Start Here: The Short Version

This method turns each item into a very long code of simple values. Similar items get similar codes. The system compares a new code with stored class codes. The closest match gives the answer.

What it can do

- It learns a class from one example or a few examples. A teacher shows the system one good sample. The system saves that sample as the class code. Later samples match that code with no long training.
- It joins facts into one code and can pull the facts back out. One fact can name an item. One fact can name a place. The system joins both facts into one code for later use.
- It keeps working when part of the code has noise or damage. Noise changes a few parts of the code. The rest of the code still holds the pattern. The system can still find the closest match.

Where people can use it

- Small sensors can sort signals on the device. A sensor on a pump can watch sound and shaking. The sensor can label a normal pump and a weak pump on the spot.
- Teams can search for items with a similar pattern. A team stores one code for each item. A new item gets a new code. The system finds stored codes that sit close to the new code.
- Systems can give simple and robust labels in noisy places. Factories have dust, heat, and loud sound. Farms have wind, rain, and rough ground. Field gear takes drops and hits in daily use.

Problems it can help solve

- Deep learning often needs many examples and long training. A small team has only a few good examples. This method learns from the few examples on hand.
- Small chips have low power and little memory. A big model drains a small chip fast. This method uses simple values and simple math. A small chip can run the method all day.
- Noise and small faults can break brittle features in normal models. One bad reading can change a normal result. This method spreads each pattern across many parts. One bad part changes the full code only a little.

> The rest of this brief gives the history, the method, the 2026 gap, and a first project plan.

FOUNDATIONAL CONCEPTS: TECHNICAL BRIEF

Hyperdimensional Computing

Vector symbolic architectures store symbols in very long vectors. No product uses the method in 2026.

1,000 to 10,000
Components in one hypervector

1 pass
Training pass for prototypes

450x
VS-Graph training speed vs GNN

1988
Kanerva publishes the base work

1 Identification

Hyperdimensional computing uses very long vectors for symbols and structures.

Engineers also use the name vector symbolic architectures for the same family.

Pentti Kanerva published Sparse Distributed Memory in 1988.

Tony Plate published Holographic Reduced Representations in 1995.

Ross Gayler developed vector symbolic architectures in the 2000s.

Each hypervector holds 1,000 to 10,000 components.

Designers assign one random hypervector to each base symbol.

Random hypervectors show near-zero similarity to each other.

The property gives the system a large clean symbol space.

Purpose in one line

The method represents symbols, sets, and order in one common form.

The computer then does reasoning with vector arithmetic.

2 Purpose

The method solves a composition problem.

Normal vectors store one item well.

Normal vectors store nested structures poorly.

Hypervectors store a pair, a set, and a sequence in one vector.

The result vector keeps the same size in every case.

Programs read parts back from the result vector.

Programs also compare a new vector against stored vectors.

The method needs no gradient computation.

The method learns class examples in one training pass.

The method tolerates noise in many components.

Damage to a few components changes the result only slightly.

The property suits low-power hardware and faulty hardware.

> High dimension keeps random vectors apart. Random vectors almost never collide in this space.

3 Operation: The Three Operations

The bind operation combines two hypervectors into one hypervector.

The result represents an association between the two inputs.

The bundle operation adds hypervectors into one set vector.

A bundle also forms a class prototype from many examples.

The permute operation shifts the components of one hypervector.

The shift marks a position or a sequence order.

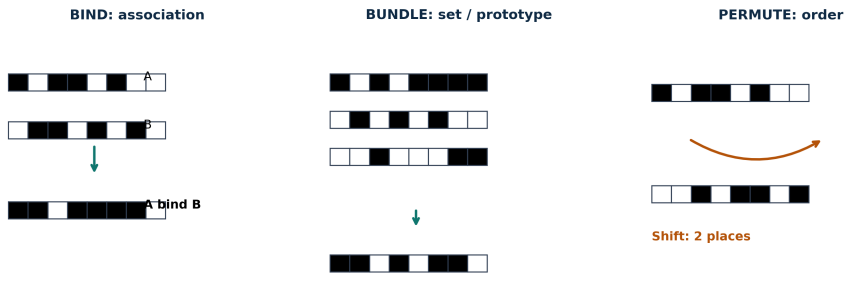

Figure 1: Bind, bundle, and permute. Each row shows one hypervector in short form.

3 Operation: Similarity and Classification

Programs measure similarity with Hamming distance for binary vectors.

Programs measure similarity with cosine similarity for real vectors.

A high score means two vectors share structure.

Classification procedure

1. The encoder converts the input signal into one hypervector.
2. The system bundles example hypervectors for one class.
3. The bundle forms one prototype for that class.
4. The system repeats the bundle step for every class.
5. The system compares the query hypervector against every prototype.
6. The system assigns the class with the highest similarity.

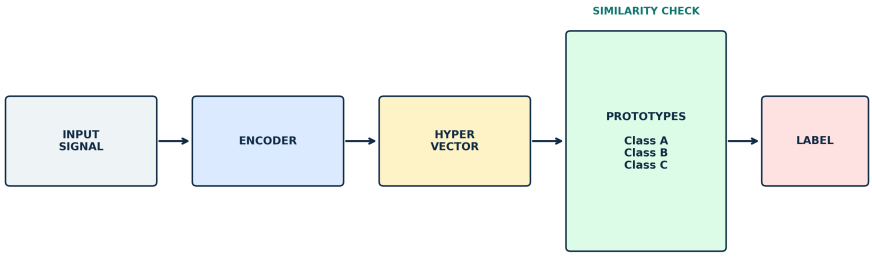

Figure 2: Classification flow. The similarity check selects the output label.

Analogy computation

An analogy computation uses bind and the inverse bind operation.

The computation uses four concept vectors.

The system extracts the relation between the first pair.

The system applies the relation to the third vector.

The result lands near the fourth vector.

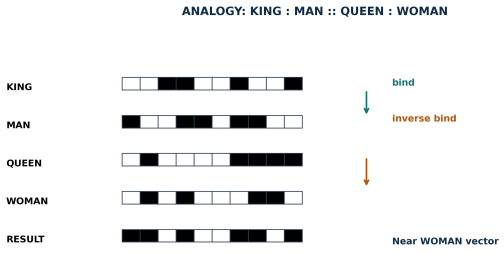

Figure 3: Analogy. Bind extracts a relation. Inverse bind applies the relation.

4 History: Why Use Stopped

Researchers built the theory between 1988 and the 2000s.

The theory worked on small symbolic tasks.

Deep networks then won the main accuracy contests.

Teams chose deep networks for images, speech, and text.

Learned embeddings took the vectors-for-meaning field.

Word vectors and neural embeddings gave teams a simpler story.

Developer tooling for hypervectors stayed absent.

No standard library reached wide use.

No training course built a user base.

The method stayed in papers and small laboratories.

5 The 2026 Gap

Research activity continues in 2025 and 2026.

VS-Graph classified graphs in hypervector space in December 2025.

VS-Graph trained up to 450 times faster than graph neural networks.

VS-Graph held accuracy at a dimension of 128.

The result proves speed on graph tasks only.

No developer platform productizes hyperdimensional computing for builders today.

No library ecosystem gives builders a complete stack today.

No hosted service sells hypervector classification today.

A builder still assembles every part by hand.

This gap creates an opening for a new project.

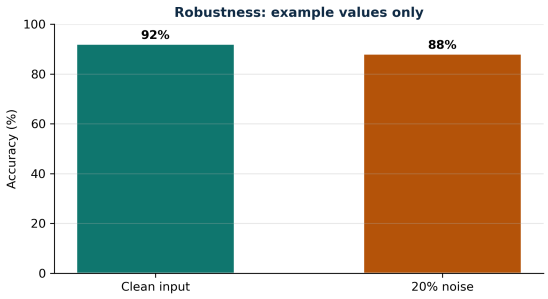

Figure 4: Robustness. Example values only. Noise damages 20 percent of components in the second bar.

Noise spreads across many components. The prototype match survives the noise.

6 Concrete Examples: The Audit-Trail Index

This section gives concrete uses for the method.

Example 1: an index over document parameters and audit trails

A document repository stores parameters and audit events for each document.

An encoder assigns one random hypervector to each parameter name and each value.

The encoder binds each name to its value.

The encoder bundles the bound pairs into one record vector per document.

Documents carry different parameters in the same fixed form.

A new parameter type needs only one new codebook entry.

The encoder binds each actor to each audit action.

The encoder rotates each pair by its step number.

The encoder bundles the rotated pairs into one history vector per document.

A query binds the record vector with a parameter name.

The cleanup step compares the result against the value codebook.

A reference test used 10,000-bit vectors and four parameter pairs.

The author query returned the correct value at similarity 0.685.

The status query returned the correct value at similarity 0.692.

An unrelated symbol scored 0.499 against the same record.

A three-step history query returned the correct actor at similarity 0.745.

A capacity test placed 30 pairs in one vector.

Field queries failed zero times in 12 trials.

The compass pattern

An investigator compares one suspect history vector against stored history vectors.

High similarity ranks candidate documents for review.

The investigator opens the exact audit entries in the repository table.

The hypervector index gives speed.

The repository table gives proof.

The index never replaces the audit table.

Bundle retrieval gives a probability, not a fact.

A bundle cannot list its own events back in order.

Long audit trails exceed the capacity of one vector.

The exact store remains the evidence in every design.

6 Concrete Examples: The Index at Work, and Five More Uses

Example 2: similar-case retrieval in health care

A clinic bundles symptom vectors and lab-value vectors into one case vector.

A new case queries the store for the nearest past cases.

The record system returns the full charts for those cases.

The vector gives the shortlist.

The chart gives the facts.

Example 3: fraud-pattern matching in banking

A bank encodes each transaction sequence with rotation by position.

Confirmed fraud cases form prototype vectors from a few examples.

Each new sequence scores against every prototype.

High scores route the account to human review.

The ledger remains the record of truth.

Example 4: support-ticket matching

A support desk encodes each ticket from word n-gram vectors.

Each solved ticket stores one vector with a link to its fix.

A new ticket retrieves the nearest solved tickets.

Different wording for the same fault still matches.

Agents start from a proven fix, not a blank page.

6 Concrete Examples: Field Devices and the Wearable

Example 5: failure signatures on field devices

A field device bundles vibration and temperature windows into a state vector.

Failure classes form prototypes from one or a few examples.

The device compares each new window on the chip itself.

A match raises a local alert with no network round trip.

Example 6: a memory index for a wearable assistant

A wearable device records sound, images, and sensor readings through the day.

An encoder turns each short segment into one hypervector.

The encoder binds each segment vector to time and place vectors.

The device stores the vectors beside the original recordings.

A wearer asks about a past moment in plain words.

The query vector retrieves the nearest segment vectors.

The system opens the original recordings at those points.

The index searches a long record at low power on the device.

The recordings remain the true memory.

The index only finds the moment faster.

> One rule governs every example in this section. Hypervectors act as an index and a compass. Exact systems keep the data and the proof.

7 Project Proposal: HDC Toolkit and Edge Classifier

Goal. The desired end state is one developer toolkit and one reference edge classifier in daily use. Builders encode signals into hypervectors and classify new signals against stored class prototypes. Training takes one pass over the examples.

Proof target. The project proves one claim on one sensor or signal benchmark. The classifier reaches usable accuracy with one to ten examples per class. Training runs much faster than a small neural network baseline, and accuracy falls slowly as noise grows.

Product at the end. The product is the HDC Toolkit, version 1, and one reference classifier build. The toolkit contains an encoder library, a prototype store, a similarity engine, and a small API. The reference build runs on low-power hardware and sorts one signal type on the device.

The project builds a small toolkit for builders.

The toolkit targets edge hardware first.

Language models form the second target.

Components

Component	Function
Encoder library	Record-based encoding and N-gram encoding
Prototype store	One bundled prototype per class
Similarity engine	Hamming distance and cosine similarity
Small API	Train, query, and explain calls

7 Project Proposal: Targets and Build Sequence

First target

The first target classifies sensor patterns and short signals.

The classifier learns each class from few examples.

The classifier runs on low-power hardware.

One training pass builds the prototypes.

Second target

The second target binds language-model outputs into compositional structures.

The structures form an audit trail.

An auditor reads the parts back from the structure.

Build sequence

1. The team defines the hypervector size and the value type.
2. The team builds the encoder library.
3. The team builds the prototype store and the similarity engine.
4. The team adds the small API and example programs.
5. The team tests accuracy, speed, and noise tolerance.
6. The team releases a sensor classifier as the first demo.

8 Risks and Limits

Deep networks beat hypervectors on large perception tasks.

The toolkit does not replace deep networks.

Encoder design controls accuracy strongly.

A poor encoder gives poor prototypes.

Teams test encoders on real signals early.

Large bundles lose detail after many additions.

Designers set bundle limits and test retrieval.

Buyers know neural products well.

Buyers know hypervectors poorly.

A working demo shows buyers the value directly.

Research results come from narrow tasks.

Graph speed does not prove sensor speed.

The team measures the target task directly.

> Recent research gives fresh proof for the science. No company owns a product layer for the method today. A narrow edge toolkit tests the gap at low cost.

Decision test

The project passes the test under three conditions.

The sensor demo matches a small neural baseline on accuracy.

The sensor demo trains faster than the baseline.

The sensor demo keeps accuracy under component noise.

Failure on two conditions ends the project.

Sources

Kanerva, P. Sparse Distributed Memory. 1988. Plate, T. Holographic Reduced Representations. 1995. Gayler, R. Vector symbolic architectures. 2000s. Survey report: Foundational Concepts series research, section 8, 2026-10-23. VS-Graph result, December 2025, as cited in the survey.

Figure 4 uses example values. The values illustrate robustness only. The values do not report a measured test.

Section 6 reports values from a 10,000-bit binary reference run by the series authors in 2026.