The ArrowSpace Algorithm: From Graph Wiring to τ-Mode Spectral Search

Lorenzo Moriondo
tuned.org.uk
ORCID: 0000-0002-8804-2963
tunedconsulting@gmail.com

July 29, 2026

Abstract

ArrowSpace is a graph-native indexing engine that replaces (or augments) purely metric vector search with a *spectral* view of a dataset. Rather than building a graph over items, ArrowSpace transposes the data matrix into feature space, builds a sparse k-nearest-neighbour Laplacian over features ("graph wiring"), and attaches a single scalar — the synthetic τ-mode index λ — to every item by evaluating a bounded Rayleigh-quotient functional on that Laplacian. The resulting index supports λ-aware nearest-neighbour search, clustering, classification, compression and mechanistic analysis, depending on which *Index Score Phase* is plugged into the pipeline. This report documents the full algorithmic pipeline: graph wiring, clustering/sampling/compression, FastPair-accelerated Laplacian construction, τ-mode synthesis, λ-aware search, and the test-driven experimental evidence supporting each stage. τ-mode synthetic score, in particular, is an effective encoding for position and connectivity of a node into a single scalar value .

Contents

1 Introduction

Classical vector search treats a dataset $X \in \mathbb{R}^{N \times F}$ as a point cloud in $\mathbb{R}^F$ and ranks candidates by cosine or Euclidean distance. ArrowSpace instead treats the *features* as the sampled objects: it transposes X into $X^\top \in \mathbb{R}^{F \times N}$, builds a sparse neighbourhood graph over the F feature vectors,

and constructs a graph Laplacian L_F that acts as a discrete Laplace–Beltrami operator on the feature manifold. Every item $x \in \mathbb{R}^F$ can then be scored by its Rayleigh quotient $x^\top L_F x / x^\top x$ against this learned manifold, producing a single bounded scalar $\lambda \in [0, 1]$ that measures how "smooth" the item is with respect to the feature-space wiring. This construction is called *Graph Wiring*, and it is the common substrate underneath every capability ArrowSpace exposes.

The key engineering idea explored in this report is that Graph Wiring is a *generic* pipeline with a pluggable final stage. Once the feature Laplacian is built, one can attach different scoring phases to obtain different tools from the same substrate:

- **Search** – `TauModeScore`: λ-aware semantic retrieval.
- **Analysis** – `FeatureSpectralScore`: mechanistic interpretability via the $F \times F$ feature graph.
- **Clustering** – `FiedlerVectorScore`: bipartite graph partitioning via the second eigenvector.
- **Classification** – `BoundaryEnergyScore`: active-learning flags via Dirichlet energy to class centroids.
- **Compression** – `EnergyDiffusionScore`: redundancy elimination via diffusion and optical binning.

We refer to this design as *Graph Wiring as a generic algorithm*: a build pipeline of the form

$$\mathsf{cluster} \;\to\; \mathsf{sample/compress} \;\to\; \mathsf{Laplacian} \;\to\; \mathsf{IndexScorePhase},$$

where only the last stage changes to switch capability. A key practical enabler of this pipeline's scalability, detailed in Section 4, is that the k-nearest-neighbour search underlying Laplacian construction is itself sub-quadratic, via the `FastPair`/`CosinePair` data structure. Section 11 then documents, stage by stage, which of these design claims are directly verified by the current test suite rather than asserted on theoretical grounds alone.

2 Mathematical Preliminaries

2.1 Graph Laplacians and the Rayleigh Quotient

Given a weighted adjacency matrix $W = (w_{ij})$ on n nodes, with degree matrix $D = \mathrm{diag}(D_{11}, \ldots, D_{nn})$, $D_{ii} = \sum_j w_{ij}$, the *unnormalised Laplacian* is

$$L = D - W,$$

and the *symmetric normalised Laplacian* is $L_{\mathrm{sym}} = D^{-1/2} L D^{-1/2}$. For $f \in \mathbb{R}^n \setminus \{0\}$, the *Rayleigh quotient* is

$$R(f) = \frac{f^\top L f}{f^\top f} = \frac{\frac{1}{2}\sum_{i,j} w_{ij}(f_i - f_j)^2}{\sum_i f_i^2}.$$

This is the discrete analogue of the continuum Dirichlet energy $E_M(f) = \frac{1}{2}\int_M |\nabla_M f|^2 \, dV_M$, and it converges to it under standard manifold-sampling assumptions (Belkin–Niyogi eigenmap convergence [12]). Minimising $R(f)$ is therefore a graph-native surrogate for minimising smoothness-violating structure on the underlying manifold.

2.2 Feature-First Wiring

ArrowSpace departs from item-graph constructions (used by HNSW-style indexes) by building the Laplacian over *features*, not items:

$$L_F = \mathrm{Laplacian}\big(\mathrm{Transpose}(C)\big),$$

where $C \in \mathbb{R}^{K \times F}$ is a matrix of K item *representatives* (raw items or, after compression, centroids). This design choice means that the Rayleigh energy of an item vector $x \in \mathbb{R}^F$,

$$E(x) = \frac{x^\top L_F x}{x^\top x},$$

measures roughness *relative to learned feature co-variation*, not relative to item adjacency. This is what allows a single $F \times F$ Laplacian to score arbitrarily many items in $O(\text{nnz}(L_F))$ each, independent of N.

2.3 Relation to Classical Kernel Methods (RKHS Theory)

A complementary body of theory addressing positive semi-definite operators on data is the classical kernel-methods literature, as taught in graduate courses such as the MVA (Mathématiques, Vision, Apprentissage) kernel methods curriculum [16]. That tradition builds a reproducing kernel Hilbert space (RKHS) from a positive-definite kernel $k(x, y)$, characterises valid kernels via Mercer, Bochner, and Herglotz representer theory, and reduces learning to convex optimisation over this implicit feature space, as in large-margin classifiers, kernels defined directly on graph-structured data [17], and kernel mean embeddings of probability distributions, with scalable variants addressing the $O(N^2)$ Gram-matrix bottleneck at large N [18].

ArrowSpace's Graph Wiring shares the same underlying mathematical substrate — positive semi-definite operators and the quadratic (Rayleigh) forms they induce — but inverts the direction of construction. Kernel methods start from a chosen similarity function $k(x, y)$ between *items* and derive an implicit feature map; ArrowSpace instead transposes the data matrix so that *features* become graph nodes, builds an explicit sparse k-NN Laplacian L_F over them (Section 5), and evaluates the Rayleigh quotient $x^\top L_F x / x^\top x$ of an item against that learned feature manifold, justified by convergence to the continuum Dirichlet energy and Laplace–Beltrami operator [12, 13] rather than by Mercer-style positive-definiteness arguments. Where a kernel machine fixes one similarity function per task (SVM margin, graph kernel, mean embedding), a single wired Laplacian in ArrowSpace supports a *pluggable* final scoring stage (Section 14), so search, analysis, clustering, classification, and compression are obtained by swapping the Index Score Phase rather than by re-deriving a new kernel.

Encoding the feature space Classical kernel methods build a Gram matrix $K = XX^\top \in \mathbb{R}^{N \times N}$ over the N *items* (rows of $X \in \mathbb{R}^{N \times F}$), since the kernel trick evaluates $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$ pairwise between data points [21]. By the singular value decomposition $X = U \Sigma V^\top$, this item-Gram matrix and its *feature*-Gram counterpart $X^\top X \in \mathbb{R}^{F \times F}$ share the same non-zero eigenvalues σ_i^2, related by $U = XV\Sigma^{-1}$ [10]: this is precisely the primal/dual duality exploited by kernel PCA, where the eigenproblem can be solved equivalently in the $N \times N$ dual (kernel) space or the $F \times F$ primal (covariance) space [21]. Graph Wiring's transpose $X \mapsto X^\top$ (Section 2.3) is the operational choice to work in exactly this *primal, feature-indexed* space rather than the dual, item-indexed one: instead of forming $K = XX^\top$ and reasoning about similarity between items, ArrowSpace forms a k-NN adjacency directly over the columns of $X^\top$ (features as nodes) and builds L_F from it. In the classical language, this is the same duality that lets kernel PCA be computed on $F \times F$ covariance-like matrices for $F << N$, except that Graph Wiring replaces the dense inner-product structure $X^\top X$ with a *sparse k-NN graph* over feature vectors, so that L_F is a graph-Laplacian regulariser of the feature covariance structure rather than the covariance matrix itself.

2.4 Graph Wiring as a Regularization-Operator Kernel

The Rayleigh-quotient construction underlying Graph Wiring (Section 2.3) admits a direct restatement in classical kernel-theoretic terms via the *kernels-from-regularization- operators* correspondence of Smola and Kondor [20]. Given the feature Laplacian L_F with eigendecomposition $L_F = \sum_i \lambda_i \phi_i \phi_i^\top$, any non-negative, non-decreasing spectral transform $r(\lambda)$ defines a positive semi-definite kernel

$$K \;=\; \sum_i r(\lambda_i)^{-1}\, \phi_i \phi_i^\top, \tag{1}$$

whose reproducing-kernel norm satisfies $\|f\|_K^{-2} = f^\top r(L_F)\, f$ [20]. Taking $r(\lambda) = \lambda$ recovers the (Moore–Penrose) pseudoinverse of the Laplacian, $K = L_F^+$, as the associated Mercer kernel, and the Rayleigh quotient $x^\top L_F x / x^\top x$ used throughout Sections 5–14 is exactly the regularization functional $f^\top r(L_F) f$ that this kernel penalises. In this sense, ArrowSpace's TauModeScore is not merely *analogous* to a kernel method but instantiates the $r(\lambda) = \lambda$ member of a well-defined family of graph regularization kernels [20].

A second, equally classical instantiation replaces the linear spectral transform with an exponential one. The *diffusion* (heat) kernel of Kondor and Lafferty,

$$K_t \;=\; e^{-tL_F}, \tag{2}$$

solves the heat equation on the wired feature graph and is guaranteed positive semi-definite because $-L_F$ is negative semi-definite by construction [22]. Equation (2) is the graph analogue of the Gaussian RBF kernel in Euclidean space, and it is the natural classical-kernel restatement of ArrowSpace's EnergyDiffusionScore compression phase (Section 3), which already diffuses the wired graph via `diffuse_and_split_subcentroids` before optical binning.

The remaining conceptual link is to *manifold regularization*, in which Belkin, Niyogi, and Sindhwani add the Laplacian quadratic form $f^\top L f$ as a smoothness penalty to a classical RKHS loss (kernel ridge regression or an SVM), and prove a representer theorem showing that the joint minimiser still lies in a finite-dimensional space spanned by kernel evaluations and Laplacian eigenfunctions [23]. This gives a precise sense in which the tau-mode synthetic index (Equation (3)) and a classical Mercer kernel can coexist as two terms of a single convex objective, connected by the same representer-theorem machinery that the MVA kernel-methods curriculum applies to Mercer kernels alone [16]. Table 1 summarises the correspondence.

ArrowSpace object	**Classical kernel-theoretic equivalent**
Feature Laplacian L_F	Spectral generator of a regularization-operator kernel family [20]
Rayleigh quotient $x^\top L_F x / x^\top x$	RKHS regularisation functional $f^\top r(L_F) f$ with $r(\lambda) = \lambda$ [20]
EnergyDiffusionScore (graph diffusion)	Kondor–Lafferty diffusion/heat kernel $K_t = e^{-tL_F}$ [22]
Tau-mode as a smoothness prior on items	Belkin–Niyogi–Sindhwani manifold-regularisation penalty in a joint RKHS+graph objective [23]

Table 1: Graph Wiring constructs restated as classical kernel-theoretic objects.

3 Stage 1: Clustering, Sampling, and Compression

The build pipeline begins by reducing the effective size of the dataset that the Laplacian is computed over, while preserving spectral fidelity.

3.1 Optimal-K Clustering

Given $X \in \mathbb{R}^{N \times F}$, ArrowSpace selects a number of clusters K heuristically (density- or elbow-based), and partitions rows into K groups, producing a centroid matrix $C \in \mathbb{R}^{K \times F}$ with $K \ll N$. Clustering can be made deterministic (fixed seed) or non-deterministic (random restarts), and supports inline density-adaptive sampling to bias centroid placement toward under-represented regions. As discussed in Section 11, this heuristic is deliberately conservative under high-noise conditions, preferring fewer, more stable clusters over exact recovery of a planted cluster count.

3.2 Johnson–Lindenstrauss Projection (optional)

When F is large, ArrowSpace can apply a random projection $R \in \mathbb{R}^{r \times F}$, $r = O(\log F/\varepsilon^2)$, before wiring:

$$\tilde{c} = Rc, \qquad c \in \mathbb{R}^F,\ \tilde{c} \in \mathbb{R}^r.$$

By the Johnson–Lindenstrauss lemma, pairwise distances are preserved up to a factor $(1 \pm \varepsilon)$ with high probability, so k-NN neighbourhoods computed in the reduced space remain faithful to the original feature manifold. Query vectors are projected on-the-fly at search time using the same matrix R, which is persisted as part of the index metadata. The target reduced dimension r is itself deterministic given (N, F, ε), even though the projection matrix's random entries are seeded independently per build.

3.3 Compression via Energy Diffusion (EnergyDiffusionScore)

For pure compression objectives, ArrowSpace diffuses the item graph and splits centroids into sub-centroids (`diffuse_and_split_subcentroids`), then applies *optical binning* (`optical_compress_centroids`) to merge sub-centroids whose energy bounds overlap. The compression objective evicts items that fall in densely populated, low-λ regions of the manifold (redundant, "typical" points) while retaining rare, high-energy sub-centroids that are structurally informative – the items most valuable for fine-tuning or curriculum construction.

Algorithm 1 Stage 1 – Cluster, Project, Compress

Require: Data matrix $X \in \mathbb{R}^{N \times F}$, target cluster budget $K_{\max}$

$K \leftarrow \textsc{SelectOptimalK}(X, K_{\max})$

$C \leftarrow \textsc{Cluster}(X, K)$ ▷ $C \in \mathbb{R}^{K \times F}$

if `use_dims_reduction` **then**

  $R \leftarrow \textsc{RandomProjection}(F, \varepsilon)$

  $C \leftarrow CR^{\top}$ ▷ project rows to reduced dim r

end if

if `compression mode` **then**

  $S \leftarrow \textsc{DiffuseAndSplitSubcentroids}(C)$

  $S \leftarrow \textsc{OpticalCompressCentroids}(S)$

  return S ▷ sub-centroid matrix replaces C

end if

return C, R

4 Fast Nearest-Neighbour Pairing with FastPair

Graph Wiring requires, for every one of the n feature (or item) vectors, retrieval of its k nearest neighbours under cosine distance. A naive approach compares every pair of vectors, costing $O(n^2 d)$ for n vectors of dimension d – prohibitive once n reaches the tens of thousands. ArrowSpace instead builds the k-NN graph using a **FastPair**-derived structure, implemented as `CosinePair` in the `smartcore` crate, which organises the vectors into a similarity-aware search tree at construction time and answers repeated k-NN queries against that tree rather than against the raw point set.

4.1 Why FastPair

FastPair [19] is a classical algorithm for the *closest-pair* and *repeated nearest-neighbour* problem: it maintains, for each point, a candidate nearest neighbour, and updates only the candidates invalidated by a change, rather than rescanning all pairs. ArrowSpace repurposes the same principle for a *static* batch setting: instead of incremental updates, it builds the neighbour-candidate structure once over the full vector set and then issues n independent top-k queries against it, which is precisely the workload required at Laplacian-construction time. The structure amortises the pairwise-comparison cost across queries by pruning comparisons that cannot possibly improve on the current k-best candidates, using triangle-inequality-style bounds on cosine distance.

4.2 Construction and Query Complexity

Let n be the number of vectors, d the (possibly JL-projected) feature dimension, and k the number of neighbours requested per vector. Building the `CosinePair` structure costs

$$O(d\, n \log n),$$

and each of the n top-k queries costs

$$O(k\, d\, \log n)\,,$$

since each query performs an expected $O(\log n)$-depth tree traversal with an $O(d)$ distance evaluation per candidate node, and every distinct distance computation contributes at most $O(k)$ times to the returned candidate set. The total cost of the k-NN phase across all n vectors is therefore

$$O(d\, n \log n) \;+\; O(n\, k\, d \log n) \;=\; O(n\, k\, d \log n),$$

compared with $O(n^2 d)$ for the brute-force pairwise scan. For typical production parameters ($n =$ 10,000 items, $k = 10$ neighbours, $d = 384$ features), this is a reduction from roughly 3.84×10^{10} raw operations to roughly 5.1×10^7, a speedup on the order of 750×.

Phase	**Brute force**	**FastPair (CosinePair)**
Build structure	—	$O(d\, n \log n)$
All-pairs k-NN	$O(n^2 d)$	$O(n\, k\, d \log n)$
Per-query cost	$O(nd)$	$O(k\, d \log n)$

Table 2: Complexity of k-NN retrieval for Laplacian construction, brute force vs. FastPair-derived `CosinePair`.

4.3 Integration into Adjacency Construction

The adjacency step (Section 5) first builds the `CosinePair` structure over the n representative vectors, then issues one `topk` query per vector to obtain its candidate neighbour list together with cosine distances. Node degrees from this initial pass are used to decide, per dataset, whether *inline sparsification* should trigger (Section 5.2): if the average node degree from the FastPair query exceeds a fixed threshold, ArrowSpace scores each retained edge by weight scaled by the geometric mean of endpoint degrees, and truncates each node's edge list to the top half by that score, before kernel weights and symmetrisation are applied.

Algorithm 2 FastPair-Based Adjacency Construction

Require: Vectors $\{v_i\}_{i=1}^n \subset \mathbb{R}^d$, neighbour budget k, radius ε

$\mathcal{T} \leftarrow$ CosinePair.Build($\{v_i\}$) ▷ $O(d\, n \log n)$
parfor $i = 1$ **to** n **do**
  $\mathcal{N}_i \leftarrow \mathcal{T}$.QueryTopK($v_i, k+1$) ▷ $O(k\, d \log n)$, self-excluded
  $\deg_i \leftarrow |\{j \in \mathcal{N}_i : \mathrm{dist}(v_i, v_j) \le \varepsilon\}|$
end parfor
$\bar{d} \leftarrow \frac{1}{n} \sum_i \deg_i$
sparsify $\leftarrow (\bar{d} > 10)$
parfor $i = 1$ **to** n **do**
  $\mathrm{rows}_i \leftarrow \{(j, w_{ij}) : j \in \mathcal{N}_i,\ \mathrm{dist}(v_i, v_j) \le \varepsilon\}$ ▷ kernel weight, Sec. 5
  if sparsify **then**
    score each (j, w_{ij}) by $w_{ij} \sqrt{\deg_i \cdot \deg_j}$; keep top half by score
  end if
end parfor
return $\{\mathrm{rows}_i\}$ ▷ fed into symmetrisation, Sec. 5

Remark 1. *FastPair's role in ArrowSpace is purely at* index-build *time: once the k-NN adjacency has been extracted and converted to a sparse Laplacian, the `CosinePair` tree is discarded. This mirrors the "compute once, store scalar" philosophy of Stage 3 (Section 14): the expensive geometric structure is transient, and only the resulting sparse matrix and lambda scores are retained for query time.*

5 Stage 2: Graph Wiring / Laplacian Construction

5.1 The λ-graph

Given the (possibly projected, possibly compressed) representative matrix $C \in \mathbb{R}^{K \times r}$, ArrowSpace builds a k-NN or ε-graph over items (rows of C) with a set of parameters $(\varepsilon, k, \text{topk}, p, \sigma, \text{normalise})$, using the FastPair-derived `CosinePair` retrieval of Section 4 in place of brute-force pairwise distance computation:

- ε: neighbourhood radius (for ε-graphs);
- k: number of nearest neighbours retained per node;
- topk: number of edges retained after pruning (sparsification);
- p: exponent for the distance kernel $w_{ij} \propto d_{ij}^{-p}$;
- σ: bandwidth for the Gaussian heat kernel $w_{ij} = e^{-d_{ij}^2/\sigma^2}$;
- normalise: whether to use L or L_{sym}.

The resulting adjacency W is symmetrised, weighted, and converted to a sparse CSR Laplacian L (the `GraphLaplacian` type), which is transposed internally so that item indices correspond to Laplacian rows. As documented in Section 11, the symmetry of this matrix, the preservation of its construction parameters, and the non-negativity of its diagonal are directly unit-tested rather than assumed.

Algorithm 3 Stage 2 – Laplacian Construction (`eigenmaps`)

Require: Centroids $C \in \mathbb{R}^{K \times r}$, params $(\varepsilon, k, \text{topk}, p, \sigma, \text{norm})$

$G \leftarrow \textsc{FastPairAdjacency}(C, k, \varepsilon)$ ▷ Alg. 2
for each edge $(i, j) \in G$ **do**
  $d_{ij} \leftarrow \textsc{Dist}(c_i, c_j)$
  $w_{ij} \leftarrow \textsc{Kernel}(d_{ij}, p, \sigma)$
end for
$D \leftarrow \text{diag}\big(\sum_j w_{ij}\big)$
$L \leftarrow D - W$
if norm **then** $L \leftarrow D^{-1/2} L D^{-1/2}$
end if
$L \leftarrow \textsc{VerifySparsity}(L)$ ▷ sparsity-check guard
return L ▷ `GraphLaplacian`, nnodes $= N$

5.2 Inline Sparsification

When the average node degree returned by FastPair queries is high, edges are pruned before kernel weighting is finalised, using the degree-scaled score described in Section 4. This keeps the resulting adjacency (and hence L) sparse without a separate global sparsification pass, and its cost is absorbed into the FastPair query loop rather than requiring an additional $O(n^2)$ scan.

5.3 Optional Stage 2b: Feature-Feature Spectral Laplacian

When `prebuilt_spectral` is enabled, ArrowSpace performs a *Laplacian-of-Laplacian* step: the item Laplacian is transposed and a second Laplacian is computed over *features* (columns), producing an $F \times F$ matrix stored in `signals`. Because this matrix encodes feature-to-feature correlation modulated by the item graph rather than a proper metric adjacency, it need not be positive semi-definite – negative Rayleigh quotients are mathematically valid here and are interpreted as sign-

indefinite feature couplings. This feature graph is the substrate for the `FeatureSpectralScore` analysis capability (mechanistic interpretability, spectral gap / motif extraction), and its shape ($F \times F$ when enabled, empty otherwise) is directly checked by the test suite.

6 Stage 3: The Pluggable Index Score Phase

Once L (and optionally the feature graph L_F) is available, ArrowSpace computes exactly one scalar per item and discards the graph if it is not needed further. This is the "compute once, store scalar" design: spectral information is preserved in λ_i without retaining L in memory for later queries (search re-derives what it needs from the persisted matrices).

Table 3 summarises the five interchangeable `IndexScorePhase` implementations and the capability each one confers on the resulting `ArrowSpace`.

Score	Capability	Mechanism	Output
`TauMode`	Search	Rayleigh energy + Dirichlet dispersion, blended via τ	bounded $\lambda \in [0,1]$ per item
`FeatureSpectral`	Analysis	$F \times F$ Laplacian-of-Laplacian, spectral gaps	feature-circuit communities
`FiedlerVector`	Clustering	2nd-smallest eigenvector of normalised L	bipartite item score
`BoundaryEnergy`	Classification	Dirichlet energy of item to class centroids	decision-boundary flag
`EnergyDiffusion`	Compression	diffusion + optical binning	eviction / retention scalar

Table 3: Pluggable Index Score Phases and the capabilities they unlock.

6.1 A Proposed Sixth Score: `LearnableNoveltyScore` (Epiplexity)

Alongside the five implemented `IndexScorePhase` variants of Table 3, a sixth score is currently under design discussion: `LearnableNoveltyScore`, informally referred to as *epiplexity* [8, 9]. Unlike `TauModeScore`, which asks "how spectrally rough is this vector relative to the wired feature manifold?", epiplexity asks a different question: "how much stable, compressible new structure does this item, batch, or trajectory add for a bounded observer?" This subsection describes the proposal as it currently stands; none of the properties below carry the empirical backing established for `TauModeScore` in Section 11, and the design remains open at the time of writing [8].

6.1.1 Motivation and Distinction from Rayleigh Energy

The core risk the proposal identifies is that Rayleigh/Dirichlet energy, the basis of τ-mode (Eq. 3), cannot by itself distinguish a rare but structurally coherent item from pure noise: both produce high graph energy [8]. Epiplexity instead defines novelty as the *description length of the model a bounded learner can extract* from an item, explicitly excluding both trivial regularity (already captured by low λ) and irreducible noise (which a frozen reservoir cannot compress) [9]. The practical estimator proposed is a frozen random reservoir combined with a closed-form ridge readout, scored via the log-determinant of the readout's singular values:

$$S_\phi(Y \mid X) = \frac{1}{2} \log_2 \det\big(I + \eta\, W_\lambda W_\lambda^\top\big),$$

where W_λ is the ridge-optimal linear map from frozen reservoir features of X to targets Y, and η is a fixed spectral-resolution parameter [9].

6.1.2 Graph-Conditioned Targets

The proposal's central architectural idea is to make the wired feature Laplacian L_F part of the observer, rather than scoring epiplexity on raw embeddings. For an item x_i, graph-native targets are derived from the existing Stage 2 output:

$$Y_i = \big[\, x_i L_F,\ x_i L_F^2,\ \text{Rayleigh}(x_i),\ \text{diffusion}_t(x_i) \,\big],$$

so that the frozen reservoir is asked whether a bounded readout can reliably recover an item's *graph response* from its embedding, local neighbourhood, or prior state [8]. This reuses L_F exactly as constructed for `TauModeScore` and `FeatureSpectralScore`, requiring no change to Stages 1–2 of the pipeline.

6.1.3 Global and Marginal Products

The proposal specifies two outputs rather than one. The *global* score, S_ϕ, answers "how much learnable structure does this wired dataset contain?" The *marginal* score, intended for the per-item slot that τ-mode currently fills,

$$\Delta S_i = S_\phi(\mathcal{D}_{1:i}) - S_\phi(\mathcal{D}_{1:i-1}),$$

or a rank-one-update approximation of it via ridge leverage, measures whether a specific item changes the learnable model rather than merely being distant or high-energy [8]. This marginal formulation is what would make the score usable inside a streaming `ArrowSpaceBuilder` pipeline, analogous to how τ-mode is computed per item in Algorithm 5.

6.1.4 Proposed Trait Shape

The proposal fits the existing `IndexScorePhase` abstraction with an explicit fit/score split, separating a per-dataset reservoir fit from per-item and global scoring:

Algorithm 4 Proposed Interface – `LearnableNoveltyScore` (design sketch, not yet implemented)

Require: Data X, feature Laplacian L_F, reservoir seed, ridge parameter λ_r, spectral resolution η

targets $\leftarrow$ GraphConditionedTargets(X, L_F) ▷ Y_i per item
$\mathcal{R} \leftarrow$ FrozenReservoir(seed) ▷ fixed random feature map
$W_\lambda \leftarrow$ RidgeReadout$(\mathcal{R}(X), \text{targets}, \lambda_r)$
$S_{\text{global}} \leftarrow \frac{1}{2}\log_2\det(I + \eta\, W_\lambda W_\lambda^\top)$
parfor $i = 1$ **to** N **do**
    $\Delta S_i \leftarrow$ MarginalContribution$(i, W_\lambda, \text{targets})$ ▷ leverage-based rank-one update
end parfor
return $S_{\text{global}}, \{\Delta S_i\}$

6.1.5 Intended Applications and Open Safeguards

The proposal outlines four candidate uses that would reuse the same wired graph without rebuilding it: a curation policy that retains high-λ, high-ΔS items while evicting low-λ, low-ΔS redundancy; a second-stage diversity term added to the Stage 4 ranking function of Section 7

(rank$(q, d) = \alpha \cos(q, d) + \beta\, \tau$-compatibility $+ \gamma\, \Delta S$); a two-axis active-learning queue that pairs `BoundaryEnergyScore` with ΔS to separate genuinely novel decision-boundary cases from noise; and an additional coordinate in ArrowSpace's dataset-fingerprint mechanism, alongside the truncated Laplacian spectrum and τ-distribution already used for drift monitoring [8].

The proposal is explicit that the score is *observer-relative*: its value depends on reservoir width, random-feature seed, ridge penalty, and target construction, and must not be presented as an observer-free complexity measure [9, 8]. Accordingly, the issue specifies several non-negotiable safeguards before any production use: a frozen (never per-batch-retrained) reservoir, out-of-sample evaluation to prevent memorisation of self-generated graph targets, multi-seed confidence intervals over reservoir and graph seeds, and explicit magnitude controls against norm, entropy, and Rayleigh energy so that the score cannot be trivially reproduced by scale alone [8]. The proposal also recommends the lowest-risk first experiment be offline graph-conditioned curation on a labelled corpus, comparing random, density-only, energy-only, and energy-plus-epiplexity retention policies at equal storage budget, rather than attempting end-to-end differentiable representation training against the score immediately [8].

Remark 2. *As of this writing,* ***`LearnableNoveltyScore`*** *has no implementation, no unit tests, and no entry in Table 3; it is included here only as a worked example of how the pluggable* ***`IndexScorePhase`*** *abstraction of Section 14 is intended to accommodate scoring objectives beyond spectral roughness, without requiring changes to Stages 1–2 of the pipeline [8].*

7 The τ-Mode Synthetic Index

The remainder of this paper focuses on `TauModeScore` (search), which is the most extensively validated phase, with cross-phase agreement checked separately via motif discovery in Section 11. Leaving the others for dedicated literature.

7.1 Rayleigh Energy and Dispersion

Given an item vector $x \in \mathbb{R}^F$ (projected into the wiring's coordinate system if JL projection was used) and the feature Laplacian L_F (CSR sparse matrix), TauMode computes two quantities.

Raw Rayleigh energy

$$E_{\text{raw}}(x) = \frac{x^\top L_F x}{x^\top x} = \frac{\sum_{i,j} l_{ij}\, x_i x_j}{\sum_i x_i^2}.$$

Edge dispersion $G(x) \in [0, 1]$, a Gini-like statistic over the edge-wise energy distribution:

$$w_{ij} = \max(-l_{ij}, 0), \qquad d_{ij} = (x_i - x_j)^2, \qquad \text{TotalEdgeEnergy}(x) = \sum_{i \neq j} w_{ij}\, d_{ij},$$

$$G(x) = \text{clamp}\left(\sum_{i \neq j} \left(\frac{w_{ij}\, d_{ij}}{\text{TotalEdgeEnergy}(x)} \right)^2,\ 0,\ 1 \right).$$

$G(x)$ measures how concentrated the Dirichlet energy is on a few edges (high G: energy dominated by a small number of feature pairs, i.e. a "spiky" roughness) versus spread uniformly (low G).

7.2 Bounding and Synthesis

The raw energy is unbounded, so it is squashed into $[0, 1)$ using a *tau-parametrised* bounded transform:

$$E_{\text{bounded}}(x) = \frac{E_{\text{raw}}(x)}{E_{\text{raw}}(x) + \tau}.$$

The final **synthetic index** blends bounded energy and dispersion:

$$\lambda(x) \;=\; \tau \cdot E_{\text{bounded}}(x) \;+\; (1 - \tau) \cdot G_{\text{clamped}}(x) \;\in\; [0, 1]. \tag{3}$$

This is the ArrowSpace τ-mode formula: λ is a single scalar per item that jointly encodes Rayleigh-quotient smoothness and edge-energy concentration on the learned feature manifold.

Remark 3. *$\lambda(x)$ is provably scale-invariant: since both E_{raw} and G are ratios of homogeneous-degree-2 quantities in x, scaling $x \mapsto cx$ leaves λ unchanged for any $c \neq 0$. This property is verified numerically in Section 11 to 10^{-10} tolerance.*

7.3 Selecting τ: the TauMode Policy

The blending weight τ is not a single global constant but is selected *per item* from a distribution of raw energies according to a policy enum, `TauMode`:

- `Fixed(`t`)`: constant $\tau = t$ (clamped to a numerical floor $\tau_{\text{floor}} = 10^{-10}$ if non-finite or negative).
- `Mean`: $\tau = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} e$ over finite energies $\mathcal{E}$.
- `Median`: $\tau =$ median of the finite energy distribution.
- `Percentile(`p`)`: $\tau =$ the p-th percentile ($p \in [0, 1]$) of the sorted finite energies.

`Median` is the default production policy, chosen empirically for stability; other policies allow deployments to tune the search bias toward smoothness (`Fixed` with large τ) or dispersion-sensitivity (`Fixed` with small τ). Each policy, including its NaN/Inf and empty-input fallback behaviour, is unit-tested individually (Section 11).

Algorithm 5 Stage 3 – Parallel TauMode Lambda Computation

Require: Item matrix (or stream) of rows $\{x_i\}_{i=1}^{N}$, graph L_F (feature Laplacian, or precomputed `signals`), policy mode

graph $\leftarrow$ `signals` if available, else L_F
parfor $i = 1$ **to** N **do** ▷ Rayon parallel iterator
  $x \leftarrow \textsc{Project}(x_i)$ ▷ apply JL projection if configured
  if $x = 0$ **then** $\lambda_i \leftarrow 0$; **continue**
  end if
  $\tau \leftarrow \textsc{SelectTau}(x, \text{mode})$
  $E_{\text{raw}} \leftarrow \dfrac{x^\top \text{graph}\, x}{x^\top x}$ ▷ sparse CSR row iteration
  $G \leftarrow \textsc{ComputeDispersion}(x, \text{graph})$
  $\lambda_i \leftarrow \tau \cdot \dfrac{E_{\text{raw}}}{E_{\text{raw}} + \tau} + (1 - \tau) \cdot \text{clamp}(G, 0, 1)$
end parfor
$\textsc{UpdateArrowSpace}(\lambda_1, \ldots, \lambda_N)$
return $\{\lambda_i\}$

The implementation adaptively switches between sequential and parallel per-item kernels based on graph size (sequential below 1000 nodes or 10,000 non-zeros, parallel chunked above), giving near-linear scaling with core count and $O(\text{nnz})$ total cost, independent of a brute-force $O(N^2)$ pairwise comparison.

7.4 Determinism and Bounds

The construction guarantees:

- **Determinism**: repeated evaluation of $\lambda(x)$ for a fixed x, graph, and τ-policy is bit-stable (verified to 10^{-12} tolerance in regression tests), including under JL projection with a fixed random seed.
- **Boundedness**: $\lambda(x) \in [0, 1]$ for all finite, non-degenerate x.
- **Scale invariance**: $\lambda(cx) = \lambda(x)$ for $c \neq 0$.
- **Consistency**: the λ recomputed for a query at search time matches the indexed λ for the same vector to numerical precision, which is what allows lambda-based candidate pruning to be trusted at query time.

Each of these four properties is exercised by a dedicated test, described in Section 11.

8 Sub-Centroid Assignment (Energy / Compression Mode)

When ArrowSpace is built in *energy* mode (compression-oriented), items are not directly assigned a Laplacian row; instead, τ-mode is computed once on a smaller set of sub-centroids, and each item is mapped to its nearest sub-centroid in λ-*space*:

Algorithm 6 Item-to-Subcentroid Assignment via Lambda-Distance

Require: Item x, sub-centroid lambdas $\{\lambda^{(c)}\}_{c=1}^{M}$, tie tolerance $\epsilon = 10^{-11}$

$\lambda_x \leftarrow \textsc{PrepareQueryItem}(x)$ ▷ Eq. 3

$c^\star \leftarrow \arg\min_c |\lambda_x - \lambda^{(c)}|$

$\mathcal{C} \leftarrow \{c : \big||\lambda_x - \lambda^{(c)}| - |\lambda_x - \lambda^{(c^\star)}|\big| \leq \epsilon\}$ ▷ candidates within tie band

if $|\mathcal{C}| > 1$ **then**

  $c^\star \leftarrow \arg\max_{c \in \mathcal{C}} \cos(x, \text{centroid}_c)$ ▷ cosine tie-break

end if

return $c^\star, \lambda^{(c^\star)}, \|x\|$

This design lets ArrowSpace preserve the $O(\text{nnz})$ cost of computing λ once over $M \ll N$ sub-centroids, while giving each of the N original items a consistent lambda by nearest-neighbour lookup in a one-dimensional space, with a cosine tie-break to disambiguate degenerate lambda collisions. The internal consistency of this mapping (subcentroid count matching graph node count, centroid-map indices remaining in-bounds, subcentroid-lambda array length matching subcentroid count) is directly unit-tested (Section 11).

9 Stage 4: λ-Aware Search

9.1 Query Preparation

A query vector $q \in \mathbb{R}^F$ is projected (if JL was used) and scored identically to indexed items via Algorithm 5's per-item kernel, yielding λ_q. This guarantees that query-time and index-time λ

values are computed by the exact same functional, so that lambda-band pruning is meaningful – a guarantee that is not merely architectural but is checked numerically (Section 11).

9.2 Blended Ranking

ArrowSpace ranks candidates by a convex combination of semantic (cosine) similarity and λ-proximity:

$$\text{score}(q, x_i) = \alpha \cdot \cos(q, x_i) \; + \; (1 - \alpha) \cdot \big(1 - |\lambda_q - \lambda_i|\big), \qquad \alpha \in [0, 1],$$

where $\alpha = 1$ recovers pure cosine search and $\alpha = 0$ is pure λ-proximity search. A companion structure, `SortedLambdas` (a `BTreeMap` keyed by OrderedFloat(λ)), supports $O(\log N + k)$ range queries and a k-nearest-by-lambda expanding-window search that starts from a bandwidth derived from the empirical stddev(λ) and grows geometrically until k candidates are found.

Algorithm 7 Stage 4 – λ-Aware Search

Require: Query q, Laplacian L (or `signals`), k, blend weight α
Require: Precomputed $\{\lambda_i\}_{i=1}^{N}$ (Algorithm 5 must have run)
$q_{\text{proj}} \leftarrow \textsc{Project}(q)$
$\lambda_q \leftarrow \textsc{ComputeSyntheticLambda}(q_{\text{proj}}, L)$ ▷ Eq. 3
$\mathcal{N} \leftarrow \textsc{SortedLambdas.KNearestByLambda}(\lambda_q, k', \text{growth})$ ▷ expanding window, $k' \geq k$
for $x_i \in \mathcal{N}$ **do**
    $s_i \leftarrow \alpha \cdot \cos(q, x_i) + (1 - \alpha) \cdot (1 - |\lambda_q - \lambda_i|)$
end for
return top-k items by s_i, descending

9.3 Complexity

Let N be the number of items, F the (possibly reduced) feature dimension, nnz the non-zeros of the feature Laplacian, and k the requested result size.

- **Index build**: $O(Nkd \log N)$ for FastPair-based wiring (Sec. 4) plus $O(\text{nnz})$ per item for τ-mode computation, parallelised across cores $\Rightarrow O((Nkd \log N + N \cdot \text{nnz})/P)$ wall time on P cores.
- **Query**: $O(\text{nnz})$ to compute λ_q, plus $O(\log N + w)$ to retrieve a window of size $w \geq k$ from `SortedLambdas`, plus $O(w \cdot F)$ for the cosine re-rank.
- **No $O(N^2)$ term ever appears**, since neither wiring (thanks to FastPair) nor search requires pairwise item comparison; this is the central practical claim distinguishing Graph Wiring from naive geometric approaches at scale.

10 The Complete Build-and-Search Pipeline

Putting all stages together, the full `ArrowSpaceBuilder::build` path is:

Algorithm 8 Full ArrowSpace Build + Search Pipeline

Require: $X \in \mathbb{R}^{N \times F}$, λ-graph params, TauMode policy, IndexScorePhase Φ

$C, R \leftarrow$ Stage1_ClusterProjectCompress(X) ▷ Alg. 1
$L \leftarrow$ Stage2_BuildLaplacian(C, params) ▷ Alg. 3, FastPair-backed (Alg. 2)
if $\Phi =$ FeatureSpectralScore **then**
    $L_F \leftarrow$ BuildSpectralLaplacian(L) ▷ Laplacian-of-Laplacian, Stage 2b
end if
$\{\lambda_i\} \leftarrow \Phi$.Compute($L$ or L_F) ▷ Alg. 5 for $\Phi =$ TauModeScore
Index $\leftarrow$ SortedLambdas.BuildFrom($\{\lambda_i\}$)

function Search(q, k, α)
    return Stage4_LambdaAwareSearch(q, L, k, α) ▷ Alg. 7
end function

11 Experiments

The claims made in the preceding sections are grounded in a test-driven experimental suite spanning synthetic clustering scenarios, graph-structural invariants, TauMode determinism and boundedness, energy-mode consistency, and motif discovery on planted-clique data. Unlike the retrieval-quality benchmarks discussed in Section 12, the results below are direct unit-test assertions against the implementation.

11.1 TauMode Determinism, Boundedness, and Invariance

The τ-mode synthetic lambda (Eq. 3) is tested directly against its defining properties rather than only against downstream retrieval quality [4]. Lambda values are verified finite and bounded in $[0, 1]$ across multiple synthetic datasets, including high-noise Gaussian blobs and high-dimensional moons data [4]. The Rayleigh-quotient component is explicitly checked for scale invariance: computing λ on a vector x and on $2x$ produces identical values to 10^{-10} tolerance [4]. Repeated evaluation of $\lambda(x)$ for a fixed built model is checked to be bit-stable to 10^{-12} tolerance, including under Johnson–Lindenstrauss projection with a fixed seed, and the query-time lambda is verified to match the indexed lambda for the same item to 10^{-9} tolerance [4]. All four `TauMode` policies (`Fixed`, `Mean`, `Median`, `Percentile`) are unit-tested against hand-computed expected values, including edge cases with NaN/Inf energies and empty distributions, all of which correctly fall back to the numerical floor τ_{floor} [4]. A separate test confirms that different `TauMode` policies applied to the same underlying graph produce measurably different lambda distributions, validating that the policy selection is not a cosmetic parameter [4].

11.2 Laplacian Structural Invariants

The graph produced by Stage 2 is tested for the structural properties a Laplacian must satisfy [5]. Diagonal entries are checked to be present, finite, and non-negative for every node [5]. Off-diagonal symmetry is verified exhaustively: for every stored edge (i, j), the mirrored entry (j, i) is located and checked to match within 10^{-10} relative tolerance [5]. Graph-construction parameters $(\varepsilon, k, \text{topk}, p, \sigma, \text{normalise})$ are checked to be preserved unchanged through the builder [5]. Enabling `prebuilt_spectral` is checked to produce a signal matrix of shape exactly $F \times F$, while disabling it leaves the signal matrix empty [5].

11.3 Clustering Robustness Under Noise

The clustering stage that feeds Stage 2 is tested across dimensionality and noise regimes [5]. On low-, medium-, and high-dimensional synthetic moons data, the builder is verified to always produce a non-empty, valid clustering [5]. Under normalisation, clustering is tested for scale invariance: rescaling all inputs by a constant factor $c = 5.7$ changes the resulting cluster count by at most 3, and leaves the graph node count exactly unchanged [5]. On Gaussian-blob data with high noise (overlap parameter 0.9), the heuristic is explicitly tested to prefer *under*-clustering (choosing $K = 2$ instead of the planted $K = 3$) rather than over-fragmenting, and this behaviour is documented in the test as correct rather than as a defect [5].

11.4 Energy-Mode (Compression) Consistency

For the compression-oriented energy pipeline, tests verify that the number of subcentroids matches the energy graph's node count, that every item's centroid-map entry references a valid subcentroid index, and that the stored subcentroid lambdas align in length with the subcentroid count [4]. A dedicated test confirms that energy-based search returns a non-empty result set without panicking on a representative query [4]. Lambda bounds are additionally verified end-to-end for the energy path on high-dimensional Gaussian data with dimensionality reduction enabled [4].

11.5 Motif Discovery Cross-Validation

The `FeatureSpectralScore`-adjacent motif-discovery machinery is cross-validated between the `EigenMaps` and `EnergyMaps` pipelines on planted-clique synthetic data [6]. Both pipelines are independently required to discover a non-empty set of motifs on the same underlying planted-clique structure, and the top-ranked motif recovered by each pipeline is compared via Jaccard overlap after deduplication, testing that the two independently-derived scoring mechanisms agree on the dominant structural motif [6]. Subgraph-level tests further confirm that extracted motifs preserve dimensional invariants (feature-Laplacian shape $F \times F$, node-index bounds, item-to-centroid mapping validity under energy mode) and that relaxing the Rayleigh-quotient filter threshold monotonically admits at least as many subgraphs as a stricter threshold [6].

11.6 Storage Round-Trip and Persistence

A separate suite validates that a built `ArrowSpace` and its `GraphLaplacian` can be serialised to disk and reloaded with lambda values, cluster metadata, sorted-lambda index ordering, and graph parameters all reproduced exactly, including a negative test that correctly rejects storage where the lambda count does not match the item count [7].

12 Benchmarks

Distinct from the unit-test evidence of Section 11, the following are external retrieval-quality benchmarks rather than implementation-correctness assertions. On a dense-embedding CVE (vulnerability description) benchmark of 18 real queries, τ-mode-aware search delivers a consistent lift over pure cosine: mean head/tail quality improves from 0.833 (cosine) to 0.887 (τ-mode), a 0.054 absolute gain, and τ-mode wins top-1 head/tail quality on all 18/18 queries. The advantage is not confined to head results: cumulative score advantage grows roughly linearly to ≈ 0.65 by rank 15, indicating that the spectral signal specifically improves *tail* quality – the hardest regime for cosine-only retrieval. τ-mode also achieves the best head/tail ratio on 14/18 queries and the lowest

tail coefficient of variation on 14/18 queries, indicating more predictable deep-rank behaviour under multi-query workloads. On the sparse, high-dimensional Dorothea dataset, the same substrate (with the `FiedlerVectorScore`-style partitioning rather than `TauModeScore`) shows promise as a clustering tool comparable to UMAP, while τ-mode itself under-performs there – consistent with τ-mode being purpose-built for search rather than for general classification, and motivating the pluggable-score design of Section 14 rather than a single fixed scoring rule. Independently of scoring choice, the FastPair-backed wiring stage (Section 4) reduces Laplacian-construction time by roughly 750× relative to brute-force pairwise search at the $n = 10{,}000$, $k = 10$, $d = 384$ scale used in these experiments, without changing the resulting graph's downstream quality.

13 External Validation: The SPIN Retrieval Study

Independently of the internal unit-test suite of Section 11 and the CVE/Dorothea benchmarks of Section 12, the τ-mode search capability of ArrowSpace has been evaluated end-to-end as *SPectral INdexing* (SPIN) in a dedicated retrieval study built directly on the `arrowspace` library [11]. SPIN implements the same λ_τ-blended ranking $\text{score}(q, x_i) = \alpha \cdot \text{sim}_{\cos}(q, x_i) + (1-\alpha)\,\text{sim}_\lambda(q, x_i)$ described in Section **??**, and evaluates it on two retrieval benchmarks distinct from those in Section 12: a 347,113-document CVE vulnerability corpus with 50 LLM-generated queries, and TREC-COVID, a 171,000-document biomedical benchmark with 50 expert queries and human relevance judgments [11].

CVE experiment. Across three blending settings — pure cosine ($\tau = 1.0$), hybrid ($\tau = 0.72$), and taumode ($\tau = 0.42$) — taumode retrieval achieved the highest Tail/Head ratio (0.991 vs. 0.986 for cosine), the lowest Tail coefficient of variation (0.00254 vs. 0.00396), and the lowest Tail Decay (0.000359 vs. 0.000512) at $K_h = 3$, winning on these three tail-shape metrics in 40–43 of 50 queries [11]. Tolerant Recall@25 remained near-perfect at 0.993 for taumode, and Semantic Uplift (the gap between Tolerant and Traditional Recall) averaged +0.573, confirming that spectral reranking does not sacrifice retrieval fidelity while flattening the tail-quality decay that pure cosine search exhibits [11].

TREC-COVID experiment. On the human-labelled TREC-COVID benchmark, intermediate τ values in $[0.65, 0.80]$ improved Relevance Recall@10 from a cosine baseline of 0.508 to a peak of 0.522, and NDCG@10 from 0.455 to approximately 0.467 [11]. Tail-shape metrics improved monotonically as spectral weight increased: taumode ($\tau = 0.4$) reached a Tail/Head ratio of 0.980 versus 0.949 for cosine, with Tail CV dropping from 0.01362 to 0.00459 [11]. Semantic Uplift rose from 0.00 under pure cosine to 0.334 under taumode, with 42 of 50 queries showing measurable gains [11].

Relation to the internal test suite. These results externally corroborate two claims already supported by unit tests in Section 11: that the λ_τ scalar of Equation (3) is stable enough to be trusted as a ranking signal under real query workloads, and that its principal practical benefit is in the tail of the ranked list rather than at the head — consistent with the CVE benchmark reported in Section 12, where τ-mode gains grew roughly linearly with rank depth. The SPIN study further grounds this behaviour theoretically via an *epiplexity* argument, showing via a two-part MDL criterion that the feature-space Laplacian L_F encodes structural information beyond what is available to geometric similarity alone [11]. This is the same information-theoretic motivation

informally proposed for the sixth, not-yet-implemented score of Section 6.1, suggesting a convergent design rationale between the production τ-mode search phase and the speculative epiplexity phase.

14 Discussion: Why a Pluggable Score Phase?

The test evidence of Section 11, taken together with the benchmark split of Section 12, supports a specific reading of the ArrowSpace design: the wiring substrate (clustering, FastPair-backed k-NN, Laplacian construction) is validated as a robust, invariant-preserving geometric operation, while each `IndexScorePhase` is validated separately against the mathematical properties it claims, rather than against a single universal benchmark.

The strongest and most exhaustively tested property is determinism and boundedness of the τ-mode scalar. Repeated computation, cross-checking query-time against index-time lambda, and scale-invariance under vector rescaling are all directly exercised [4]. This matters practically because it is what allows lambda-band pruning at search time (Algorithm 7) to be trusted rather than merely hoped for: if λ were not reproducible to high numerical tolerance, expanding-window lambda-nearest-neighbour search could silently miss valid candidates.

A second consistent pattern is that clustering behaviour is intentionally conservative under noise rather than exact under a fixed target K [5]. The high-noise Gaussian-blob test is explicit about this: when cluster overlap makes $K = 3$ statistically ambiguous, the heuristic is documented and tested to prefer $K = 2$, treating under-clustering as the safer failure mode for downstream graph construction [5]. This is consistent with the broader design goal of Stage 1 (Section 3): clustering quality is judged by its effect on downstream Laplacian usability, not by exact recovery of a planted partition.

Third, the motif cross-validation tests are the closest available evidence for the claim that different `IndexScorePhase` implementations operating on the same wiring substrate agree on dominant structure [6]. The Jaccard-overlap comparison between `EigenMaps`-derived and `EnergyMaps`-derived motifs on identical planted-clique data is a direct, if narrow, test of the central architectural claim of this section: that wiring is capability-agnostic and only the final scoring stage differs.

Finally, the search-versus-classification split observed in the CVE and Dorothea benchmarks (Section 12) is the practical justification for decoupling *graph wiring* from *index scoring* in the first place. Graph wiring is a dataset-agnostic geometric operation that is expensive to redo; the scoring phase is comparatively cheap and can be swapped per use case without rebuilding the graph:

- Swap in `FiedlerVectorScore` to get bipartite partitions from the same L used for search, useful when a dataset is discovered to have a strongly bimodal topology (e.g. Dorothea-style sparse binary features).
- Swap in `BoundaryEnergyScore` once class labels become available, turning a search index into an active-learning triage tool that flags items near decision boundaries without any additional wiring cost.
- Swap in `EnergyDiffusionScore` periodically as a maintenance job to compress a growing production index, evicting redundant low-λ items while protecting rare high-energy sub-centroids.
- Swap in `FeatureSpectralScore` offline for mechanistic interpretability sweeps (e.g. SAE decoder-feature communities), reusing the same transpose-and-wire machinery used for the item-level index.

What the test suite does *not* yet establish is a formal argument for why bounded Rayleigh energy plus edge dispersion (Eq. 3) is the correct blend for search specifically, as opposed to classification or clustering; that separation currently rests on the CVE-versus-Dorothea retrieval comparison

of Section 12 rather than on a property proven by the unit-test suite. Likewise, while Laplacian symmetry and parameter preservation are tested exhaustively [5], the FastPair-backed adjacency construction of Section 4 is validated indirectly, through the correctness of the Laplacian it produces, rather than through a dedicated benchmark of the `CosinePair` query structure itself.

In this sense, ArrowSpace is best understood not as a vector index with one scoring rule, but as a *graph-wiring runtime* on top of which `IndexScorePhase` implementations are hot-swappable, capability-defining plugins – with FastPair providing the sub-quadratic backbone that makes rebuilding or re-wiring that graph practical at scale, and with the test suite of Section 11 providing evidence that the substrate itself is invariant-preserving even as the scoring layer above it changes.

15 Conclusion

Graph Wiring reduces an $N \times F$ dataset to a sparse feature-space Laplacian and a bounded, scale-invariant, per-item spectral scalar $\lambda \in [0, 1]$, computed once in $O(\text{nnz})$ per item and reused at query time without ever requiring an $O(N^2)$ pairwise comparison. The wiring stage itself avoids $O(N^2)$ cost by building its k-NN adjacency with a FastPair-derived structure (`CosinePair`), reducing construction from $O(N^2 d)$ to $O(Nkd \log N)$. The τ-mode instantiation of this substrate – selecting τ from an energy distribution and blending bounded Rayleigh energy with edge dispersion – yields a stable, deterministic, scale-invariant index that measurably improves tail-quality retrieval over cosine-only search. These properties are not merely asserted: determinism, boundedness, scale invariance, and query/index consistency are each directly exercised by the test suite documented in Section 11, alongside structural invariants of the Laplacian itself and cross-pipeline agreement on discovered motifs. Replacing the scoring phase while keeping wiring fixed extends the same machinery to clustering, classification, compression and mechanistic analysis, motivating the description of ArrowSpace as a generic "Graph Wiring" algorithm rather than a single-purpose search index. What remains open is a formal, rather than empirical, account of why the τ-mode blend is specifically well suited to search among the five pluggable scoring phases – a question we leave to future theoretical work. This effectiveness is corroborated independently by the SPIN retrieval study [11], which applies the same τ-mode blended ranking to two benchmarks external to this report's own test suite: a 347,113-document CVE corpus and the 171,000-document, human-labelled TREC-COVID benchmark. On both, λ-aware retrieval consistently outperformed pure cosine search on tail-quality metrics — raising the Tail/Head ratio, lowering Tail coefficient of variation, and lowering Tail Decay — while simultaneously improving Relevance Recall@10 and NDCG@10 on TREC-COVID and preserving near-perfect Tolerant Recall@25 on the CVE corpus [11]. That gains concentrate in the tail of the ranked list, precisely where pure cosine similarity is weakest, on two independent, human- and LLM-judged benchmarks, reinforces the central claim of this report: a single bounded, deterministic, scale-invariant scalar derived from Graph Wiring is sufficient to measurably improve retrieval quality without discarding semantic similarity, and does so consistently across datasets [11].

Acknowledgments

The authors acknowledge the use of AI-assisted tools during the preparation of this paper, including Perplexity for literature discovery, OpenAI GPT models for drafting and result analysis, NVIDIA Nemotron for exploratory formal verification analysis, Claude Sonnet 4.6 for coding support and implementation checks, and GLM-5.2 by Z.AI for code generation and experimental iteration [24,

25, 26, 27, 28]. All technical claims, interpretations, experiments, and final editorial decisions remain the sole responsibility of the authors.

References

[1] L. Moriondo. *Graph Wiring: Eigenstructures for Vector Datasets and LLMs*. Independent Researcher, 22 February 2026.

[2] L. Moriondo. *ArrowSpace: introducing Spectral Indexing for vector search*. Journal of Open Source Software, 10, 2025.

[3] L. Moriondo. *ArrowSpace-rs: Spectral Vector Search with Lambda-Tau Indexing*. Rust codebase, 2025–2026. `https://github.com/Mec-iS/arrowspace-rs`

[4] ArrowSpace contributors, *test_taumode.rs*, in *arrowspace-rs*. Covers TauMode policy selection (`Fixed`, `Mean`, `Median`, `Percentile`), floor enforcement, lambda boundedness and finiteness, Rayleigh-quotient scale invariance, query/index lambda consistency, and energy-mode dimension/centroid-map consistency. `https://github.com/Mec-iS/arrowspace-rs/blob/main/src/tests/test_taumode.rs`

[5] ArrowSpace contributors, *test_graph_factory.rs*, in *arrowspace-rs*. Covers Laplacian diagonal non-negativity and finiteness, adjacency symmetry, graph-parameter preservation, clustering robustness under noise and dimensionality, scale invariance under normalisation, and spectral Laplacian shape validation ($F \times F$). `https://github.com/Mec-iS/arrowspace-rs/blob/main/src/tests/test_graph_factory.rs`

[6] ArrowSpace contributors, *test_motives.rs* and *test_subg_motives.rs*, in *arrowspace-rs*. Covers motif discovery on planted-clique synthetic data, cross-validation between `EigenMaps` and `EnergyMaps` pipelines via Jaccard overlap of top-ranked motifs, and subgraph dimensional invariants under Rayleigh-quotient filtering. `https://github.com/Mec-iS/arrowspace-rs/blob/main/src/tests/test_motives.rs`

[7] ArrowSpace contributors, *test_load_from_storage.rs*, in *arrowspace-rs*. Covers parquet-based persistence round-trips for `ArrowSpace` and `GraphLaplacian`, sorted-lambda index correctness after reload, and rejection of mismatched lambda/item counts. `https://github.com/Mec-iS/arrowspace-rs/blob/main/src/tests/test_load_from_storage.rs`

[8] ArrowSpace contributors, *Proposal: LearnableNoveltyScore (Epiplexity) as a new IndexScorePhase*. GitHub Issue #118, *arrowspace-rs*, tuned-org-uk, 2026. `https://github.com/tuned-org-uk/arrowspace-rs/issues/118`

[9] Authors withheld pending citation confirmation. *Learnable Novelty: A Bounded-Observer Complexity Measure for Structured Data*. arXiv:2607.18433, 2026. `https://arxiv.org/pdf/2607.18433`

[10] J. Shawe-Taylor, C. K. I. Williams, N. Cristianini, and J. Kandola. On the Eigenspectrum of the Gram Matrix and the Generalization Error of Kernel-PCA. *IEEE Transactions on Information Theory*, 51(7):2510–2522, 2005.

[11] L. Moriondo and I. Azizi. From Embedding Geometry to Spectral Search: Energy Dispersion Networks For Vector Retrieval. arXiv:2606.21535, 2026. `https://arxiv.org/abs/2606.21535`

[12] M. Belkin and P. Niyogi. *Convergence of Laplacian Eigenmaps*. NeurIPS, 2008.

[13] N. García Trillos and D. Slepčev. *Continuum limit of total variation on point clouds*. Archive for Rational Mechanics and Analysis, 220(1):193–241, 2016.

[14] Meng, Piazza, Barabási et al. *Surface optimization governs the local design of physical networks*. Nature, 2026. DOI:10.1038/s41586-025-09784-4.

[15] D. Tong. *String Theory*. Lecture notes, University of Cambridge, 2009.

[16] MVA (Mathématiques, Vision, Apprentissage) Master's Programme, *Kernel Methods for Machine Learning*, lecture course, ENS Paris-Saclay. Covers RKHS theory, Mercer/Bochner/Herglotz representer theorems, SVMs, graph kernels, kernel mean embeddings, and scalable kernel machines.

[17] T. Gärtner, P. Flach, and S. Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. In *Learning Theory and Kernel Machines (COLT/Kernel 2003)*, Lecture Notes in Computer Science, vol. 2777, pp. 129–143, 2003.

[18] A. Rahimi and B. Recht. Random Features for Large-Scale Kernel Machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

[19] D. Eppstein. *Fast Hierarchical Clustering and Other Applications of Dynamic Closest Pairs*. ACM Journal of Experimental Algorithmics, 5, 2000.

[20] A. J. Smola and R. I. Kondor. Kernels and Regularization on Graphs. In *Learning Theory and Kernel Machines (COLT/Kernel 2003)*, Lecture Notes in Computer Science, vol. 2777, pp. 144–158, 2003.

[21] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, 2002.

[22] R. I. Kondor and J. Lafferty. Diffusion Kernels on Graphs and Other Discrete Input Spaces. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, pp. 315–322, 2002.

[23] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

[24] Perplexity AI, Perplexity: AI-powered answer engine, https://www.perplexity.ai, 2026.

[25] OpenAI, GPT-5: Generative Pre-trained Transformer 5, https://openai.com, 2026.

[26] NVIDIA, Nemotron: Large Language Model for Formal Verification, https://www.nvidia.com, 2026.

[27] Anthropic, Claude Sonnet 5, https://www.anthropic.com, 2026.

[28] Z.AI, GLM-5.2: General Language Model 5.2, https://z.ai, 2026.