# A Hypercube-Based Kernel for Generative Theomimesis

Domain-Agnostic Blueprint Mining on High-Dimensional State Spaces

Sebastian Klemm

GitHub: github.com/

Email:

2025

**Abstract**

This document specifies a domain-agnostic kernel for *generative theomimesis*:
a hypercube-based blueprint mining and transformation framework that operates on high-dimensional state spaces instead of symbolic rule sets.
The kernel is designed as a neutral, re-usable base technology which can be integrated into existing systems such as hypercube engines, HDAG (hypercube-directed acyclic graph) frameworks and deterministic simulation cores. Previous paradigm-specific layers (names, mythological references, domain-specific vocabulary) are stripped away and replaced by a universal information-geometric formulation.
At its core, the kernel consists of: (i) a family of domain adapters projecting arbitrary artefacts into a common high-dimensional state space, (ii) a hypercube and HDAG representation of candidate blueprints, (iii) a mining kernel for operator-space exploration, and (iv) an irreversible materialization pipeline from blueprint to artefact, backed by a ledger.
This specification is intended to serve as a reference document that can be included as a PDF in a public repository and used as a foundation for implementations in Rust, Python or hybrid stacks (e.g. Tauri-based hypercube labs).

# Contents

**1 Introduction** **3**

**2 Foundations** **4**
2.1 State Space . . . . . 4
2.2 Hypercube and HDAG Representation . . . . . 4
2.3 Blueprints and Artefacts . . . . . 4

**3 Domain Adapters** **5**
3.1 Adapter Definition . . . . . 5
3.2 Blueprint Embedding . . . . . 5

**4 Resonance Model** **6**
4.1 Resonance Function . . . . . 6
4.2 Theomimesis as Ideal Approximation . . . . . 6

**5 Core Operators** **7**
5.1 Extract Operator $E_x$ . . . . . 7
5.2 Compose Operator $C_o$ . . . . . 7
5.3 Materialize Operator $M$ . . . . . 7

**6 Mining Kernel** **8**
6.1 Kernel Definition . . . . . 8
6.2 Search and Filtering . . . . . 8

**7 System Architecture** **9**
7.1 Layered Architecture . . . . . 9
7.2 Integration into Existing Hypercube Systems . . . . . 9

**8 Outlook** **10**

# 1 Introduction

Most generative systems—from large language models to classical search and optimization pipelines—operate in symbolic spaces: strings, tokens, syntactic graphs, parameter vectors. At the same time, many deep structural patterns in science, design and cultural artefacts can be more naturally described as *geometric* or *topological* objects: state manifolds, hypercubes, flows on graphs, and resonance landscapes.

*Generative theomimesis*, in the sense used in this document, does not refer to any religious or metaphysical perspective. Instead, it denotes the attempt to emergently generate artefacts that approximate *idealised pattern families*: structures that are internally coherent, constrained, and stable under transformation, regardless of the concrete domain (code, protocols, blueprints, architectures, texts, etc.).

The goal of this document is to define a domain-agnostic kernel that:

- treats blueprints as points and trajectories in a high-dimensional state space,
- embeds them into a hypercube and HDAG structure,
- explores and mines operator spaces deterministically,
- and materializes selected blueprints into concrete artefacts with a persistent ledger.

All domain-specific names and references from previous work are replaced by neutral terms. The resulting kernel is meant to be integrated into hypercube frameworks such as QOPS-like engines, where hypercubes, cascades and topology modules already exist.

# 2 Foundations

## 2.1 State Space

We assume a high-dimensional state space $H^n$ with coordinates

$$\vec{v} = (v_1, v_2, \ldots, v_n) \in H^n,$$

where $n$ is flexible and can be adapted to the domain and implementation.

For many practical purposes, a five-dimensional core signature

$$\vec{v}_{\text{core}} = (\psi, \rho, \omega, \chi, \eta)$$

is used as a compact description of a blueprint's informational profile. Additional dimensions may hold topological features, spectral coefficients, or other invariants.

## 2.2 Hypercube and HDAG Representation

We model the computational geometry of the system using:

- a *hypercube* $Q$ as a discrete high-dimensional state lattice,
- an associated hypercube-directed acyclic graph (HDAG) $G = (V, E)$, where each node $v \in V$ represents a state or partial blueprint, and each edge $e \in E$ corresponds to a transformation or operator application.

A mapping

$$f : Q \to G$$

associates hypercube positions with HDAG nodes, preserving adjacency and relevant constraints. This allows both lattice-based and graph-based views of the generative process.

## 2.3 Blueprints and Artefacts

We distinguish between two central concepts:

- **Blueprint** $B$: a structured candidate object in abstract form. Examples include:
  - a protocol specification,
  - a software architecture,
  - a generative model configuration,
  - an operator network.
- **Artefact** $A$: a materialised instance of a blueprint in a concrete domain:
  - code repositories,
  - configuration files,
  - rendered documents,
  - deployed services.

The kernel focuses on transforming blueprints $B$ in the hypercube/HDAG space and selectively materialising them into artefacts $A$.

# 3 Domain Adapters

## 3.1 Adapter Definition

For each domain $d$ (e.g. code, text, diagrams, architectures) we define a *domain adapter*

$$D_d : \text{Object}_d \to \vec{v} \in H^n,$$

which maps domain-specific objects into the common state space.

In practice, this consists of:

- feature extraction (e.g. structural, syntactic, semantic),
- projection into the core signature $(\psi, \rho, \omega, \chi, \eta)$,
- optional augmentation with topological or spectral features,
- normalisation to maintain stability across domains.

## 3.2 Blueprint Embedding

Given a blueprint $B$ in domain $d$, the adapter produces:

$$\vec{v} = D_d(B) \in H^n.$$

This state can then be:

- inserted into the hypercube $Q$,
- mapped onto a node in the HDAG $G$,
- used as input to mining and transformation operators.

The adapter layer is the only place where domain-specific logic is required. All subsequent steps operate on $H^n$, $Q$ and $G$ only.

# 4 Resonance Model

## 4.1 Resonance Function

To capture how well a state aligns with an internal notion of coherence, consistency or "ideal form", we define a *resonance function*

$$R : H^n \to \mathbb{R}.$$

A simple core example is

$$R(\vec{v}) = \psi \cdot \rho \cdot \omega,$$

with a threshold $\tau > 0$ such that

$$R(\vec{v}) \geq \tau$$

is required for a state or blueprint to be accepted for further transformation.

The exact functional form of $R$ is domain-agnostic; it can incorporate:

- structural consistency scores,
- topological invariants,
- cost or complexity metrics,
- heuristic or learned quality indicators.

## 4.2 Theomimesis as Ideal Approximation

In this framework, theomimesis is interpreted as:

> Iterative approximation of blueprint states to an idealised family of patterns in $H^n$, as measured by the resonance function $R$ and additional constraints.

# 5 Core Operators

We define three principal operators, forming a generative pipeline:

(a) **Extract** $E_x$ – candidate generation and normalisation,

(b) **Compose** $C_o$ – filtering, combination and consolidation,

(c) **Materialize** $M$ – irreversible artefact creation.

## 5.1 Extract Operator $E_x$

The Extract operator takes input data (prompts, existing artefacts, partial blueprints) and produces a set of candidate blueprints:

$$E_x : \text{Input} \to \{B_1, \ldots, B_k\}.$$

Each $B_i$ is then embedded into the state space via the appropriate domain adapter:

$$\vec{v}_i = D_d(B_i).$$

## 5.2 Compose Operator $C_o$

The Compose operator operates on a set of candidate blueprints and their embedded states:

$$C_o : \{(B_i, \vec{v}_i)\}_{i=1}^{k} \to \{(B'_j, \vec{v}'_j)\}_{j=1}^{m}.$$

Typical actions include:

- enforcing constraints (hard or soft),
- merging compatible blueprints,
- pruning low-resonance states,
- projecting onto specific subspaces in $H^n$,
- updating the HDAG structure accordingly.

## 5.3 Materialize Operator $M$

The Materialize operator selects a single blueprint $B$ and produces a concrete artefact $A$:

$$M : B^{\to A.}$$

This step is *irreversible* at the level of the ledger: the transformation $B^{\to A}$ is recorded and becomes part of the system history.

# 6 Mining Kernel

## 6.1 Kernel Definition

We define a mining kernel

$$\mathcal{M} = (Q, S, F, R),$$

where:

- $Q$ is the hypercube state space,
- $S$ is a search procedure on $Q$ and the HDAG $G$,
- $F$ is a set of filters and constraints,
- $R$ is the resonance function described above.

The role of $\mathcal{M}$ is to explore high-dimensional operator and blueprint spaces and to output an ordered set of viable candidates.

## 6.2 Search and Filtering

The mining process can be sketched as:

1. initialise a frontier of states in $Q$ or nodes in $G$,
2. apply expansion steps according to the HDAG and operator rules,
3. evaluate each new state with $R$ and other metrics,
4. filter using $F$ (constraints, bounds, stability requirements),
5. update the frontier based on an exploration/exploitation policy (e.g. softmax over scores),
6. repeat until convergence or resource limits are reached.

The result is a ranked list of candidates:

$$\mathcal{M}_{\text{out}} = \{B_1, \ldots, B_k\},$$

each with associated state vectors $\vec{v}_i$ and scores $R(\vec{v}_i)$.

# 7 System Architecture

## 7.1 Layered Architecture

A practical implementation of the kernel can follow this layered structure:

(L1) **Domain Adapter Layer**
Maps domain objects into $H^n$ via $D_d$.

(L2) **Spectral / Signature Layer**
Computes core signatures and resonance values $R(\vec{v})$.

(L3) **Hypercube / HDAG Layer**
Maintains the hypercube $Q$ and graph $G$; updates nodes and edges during exploration and composition.

(L4) **Mining Layer**
Implements $\mathcal{M} = (Q, S, F, R)$; explores operator and blueprint spaces.

(L5) **Materialization Layer**
Applies $M$ to selected blueprints to create artefacts $A$; integrates with file systems, repositories or external services.

(L6) **User / Integration Layer**
Provides GUI, API endpoints, CI integration and export interfaces.

(L7) **Ledger / Governance Layer (optional)**
Records irreversible transformations, roles, approvals and economic parameters if needed.

## 7.2 Integration into Existing Hypercube Systems

In an existing hypercube engine, many components are already present:

- state representation and cascades (dimension reduction),
- topology dashboards and invariants,
- mining or slot-based discovery mechanisms,
- GUI frontends (e.g. Tauri, Svelte, web-based labs).

The generative theomimesis kernel can be integrated by:

- formalising domain adapters $D_d$,
- implementing a generic resonance module $R$,
- wiring existing mining code into $\mathcal{M}$,
- routing final selections through a materialization layer with a ledger.

# 8 Outlook

The kernel presented here is intentionally minimal and neutral. It abstracts away from any particular symbolic tradition and focuses exclusively on the information-geometric and algorithmic aspects of generative theomimesis.

Future work may include:

- adding explicit interfaces for machine learning components (e.g. learned resonance or constraint functions),
- extending the topology layer with additional invariants,
- building domain-specific adapter libraries (e.g. code, circuits, architectural plans),
- benchmarking different mining strategies and their convergence behaviour.