant to the task family.

4.4 Program Induction as Geometric Search

Once valid latent programs are identified with the structured subset $\mathcal{L}(B)$, the search problem becomes concrete. The demonstrations induce an approximate latent effect $\bar{\Delta}$. Program induction then consists of selecting the valid latent program nearest to this estimate. The selected program is

$$p^\dagger = \arg\min_{v \in \mathcal{L}(B)} \|v - \bar{\Delta}\|.$$

This is the geometric form of program induction in the present framework. The search is restricted to valid discrete hypotheses embedded in the ambient latent space.

The ambient continuous geometry remains useful because it allows noisy or approximate task evidence to be compared against valid candidates. The discrete valid set remains essential because it imposes a strong structural bias on what counts as a plausible explanation.

This search problem is closely related to the Closest Vector Problem (CVP) in lattice theory. In the present context, the significance of this connection is conceptual rather than complexity-theoretic. It gives a precise meaning to the claim that program induction is search over a structured latent program space. The induced latent program is obtained by projection to the nearest valid compositional hypothesis, not by arbitrary decoding.

4.5 Fast and Deliberate Inference

The same structured latent program space supports two modes of inference. The first is direct one-step inference. Given the demonstrations, one infers $\bar{\Delta}$, projects it to the nearest valid latent program $p^\dagger$, and applies that effect to the latent encoding of the query input. The predicted output is

$$\hat{y} = D(E(x) + p^\dagger).$$

This mode is computationally economical. It treats program induction as recognition of the nearest valid composed effect.

The second mode is deliberate inference. The decomposition of $p^\dagger$ in the latent basis can guide a sequence of simpler latent updates. Each step can be followed by decoding and re-encoding, allowing the latent state to be corrected as the program is applied. In this mode, the latent program space serves both as a target for projection and as the substrate for a structured walk.

This distinction is important because the one-step projection is only a first-order approximation. If the demonstrations induce a noisy or imperfect aggregate effect, then direct application may accumulate error. The stepwise mode uses additional computation to improve fidelity while preserving the same underlying representation. It is a

second way of using the same structured latent program space, not a separate symbolic system.

4.6 Basis Refinement and Abstraction Discovery

The same framework also provides an interpretation of abstraction discovery. If valid latent programs are generated by a basis B, then another basis B' may generate exactly the same structured set $\mathcal{L}(B)$ while providing a more useful coordinate system for search. In geometric terms, one seeks generators that are shorter, more independent, or better conditioned. In representational terms, one seeks latent components that make the valid set easier to navigate.

Basis refinement is significant in this sense. Even if the initial latent generators correspond to relatively coarse or non-atomic program effects, a change of basis can reveal a more efficient latent organization. In the lattice setting, this suggests an interpretation of lattice reduction as a form of abstraction refinement. The process does not invent new valid programs; it re-expresses the same valid program set in a more useful coordinate system.

This interpretation should remain appropriately modest. A geometrically improved basis is not automatically a semantically meaningful basis. Nevertheless, basis refinement is important because it links abstraction discovery to the search geometry itself, rather than to a separate symbolic library-building procedure.

4.7 What the First-Order Structure Establishes

The structured latent space developed in this chapter establishes several properties. First, it makes validity explicit, since not every latent point is a candidate program. Second, it gives compositionality a concrete geometric form through the discrete latent span. Third, it turns program induction into a precise search problem over that structured set. Fourth, it allows the same representation to support both direct and iterative inference. Finally, it provides a representational interpretation of abstraction refinement.

For these reasons, the proposal should be understood as a first explicit structural answer to the problem raised in the previous chapter. It makes the central commitments of the thesis operational, while leaving several important difficulties unresolved.

4.8 Limits of the First-Order Structure

Making the latent program structure explicit also makes its limitations explicit. The first structured proposal reveals several mismatches between the algebra of programs and the geometry used to represent them.

The first mismatch concerns order sensitivity. The representation above identifies valid programs with integer combinations of basis vectors and is therefore fundamentally commutative. Many program spaces are not commutative. If composition order matters, then two different compositions may collapse to the same point under a purely additive representation. The second limitation is cyclicity. Some transformation families return to identity after repeated application, whereas a globally flat translational model does not close naturally. The third limitation is state dependence. A single global latent effect may be too coarse when a program's action depends strongly on local context or intermediate state.

These limitations do not invalidate the structure proposed here. They show why an explicit first-order formulation is useful. Once the commitments are visible, the next representational requirements can be stated clearly. The next chapter builds on this point by examining the limits of the first-order structure in greater detail.

Chapter 5
Limits of the First-Order Structure

The previous chapter proposed a first explicit structured latent program space. Its value lies in the representational commitments it makes visible as well as in the inference procedures it enables. Once validity, compositional span, and geometric search are formulated explicitly, the limitations of the first-order structure can be examined directly. This chapter analyzes those limitations and uses them to derive the next set of representational requirements.

5.1 What the First-Order Structure Preserves

Before examining its limitations, it is useful to state what the first-order structure achieves. It distinguishes valid latent programs from the ambient continuous latent space. It gives compositionality a concrete geometric realization through a discrete structured subset. It turns program induction into a geometric search problem over that subset. It also allows the same latent structure to support both direct one-step inference and iterative refinement. Finally, it provides an interpretation of abstraction discovery as refinement of the latent basis.

These are substantial representational gains. The first-order structure should therefore be understood as a useful structural approximation. The relevant question is what it preserves and what it necessarily collapses.

5.2 Abelianity

The first limitation of the first-order structure is that it is abelian. Valid latent programs were represented as integer combinations of latent generators,

$$p = Bz, \qquad z \in \mathbb{Z}^d,$$

and the underlying composition rule in this representation is vector addition. Vector addition is commutative, whereas program composition is generally not. If two primitive programs are denoted by π_i and π_j, then in many task families

$$\pi_j \circ \pi_i \neq \pi_i \circ \pi_j.$$

The issue is algebraically substantive. If order matters in the task family, then a representation that identifies programs through a commutative operation can preserve only an abelian projection of the underlying structure. It may still preserve coarse compositional validity, but it cannot represent ordered interaction as a native property of the space.

This observation reinterprets the first-order proposal. It is better understood as a useful first approximation that retains the part of composition expressible through additive structure, rather than as a faithful model of arbitrary program composition.

5.3 Order Sensitivity and Loss of Uniqueness

The abelianity problem directly affects inference. If two distinct ordered compositions collapse to the same latent point, then the induced latent program need not be unique even when the representation is otherwise sharp. In that case, ambiguity is introduced by the representation itself, rather than only by sparse demonstrations.

This matters because the objective of abstraction in this thesis is to sharpen latent program induction. A useful abstraction should make the posterior over latent programs narrow enough for sparse demonstrations to constrain the induced hypothesis strongly. If the representation identifies distinct compositions by construction, then this narrowing cannot be achieved at the level of latent program identity. Even when the observed demonstrations do not distinguish two compositions, the representation should still possess the capacity to do so when the task requires it.

The problem is therefore more than theoretical inaccuracy. Commutativity interferes directly with the selectivity requirement developed in earlier chapters.

5.4 Perturbative Encodings of Order

One possible response is to preserve the additive backbone and supplement it with small perturbative corrections intended to encode order. This approach can be useful in restricted settings, but it does not resolve the structural issue. The underlying geometry remains commutative, and order information is added as an auxiliary component around that backbone.

This approach becomes inadequate as composition length grows. The number of distinct ordered paths through a fixed set of primitive programs grows rapidly, while the available local capacity around each coarse additive composition remains limited. The resulting limitation is structural, since order is not represented as a native relation of the latent space.

For this reason, bounded corrective devices should be viewed as local approximations or bookkeeping mechanisms. They do not remove the need for a richer representational mechanism when ordered interaction is central to the task family.

5.5 Cyclicity and Flat Translation

The second major limitation concerns cyclicity. Some transformation families contain programs of finite order. For such a program p, repeated application eventually returns to identity,

$$p^k = \mathrm{id}$$

for some finite k. A standard example is rotation by a fixed angle in a finite discrete setting.

The first-order latent structure, however, models programs primarily as translational effects. Repeated addition of a nonzero vector in a globally flat space does not return to the origin; it produces indefinite drift. The mismatch is therefore structural rather than incidental. A translational geometry can represent one-step displacement, but by itself it does not provide a mechanism for finite-order closure.

This limitation matters because finite-order behavior occurs in many transformation

spaces generated by symmetries, periodic rules, and repeated operations on discrete objects. A latent program space intended to support such families should therefore possess the capacity to represent cyclic behavior when needed.

5.6 State Dependence

The first-order structure also assumes a relatively uniform notion of program effect. A latent program is represented as a single displacement or as an element of a structured discrete span that acts similarly across the latent state space. This is useful as a first approximation, but it can be too coarse when the effect of a program depends strongly on state, context, or intermediate configuration.

In such cases, the same symbolic program may act differently depending on where it is applied. The program itself has not changed, but the underlying objects and relations in the state have changed. A single global latent effect may then fail to capture the relevant behavior, even if the compositional span and search procedure are otherwise well-defined.

This limitation is especially important because the state geometry and the program geometry are coupled in the shared latent-space formulation. If the latent state space hides important context, then the latent program representation will inherit that inadequacy. Conversely, a richer state geometry may enable more faithful program effects.

5.7 Computational Conditioning and Search Hardness

A separate computational limitation also arises. Even if valid latent programs are represented by a discrete structured subset, search over that subset need not be easy. The geometry of the basis matters. Poorly conditioned generators can make nearest-valid-point search unstable or computationally expensive.

This is one reason why basis refinement has substantive computational value. A better basis can make the same valid set easier to search. However, this computational gain should not be confused with a solution to the structural mismatches described earlier. Basis conditioning can improve the tractability of search inside the first-order

structure, but it cannot by itself repair commutativity, lack of closure, or inadequate state dependence.

This distinction is important for the logic of the thesis. Some difficulties arise because the search problem is computationally hard. Others arise because the underlying representation preserves insufficient or inappropriate structure. The latter are more fundamental.

5.8 What a Richer Latent Program Space Must Preserve

Once the limitations of the first-order structure are stated explicitly, the requirements for a richer latent program space become clearer. Such a space should preserve discrete validity, because not every latent point ought to count as a plausible program. It should preserve compositional closure, because the program family is generated compositionally. It should also represent order sensitivity when the underlying task family is noncommutative. It should allow cyclic or finite-order behavior when repeated application returns to identity. Finally, it should remain searchable, since representational gains have limited value if inference becomes impractical.

These requirements should not be interpreted as a demand for perfect global faithfulness. The thesis does not require arbitrary program algebras to be embedded losslessly into latent geometry. The point is to identify which structures must be retained strongly enough for the desired level of reasoning and generalization to become plausible.

5.9 Toward a Richer Latent Program Space

The first-order structured latent program space developed in the previous chapter should therefore be interpreted as a lower-order approximation. It captures validity, discreteness, and searchability, while preserving only part of the algebraic structure of program space. The limitations identified in this chapter motivate extensions of the framework.

The next stage of the thesis is therefore a search for richer latent geometry that can

preserve more of what the first-order approximation leaves out. Once these limitations are made explicit, higher-order corrections and richer topological structure become the next representational targets.

Chapter 6
Toward Richer Latent Geometry

The previous chapter argued that the first-order structured latent program space should be retained as a useful approximation, while remaining distinct from a complete representation of compositional program structure. The present chapter addresses the next question of what it would mean to preserve more of that structure while retaining the central commitments of the thesis. The aim is to articulate the next representational layer suggested by the limitations of the first-order model.

6.1 Beyond the First-Order Approximation

The first-order proposal preserves a coarse compositional skeleton. Validity is discrete, latent search is well-defined, and program induction can be expressed geometrically. These are substantial gains. However, the previous chapter showed that several important aspects of program structure are compressed away. Ordered interaction is treated as if it were commutative, cyclicity is treated as if it were unbounded translation, and state dependence is treated as if it were an approximately uniform effect.

The task is therefore to determine how the first-order structure should be extended. In the language used throughout this thesis, the additive structured latent space should be interpreted as a lower-order approximation to a richer latent geometry. The next question is what additional structure must be introduced so that the representation preserves more of the algebra of programs while remaining searchable.

6.2 Higher-Order Correction

A central deficiency of the first-order structure is its inability to represent ordered interaction natively. One response is to retain the additive approximation as a base layer and introduce correction terms that account for what the additive layer misses.

In this view, the latent program lattice is the first term of an expansion, not the final representation of composition.

This idea should be understood structurally rather than formally. The lower-order representation captures what remains invariant under coarse additive approximation. The residual between true composition and additive composition is therefore structured discrepancy, not noise. The representation should include a mechanism for modeling that discrepancy, rather than treating it as an implementation artifact.

A useful mathematical analogy comes from noncommutative settings, where naive addition fails because composition contains interaction terms that are invisible at first order. A richer latent representation should be able to encode such interaction terms, whether through explicit higher-order coordinates, learned correction layers, or another construction that preserves ordered effects as genuine representational content.

6.3 Order as Structure, Not Decoration

The previous chapter argued that small perturbative corrections attached to an additive latent backbone are not sufficient in general. The underlying reason is that order is not a minor attribute of some program spaces; it is part of their algebraic identity. A representation that preserves only the commutative projection of composition may still be useful in restricted regimes, but it cannot be adequate when ordered interaction determines behavior.

This suggests a stronger requirement for the latent program space. Distinct ordered compositions should not depend on external bookkeeping for their recovery. The representation itself should possess enough capacity to distinguish them when the task family demands it. Order should therefore be part of the latent geometry, not a side channel attached after the fact.

The precise mathematical object satisfying this requirement remains open in the present thesis. The important point is the requirement itself. Once order is recognized as native structure, any adequate extension of the first-order model must preserve it directly rather than implicitly.

6.4 Topological Capacity for Cyclicity

The same logic applies to cyclicity. A globally flat translational latent model treats every program effect as if repeated application produces unbounded drift. Yet many transformation families contain periodic or finite-order behavior. The latent space should therefore admit transformations whose repeated application closes instead of diverging indefinitely.

This requirement points beyond globally flat geometry. One possibility is to provide the latent space with topological capacity for closure, so that repeated application of a latent program can trace an orbit and return to identity. The toroidal construction considered in this thesis is motivated by precisely this requirement. It should not be read as a universal claim about the correct latent topology for all tasks. It is instead a minimal way to ensure that cyclic program behavior is representable at all.

More generally, the issue is not toroidal geometry in itself. The broader point is that the ambient latent geometry must be rich enough to support the algebraic phenomena present in the task family. If periodicity is possible, the latent space should not rule it out by construction.

6.5 From Uniform Effects to Local Program Flows

The first-order model also treats latent programs as approximately uniform effects. Even when applied iteratively, the local move is still intended to be simple and largely state-independent. This is an informative starting point, but it becomes limiting when the same symbolic program has substantially different consequences depending on state.

One possible extension is to move from globally uniform transports toward local program flows. A latent program need not be represented as a single displacement that acts identically everywhere. It may instead be represented by a family of locally coherent transformations whose action depends on where in the latent state space the program is applied. This extension preserves the geometric viewpoint while relaxing the strongest uniformity assumption.

The iterative mode introduced earlier already points in this direction. Once pro-

grams are applied through repeated local steps instead of only through one-shot displacement, the latent dynamics begin to resemble the flow of a vector field rather than the application of a single constant vector. This does not solve the full state-dependence problem, but it shows that the original architecture already contains the first indication of a richer geometric interpretation.

6.6 What Must Remain Invariant

In extending the first-order structure, the features that made it valuable should be preserved. A richer latent program space should continue to distinguish valid and invalid hypotheses. It should continue to support search, rather than replacing induction with unconstrained decoding. It should remain tied to sparse evidence, so that the latent hypothesis remains sharply constrained by a small number of demonstrations. Finally, it should preserve the coupling between state geometry and program geometry, since that coupling is what makes representation learning useful for program induction.

These invariants provide criteria for evaluating candidate extensions. A representation may preserve more algebraic structure but become too unconstrained to search effectively. It may support richer dynamics but lose the distinction between valid and arbitrary latent points. It may become expressive enough to emulate many task families while ceasing to provide sharp inference from sparse demonstrations. Such extensions would not satisfy the objective of the thesis.

6.7 Open Questions

Several concrete questions remain open. First, what mathematical object should describe latent program structure once commutativity is relaxed? The first-order lattice is easy to define and easy to search, but it cannot represent order natively. A richer object may be group-like, Lie-like, manifold-valued, or hybrid; the thesis does not commit to a single final answer.

Second, how should cyclic and finite-order transformations be embedded? Toroidal geometry provides one possible answer, although it may not be sufficient for all task

families. More generally, the relationship between algebraic periodicity and latent topology remains to be clarified.

Third, how should abstraction discovery be understood in richer settings? Basis refinement is meaningful in the first-order structured space, but once higher-order interaction enters, the appropriate analogue of basis improvement is no longer obvious.

Finally, what level of structural faithfulness is necessary? The thesis has consistently avoided the stronger claim that program algebra must be represented globally and losslessly. The open question is therefore how much structure must be preserved for sparse program induction to remain selective and useful, not how to achieve perfect representation.

6.8 Closing Perspective

The argument of the thesis is now structurally complete. Chapter 3 motivated the need for abstraction as a search-enabling representation. Chapter 4 gave that representation a first explicit form by making validity and compositional span concrete. Chapter 5 used the explicit structure to reveal the next set of representational requirements. The present chapter has treated those requirements as guides toward richer latent geometry.

The final summary returns to the thesis as a whole and states its conceptual contribution. It presents a view of program induction in which abstraction is evaluated by whether it organizes latent space so that hidden transformations become sharply inferable from sparse observations.

References

1. H. N. Mhaskar, E. Tsoukanis, and A. D. Jagtap, "An approximation theory perspective on machine learning," 2026.

2. K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain generalization: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 4, pp. 4396–4415, 2022.

3. H. Lieberman, *Your wish is my command: Programming by example*. Morgan Kaufmann, 2001.

4. S. Gulwani, "Programming by examples: Applications, algorithms, and ambiguity resolution," in *International Joint Conference on Automated Reasoning*, pp. 9–14, Springer, 2016.

5. A. Menon, O. Tamuz, S. Gulwani, B. Lampson, and A. Kalai, "A machine learning framework for programming by example," in *International Conference on Machine Learning*, pp. 187–195, PMLR, 2013.

6. E. Kitzelmann, "Inductive programming: A survey of program synthesis techniques," in *International workshop on approaches and applications of inductive programming*, pp. 50–73, Springer, 2009.

7. Z. Manna and R. J. Waldinger, "Toward automatic program synthesis," *Communications of the ACM*, vol. 14, no. 3, pp. 151–165, 1971.

8. C. David and D. Kroening, "Program synthesis: challenges and opportunities," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 375, no. 2104, p. 20150403, 2017.

9. E. Parisotto, A.-r. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli, "Neuro-symbolic program synthesis," *arXiv preprint arXiv:1611.01855*, 2016.

10. J. Devlin, R. R. Bunel, R. Singh, M. Hausknecht, and P. Kohli, "Neural program meta-induction," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

11. M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. Eslami, and Y. W. Teh, "Neural processes," *arXiv preprint arXiv:1807.01622*, 2018.

12. M. Macfarlane and C. Bonnet, "Searching latent program spaces," *Advances in Neural Information Processing Systems*, vol. 38, pp. 103463–103520, 2026.

13. C. Allen and T. Hospedales, "Analogies explained: Towards understanding word embeddings," in *International Conference on Machine Learning*, pp. 223–231, PMLR, 2019.

14. D. Korchinski, D. Karkada, Y. Bahri, and M. Wyart, "On the emergence of linear analogies in word embeddings," *Advances in Neural Information Processing Systems*, vol. 38, pp. 62069–62097, 2026.

Acknowledgements

I would like to thank my advisor for their guidance and support throughout this work. I am also grateful to the collaborators in the lab for helpful discussions and feedback. Finally, I thank my family and friends for their patience and encouragement during the completion of this thesis.