Learning-Based Hashing for ANN Search: Foundations and Early Advances

Sean Moran   SEAN.J.MORAN@GMAIL.COM

Abstract

Approximate Nearest Neighbour (ANN) search is a fundamental problem in information retrieval, underpinning large-scale applications in computer vision, natural language processing, and cross-modal search. Hashing-based methods provide an efficient solution by mapping high-dimensional data into compact binary codes that enable fast similarity computations in Hamming space. Over the past two decades, a substantial body of work has explored *learning to hash*, where projection and quantisation functions are optimised from data rather than chosen at random.

This article offers a foundational survey of early learning-based hashing methods, with an emphasis on the core ideas that shaped the field. We review supervised, unsupervised, and semi-supervised approaches, highlighting how projection functions are designed to generate meaningful embeddings and how quantisation strategies convert these embeddings into binary codes. We also examine extensions to multi-bit and multi-threshold models, as well as early advances in cross-modal retrieval.

Rather than providing an exhaustive account of the most recent methods, our goal is to introduce the conceptual foundations of learning-based hashing for ANN search. By situating these early models in their historical context, we aim to equip readers with a structured understanding of the principles, trade-offs, and open challenges that continue to inform current research in this area.

1. Motivation

The rapid growth of the World Wide Web (WWW) over the past two decades has brought with it a phenomenal increase in the amount of image, video and text based data being collected, stored and shared across the world. This phenomenon has been fuelled by the popularity of social media networks, cheap disk storage and the wide availability of Internet-enabled smartphones. For example it has been estimated that Facebook has in the order of 300 million images uploaded per day[1], YouTube receives 10 years worth of content per day[2] and there are now estimated to be well over 1 trillion web pages[3] in existence ([60]). Figure 1 illustrates the explosive growth of images being uploaded onto popular social media websites and applications during

---

1. Velocity 2012: Jay Parikh, "Building for a Billion Users"
2. `http://www.youtube.com/yt/press/en-GB/statistics.html`
3. `http://googleblog.blogspot.co.uk/2008/07/we-knew-web-was-big.html`
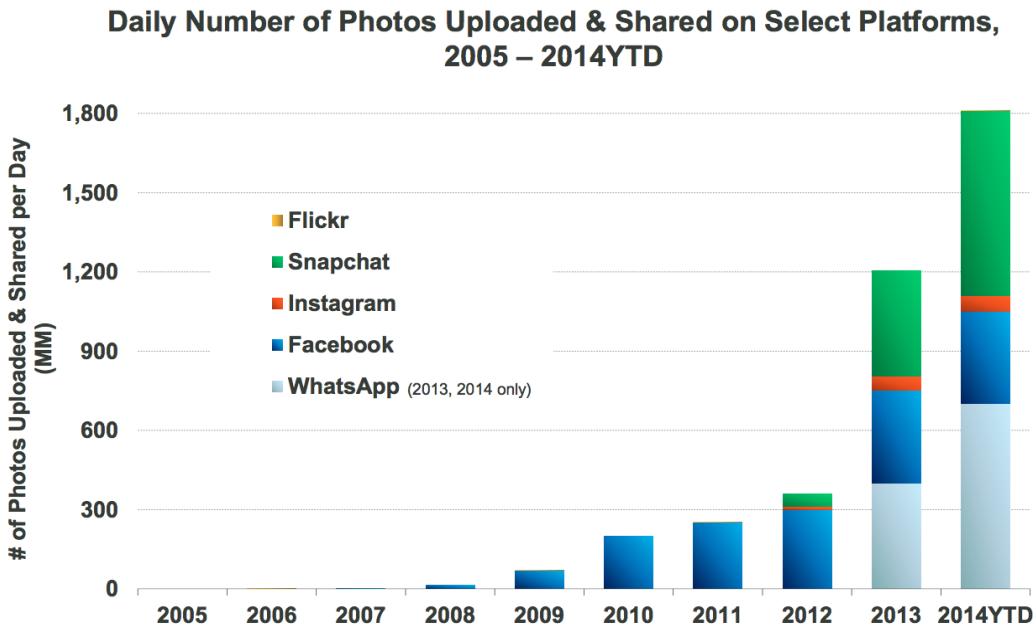
Figure 1: The amount of images being uploaded to popular social media websites (Facebook, Flickr) and mobile applications (Instagram, SnapChat, WhatsApp) has undergone a dramatic growth since 2005. Efficient algorithms for searching through such large image datasets are needed now more than ever. This chart has been copied directly from slide 62 of the talk *"Internet Trends 2014 - Code Conference"* given by the venture capitalist Mary Meeker of Kleiner Perkins Caufield Byers (KPCB): `http://www.kpcb.com/blog/2014-internet-trends` (URL accessed on 16/12/15).

the period 2005-2014. The trend towards real-time video sharing over the Internet with applications such as Periscope, involving a medium that is many times the size of individual images or documents, will severely exacerbate this torrent of data. In the near-term future the emergence of the Internet-of-Things (IoT) and Smart Cities promise to add further fuel to this fire, hinting at a connected society in which Internet linked sensors embedded in everyday objects, such as CCTV cameras and thermostats, produce an abundance of data that is automatically captured, stored and analysed so as to produce actionable insights for interested citizens and government stakeholders ([2]). The sheer scale of the data being produced around the world brings with it a need for computationally efficient algorithms that ensure the storage requirements and processing overhead do not grow with the quantity of the data being produced.

In this article we focus on the problem of nearest neighbour (NN) search where the goal is to find the most similar data-point(s) to a query in a large database. Similarity is typically judged by representing the data-points as fixed dimensional vectors in a vector space and computing a distance metric such as the Euclidean or

cosine distance. In this case data-points with a sufficiently low distance are judged to be nearest neighbours. Due to its generality and usefulness nearest-neighbour search finds application in many areas of Science, ranging from the field of Information Retrieval (IR) where we wish to find documents relevant to a query, to the problem of Genomic assembly in the field of Bioinformatics. The näive way to solve the nearest-neighbour search problem would be to compare the query to every data-point in the database, a method known as *brute-force search*. Brute-force search is only feasible in relatively small databases where performing the number of required comparisons between the data-points remains computationally tractable. Given the linear scaling of the query time with respect to the dataset size it is impossible to exhaustively search large-scale datasets consisting of millions to billions of data-points for nearest neighbours in a reasonable amount of time. This problem is compounded in the streaming data scenario where data-points need to be processed sequentially in real-time with potentially no end to the amount of incoming data. To efficiently find nearest-neighbours in large-scale datasets, algorithms are required that offer a query time that is independent of the dataset size.

Hashing-based approximate nearest neighbour (ANN) search methods are a popular class of algorithms that permit the nearest neighbours to a query data-point to be retrieved in constant time, independent of the dataset size. Hashing has proved to be an extremely useful method for ANN search over high-dimensional, large-scale datasets that are prevalent in the modern data-rich world. Hashing permits constant time search per query by condensing both the database and the query into fixed-length compact binary hashcodes or fingerprints. The hashcodes exhibit the neighbourhood preserving property that similar data-points will be assigned similar (low Hamming distance) hashcodes. Crucially, unlike cryptographic hash functions such as MD5 or SHA-1, the data-points need not be identical to receive matching hashcodes. Rather the degree of similarity between the hashcodes is a direct function of the similarity between the feature representation of the data-points. This property is ideal for the task of image retrieval where we rarely wish to find only those images that are identical down to the pixel level. Most people would deem two images to be related even if the semantically equivalent objects (e.g. a tiger) depicted in both images are in widely different poses, and therefore the images have a completely different pixel consistency.

The aforementioned similarity preserving property enables the hashcodes to be used as the keys into the buckets of hashtables so that similar, but not necessarily identical, images will collide in the same buckets (Figure 2). This is a rather different use-case to the typical application of hashtables in Computer Science in which it is imperative to avoid collisions between non-identical data-points. In hashing-based ANN search we are actively encouraging collisions between similar data-points. The bucketing of the data-points drastically reduces the computational overhead of nearest neighbour search by reducing the number of comparisons that are required between the data-points: at query time we need only compare our query to those data-points
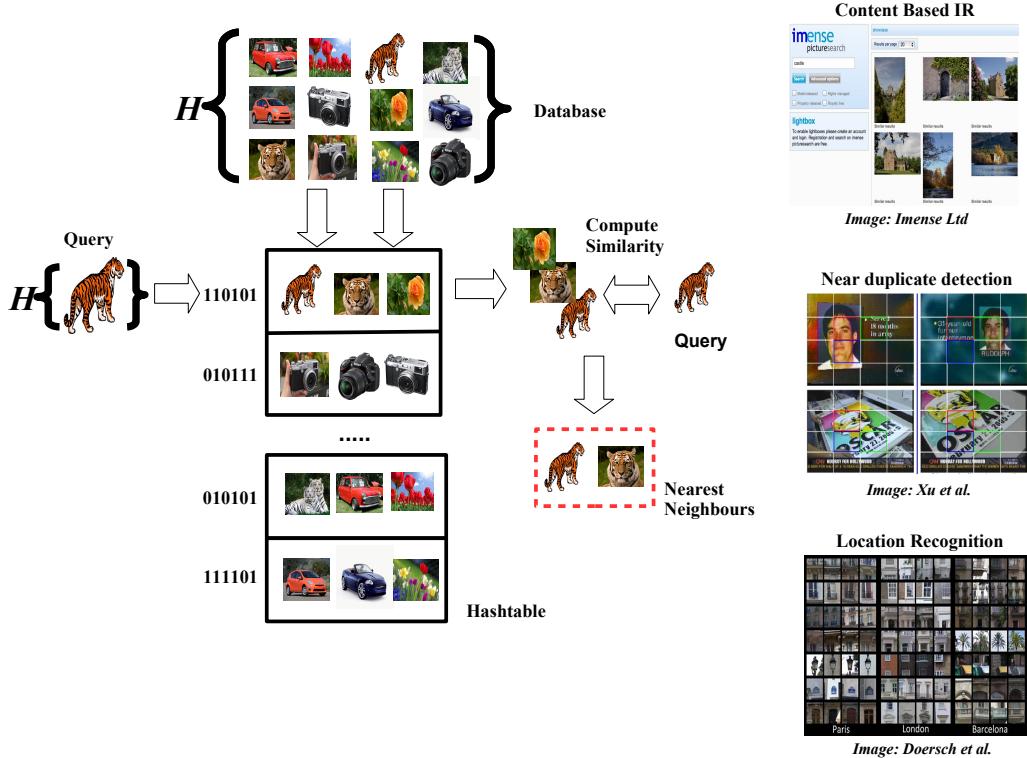
Figure 2: Nearest neighbour search with hashcodes. Similarity preserving binary codes generated by a hash function $\mathcal{H}$ can be used as the indices into the buckets of a hashtable for constant time search. Only those images that are in the same bucket as the query need be compared thereby reducing the size of the search space. The focus of this review is learning the hash function $\mathcal{H}$ to maximise the similarity of hashcodes for similar data-points. On the right-hand side we present examples of tasks for which nearest neighbour search has proved to be fundamental: from content-based information retrieval (IR) to near duplicate detection and location recognition. The three images on the right have been taken from Imense Ltd (`http://www.imense.com`) and [18, 93, 29].

colliding in the same buckets. There is no free lunch however as we pay for the reduced query time with a non-zero probability of failing to retrieve the closest nearest neighbours in the case where they happen to fall in different buckets. Nevertheless this quantifiable non-zero false negative probability turns out to be an acceptable trade-off in many application areas in which sub-optimal nearest neighbours can be almost as good as finding the exact nearest neighbour ([17, 64]).

Hashing-based ANN search is commonly applied to the task of content-based image retrieval, a signature problem encountered within the fields of IR and Computer Vision that is characterised by an abundance of data and the need for accurate and efficient search. Hashing-based ANN has also shown great promise in terms of efficient

query processing and data storage reduction across a wide range of other interesting application areas within IR and Computer Vision. For example [64] present an efficient method for event detection in Twitter that scales to unbounded streams through a novel application of Locality Sensitive Hashing (LSH), a seminal randomised approach for ANN search ([36]). In the streaming data scenario of [64] the $\mathcal{O}(N)$ worst case complexity of inverted indexing is undesirable, motivating the use of LSH to maintain a constant $\mathcal{O}(1)$ query time[4]. Hashing-based ANN has also proved particularly useful for search over dense and much lower dimensional (compared to text) feature vectors, such as GIST ([61]), that are commonly employed in the field Computer Vision. In one recent application of LSH within Computer Vision, similarity preserving hashcodes have been successfully used for fast and accurate detection of 100,000 object classes on just a single machine ([17]).

---

4. This is only true if we ignore the hashing cost (cost of generating the hashcode) and assume that each database data-point goes into its own hashtable bucket. In practice the LSH computational cost for a single hashtable and a single data-point is a sum of the hashing cost ($\mathcal{O}(KD)$), lookup cost ($\mathcal{O}(1)$) and the candidate test cost ($\mathcal{O}(ND/2^K)$), where K is the hashcode length and assuming a uniform distribution of data-points to buckets. Observe that there is a trade-off between the hashing cost and the candidate test cost, both of which are dependent on K. For example, in the situation where the data-points are evenly distributed into their own hashtable bucket ($N = 2^K$), the total computational cost for LSH is actually sub-linear ($\mathcal{O}(D \log N)$).
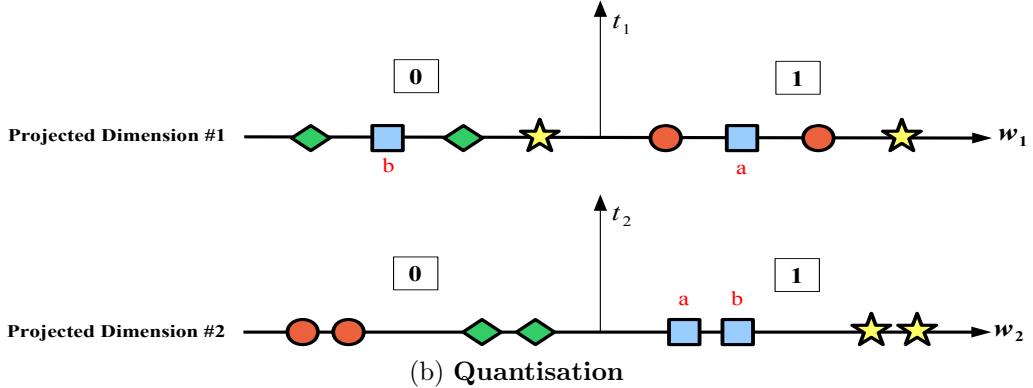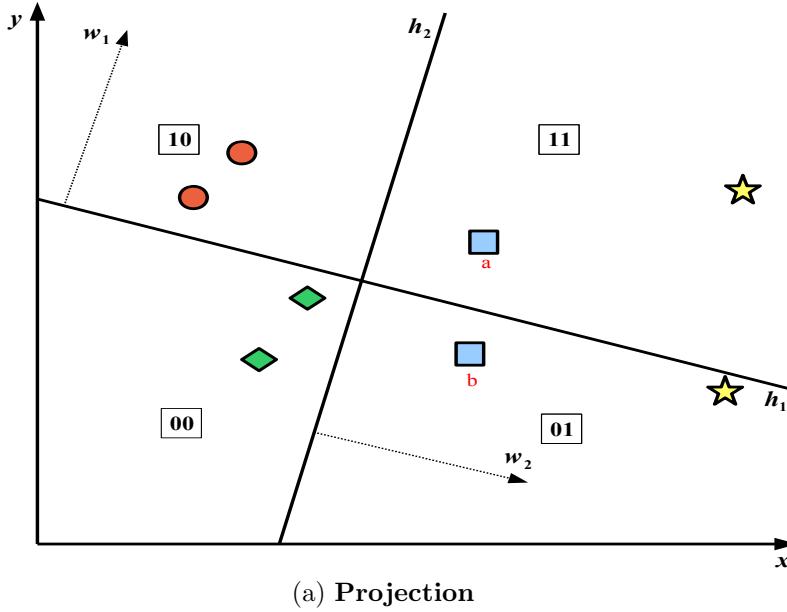
(a) **Projection**

(b) **Quantisation**

Figure 3: The projection and quantisation operations. In Figure (a) a 2D space is partitioned with two hyperplanes $\mathbf{h}_1$ and $\mathbf{h}_2$ with normal vectors $\mathbf{w}_1, \mathbf{w}_2$ creating four buckets. Data-points are shown as coloured shapes, with similar data-points having the same colour and shape. The hashcode for each data-point is found by taking the dot-product of the feature representation onto the normal vectors ($\mathbf{w}_1$, $\mathbf{w}_2$) of each hyperplane. The resulting projected dimensions are binarised by thresholding at zero (Figure (b)) with two thresholds t_1, t_2. Concatenating the resulting bits yields a 2-bit hashcode for each data-point (indicated by the unfilled squares). For example the projection of data-point a is greater than threshold t_1 and so a '1' is appended to its hashcode. Data-point a's projection onto normal vector $\mathbf{w}_2$ is also greater than t_2 and so a '1' is further appended to its hashcode. The hashcode for data-point a is therefore '11' which is also the label for the top-right region of the feature space in Figure (a).

The ANN search hashing models we will review in this article all partition the input feature space into disjoint regions with a set of hypersurfaces, either linear (hyperplanes) or non-linear. In the case of linear hypersurfaces the polytope-shaped regions formed by the intersecting hyperplanes constitute the hashtable buckets (Figure 3). The hashtable key for a data-point is generated by simply determining which side of the hyperplanes the data-point lies. Depending on which side it falls a '0' or a '1' is appended to the hashcode for that data-point. By repeating this procedure for each hyperplane we can build up a hashcode for each data-point that is the same length as the number of hyperplanes partitioning the space. Intuitively, the hashcode can be thought of as an identifier that captures the geometric position of the data-points within the input feature space with each bit encoding the position of the data-point with respect to a given hyperplane. Algorithmically this hashcode generation procedure can be accomplished in two separate steps performed in a pipeline: *projection* followed by *quantisation*. This procedure is illustrated with a toy example in Figure 3. Projection involves a dot product of the feature vector representation of a data-point onto the hyperplane normal vectors positioned either randomly or in data-aware positions in the feature space. The hyperplanes should ideally partition the space in a manner that gives a higher likelihood that similar data points will fall within the same region, and therefore assigned the same hashcode. In the second step the real-valued projections are quantised into binary ('0' or '1') by thresholding the corresponding projected dimensions[5] typically with a single threshold placed at zero for mean centered data.

Despite the success and wide application of algorithms for hashing-based ANN search there still remains considerable downsides to the manner in which the projection and quantisation steps are typically performed. Locality Sensitive Hashing (LSH), one of the arguably most well-known and widely applied methods for hashing-based ANN search, sets the hashing hyperplanes and the quantisation thresholds in a manner that is *independent* of the distribution of the data. For example, in the variant of LSH for preserving the cosine similarity, the normal vectors of the hashing hyperplanes are randomly sampled from a zero mean unit variance multidimensional Gaussian distribution. This data-oblivious mechanism for generating the hashing hypersurfaces runs a high risk of separating dense areas of the feature space and therefore partitioning related data-points into different hashtable buckets (e.g. points a and b in Figure 3). To ameliorate this low recall problem a typical LSH deployment involves partitioning the data with multiple independent hashtables and presenting the union of all the data-points in the colliding hashtable buckets as candidate nearest neighbours. Unfortunately, the greater the number of hashtables the higher the memory requirements needed for an LSH deployment. The quantisation thresholds are also set in a data-independent manner, typically by thresholding at zero along a projected dimension. In this context a projected dimension is formed from collating

---

5. We define a projected dimension as the collection of the real-valued projections (dot products) of all data-points onto the normal vector to a hyperplane.

the projections from all data-points onto the normal vector to a hyperplane. Unfortunately, the region around zero on a projected dimension is usually the area of highest point density which means that there is a high chance of related data-points falling on opposite sides of the threshold and therefore being assigned different bits.

There remains substantial opportunity to enhance the retrieval effectiveness of Locality-Sensitive Hashing (LSH) and other foundational but data-oblivious algorithms for hashing-based approximate nearest neighbour search. This survey focuses on addressing two critical challenges in this domain: improving the neighbourhood preservation at each stage of the hashing pipeline and optimizing hashcode efficiency. In particular, we emphasize the importance of preserving relative distances from the original feature space when mapping to the Hamming space. Strong neighbourhood preservation ensures that similar data points yield similar binary codes, thereby improving retrieval performance. Additionally, we highlight the practical constraint of achieving this with minimal hashcode length to reduce both storage requirements and computational overhead.

2. Survey Contributions

This article presents a comprehensive survey of the projection and quantisation operations in hashing-based approximate nearest neighbour (ANN) search. As discussed earlier, both stages are fundamental to the process of generating similarity-preserving hashcodes. The central hypothesis explored in this survey can be succinctly stated as follows:

The retrieval effectiveness of existing hashing-based ANN search methods can be significantly enhanced by learning both the quantisation thresholds and hashing hyperplanes in a manner informed by the distribution of the data.

This hypothesis underpins all contributions discussed in the article. Locality-Sensitive Hashing (LSH), a widely used baseline, illustrates the importance of this investigation. LSH randomly selects a projection dimension and then applies a fixed threshold at zero to quantise each data point into a binary value. Repeating this operation K times yields a K-bit hashcode. This procedure rests on three key assumptions:

- A_1: A single, static threshold placed at zero (assuming mean-centered data)

- A_2: Uniform allocation of thresholds across all dimensions

- A_3: Linear hyperplanes positioned randomly

These assumptions, commonly found in the literature [36, 91, 51, 27, 65, 47], simplify the hash function design but often lead to suboptimal performance. In

response, this article surveys novel, data-driven techniques designed to relax each of these assumptions. The proposed algorithms are evaluated against strong baselines on standard image retrieval benchmarks, with statistically significant improvements in retrieval effectiveness consistently observed.

There are three notable advantages of the algorithms reviewed: (i) they can be integrated with a wide range of existing hashing models beyond LSH; (ii) they support extension to cross-modal retrieval tasks, such as retrieving images using textual queries; and (iii) they can be combined synergistically, yielding further gains in performance.

3. Survey Overview

This survey begins by motivating the problem of efficient similarity search in high-dimensional data spaces, highlighting its relevance in large-scale information retrieval and machine learning. The introductory section outlines the limitations of data-oblivious methods such as Locality-Sensitive Hashing (LSH), particularly in terms of neighbourhood preservation and retrieval effectiveness.

The article then presents a detailed review of projection and quantisation functions as they relate to the construction of similarity-preserving hashcodes. This includes an examination of common assumptions made by existing methods—such as the use of static thresholds, uniform threshold allocation, and random projections—and motivates the need for data-dependent alternatives.

Following this, the survey introduces and analyses a series of data-driven hashing approaches aimed at relaxing these limiting assumptions:

- Techniques for learning multiple quantisation thresholds per projection to improve alignment between hashcodes and data structure.

- Adaptive strategies for allocating variable numbers of bits to more informative projections, enhancing code discriminativeness.

- Techniques for learning the hashing hyperplanes in a data-adaptive manner with extensions to cross-modal retrieval tasks.

In summarising these contributions, the article provides a structured synthesis of the field and highlights promising directions for future research in adaptive, learned hashing methods for approximate nearest neighbour search.

4. Background into Hashing-Based Approximate Nearest Neighbour Search

This section provides a structured overview of prior research relevant to hashing-based approximate nearest neighbour (ANN) search. We begin by introducing the

standard nearest neighbour (NN) search problem, its computational challenges, and the motivation for adopting approximate variants. A seminal method in this space, Locality-Sensitive Hashing (LSH), is then described as a foundational approach to ANN search with sublinear query complexity.

The limitations of LSH, particularly in its data-oblivious design and simplistic assumptions, motivate a discussion of more recent, data-driven approaches that aim to improve retrieval effectiveness. The survey is structured around the two main components of the hashcode generation pipeline: (i) binary quantisation and (ii) projection function learning. This division mirrors the practical architecture of most hashing algorithms and guides the organisation of the remainder of this section.

4.1 Preliminaries and Notation

We adopt the following notational conventions. Vectors are denoted by bold lowercase symbols (e.g., $\mathbf{x}$) and matrices by bold uppercase symbols (e.g., $\mathbf{X}$). The (i, j)-th entry of matrix $\mathbf{X}$ is written as X_{ij}. A vector $\mathbf{x} = [x_1, x_2, \ldots, x_N]^\mathsf{T}$ is assumed to be a column vector, and a matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\mathsf{T} \in \mathbb{R}^{N \times D}$ represents N data points in D-dimensional space. The c-th column and r-th row of $\mathbf{X}$ are denoted by $\mathbf{x}^c = \mathbf{X}_{\bullet c}$ and $\mathbf{x}_r = \mathbf{X}_{r\bullet}$, respectively. Scalar-valued functions are represented by regular lowercase letters, e.g., $d(\cdot, \cdot)$.

Given a dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$, consisting of N points $\mathbf{x}_i \in \mathbb{R}^D$, the goal is to learn a set of K binary hash functions $\{h_k : \mathbb{R}^D \to \{0, 1\}\}_{k=1}^K$. The outputs of these functions are concatenated to form a binary embedding function $g : \mathbb{R}^D \to \{0, 1\}^K$ such that $g(\mathbf{x}_i) = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_K(\mathbf{x}_i)] = \mathbf{b}_i$. For effective ANN search, the resulting hashcodes $\mathbf{b}_i$ and $\mathbf{b}_j$ should be similar (e.g., under Hamming distance) when the corresponding input vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar in the original space.

The remainder of this section reviews how similarity-preserving hash functions have been constructed in the literature, with a focus on their respective approaches to quantisation and projection. An overview of the organisational structure of this review is illustrated in Figure 4.

While the hashing models discussed here are primarily evaluated on image datasets, their design principles are generally applicable to any data that can be represented as fixed-length vectors, including text and audio. That said, model performance may vary by data modality; for example, eigen-decomposition methods often excel on low-dimensional visual features but may face scalability challenges on high-dimensional sparse text data. Understanding the behaviour of these models across different data types remains a promising direction for future investigation.

4.2 Approximate Nearest Neighbour (ANN) Search

This section formally introduces the nearest neighbour (NN) search problem and its widely adopted approximation, which forms the foundation of much of the recent work in similarity search.
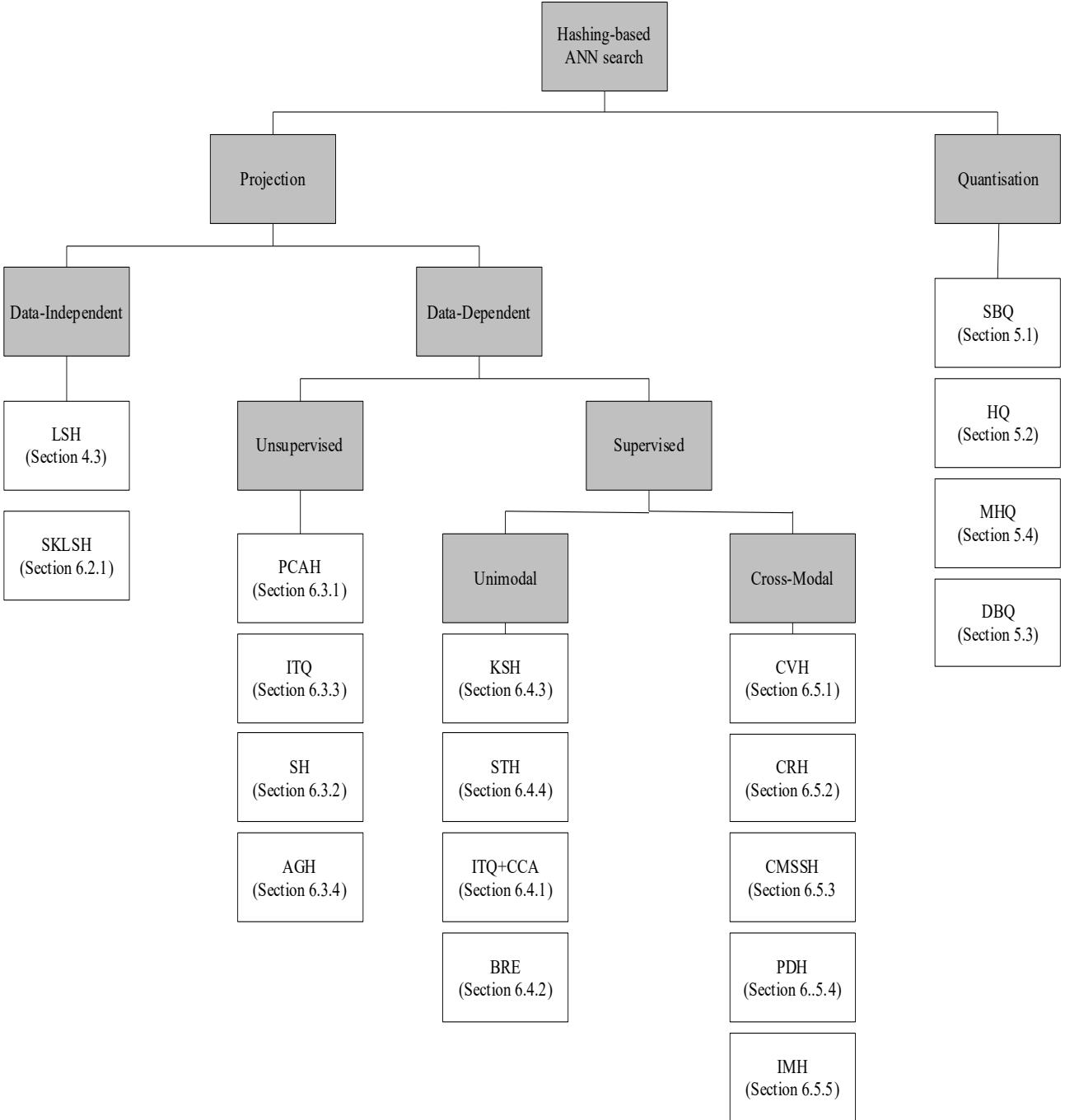
Figure 4: Overview of one possible categorisation of the field of hashing-based ANN search. The main categories are shown in grey, while specific models are listed in white alongside their corresponding section numbers.

The classic 1-NN problem is defined as finding the data point $NN(\mathbf{q})$ that is closest to a given query $\mathbf{q} \in \mathbb{R}^D$ within a database $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\mathsf{T}$, where each $\mathbf{x}_i \in \mathbb{R}^D$. Similarity is typically quantified using a distance function $d : \mathbb{R}^D \times \mathbb{R}^D \to [0, 1]$, and the retrieval task is specified as:

$$NN(\mathbf{q}) = \arg \min_{\mathbf{x}_i \in \mathbf{X}} d(\mathbf{x}_i, \mathbf{q}) \tag{1}$$

This can be extended to the k-NN problem, where the k closest neighbours to $\mathbf{q}$ are retrieved. Distance functions used in practice include the l_p-norm:

$$d_{pnorm}(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^{D} |x_{ik} - x_{jk}|^\rho \right)^{1/\rho} \tag{2}$$

where $\rho \in \mathbb{R}_+$, with common settings such as $\rho = 1$ (Manhattan), $\rho = 2$ (Euclidean), and $\rho < 1$ yielding Minkowski distances. Another frequently used metric is the cosine distance:

$$d_{cosine}(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{\sum_{k=1}^{D} x_{ik} x_{jk}}{\sqrt{\sum_{k=1}^{D} x_{ik}^2} \sqrt{\sum_{k=1}^{D} x_{jk}^2}} \tag{3}$$

In the context of binary hashcodes, Hamming distance is the standard metric:

$$d_{hamming}(\mathbf{b}_i, \mathbf{b}_j) = \sum_{k=1}^{D} \delta[b_{ik} \neq b_{jk}] \tag{4}$$

where $\delta(\cdot) = 1$ if the condition is true, and 0 otherwise. Hamming distance simply counts the number of differing bits between two binary vectors.

These generic distance functions are data-agnostic and assume that distances meaningfully capture semantic similarity across all domains. This is often an unrealistic expectation—particularly in high-dimensional spaces—leading to suboptimal retrieval performance. Metric learning methods attempt to overcome this by adapting the distance function to the specific characteristics of the dataset, often improving nearest neighbour quality significantly [46]. This notion of data-adaptive distance learning underpins many of the recent advances in hashing-based ANN search, particularly in the design of projection functions (see Section 6).

The simplest algorithm for nearest neighbour retrieval is brute-force search, which compares the query against all N database points in $\mathcal{O}(ND)$ time. While exact and easy to implement, this approach becomes computationally prohibitive for large datasets or high-dimensional features. Traditional spatial indexing structures such as KD-trees [7], quad-trees [22], X-trees [8], and SR-trees [39] offer efficiency gains in low-dimensional settings. However, in high-dimensional spaces these structures degrade to linear time, due to the well-known *curse of dimensionality* [56, 90].
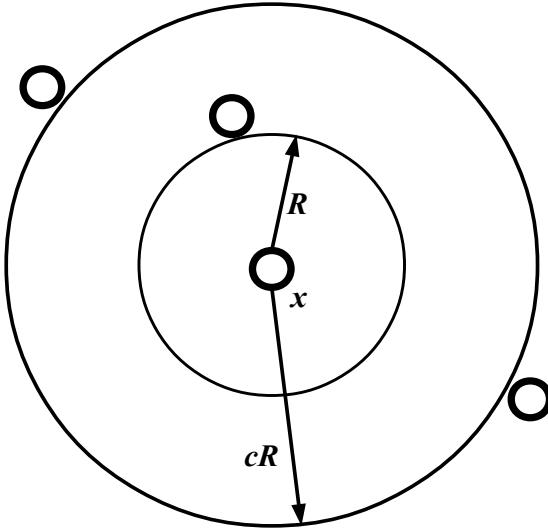
Figure 5: The (c, R)-approximate NN problem: in many applications it is acceptable to retrieve a data point (circle) within distance cR of the query point $\mathbf{x}$, where R is the distance to the exact NN.

Approximate Nearest Neighbour (ANN) search addresses this issue by relaxing the requirement for exact results. Instead, the goal is to retrieve neighbours that are sufficiently close to the query, accepting a trade-off between accuracy and efficiency. This relaxation is formalised in the (c, R)-approximate nearest neighbour problem:

Definition 4.1 (Randomised c-approximate R-near neighbour problem). *Given a set of N data points in a D-dimensional space, return each cR-nearest neighbour of the query point $\boldsymbol{q}$ with probability at least $1 - \delta$, where $\delta > 0$ and $c > 1$.*

The approximation factor c controls how close a retrieved neighbour must be to the optimal one. Larger values of c relax the requirement and can substantially improve query efficiency.

A related problem is the R-near neighbour search, which omits the approximation factor:

Definition 4.2 (Randomised R-near neighbour problem). *Given a set of N data points in a D-dimensional space, return each R-nearest neighbour of the query point $\boldsymbol{q}$ with probability at least $1 - \delta$, where $\delta > 0$ and $R > 0$.*

Subsequent sections introduce Locality-Sensitive Hashing (LSH), a widely studied algorithmic framework that enables efficient solutions to these ANN decision problems, particularly in high-dimensional settings.

4.3 Locality-Sensitive Hashing (LSH)

A key objective in hashing-based approximate nearest neighbour (ANN) search is to pre-process the dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ such that query-time retrieval is significantly faster

than brute-force search. Locality-Sensitive Hashing (LSH) [36] is among the most influential algorithms in this domain, offering the first sub-linear time solution to the randomised (c, R)-approximate near neighbour problem. It has been widely applied across computer vision and retrieval tasks, including large-scale object recognition [17], pose estimation [76], image indexing [15], and shape matching [28].

LSH operates by hashing input vectors into buckets such that similar points are more likely to fall into the same bucket. This transforms the nearest neighbour search problem into the task of probing a small number of candidate buckets, greatly improving efficiency. The central idea is to use a family of hash functions with the locality-sensitive property, which ensures that closer points collide with higher probability than distant ones.

Definition 4.3 (Locality-sensitive hash function family). *A hash function family $\mathcal{H}$ is (R, cR, P_1, P_2)-sensitive with respect to a distance function $d(\cdot, \cdot)$ if, for any two points $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^D$:*

$$if\ d(\boldsymbol{p}, \boldsymbol{q}) \leq R \Rightarrow \Pr_{\mathcal{H}}[h(\boldsymbol{p}) = h(\boldsymbol{q})] \geq P_1$$

$$if\ d(\boldsymbol{p}, \boldsymbol{q}) \geq cR \Rightarrow \Pr_{\mathcal{H}}[h(\boldsymbol{p}) = h(\boldsymbol{q})] \leq P_2$$

Here, $P_1 > P_2$ and $c > 1$ are required to make the hash family useful for ANN search. That is, close pairs should collide more often than distant pairs. Hash function families with this property have been developed for several distance measures, including L_p norms [16], cosine similarity [13], Jaccard similarity [11], and angular distances on hyperspheres [82].

The probability gap between P_1 and P_2 can be amplified by concatenating K independently sampled hash functions $h_k \in \mathcal{H}$ into a composite function:

$$g(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_K(\mathbf{x})] \tag{5}$$

This embedding maps each point to a K-bit hashcode, significantly reducing the chance that distant points collide. The composite function g is drawn from a higher-order family $\mathcal{G}$ and can be used to index data into 2^K buckets per hashtable.

To maintain recall while increasing precision, multiple such hashtables (indexed by L independent g_l functions) are used. The collision probability for at least one of the L tables is given by $P_1'' = 1 - (1 - P_1^K)^L$, which can be tuned to ensure desirable trade-offs between recall, precision, and query time. Figure 6 illustrates the interaction between these parameters.

The pre-processing step (Algorithm 1) indexes data points into multiple hashtables. During querying, the query point is hashed with the same g_l functions, and candidate neighbours are drawn from matching buckets:

Different query strategies support both the (c, R)-approximate and the R-near neighbour decision problems. The former is typically addressed by early stopping

---

Algorithm 1: LSH Pre-processing

Input: Data $\mathbf{X} \in \mathbb{R}^{N \times D}$, L embedding functions g_l, each composed of K hash functions from family $\mathcal{H}$
Output: L hashtables $H_1, \ldots, H_L$ with indexed data

1 **for** $l \leftarrow 1$ **to** L **do**
2   **for** $j \leftarrow 1$ **to** N **do**
3   Insert $\mathbf{x}_j$ into bucket $H_l[g_l(\mathbf{x}_j)]$
4   **end**
5 **end**
6 **return** $H_1, \ldots, H_L$

Algorithm 2: LSH Querying Step

Input: Query $\mathbf{q} \in \mathbb{R}^D$, hashtables $H_1, \ldots, H_L$, embedding functions g_l
Output: Set S of approximate nearest neighbours

1 **for** $l \leftarrow 1$ **to** L **do**
2   **for** *each* $\boldsymbol{x}_j$ *in* $H_l[g_l(\boldsymbol{q})]$ **do**
3   **if** $d(\boldsymbol{x}_j, \boldsymbol{q}) < R$ **then**
4   Add $\mathbf{x}_j$ to S
5   **end**
6   **end**
7 **end**
8 **return** S

(e.g., after retrieving $3L$ candidates), while the latter exhaustively examines all collisions.

The effectiveness of an LSH system depends heavily on the chosen hash function family, as well as on parameters K and L, which govern the precision/recall trade-off. Guidelines for parameter selection, including analytical bounds and empirical tuning heuristics, are well-documented in the literature [63, 62, 67]. LSH remains a foundational approach in modern ANN search and provides the basis for numerous extensions discussed in subsequent sections.

4.4 LSH with Sign Random Projections

A particularly important instance of locality-sensitive hashing is the use of random hyperplane projections for measuring inner product or cosine similarity. This hash function family, denoted $\mathcal{H}_{cosine}$, is frequently used as a baseline in the learning-to-hash literature.

The inner product similarity is defined as:

$$d(\mathbf{p}, \mathbf{q}) = \sum_{k=1}^{D} p_k q_k = \mathbf{p}^{\mathsf{T}}\mathbf{q} \qquad (6)$$

When vectors are L_2-normalised to lie on the unit sphere, this expression becomes equivalent to cosine similarity, which measures the angle between two data vectors. As this angle increases, similarity decreases, and the probability of hash collision under $\mathcal{H}_{cosine}$ is reduced accordingly.

In this scheme, K random hyperplanes are sampled from a Gaussian distribution with zero mean and unit variance. Each hash function h_k assigns a bit to a data point $\mathbf{q}$ depending on which side of the hyperplane it lies:

$$h_k(\mathbf{q}) = \frac{1}{2}(1 + \mathrm{sgn}(\mathbf{w}_k^{\mathsf{T}}\mathbf{q})) \qquad (7)$$

Here, $\mathbf{w}_k$ is the normal vector of the k-th hyperplane, and sgn is the sign function (with $\mathrm{sgn}(0) = -1$). This hash function can be viewed as a composition of two components: a projection function $p_k : \mathbb{R}^D \to \mathbb{R}$ that computes the dot product, and a quantisation function $q_k : \mathbb{R} \to \{0, 1\}$ that thresholds at zero.

The key property of $\mathcal{H}_{cosine}$ is that the probability of a collision (i.e., $h(\mathbf{p}) = h(\mathbf{q})$) is proportional to the angle θ between the vectors:

$$\Pr_{\mathcal{H}_{cosine}}(h(\mathbf{p}) = h(\mathbf{q})) = 1 - \frac{\theta(\mathbf{p}, \mathbf{q})}{\pi} \qquad (8)$$

This angular sensitivity makes $\mathcal{H}_{cosine}$ an effective choice for cosine-similarity-based ANN search. The amplification strategy discussed in Section 4.3 applies here as well: concatenating multiple hash functions (K) and using multiple hashtables (L) increases the discriminative power and collision selectivity of the system.

The computational complexity of querying using this hash family is given by:

$$\mathcal{O}(KDL) + \mathcal{O}(1) + \mathcal{O}\left(\frac{NDL}{2^K}\right)$$

The first term corresponds to hashcode computation, the second to bucket lookup, and the third to distance comparisons with candidate points. Increasing K reduces the number of candidates by creating more selective buckets, but increases hashing cost. Increasing L improves recall by spreading the query across more hashtables but increases total runtime.

This framework forms the foundation for many modern hashing-based methods. However, while effective, the random projection and thresholding design of LSH is inherently data-independent and may lead to inefficiencies. In particular, it often requires many bits and multiple hashtables to achieve adequate performance. Recent advances in the literature seek to address these limitations by learning data-dependent projection and quantisation functions. These data-driven approaches are reviewed in the following sections.
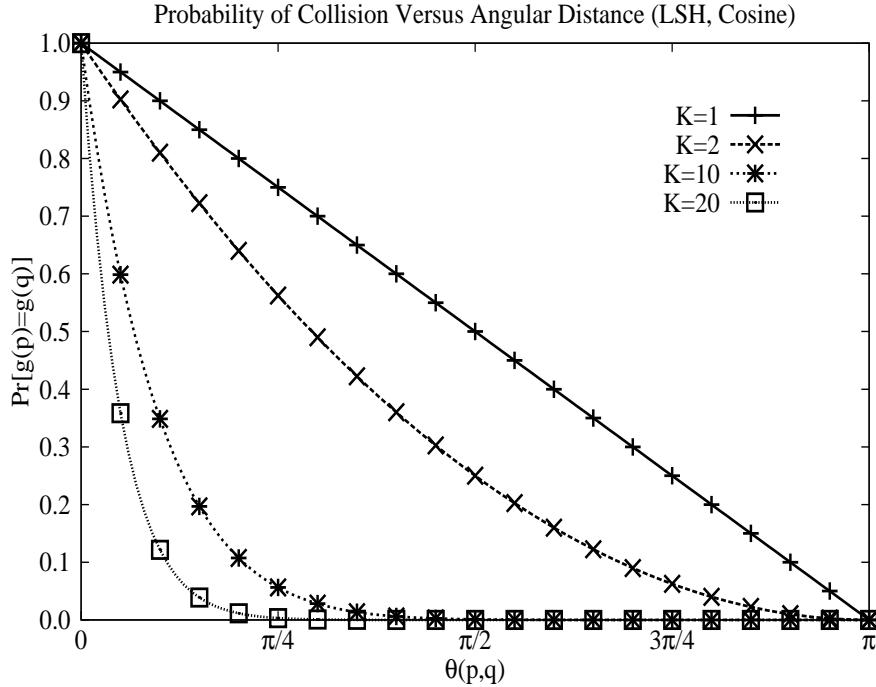
Probability of Collision Versus Angular Distance (LSH, Cosine)

Figure 6: Probability of two hashcodes $g(\mathbf{p})$ and $g(\mathbf{q})$ matching as a function of hashcode length K, using the $\mathcal{H}_{cosine}$ hash family. As K increases, collision probability declines sharply for vectors with larger angular separation. Adapted from [63].

5. Quantisation for Nearest Neighbour Search

In this section, we review prior research on binary quantisation methods for hashing-based approximate nearest neighbour (ANN) search. As discussed in Section 4.4, one of the key steps in generating locality-sensitive hashing (LSH) codes is the conversion of real-valued projections into binary bits. Our focus here is on algorithms that aim to reduce the information loss introduced by discretising continuous values into binary form, with particular attention to approaches that go beyond the simple sign function in Equation 7. We will clarify later in this section what constitutes an improvement over this baseline. Broadly, quantisation refers to the process of mapping a representation with infinite cardinality (such as real numbers) to a finite discrete set (e.g., binary codes). Quantisation has been extensively studied in information theory [31] and widely applied in engineering, motivated by the practical impossibility of storing and manipulating values at infinite precision. The present review is restricted to quantisation methods developed specifically for hashing-based ANN search.

Two primary categories of quantisation have been proposed for nearest neighbour search: scalar quantisation and vector quantisation, distinguished by whether the input and output of the quantiser are scalar or vector quantities. Scalar quantisation is commonly applied to the real-valued projections obtained by computing the dot product between a data-point's feature vector and the normal vectors of a set of
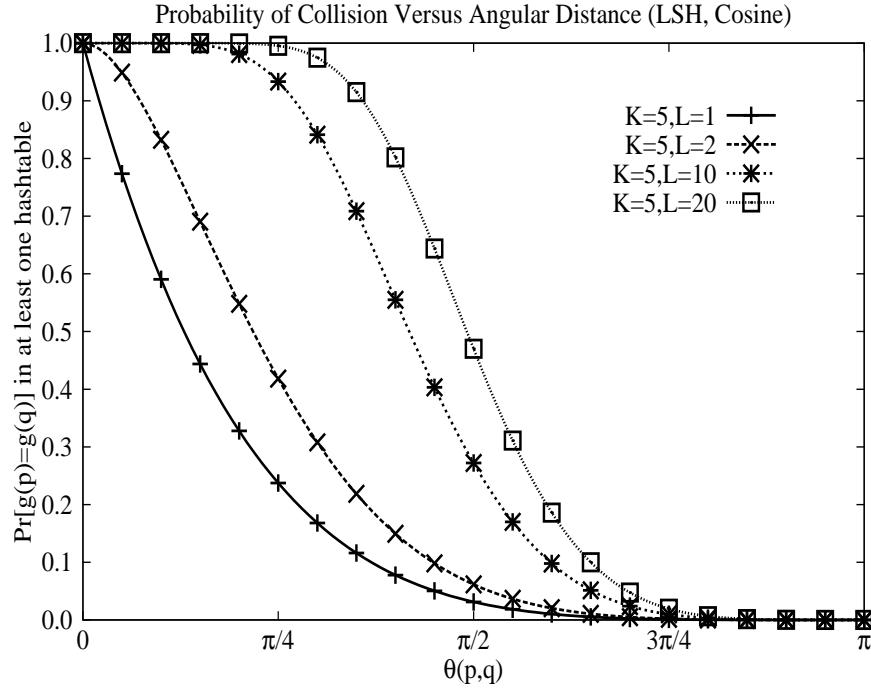
17

Figure 7: Effect of increasing the number of hashtables L on the collision probability under $\mathcal{H}_{cosine}$ with fixed hashcode length $K = 5$. Higher values of L increase the likelihood that nearby points collide in at least one bucket. Adapted from [63].

random hyperplanes partitioning the feature space (Figure 8). As discussed further in Section 5.1, each dot product produces a scalar value, which is then thresholded to yield a binary code (0/1).

In contrast, vector quantisation assigns each data-point to the nearest centroid from a codebook discovered, for example, via the k-means algorithm [54]. Each input vector is thus represented by one of a much smaller set of centroids, with k-means dividing the space into Voronoi regions and providing a more flexible, data-driven partitioning (see Figure 8). While vector quantisation will not be discussed in detail in this article, it is worth noting that it has often been found more effective in computer vision tasks due to its lower reconstruction error [37]. Its principal drawback is the need to store a lookup table at query time to retrieve inter-cluster Euclidean distances from centroid indices. This decoding step introduces additional computational overhead: distance computation with vector quantisation has been reported to be 10–20 times slower than Hamming distance comparisons of binary hash codes on standard datasets [34].

Scalar quantisation avoids this overhead, as distances can be computed directly from binary codes, a property that has proven advantageous in applications such as mobile product search [21]. A further advantage of hyperplane-based scalar quantisation is its exponential partitioning power: K hyperplanes induce 2^K distinct regions,

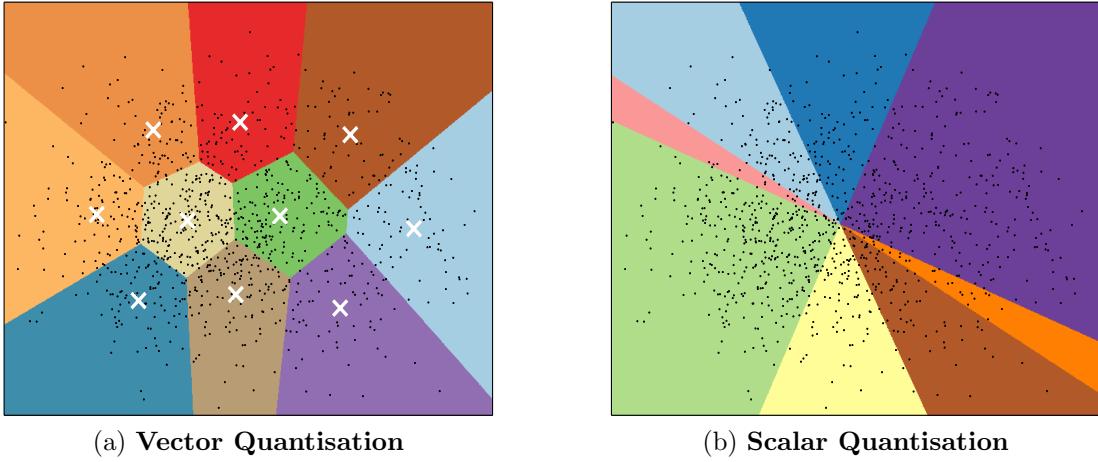(a) **Vector Quantisation**   (b) **Scalar Quantisation**

Figure 8: In the context of nearest neighbour search two variations on quantisation are typically employed: vector quantisation (Figure (a)) partitions the feature space into Voronoi cells ([37]). Centroids are marked with a white cross while data-points are shown as black dots. The distance between query and database points is computed by determining the distance to their closest centroids. In contrast, scalar quantisation (Figure (b)) is frequently used to binarise a real-valued projection resulting from a dot product of a data-point with a hyperplane normal vector. The space is partitioned with multiple such hyperplanes and each usually contribute 1-bit to the final hashcode. The hashcode is effectively the index of the polytope-shaped region containing the associated data-point. The data-points appearing in this example are a 2D PCA projection of the CIFAR-10 image dataset.

Figure 9: Illustration of single-bit quantisation (SBQ). A threshold $t_1 \in \mathbb{R}$ is applied to a projected dimension: values below the threshold (left, coloured shapes) are assigned a bit '0', while values above the threshold (right) are assigned a bit '1'.

whereas vector quantisation would require 2^K centroids to achieve comparable granularity. Efficient methods, such as product quantisation, have been proposed to scale vector quantisation to this regime [37]. More recently, researchers have begun exploring approaches that unify the strengths of scalar and vector quantisation, and the reader is referred to [34] and references therein for a detailed survey of this line of work.

In the context of hashing-based ANN search, a scalar quantiser

$$q_k : \mathbb{R} \to \{0,1\}^B$$

maps a real-valued projection $y_{we} \in \mathbb{R}$ to either a single-bit (Section 5.1) or multi-bit (Sections 5.3–5.4) binary codeword

$$\mathbf{c}_{we} \in \mathcal{C}, \quad \mathbf{c}_{we} \in \{0,1\}^B, \quad we \in \{1,2,\ldots,T+1\},$$

where T denotes the number of quantisation thresholds, B the number of bits per projected dimension, and $\mathcal{C}$ the binary codebook.

Following [44], we define a *projected dimension* $\mathbf{y}^k \in \mathbb{R}^{N_{\mathrm{trd}}}$ as the set of real-valued projections

$$\{y^k_{we} \in \mathbb{R}\}^{N_{\mathrm{trd}}}_{we=1}$$

obtained from all data-points $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_{\mathrm{trd}}}]$ for a given hyperplane $\mathbf{h}_k$, where each projection is computed as

$$y^k_{we} = \mathbf{w}^\mathsf{T}_k \mathbf{x}_i.$$

The quantisation function q_k binarises each projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{\mathrm{trd}}}$ independently by placing one or more thresholds along the dimension. Projected values that fall into a given thresholded region are assigned the corresponding codeword.

A simple example is shown in Figure 9, where the projected dimension is represented as a line with sample data-points (coloured shapes). In this illustration, a single threshold partitions the dimension into two disjoint regions: projections below the threshold are assigned the codeword '0', while projections above the threshold are assigned the codeword '1'. The corresponding codebook is

$$\mathcal{C} = \{\mathbf{c}_{we} \in \{0,1\} \mid we \in \{1,2\}\}.$$

Formally, we denote the set of threshold positions along a projected dimension as

$$\boldsymbol{\tau}_k = [t_1, t_2, \ldots, t_T], \quad t_{we} \in \mathbb{R}, \quad t_1 \leq t_2 \leq \cdots \leq t_T,$$

with extremities defined as $t_0 = -\infty$ and $t_{T+1} = +\infty$. These thresholds partition the dimension into $T+1$ disjoint regions

$$\mathcal{R}_{we} = \{y_j \mid t_{we-1} < y_j \leq t_{we}, y_j \in \mathbf{y}^k\}, \quad we \in \{1, \ldots, T+1\}.$$

Most scalar quantisation schemes adopt $T = 2^B - 1$ thresholds for a budget of B bits per projected dimension. Each of the $T+1$ regions $\{\mathcal{R}_{we} \subset \mathbf{y}^k\}^{T+1}_{we=1}$ is then associated with a unique codeword $\mathbf{c}_{we} \in \mathcal{C}$.

The retrieval effectiveness of quantisation is strongly influenced by both the choice of codebook and the positioning of quantisation thresholds [58, 44, 42]. An effective encoding scheme must preserve the relative distances between data-points in the input space when mapped to binary hashcodes. For instance, data-points that are far

Method	Encoding	Optimisation	Thresholds (T)	Complexity	Section
SBQ	0/1	Mean thresholding	1	$\mathcal{O}(1)$	5.1
HQ	00/01/10/11	Spectral partitioning	1 and 2	$\mathcal{O}(CN_{trd}^{+})$	5.2
DBQ	00/10/11	Squared error	2	$\mathcal{O}(N_{trd}\log N_{trd})$	5.3
MHQ	NBC	1D K-means	3+	$\mathcal{O}(2^{B}N_{trd})$	5.4

Table 1: Existing single (SBQ) and multi-threshold (HQ, DBQ, MHQ) quantisation schemes categorised along the three main dimensions of variability. NBC stands for Natural Binary Encoding and is explained in Section 5.4. C is the number of anchor points, N_{trd} is the number of training data-points, N_{trd}^{+} is the number of training data-points with positive projected value for the given projected dimension and B is the number of bits per projected dimension. Time complexity is for positioning thresholds along a single projected dimension.

apart in the original feature space should also be distant in Hamming space, while nearby data-points should be assigned similar codes. Ideally, the encoding of thresholded regions should yield a smooth, monotonic increase in Hamming distance as the original-space distance grows. Threshold placement is equally critical: a poorly positioned threshold that bisects a region dense in true nearest neighbours will scatter related data-points across different regions, inflating their Hamming distance. Suboptimal encoding or thresholding can therefore cause semantically related data-points to be assigned dissimilar codes, severely degrading retrieval accuracy. The state-of-the-art quantisation algorithms reviewed in this section address these issues by jointly proposing encoding schemes and threshold optimisation strategies designed to preserve relative neighbourhood structure.

From the above discussion, three properties emerge as central for distinguishing and categorising scalar quantisation methods[6]: (i) the encoding scheme used to assign symbols to thresholded regions; (ii) the strategy employed to determine threshold positions, including whether a learning procedure is used for optimisation; and (iii) the number of thresholds allocated per dimension. Table 1 summarises existing quantisation methods along these three axes of variability. In the following sections we provide detailed descriptions: Section 5.1 introduces the traditional Single-Bit Quantisation (SBQ) approach, while Sections 5.2–5.4 present more recent multi-threshold methods, including Hierarchical Quantisation (HQ), Double-Bit Quantisation (DBQ), and Manhattan Hashing Quantisation (MHQ).

5.1 Single Bit Quantisation (SBQ)

Single-Bit Quantisation (SBQ) is the binarisation strategy employed in the majority of existing hashing methods. A single threshold t_k partitions a projected dimension $\mathbf{y}^k$

---

6. Throughout the remainder of this article, the term *quantisation* refers specifically to scalar quantisation.
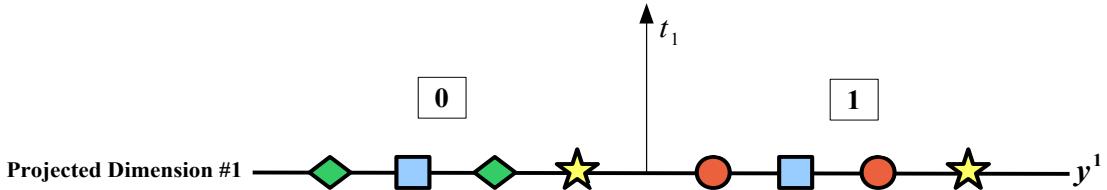
Figure 10: Illustration of Single-Bit Quantisation (SBQ). A single threshold t_1 partitions a projected dimension: values below the threshold (left, coloured shapes) are assigned the bit '0', while values above the threshold (right) are assigned the bit '1'.

into two regions: projected values below the threshold are assigned the bit '0', while values equal to or above the threshold are assigned the bit '1'. Formally, given a set of K hyperplane normal vectors $[\mathbf{w}_1, \ldots, \mathbf{w}_K]$, the kth hashcode bit for a data-point $\mathbf{x}_i$ is generated by SBQ as shown in Equation 9.

$$h_k(\mathbf{x}_{we}) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)) \tag{9}$$

In this quantisation scheme, each hyperplane contributes one bit to the hashcode of a data-point. The data is typically assumed to be *zero-centred*, and the projected dimensions are thresholded at the mean,

$$t_k = \frac{1}{N_{\text{trd}}} \sum_{we=1}^{N_{\text{trd}}} \mathbf{w}_k^\intercal \mathbf{x}_{we}.$$

For zero-centred data this reduces to placing the threshold at the origin ($t_k = 0$). No learning mechanism is employed to optimise threshold placement in SBQ, although in some cases the threshold may be set at the median rather than the mean of the projected values. Owing to the absence of threshold optimisation, SBQ is computationally inexpensive, requiring $\mathcal{O}(1)$ time for threshold computation[7] and $\mathcal{O}(1)$ time for encoding a novel query. A simple illustration of SBQ is provided in Figure 10.

The multi-threshold quantisation algorithms described in Sections 5.2–5.4 are designed to address a fundamental limitation of SBQ arising from the use of a single threshold for binarisation. In particular, SBQ may assign different bits to data-points that are located very close together along a projected dimension, while assigning identical bits to points that are much farther apart [44, 42, 58, 59]. This behaviour is at odds with the central objective of hashing, namely that nearby data-points in the original feature space should be assigned similar codes. Consequently, this limitation of SBQ can reduce retrieval effectiveness.

The problem is illustrated in Figure 11. Consider the data-points a and b (yellow stars), which lie close to one another in the projected space but fall on opposite sides of the threshold. Despite their proximity, SBQ assigns them different bits,

---

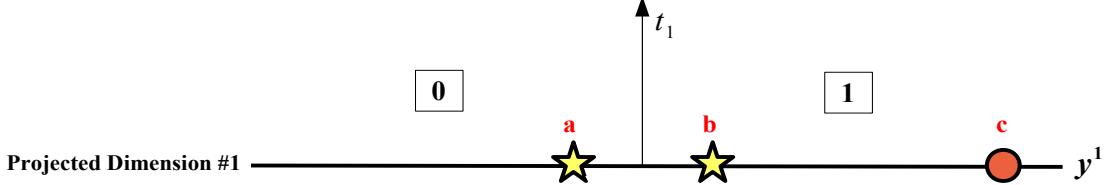7. This increases to $\mathcal{O}(N_{\text{trd}})$ if the median is used.

Figure 11: Illustration of the limitation of Single-Bit Quantisation (SBQ). True nearest neighbours such as points a and b are assigned different bits despite being close together along the projected dimension. Conversely, points b and c, which are not nearest neighbours, both lie above the threshold and are assigned the same bit ('1'), even though they are more widely separated in the projected space.

thereby increasing their Hamming distance. The hash function has, by projecting them nearby, already indicated that a and b are likely to be close in the original feature space[8]. Yet SBQ disrupts this structure by assigning opposing codes. Conversely, points b (yellow star) and c (red circle) lie far apart in the projected space—implying they are distant in the original feature space—but are both assigned the same bit, thereby bringing their codes artificially close in Hamming space.

This limitation of SBQ is particularly pronounced in practice because the vanilla scheme places the threshold at zero, where the density of projected values is often highest. This distributional pattern holds for many projection functions commonly used in practice (Figure 12). A natural remedy is to partition each projected dimension into multiple regions and assign a multi-bit encoding, thereby reducing the likelihood of separating true neighbours across a single threshold. The core principle underlying all multi-threshold quantisation schemes is the preservation of neighbourhood structure through a combination of multi-bit codebooks and threshold optimisation. We now turn to one of the earliest multi-threshold schemes, introduced in Section 5.2.
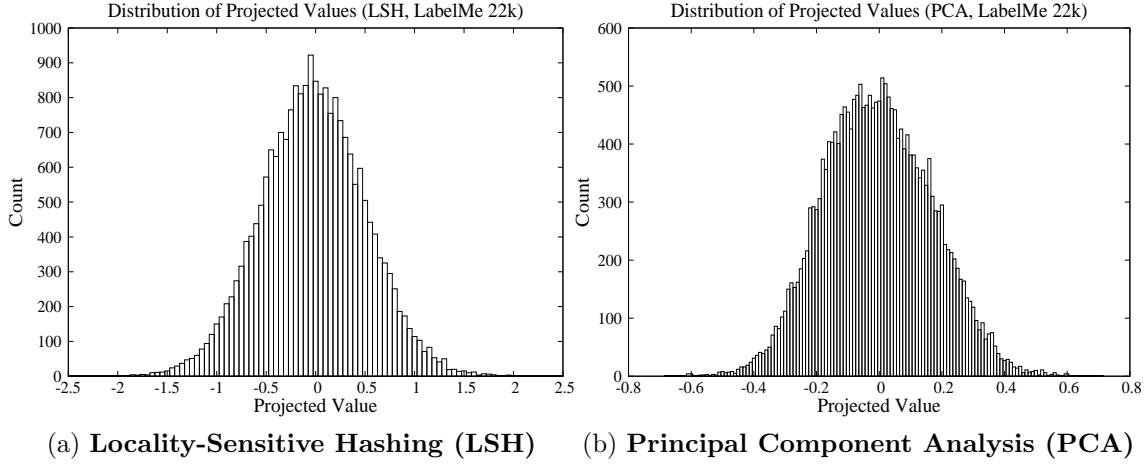
5.2 Hierarchical Quantisation (HQ)

[52] were the first to introduce the concept of multi-threshold quantisation for hashing, in which a single hyperplane contributes multiple bits to the hashcode. Their algorithm, termed Hierarchical Quantisation (HQ), was proposed as a means of quantising the projections obtained from the Anchor Graph Hashing (AGH) model. HQ employs only $\lfloor K/2 \rfloor$ of the available hyperplanes, with each selected hyperplane contributing two bits. The projection learning component of AGH is described in Section 6.3.4; here, we focus exclusively on the quantisation procedure, assuming that the projected dimensions

$$\{\mathbf{y}^k \in \mathbb{R}^{N_{\mathrm{trd}}}\}_{k=1}^K$$

---

8. This assumes that the projection function preserves neighbourhood structure. Randomised LSH guarantees this in expectation, while data-dependent projection functions explicitly learn hyperplanes to encourage similar projections for related points; see Section 6.

(a) **Locality-Sensitive Hashing (LSH)**   (b) **Principal Component Analysis (PCA)**

Figure 12: Distribution of projected values for two randomly chosen projected dimensions: (a) LSH and (b) PCA, evaluated on the LabelMe 22k image dataset [83], where images are represented as GIST features [61]. In both cases, the highest density of projected values occurs near zero. The Double-Bit Quantisation (DBQ) algorithm [42] explicitly avoids placing thresholds near zero, since doing so would separate many true nearest neighbours. DBQ is described in Section 5.3.

have already been obtained.

HQ consists of two sequential steps. In the first step, traditional SBQ (Section 5.1) is applied, thresholding the projected dimension $\mathbf{y}^k$ at zero ($t_1 = 0$). This produces the first bit of the two-bit encoding for each hyperplane. In the second step, the same projected dimension is further quantised using two additional thresholds, t_2 and t_3, which subdivide the regions produced in step one.

The thresholds t_2 and t_3 are jointly optimised to minimise the number of related data-points that are separated by the initial SBQ threshold. Both steps are illustrated in Figure 13. Liu et al. [52] formulate this optimisation as a graph-partitioning problem on the graph Laplacian

$$\mathbf{L} = \mathbf{D} - \hat{\mathbf{S}},$$

where $\hat{\mathbf{S}}$ is a low-rank *approximate* adjacency matrix and $\mathbf{D}$ is the corresponding degree matrix. The neighbourhood structure is encoded by $\hat{\mathbf{S}}$, with $\hat{S}_{ij} > 0$ indicating that points i and j are neighbours, and $\hat{S}_{ij} = 0$ otherwise. The matrix is approximate because it is not constructed by explicitly computing all N_{trd}^2 pairwise distances among the N_{trd} data-points. Instead, it is derived from an anchor graph $\mathbf{Z}$—a sparse matrix capturing similarities between the N_{trd} data-points and a much smaller set of C anchor points ($C \ll N_{\text{trd}}$), as defined in Equation 10.

$$
Z_{ij} =
\begin{cases}
\dfrac{exp(-d^2(\mathbf{x}_j, \mathbf{c}_{we})/\gamma)}{\displaystyle\sum_{we' \in \langle j \rangle} exp(-d^2(\mathbf{x}_j, \mathbf{c}_{we'}))/\gamma)} & \text{if } we \in \langle j \rangle \\[3ex]
0 & otherwise
\end{cases}
\tag{10}
$$

Here γ denotes the kernel bandwidth, $\{d(\cdot, \cdot) : \mathbb{R}^D \times \mathbb{R}^D \to [0, 1]\}$ is a distance function of interest (e.g., the L_2-norm), and $\langle j \rangle \in \{1, \dots, R\}$ are the indices of the $R \ll C$ nearest anchors to $\mathbf{x}_j$ under distance metric $d(\cdot, \cdot)$. The C anchor points $\{\mathbf{c}_{we} \in \mathbb{R}^D\}_{we=1}^C$ are obtained by running the k-means algorithm on the training data.

Using the anchor graph, the adjacency matrix is approximated as

$$
\hat{\mathbf{S}} = \mathbf{Z}\Lambda^{-1}\mathbf{Z}^\mathsf{T}, \quad \Lambda_{kk} = \sum_{i=1}^{N_{\text{trd}}} Z_{ik}.
$$

This construction approximates the affinity between two data-points $\mathbf{x}_i$ and $\mathbf{x}_j$ as the inner product of their individual affinities to the C centroids. Compared with the full adjacency matrix, $\hat{\mathbf{S}}$ is both sparse and low-rank, which offers computational advantages when extracting the graph Laplacian eigenvectors (Section 6.3.4).

Liu et al. [52] formulate an eigenvalue problem involving $\mathbf{Z}$ to compute the K graph Laplacian eigenvectors $\mathbf{y}^k$ of the approximate adjacency matrix $\hat{\mathbf{S}}$. Within AGH, these eigenvectors serve as the projected dimensions that are subsequently thresholded to generate the hashcodes for the training data-points (Equation 11).

$$
h_k(\mathbf{x}_{we}) =
\begin{cases}
\frac{1}{2}(1 + sgn(\mathbf{w}_k^\mathsf{T}\mathbf{x}_{we} - t_2)), & \text{if} \quad \mathbf{w}_k^\mathsf{T}\mathbf{x}_{we} \geq 0 \\
\frac{1}{2}(1 + sgn(-\mathbf{w}_k^\mathsf{T}\mathbf{x}_{we} + t_3)), & \text{if} \quad \mathbf{w}_k^\mathsf{T}\mathbf{x}_{we} < 0
\end{cases}
\tag{11}
$$

Binarising a graph Laplacian eigenvector partitions the graph encoded by $\hat{\mathbf{S}}$ into two groups, with each of the K eigenvectors inducing a distinct bipartition of the graph [79]. In the context of hashing-based approximate nearest neighbour search, the hope is that many of these cuts will place true nearest neighbours within the same partition. However, eigenvectors associated with the largest eigenvalues are generally unreliable and yield poor-quality partitions [79]. This observation motivates the two-step quantisation algorithm of Liu et al. [52], in which the lower-order eigenvectors are primarily responsible for generating the hashcode bits.

Let $\mathbf{y}^{k+}$ denote the set of positive projected values along dimension k,

$$
\mathbf{y}^{k+} = \{y_{we}^k \in \mathbb{R} \mid y_{we}^k \geq t_1\},
$$

and $\mathbf{y}^{k-}$ the corresponding set of negative values,

$$
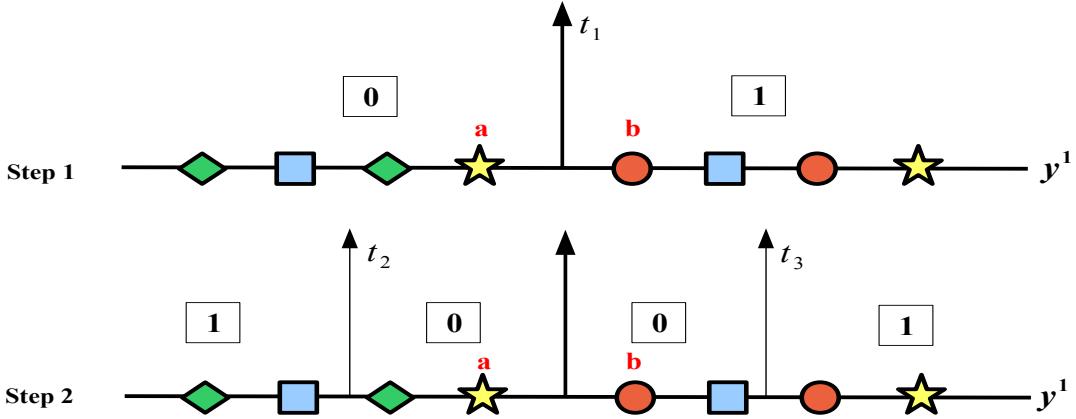\mathbf{y}^{k-} = \{y_{we}^k \in \mathbb{R} \mid y_{we}^k < t_1\}.
$$

Figure 13: Illustration of the Hierarchical Quantisation (HQ) algorithm of [52], shown using data-points a (yellow star) and b (red circle). Quantisation proceeds in two steps. **Step 1:** SBQ thresholds the projected dimension into two regions, producing the first bit of the two-bit encoding ('0' for a, '1' for b). **Step 2:** the two regions are further partitioned using thresholds t_2 and t_3, obtained via a dynamic optimisation algorithm. In this step, a is again assigned '0' while b is reassigned '0', yielding final hashcodes '00' and '10', respectively. By refining the initial SBQ partition, HQ increases the likelihood that nearby points separated by the first threshold receive similar codes, as demonstrated for a and b.

The objective of the second-level threshold optimisation is then to minimise Equation 12.

$$\text{argmin}_{t_2,t_3} \quad \mathbf{f}^\mathsf{T}\mathbf{L}\mathbf{f}$$
$$\text{where } \mathbf{f} = \begin{bmatrix} \mathbf{y}^+ - t_2\mathbf{1}_1 \\ -\mathbf{y}^- + t_3\mathbf{1}_2 \end{bmatrix} \tag{12}$$
$$\text{subject to } \mathbf{1}^\mathsf{T}\mathbf{f} = 0$$

Intuitively, the objective function positions the thresholds t_2 and t_3 so that two conditions are satisfied. First, connected nodes in $\hat{\mathbf{S}}$ (i.e., true nearest neighbours) remain as close as possible along the projected dimension, formalised by

$$\mathbf{f}^\mathsf{T}\mathbf{L}\mathbf{f} = \sum_{ij} \hat{S}_{ij}(f_{we} - f_j)^2.$$

Second, the resulting binarisation yields a balanced partition, with an equal number of '0's and '1's, expressed as the constraint $\mathbf{1}^\mathsf{T}\mathbf{f} = 0$. This balance condition has been shown to maximise the information content of the resulting bits [91]. [52] show that by setting the derivatives of Equation 12 to zero, the thresholds t_2 and t_3 that minimise the objective can be computed in closed form.

Although [52] report that their multi-threshold quantisation algorithm is more effective than SBQ, it lacks generality: the method is tied specifically to graph Laplacian eigenvectors and does not transfer easily to other projection functions. The computational complexity of solving for t_2 and t_3 is approximately $\mathcal{O}(CN_{\text{trd}}^+)$ [52], where N_{trd}^+ is the number of positive projected values in $\mathbf{y}^{k+}$. Once the thresholds are learnt, encoding a novel query point requires only $\mathcal{O}(1)$ time. HQ effectively concentrates the bit budget on the lowest-order graph Laplacian eigenvectors. [52] demonstrate that using only half the number of eigenvectors, with two bits assigned to each, achieves higher retrieval effectiveness than using all available eigenvectors with a single bit each. This is consistent with the observation that many datasets of interest have low intrinsic dimensionality: the lower eigenvectors (corresponding to the smallest eigenvalues) tend to capture most of the neighbourhood structure, while higher eigenvectors are more reflective of global input-space variation. This insight inspired subsequent work on multi-threshold quantisation algorithms for ANN search. In the following, we review subsequent research in chronological order, beginning with the Double-Bit Quantisation (DBQ) algorithm of [42].

5.3 Double Bit Quantisation (DBQ)

Double-Bit Quantisation (DBQ) [42] allocates two thresholds per projected dimension, producing three thresholded regions, each assigned a distinct two-bit code. Unlike HQ (Section 5.2), DBQ is not tied to any specific projection function. For a K-bit hashcode, DBQ therefore requires only $\lfloor K/2 \rfloor$ hyperplanes, compared with K hyperplanes under SBQ. The binary encoding scheme (Figure 14) ensures that any two adjacent regions differ by exactly one bit in Hamming space. This property is crucial for preserving relative distances between data-points in the resulting binary representation, a prerequisite for maximising retrieval effectiveness.

DBQ also introduces an adaptive thresholding algorithm to optimally determine the positions of t_1 and t_2. For a given choice of thresholds, three regions are defined:

$$\mathcal{R}_1 = \{y_{we} \mid y_{we} \leq t_1,\ y_{we} \in \mathbf{y}^k\},$$
$$\mathcal{R}_2 = \{y_{we} \mid t_1 < y_{we} \leq t_2,\ y_{we} \in \mathbf{y}^k\},$$
$$\mathcal{R}_3 = \{y_{we} \mid y_{we} > t_2,\ y_{we} \in \mathbf{y}^k\}.$$

The DBQ objective function, denoted $\mathcal{J}_{\text{dbq}}$, is then defined as the minimisation of the sum of squared Euclidean distances of projected values within each thresholded region (Equation 13).

$$\mathcal{J}_{dbq}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) = \sum_{y_{we} \in \mathbf{r}_1} (y_{we} - \mu_1)^2 + \sum_{y_j \in \mathbf{r}_2} (y_j - \mu_2)^2 + \sum_{y_l \in \mathbf{r}_3} (y_l - \mu_3)^2 \qquad (13)$$

Here, μ_{we} denotes the mean of the projected values in region $\mathcal{R}_{we}$. Since the highest density of projected values typically occurs near zero (Figure 12), DBQ explicitly avoids placing a threshold at zero by enforcing $\mu_2 = 0$, which guarantees $t_1 < 0$ and
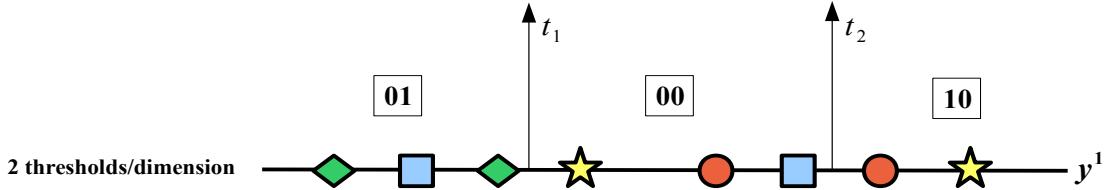
Figure 14: Illustration of Double-Bit Quantisation (DBQ). Two thresholds, t_1 and t_2, partition the projected dimension into three regions. Each region is assigned a distinct two-bit code, as shown. The encoding scheme guarantees that adjacent regions differ by exactly one bit in Hamming space.

$t_2 > 0$. Under this constraint, the objective function $\mathcal{J}_{\mathrm{dbq}}$ simplifies to the form shown in Equation 16 [42].

$$\mathcal{J}_{dbq}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) = \sum_{y_{we} \in \mathbf{y}_k} y_{we}^2 - 2\sum_{y_{we} \in \mathbf{r}_1} y\mu_1 + \sum_{y_{we} \in \mathbf{r}_1} \mu_1^2 - 2\sum_{y_l \in \mathbf{r}_3} y\mu_3 + \sum_{y_l \in \mathbf{r}_3} \mu_3^2, \quad (14)$$

$$= \sum_{y_{we} \in \mathbf{y}_k} y_{we}^2 - |\mathbf{r}_1|\mu_1^2 - |\mathbf{r}_3|\mu_3^2, \quad (15)$$

$$= \sum_{y_{we} \in \mathbf{y}_k} y_{we}^2 - \frac{\left(\sum_{y_{we} \in \mathbf{r}_1} y_{we}\right)^2}{|\mathbf{r}_1|} - \frac{\left(\sum_{y_{we} \in \mathbf{r}_3} y_{we}\right)^2}{|\mathbf{r}_3|} \quad (16)$$

Here, $|\mathcal{R}_{we}|$ denotes the number of projected values (data-points) in region $\mathcal{R}_{we}$. Since $\sum_{y_{we} \in \mathbf{y}^k} y_{we}^2$ is a constant, minimising $\mathcal{J}_{\mathrm{dbq}}$ is equivalent to maximising the alternative objective $\mathcal{J}'_{\mathrm{dbq}}$ (Equation 17).

$$\mathcal{J}'_{dbq}(\mathbf{r}_1, \mathbf{r}_3) = \frac{\left(\sum_{y_{we} \in \mathbf{r}_1} y_{we}\right)^2}{|\mathbf{r}_1|} + \frac{\left(\sum_{y_j \in \mathbf{r}_3} y_j\right)^2}{|\mathbf{r}_3|} \quad (17)$$

The objective function $\mathcal{J}'_{\mathrm{dbq}}$ is maximised by the adaptive thresholding strategy outlined in Algorithm 3. The algorithm initialises the thresholds t_1 and t_2 near zero and then gradually moves them apart: t_1 towards negative infinity $(-\infty)$ and t_2 towards positive infinity $(+\infty)$. At each step, $\mathcal{J}'_{\mathrm{dbq}}$ is evaluated for every projected value along the dimension.

A key invariant is maintained throughout: the mean of region $\mathcal{R}_2$ must remain zero (i.e., $\mu_2 = 0$; Line 9), ensuring that neither threshold partitions the densest region of the distribution around zero. To enforce this constraint, data-points are gradually shifted between regions $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$ (Lines 8–13). Specifically, if the sum of projected values in $\mathcal{R}_2$ falls below zero, a positive value from $\mathcal{R}_3$ is moved into $\mathcal{R}_2$; conversely, if the sum exceeds zero, a negative value from $\mathcal{R}_1$ is moved in.

The objective $\mathcal{J}'_{\mathrm{dbq}}$ is then recomputed (Line 15) on the updated regions. If the objective increases, the thresholds t_1 and t_2 are updated to the largest projected values in $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively, thereby defining new region boundaries. The algorithm

---

Algorithm 3: Double Bit Quantisation [42]

Input: Projected values $\mathbf{y}^k \in \mathbb{R}^N$ resulting from projection onto normal
vector $\mathbf{w}_k \in \mathbb{R}^D$ of hyperplane $\mathbf{h}_k \in \mathbb{R}^D$

Output: Optimised thresholds $t_1 \in \mathbb{R}$, $t_2 \in \mathbb{R}$

1   $\mathbf{r}_1 \leftarrow \{\, y_{we} \mid y_{we} \leq 0,\ y_{we} \in \mathbf{y}^k \,\}$

2   $\mathbf{r}_2 \leftarrow \emptyset$

3   $\mathbf{r}_3 \leftarrow \{\, y_{we} \mid y_{we} > 0,\ y_{we} \in \mathbf{y}^k \,\}$

4   Sort (ascending) projected values in $\mathbf{r}_1$

5   Sort (ascending) projected values in $\mathbf{r}_3$

6   $J_{max} \leftarrow 0$

7   $we \leftarrow 1$

8   **while** $r_1 \neq \emptyset$ **or** $r_3 \neq \emptyset$ **do**

9   **if** $\sum(r_2) \leq 0$ **then**

10         $\mathbf{r}_2 \leftarrow \mathbf{r}_2 \cup \min(\mathbf{r}_3)$ `// Remove minimum value in r`$_3$

11   **else**

12         $\mathbf{r}_2 \leftarrow \mathbf{r}_2 \cup \max(\mathbf{r}_1)$ `// Remove maximum value in r`$_1$

13   **end**

14      $we \leftarrow we + 1$

15      $J \leftarrow \mathcal{J}'_{dbq}(\mathbf{r}_1, \mathbf{r}_3)$

16   `// Equation 17`

17   **if** $J > J_{max}$ **then**

18         $t_1 \leftarrow \max(\mathbf{r}_1)$

19         $t_2 \leftarrow \max(\mathbf{r}_2)$

20         $J_{max} \leftarrow J$

21   **end**

22   **end**

23   **return** t_1, t_2

---

terminates once all data-points have been shifted into $\mathcal{R}_2$. Because all projected values are exhaustively examined, DBQ guarantees that the resulting thresholds t_1 and t_2 achieve the global maximum of $\mathcal{J}'_{\text{dbq}}$.

The implicit assumption underlying DBQ is that the hash functions minimise the squared Euclidean distance between true nearest neighbours in the low-dimensional projected space. In other words, the projected values of any two true nearest neighbours should have a small squared Euclidean distance along each projected dimension. If this assumption holds, then clustering the one-dimensional projections according to a squared-error criterion will tend to assign more true nearest neighbours to the same thresholded region, thereby producing similar hashcodes, compared with an entirely random threshold placement. While [42] demonstrate that this assumption is often reasonable in practice, we show in Chapter 8 that it is far from optimal. In

particular, substantially higher retrieval effectiveness can be achieved using a semi-supervised objective that does not rely exclusively on the quality of the projection function.

The threshold learning time complexity of DBQ is $\mathcal{O}(N_{\text{trd}} \log N_{\text{trd}})$, arising from the preprocessing step that sorts the projected values in regions $\mathcal{R}_1$ and $\mathcal{R}_3$ (Lines 4–5). Once the thresholds are learnt, DBQ has $\mathcal{O}(1)$ complexity for generating a bit for a novel query data-point.

5.4 Manhattan Hashing Quantisation (MHQ)

A key limitation of both HQ and DBQ is their arbitrary restriction to two bits per projected dimension. [44] addressed this by introducing Manhattan Hashing Quantisation (MHQ). MHQ permits an arbitrary allocation of bits per dimension: for B bits, $2^B - 1$ thresholds are used to partition the projected dimension into disjoint regions. To generate a hashcode of length K, MHQ employs $\lfloor K/B \rfloor$ hyperplanes. As with DBQ, MHQ introduces both a new encoding scheme and a threshold optimisation algorithm, designed to better preserve the relative distances between data-points in the resulting hashcodes.

The MHQ encoding scheme is illustrated in Figure 15. Each region is assigned a code using natural binary encoding (NBC). The NBC codebook is constructed by indexing the regions from left to right along the projected dimension (starting at $t_0 = -\infty$), converting each integer index to its corresponding NBC. For example, in Figure 15, the third region from the left in the bottom-most dimension corresponds to integer 2, which is encoded as '010'.

However, NBC does not preserve neighbourhood structure under Hamming distance. This becomes evident when examining the eight regions produced by placing seven thresholds along a projected dimension (Figure 15). The encoding for the fourth region is '011', while the adjacent fifth region is encoded as '100'. Their Hamming distance is 3 despite being adjacent, whereas the distance between the fourth region ('011') and the eighth region ('111')—much farther apart along the projected dimension—is only 1. Consequently, data-points projected far apart may be assigned codes that are close in Hamming space, while nearby points may be assigned codes that are far apart, degrading retrieval quality.

To overcome this issue, [44] proposed replacing Hamming distance with the Manhattan distance between the integer indices corresponding to NBC codewords. This avoids distortions caused by the NBC codebook when used with Hamming distance.[9]

To illustrate, consider the example in [44]. Suppose data-point 1 is assigned the hashcode '000100' and data-point 2 the hashcode '110000'. If $B = 2$, the Manhattan

---

9. Binary reflected Gray coding is also unsuitable as a codebook for nearest neighbour search. While Gray coding guarantees that adjacent codewords differ by unit Hamming distance—a property valuable in error correction over noisy channels [30]—it fails to preserve neighbourhood structure in this context, since codewords for data-points that are far apart can also differ by only one bit.
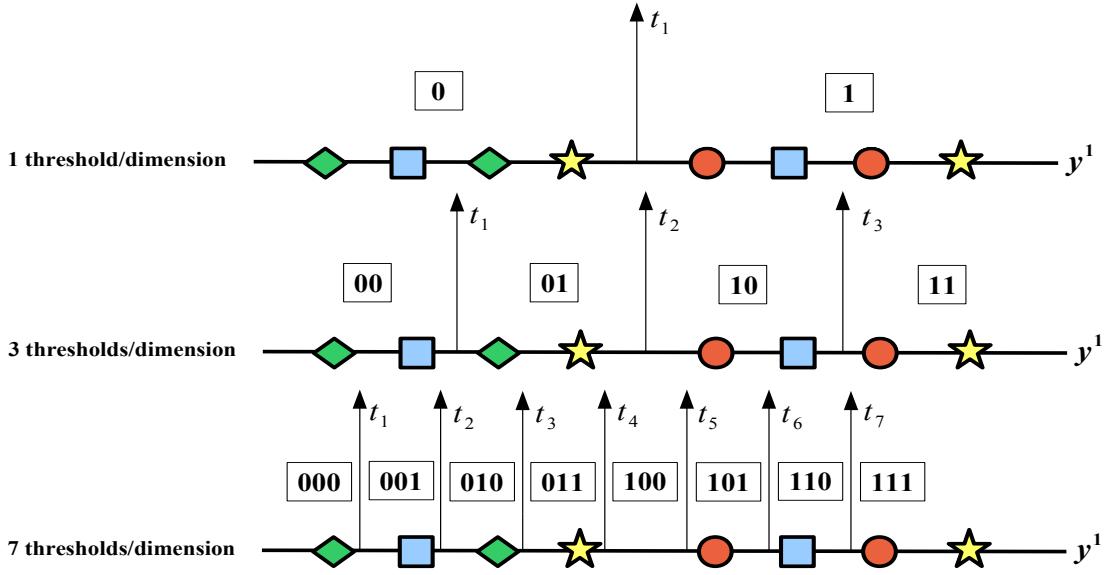
Figure 15: Illustration of Manhattan Hashing Quantisation (MHQ). Each projected dimension is partitioned using $T = 2^B - 1$ thresholds, where B is the number of bits allocated per dimension. The thresholded regions are indexed from left to right, starting at 0 and ending at $2^B - 1$. Each index is then converted to its corresponding natural binary code (NBC), which serves as the region's codeword.

distance between the corresponding codewords,

$$d_{\mathrm{MHQ}}(\cdot, \cdot) : \{0, 1\}^K \times \{0, 1\}^K \to \mathbb{Z}_+$$

is computed as shown in Equation 18.

$$d_{MHQ}(000100, 110000) = d_M(00, 11) + d_M(01, 00) + d_M(00, 00) \qquad (18)$$
$$= 3 + 1 + 0$$
$$= 4$$

For example, when $B = 3$, the Manhattan distance between codewords is computed as shown in Equation 19.

$$d_{MHQ}(000100, 110000) = d_M(000, 110) + d_M(100, 000) \qquad (19)$$
$$= 6 + 4$$
$$= 10$$

Computing the Manhattan distance between the integer indices of regions leads to substantial improvements in retrieval effectiveness, as demonstrated in [44]. This

arises from the perfect preservation of relative distances between data-points: the codewords for adjacent thresholded regions differ by a unit Manhattan distance, and the distance between any two regions increases smoothly with their separation along the projected dimension. Moreover, the scheme generalises naturally to an arbitrary number of thresholds, since the encoding is simply based on the integer index of each region.

The main drawback of computing Manhattan distance between integer indices is its higher computational cost compared with Hamming distance. On modern processors, Hamming distance can be computed extremely efficiently using a bitwise `XOR` between two hashcodes, followed by the native `POPCOUNT` instruction to count the number of set bits. In contrast, [44] do not provide evidence on whether Manhattan distance becomes a bottleneck for large-scale datasets with millions of points and high dimensionality. Subsequent work [89] suggests that it may, since Manhattan distance typically requires substantially more atomic operations on the CPU than Hamming distance. In Chapter 8, we mitigate this concern by introducing a more general quantisation model that achieves high retrieval effectiveness both under a binary encoding scheme with Hamming distance and under the NBC encoding scheme with Manhattan distance.

The MHQ threshold optimisation procedure is straightforward: k-means clustering [54] with 2^B centroids $\{c_{we} \in \mathbb{R}\}_{we=1}^{2^B}$ is applied to the projected dimension. The $2^B - 1$ thresholds $\{t_{we} \in \mathbb{R}\}_{we=1}^{2^B-1}$ are then obtained by taking the midpoints between adjacent centroids (Equation 20).

$$t_{we} = \frac{(c_{we} + c_{we+1})}{2} \tag{20}$$

MHQ requires $\mathcal{O}(2^B N_{\text{trd}})$ time for threshold learning along a single projected dimension, and $\mathcal{O}(1)$ time to generate a bit for a novel query point once the thresholds have been learnt.

5.5 A Link to the Discretisation of Continuous Attributes

It is instructive to briefly consider how this research area relates to the well-studied problem of *discretisation of continuous attributes* in machine learning [19, 24]. Many machine learning models, such as Naïve Bayes [9, 95], transform continuous attributes into nominal attributes via discretisation prior to training. The mechanism underlying this transformation is closely related to the quantisation process in multi-threshold hashing. In both cases, attributes (or projected dimensions) are partitioned by a set of cut-points (thresholds), yielding a non-overlapping division of the continuous domain. Real-valued numbers falling within each region are then assigned a discrete symbol corresponding to that region. For the quantisation algorithms studied in this section, these discrete symbols are restricted to binary codewords, whereas discretisation methods in machine learning allow for a broader range of categorical values.

The discretisation literature is extensive and proposes a wide variety of strategies for determining cut-points, ranging from unsupervised approaches (e.g., equal-interval partitioning) to supervised methods [20] and multivariate extensions, in which attributes are discretised jointly [55, 40]. Given the maturity of this field, we believe there is significant potential for established ideas in discretisation to inform the design of future scalar quantisation algorithms for hashing.

5.6 A Brief Summary

In this section, we reviewed four scalar quantisation algorithms proposed for hashing-based ANN search. Each method transforms real-valued projections into binary codewords, which are concatenated to form the hashcodes of data-points. All of the algorithms follow the same basic principle: one or more thresholds partition the projected dimension, and each region is assigned a codeword (either a single bit or multiple bits).

Single-Bit Quantisation (SBQ; Section 5.1) is the standard approach adopted in most hashing models. It places a single threshold, typically at zero, and is valued for its simplicity and computational efficiency. However, as discussed, SBQ can incur high quantisation error by assigning different codes to related data-points.

The three multi-threshold algorithms—Hierarchical Quantisation (HQ; Section 5.2), Double-Bit Quantisation (DBQ; Section 5.3), and Manhattan Hashing Quantisation (MHQ; Section 5.4)—seek to overcome this limitation by introducing novel encoding schemes and threshold optimisation strategies. Their optimisation criteria differ: HQ employs a spectral graph partitioning objective, while DBQ and MHQ minimise objectives related to squared error and variance. Their encoding schemes also vary, yet all are designed with the shared goal of maximising the preservation of relative distances between data-points in the resulting binary space.

We now turn to a complementary family of methods in Section 6, which focus on generating the projections subsequently quantised by these algorithms.

6. Projection for Nearest Neighbour Search

In Section 4.4, we identified two key steps—projection and quantisation—that together generate similarity-preserving hashcodes in the context of Locality-Sensitive Hashing (LSH). Taken sequentially, these steps determine on which side of each hyperplane a data-point lies, appending a '1' to the hashcode if the point falls on one side and a '0' otherwise. In Section 5, we reviewed prior work focused on improving the quantisation step. These algorithms aim to better preserve neighbourhood structure during binarisation, improving upon the simple sign-based projection rule of Equation 21.

We now turn our attention to research on improving the *projection* step itself, which seeks to learn more effective hyperplanes for similarity-preserving hashing.

$$h_k(\mathbf{x}_{we}) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\mathsf{T}\mathbf{x}_i + t_k)) \tag{21}$$

Here, $\mathbf{w}_k \in \mathbb{R}^D$ denotes the hyperplane normal vector and $t_k \in \mathbb{R}$ the associated quantisation threshold. Equation 21 defines the widely adopted linear hash function used in most hashing research. As discussed in Section 5, with the exception of Anchor Graph Hashing [52], most quantisation models operate independently of the projection stage, assuming that the projections to be binarised have already been generated by some existing projection method.

In this section, we review the equally important step of *projection*, focusing on algorithms that aim to generate projections which preserve relative distances between data-points along the resulting projected dimensions. In the case of a linear hash function, this corresponds to positioning a set of K hyperplanes in the input feature space such that similar data-points are likely to fall within the same polytope-shaped region. These regions form the hashtable buckets used for indexing and retrieval.

Formally, to generate a projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{\mathrm{trd}}}$ from a hyperplane $\mathbf{h}_k \in \mathbb{R}^D$, the data-points $\{\mathbf{x}_{we} \in \mathbb{R}^D\}_{we=1}^{N_{\mathrm{trd}}}$ are projected onto the normal vector $\mathbf{w}_k \in \mathbb{R}^D$ via the dot product operation $\mathbf{w}_k^\mathsf{T}\mathbf{x}_i$. Figure 16 illustrates geometrically how this dot product operation induces a projected dimension.

In Section 4.3, we introduced Locality-Sensitive Hashing (LSH), a seminal early method for solving the ANN search decision problems defined in 4.1–4.2. As discussed in Section 4.4, LSH for inner-product similarity samples hyperplanes uniformly from the unit sphere, with the asymptotic guarantee that, as the number of hyperplanes increases, the Hamming distance between hashcodes approximates the cosine similarity between data-points.[10] Nevertheless, as noted in Section 4.3, randomly sampled hyperplanes often lack discrimination and risk partitioning regions of the input space dense with related data-points. In practice, this means that many hyperplanes (bits) and multiple hash tables are required for acceptable retrieval effectiveness. Longer hashcodes and more tables, however, increase the memory footprint of an LSH system.

Recent work has therefore focused on generating more compact and discriminative hashcodes by learning hyperplanes adapted to the data distribution [52, 51, 91, 27, 65, 47, 97]. These methods form the focus of this part of the review. Projection learning methods for hashing-based ANN can be divided into three categories based on the extent to which the data distribution informs the construction of hashing hyperplanes: *data-independent* (Section 6.2), *data-dependent but unsupervised* (Section 6.3), and *data-dependent and supervised* (Sections 6.4–6.5). Table 2 summarises these approaches.

In the following sections, we review representative work under each category. The field spans a wide variety of techniques for generating hash functions, including ran-

---

10. [25] showed that the expected Hamming distance between two bit vectors formed using hash functions sampled from $\mathcal{H}_{\mathrm{cosine}}$ approximates the angle between the corresponding vectors in the input feature space.

Method	Dependency	Learning Paradigm	Hash Function	Training Complexity	Properties	Section
LSH	Independent	Unsupervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k)$	$\mathcal{O}(KD)$	E_2	4.4
SKLSH	Independent	Unsupervised	$sgn(\cos(\mathbf{w}_k^\intercal \mathbf{x} + t_k) + t_{k'})$	$\mathcal{O}(KD)$	E_2	6.2.1
PCAH	Dependent	Unsupervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k)$	$\mathcal{O}(\min(N_{trd}^2 D, N_{trd}D^2))$	E_2, E_3, E_4	6.3.1
AGH	Dependent	Unsupervised	$sgn(\mathbf{w}_k^\intercal \mathbf{z} + t_k)$	$\mathcal{O}(N_{trd}CK)$	E_1, E_2, E_3, E_4	6.3.4
ITQ	Dependent	Unsupervised	$sgn(\mathbf{R}\mathbf{W}^\intercal \mathbf{x})$	$\mathcal{O}(K^3)$	E_1, E_2, E_3, E_4	6.3.3
SH	Dependent	Unsupervised	$sgn(\sin(\frac{\pi}{2} + j\pi(\mathbf{w}_k^\intercal \mathbf{x})))$	$\mathcal{O}(\min(N_{trd}^2 D, N_{trd}D^2))$	E_1, E_2, E_3, E_4	6.3.2
STH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k)$	$\mathcal{O}(N_{trd}DK + MN_{trd}^2 K)$	E_1, E_2, E_3, E_4	6.4.4
KSH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \kappa(\mathbf{x}) + t_k)$	$\mathcal{O}(N_{trd}CK + N_{trd}^2 CK + N_{trd}C^2 K + C^3 K)$	E_1, E_2	6.4.3
BRE	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \kappa(\mathbf{x}) + t_k)$	$\mathcal{O}(KN_{trd}^2 + KN_{trd}\log N_{trd})$	E_1, E_2	6.4.2
CVH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x), sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(N_{trd}D^2 + D^3), D = \max(D_x, D_z)$	E_2, E_3, E_4	6.5.1
CMSSH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x), sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(KMN_{trd}D), D = \max(D_x, D_z)$	E_1, E_2	6.5.3
CRH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x), sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(D_x^2 D_z + D_x D_z^2 + D_z^3)$	E_1, E_2	6.5.2
PDH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x), sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(MN_{trd}^2 K)$	E_1, E_2, E_4	6.5.4
IMH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x), sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(N_{trd}^3)$	E_1, E_2, E_3, E_4	6.5.5

Table 2: Categorisation of existing projection learning algorithms, adapted in part from [86]. See Section 6 for details on each hash function. The final five algorithms listed are cross-modal. Notation: N denotes the total number of data-points, K the hashcode length, C a set of anchor data-points ($C \ll N_{\mathrm{trd}}$), N_{trd} the number of training data-points ($C < N_{\mathrm{trd}} \ll N$), and M the number of iterations. Typical values are $N_{\mathrm{trd}} = 1000$–2000, $K = 32$–128, and $C = 300$.

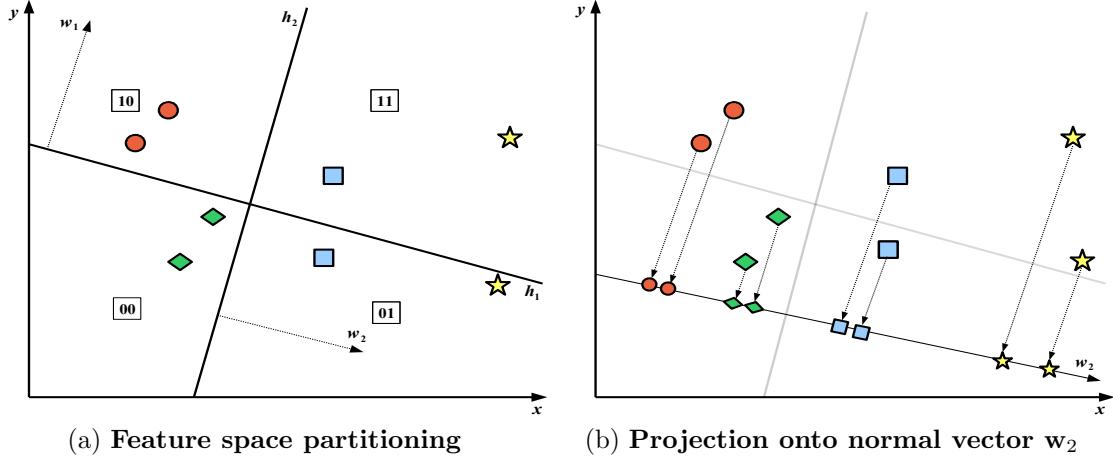(a) **Feature space partitioning**   (b) **Projection onto normal vector $\mathbf{w}_2$**

Figure 16: Illustration of the projection operation in hashing-based ANN search. Projection methods partition the feature space using a set of hyperplanes. In Figure (a), a two-dimensional feature space is partitioned by two hyperplanes, $\mathbf{h}_1$ and $\mathbf{h}_2$. To compute the bucket index (hashcode) of a data-point, the point is projected onto the normal vectors ($\mathbf{w}_1$, $\mathbf{w}_2$), followed by binary quantisation. Figure (b) illustrates the projection onto normal vector $\mathbf{w}_2$; the resulting values form the projected dimension $\mathbf{y}^2 \in \mathbb{R}^{N_{\text{trd}}}$. Section 6 reviews existing methods for positioning hyperplanes such that true nearest neighbours are likely to remain close along the resulting projected dimensions.

dom projections, kernel functions, spectral methods, and boosting. While the literature is too extensive for exhaustive coverage, we focus on influential models that have become widely adopted, particularly those accompanied by publicly available codebases. This ensures that our discussion is grounded in results collected on comparable datasets and experimental protocols, and that comparisons are made against competitive baselines. For broader surveys, we refer the reader to [85] and [29].

Finally, we note that all of the hashing models reviewed here assume search over a single hash table ($L = 1$), as is standard in the literature. Methods that explicitly *learn* multiple tables in a data-dependent manner form a promising subfield but are beyond the scope of this review; see [94] and [53] for representative research in this direction.

6.1 The Four Properties of an Effective Hashcode

Before discussing individual models for projection, it is useful to consider several properties that contribute to the effectiveness of a hashcode for nearest neighbour search. The seminal work on Spectral Hashing (SH) by [91] codified four such properties (E_1–E_4):

E_1: Hashcodes for similar data-points should have *low Hamming distance*.

E_2: Hashcodes should be *efficiently computable* for novel query points.

E_3: Each bit should take the values 0 and 1 with *equal probability*.

E_4: Different bits should be *pairwise independent*.

The importance of the first property (E_1) has already been highlighted in the context of LSH (Section 4.3) and binary quantisation (Section 5). The remaining criteria (E_2–E_4) are equally significant. Property E_2 is crucial for practical deployment: given a novel data-point, its hashcode must be computed rapidly so that query time remains low. This problem is known in the learning-to-hash literature as *out-of-sample extension*. LSH provides a straightforward solution: multiply the query vector by a matrix whose columns are hyperplane normals, followed by sign thresholding (Section 4.3).

Properties E_3 and E_4 address the *efficiency* and *compactness* of hashcodes, respectively. Property E_3 requires each hyperplane to partition the dataset evenly, i.e., $\sum_{we=1}^{N_{\text{trd}}} h_k(\mathbf{x}_{we}) = 0$. By the principle of maximum entropy, this maximises the information carried by each bit [4], ensuring balanced bucket occupancy in the hashtable.[11] Balanced partitions also prevent the degenerate case where queries require examining an excessively large number of candidates within a single bucket.

Property E_4 seeks compactness by eliminating redundant bits that encode overlapping information about the input space. An ideal hashing scheme should minimise the number of bits required to represent the data, thereby conserving both storage and computation. The majority of data-dependent projection methods developed since [91] aim to learn hyperplanes that produce hashcodes satisfying as many of these four properties as possible. In Sections 6.2–6.4, we will examine how effectively different approaches are able to jointly preserve these properties during optimisation.

6.2 Data-Independent Projection Methods

Aside from Locality-Sensitive Hashing (LSH), reviewed in Section 4.3, we consider one additional data-independent method: Locality-Sensitive Hashing from Shift-Invariant Kernels (SKLSH) [65], which extends LSH to preserve kernel-based similarity (Section 6.2.1).

6.2.1 LOCALITY SENSTIVE HASHING FROM SHIFT INVARIANT KERNELS (SKLSH)

Locality-Sensitive Hashing from Shift-Invariant Kernels (SKLSH) extends LSH to preserve similarity as defined by a kernel function $\kappa : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$. Common examples include the Gaussian kernel,

$$\kappa(\mathbf{x}_{we}, \mathbf{x}_j) = \exp\big(-\gamma \|\mathbf{x}_{we} - \mathbf{x}_j\|^2 / 2\big),$$

---

11. [88] showed that the NP-hard constraint E_3 can be relaxed by demonstrating its equivalence to variance maximisation for the kth bit. Strict enforcement of E_3 may, however, be sub-optimal if it partitions clusters of related data-points across different buckets. This issue can often be mitigated through the use of multiple independent hashtables.

and the Laplacian kernel,

$$\kappa(\mathbf{x}_{we}, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_{we} - \mathbf{x}_j\|_1 / 2),$$

where $\gamma \in \mathbb{R}$ is the kernel bandwidth parameter.

Conceptually, SKLSH is similar to LSH but employs a different hash function family $\mathcal{H}$ in order to preserve kernel-based similarity. The objective is to construct an embedding $g : \mathbb{R}^D \to \{0, 1\}^K$ such that, if two data-points are highly similar under the kernel (i.e., $\kappa(\mathbf{x}_{we}, \mathbf{x}_j) \approx 1$), their hashcodes exhibit a high degree of overlap ($d_{\text{Hamming}}(g(\mathbf{x}_{we}), g(\mathbf{x}_j)) \approx 0$), and conversely when $\kappa(\mathbf{x}_{we}, \mathbf{x}_j) \approx 0$.

To achieve this, [65] define a low-dimensional projection $\Psi^K : \mathbb{R}^D \to \mathbb{R}^K$ using the random Fourier features of [66]. These features guarantee that the inner product between transformed data-points approximates the value of a *shift-invariant* kernel,[12]

$$\Psi_k(\mathbf{x}_{we}) \cdot \Psi_k(\mathbf{x}_j) \approx \hat{\kappa}(\mathbf{x}_{we} - \mathbf{x}_j).$$

The explicit random Fourier features mapping is given in Equation 22.

$$\Psi_k(\mathbf{x}_{we}) = \sqrt{2} cos(\mathbf{w}_k^\mathsf{T} \mathbf{x}_{we} + t_k) \tag{22}$$

For the Gaussian kernel, $\mathbf{w}_k \sim \mathcal{N}(0, \gamma \mathbf{I}_{D \times D})$ and $t_k \sim \text{Unif}[0, 2\pi]$. The key contribution of [65] is to employ this embedding as the basis of a novel hash function (Equation 23).

$$h_k(\mathbf{x}_{we}) = \frac{1}{2}[1 + sgn(cos(\mathbf{w}_k^\mathsf{T} \mathbf{x}_{we} + t_k) + t_{k'})] \tag{23}$$

Here, sgn denotes the sign function, adjusted so that $sgn(0) = -1$, and $t_{k'} \sim \text{Unif}[-1, 1]$. [65] prove that hashing data-points with K randomly sampled hash functions yields a binary embedding whose Hamming distance approximates the desired shift-invariant kernel similarity. Since the hyperplanes are sampled randomly, the training time complexity of the algorithm is only $\mathcal{O}(DK)$. SKLSH therefore satisfies property E_2 of an effective hashcode, namely efficient computation of hashcodes.

6.3 Data-Dependent (Unsupervised) Projection Methods

In this section, we provide a critical appraisal of data-dependent hashing methods that learn hyperplanes in an unsupervised manner, i.e., without requiring supervisory information such as pairwise similarity constraints or class labels. All of the unsupervised approaches reviewed here learn hashing hyperplanes by formulating a *trace minimisation/maximisation* problem, solved either in closed form as an eigenvalue problem or via singular value decomposition (SVD). These methods build directly on well-established techniques for linear and non-linear dimensionality reduction, most notably Principal Components Analysis (PCA) and Laplacian Eigenmaps

---

12. A shift-invariant kernel is defined as $\kappa(\mathbf{x}_{we}, \mathbf{x}_j) = \hat{\kappa}(\mathbf{x}_{we} - \mathbf{x}_j)$.

(LapEig). Given the centrality of matrix factorisation in the learning-to-hash literature—including many methods not covered in detail here—we begin with a brief introduction to this solution strategy before reviewing individual algorithms in Sections 6.3.1–6.3.4.[13]

Broadly speaking, there are two main strategies for dimensionality reduction of a dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ into a lower-dimensional representation $\mathbf{Y} \in \mathbb{R}^{N \times K}$, where $K \ll D$. The first approach seeks an explicit linear transformation characterised by a projection matrix $\mathbf{W} \in \mathbb{R}^{D \times K}$, such that $\mathbf{Y} = \mathbf{X}\mathbf{W}$. PCA is a canonical example of this *projective* strategy. The second approach computes a non-linear embedding $\mathbf{Y} \in \mathbb{R}^{N \times K}$ directly, without requiring an explicit mapping function. Methods of this type, such as LapEig, typically impose neighbourhood constraints to ensure that nearby data-points in the original space remain close in the reduced space. Despite their differences, both categories can be expressed within a unified framework using a standard trace maximisation objective (Equation 24).

$$
\begin{aligned}
\mathrm{argmax}_{\mathbf{V} \in \mathbb{R}^{N \times K}} \quad & tr(\mathbf{V}^{\mathsf{T}}\mathbf{A}\mathbf{V}) \\
\text{subject to } & \mathbf{V}^{\mathsf{T}}\mathbf{1} = 0 \\
& \mathbf{V}^{\mathsf{T}}\mathbf{B}\mathbf{V} = \mathbf{w}\mathbf{e}^{K \times K}
\end{aligned} \tag{24}
$$

Here, $\mathbf{A}$ is a symmetric matrix, $\mathbf{B}$ is a positive-definite matrix, $\mathbf{V}$ is an orthonormal[14] matrix, and $\mathrm{tr}(\mathbf{A}) = \sum_i A_{ii}$. The exact specification of these matrices depends on the projection function under consideration. We will define $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{V}$, including their dimensionalities, in Sections 6.3.1–6.3.4.

As an intuitive example, in the context of Principal Component Analysis (PCA) we have $\mathbf{A} = \mathbf{X}^{\mathsf{T}}\mathbf{X}$, $\mathbf{V} = \mathbf{W} \in \mathbb{R}^{D \times K}$, and $\mathbf{B} = \mathbf{I} \in \mathbb{R}^{D \times D}$. Maximising the trace (Equation 24) in this case is equivalent to finding the principal directions that capture maximum variance in the input space.

The trace maximisation in Equation 24 can be solved as a generalised eigenvalue problem,

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{B}\mathbf{v}_i,$$

where $\mathbf{v}_i$ is the ith eigenvector with eigenvalue λ_i [73, 41]. Much of the unsupervised learning-to-hash literature can be distilled to this template: shape the optimisation problem into the trace maximisation form of Equation 24, and then solve for the K eigenvectors of the resulting eigenvalue problem. This optimisation can be efficiently addressed using standard solvers such as `eigs` or `svd` in Matlab.

The design of an unsupervised data-dependent hashing function typically follows four steps:

---

13. For further details on trace optimisation and eigenproblems for dimensionality reduction, see the survey of [41].

14. An orthonormal matrix $\mathbf{V}$ is a square matrix whose columns and rows are orthogonal unit vectors, satisfying $\mathbf{V}^{\mathsf{T}}\mathbf{V} = \mathbf{V}\mathbf{V}^{\mathsf{T}} = \mathbf{I}$.
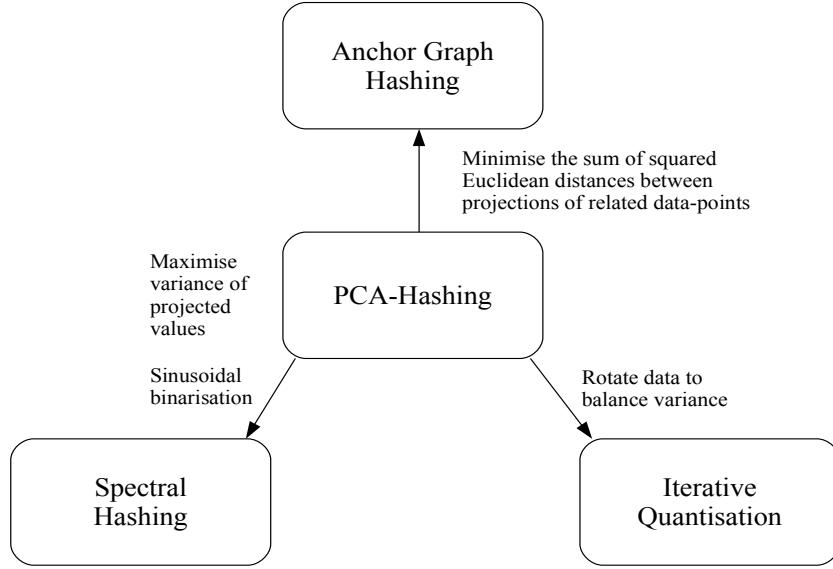
Figure 17: Illustration of the relationships between unsupervised data-dependent hashing models. PCA lies at the centre, with directed arcs labelled by the operations required to transform one model into another. See Section 6.3 for a detailed description of the four models.

1. Reformulate the problem as a matrix trace minimisation/maximisation (Equation 24).

2. Solve the optimisation as an eigenvalue problem or via SVD; the K eigenvectors (or right-singular vectors) serve as the normal vectors of the hashing hyperplanes.

3. Address the imbalanced variance resulting from matrix factorisation.

4. Construct an out-of-sample extension in the case of a non-projective mapping.

We will observe these four design principles across the unsupervised methods reviewed in this section: PCA Hashing (PCAH, Section 6.3.1), Spectral Hashing (SH, Section 6.3.2), Iterative Quantisation (ITQ, Section 6.3.3), and Anchor Graph Hashing (AGH, Section 6.3.4). In three of these methods (PCAH, SH, ITQ), PCA first extracts the directions of maximum variance, which are then used as hashing hyperplanes. The main contributions of these approaches lie in Step 3, where different strategies are proposed to mitigate the impact of imbalanced variance across hyperplanes, which otherwise reduces the quality of hashcodes derived from lower principal components. The final method, AGH, adopts a different approach, computing an eigenfunction extension of graph Laplacian eigenvectors and basing hash function learning on the Laplacian Eigenmap algorithm.

Ultimately, the goal of all these methods is to learn K hash functions $\{h_k : \mathbb{R}^D \to \{0,1\}\}_{k=1}^{K}$ that can be concatenated to generate binary hashcodes for unseen datapoints. Figure 17 summarises one interpretation of the relationships among these models.

6.3.1 PRINCIPAL COMPONENTS ANALYSIS HASHING (PCAH)

Principal Components Analysis (PCA) [35] has long been the most widely used low-dimensional embedding for data-dependent hashing schemes. A large body of influential work manipulates PCA embeddings to improve retrieval accuracy over earlier unsupervised hashing schemes [43, 27, 91, 88]. We therefore begin our review with the most basic PCA-based hashing approach: computing the principal directions of the data and directly using the singular vectors with the highest singular values as hashing hyperplanes, without further modification [87].

Without loss of generality, we assume that the training data $\mathbf{X} \in \mathbb{R}^{N_{\text{trd}} \times D}$ has been centred by subtracting the mean, i.e., $\sum_{we=1}^{N_{\text{trd}}} \mathbf{x}_{we} = 0$. The standard maximum-variance PCA objective is given by:

$$
\begin{aligned}
\underset{\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^{K}}{\text{argmax}} \quad & \frac{1}{N_{\text{trd}}} \sum_k \mathbf{w}_k^\mathsf{T} \mathbf{X}^\mathsf{T} \mathbf{X} \mathbf{w}_k \\
= \quad & \frac{1}{N_{\text{trd}}} \operatorname{tr}(\mathbf{W}^\mathsf{T} \mathbf{X}^\mathsf{T} \mathbf{X} \mathbf{W}) \\
\text{subject to} \quad & \mathbf{W}^\mathsf{T} \mathbf{W} = \mathbf{I},
\end{aligned}
\tag{25}
$$

Here, $\operatorname{tr}(\mathbf{A}) = \sum_i A_{ii}$ denotes the trace operator, and $\mathbf{W} \in \mathbb{R}^{D \times K}$ contains the vectors $\mathbf{w}_k$ as columns. The constraint $\mathbf{W}^\mathsf{T} \mathbf{W} = \mathbf{I}$ enforces orthogonality, which may be viewed as a relaxed version of the pairwise independence property (E_4, Section 6.1). Equation 25 is identical to the general trace optimisation template (Equation 24) with $\mathbf{A} = \mathbf{X}^\mathsf{T} \mathbf{X}$, $\mathbf{V} = \mathbf{W}$, and $\mathbf{B} = \mathbf{I}$. The maximisers $\{\mathbf{w}_k\}$ are the right singular vectors corresponding to the largest singular values, obtained via SVD on $\mathbf{X} \in \mathbb{R}^{N_{\text{trd}} \times D}$ in $\mathcal{O}(\min(N_{\text{trd}}^2 D, N_{\text{trd}} D^2))$ operations.

The PCA solution $\mathbf{W} \in \mathbb{R}^{D \times K}$, where each column is a principal component, can be interpreted as a rigid rotation of the feature space such that successive coordinates capture as much variance as possible. For a K-bit hashcode, it is standard to take the top-K singular vectors as hashing hyperplanes, with thresholds $t_k = 0$ since the data is mean-centred. The resulting PCAH hash function is:

$$
h_k(\mathbf{x}_{we}) = \tfrac{1}{2}\big(1 + \operatorname{sgn}(\mathbf{w}_k^\mathsf{T} \mathbf{x}_{we})\big).
\tag{26}
$$

Using PCA to generate hash functions naturally satisfies properties E_2, E_3, and E_4 of effective hashcodes (Section 6.1).

Despite its popularity, PCA has several limitations when applied to hashing. First, SVD is computationally expensive, making PCAH unsuitable for very large or high-
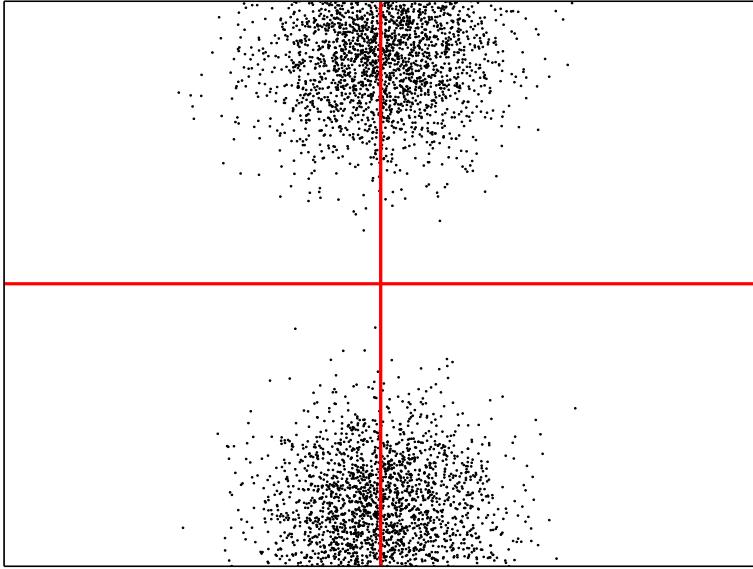
Figure 18: Illustration of PCA principal components $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^D$ (shown as perpendicular lines) for a toy two-dimensional dataset (black dots). The data is aligned to the principal axes, and the components point in the directions of greatest variance. These vectors are used as the normals to the hashing hyperplanes in PCA hashing (PCAH).

dimensional datasets. Second, the number of bits K is limited by the input dimensionality D, since $K \leq D$. In practice, hashing often requires long codes that are segmented into L parts to index multiple hash tables. PCAH thus imposes an upper bound on both K and L. Finally, the singular vectors associated with small singular values capture little variance and are unreliable. Using them as hyperplane normals (Equation 26) yields poor-quality hashcodes that fail to discriminate between data-points. This issue—termed the *imbalanced variance problem*—has motivated substantial follow-up work, including methods that better exploit top singular vectors (Sections 6.3.2, 6.3.4) or transform the data so that variance is more evenly distributed across hyperplanes (Section 6.3.3).

6.3.2 SPECTRAL HASHING (SH)

Spectral Hashing (SH) [91] was one of the earliest schemes for data-dependent hashing and is widely regarded as the work that sparked significant interest in this direction within the Computer Vision community. SH introduced a general framework for graph-based hashing and has since served as a foundation for many subsequent unsupervised and supervised models in the learning-to-hash literature.

As discussed in Section 6.1, SH placed the notion of an "effective hashcode" on firm theoretical footing by codifying four desirable properties (E_1–E_4). In contrast to PCAH (Section 6.3.1), which simply binarises projections onto the first K principal

components—a procedure unlikely to yield hashcodes with all of the desired properties—SH explicitly incorporates three of them into its formulation. Specifically, property E_1 (neighbourhood preservation) is embedded in the objective function, while properties E_3 (balanced bits) and E_4 (bit independence) are enforced as constraints.

The resulting optimisation problem introduced by SH is given in Equation 27.

$$
\begin{aligned}
\text{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & \sum_{ij} S_{ij}\|\mathbf{y}_{we} - \mathbf{y}_j\|^2 \\
= \quad & tr(\mathbf{Y}^\mathsf{T}(\mathbf{D} - \mathbf{S})\mathbf{Y}) \\
\text{subject to} \quad & \mathbf{Y} \in \{-1, 1\}^{N_{trd}\times K} \\
& \mathbf{Y}^\mathsf{T}\mathbf{1} = \mathbf{0} \\
& \mathbf{Y}^\mathsf{T}\mathbf{Y} = N_{trd}\mathbf{we}^{K\times K}
\end{aligned}
\tag{27}
$$

Here $D_{ii} = \sum_j S_{ij}$ defines the diagonal degree matrix of the adjacency matrix $\mathbf{S} \in \mathbb{R}^{N_{\text{trd}}\times N_{\text{trd}}}$. Following [91], the similarity between input data-points is modelled using the Gaussian kernel,

$$
S_{ij} = \exp\big(-\|\mathbf{x}_{we} - \mathbf{x}_j\|^2/\gamma^2\big),
$$

where $\gamma \in \mathbb{R}$ is the kernel bandwidth parameter.

The objective seeks to learn binary hashcodes $\{\mathbf{y}_{we} \in \{-1, 1\}^K\}_{we=1}^{N_{\text{trd}}}$ such that the average Hamming distance between similar neighbours is minimised, subject to bit balance and independence constraints. The constraint $\mathbf{Y}^\mathsf{T}\mathbf{1} = \mathbf{0}$ enforces property E_3, requiring each bit to partition the data evenly across buckets, while the constraint $\mathbf{Y}^\mathsf{T}\mathbf{Y} = N_{\text{trd}}\mathbf{I}_{K\times K}$ enforces property E_4 by encouraging pairwise uncorrelated bits.

Unfortunately, this optimisation problem is NP-hard even for the single-bit case, by reduction to the balanced graph partitioning problem (a classic NP-hard problem).[15] To make the optimisation tractable, [91] apply the spectral relaxation trick of [79], removing the integrality constraint and allowing the projection matrix $\mathbf{Y} \in \mathbb{R}^{N_{\text{trd}}\times K}$ to take real values (Equation 28).

$$
\begin{aligned}
\text{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & tr(\mathbf{Y}^\mathsf{T}(\mathbf{D} - \mathbf{S})\mathbf{Y}) \\
\text{subject to} \quad & \mathbf{Y} \in \mathbb{R}^{N_{trd}\times K} \\
& \mathbf{Y}^\mathsf{T}\mathbf{1} = \mathbf{0} \\
& \mathbf{Y}^\mathsf{T}\mathbf{Y} = N_{trd}\mathbf{we}^{K\times K}
\end{aligned}
\tag{28}
$$

Equation 28 is identical to the general template in Equation 24 with $\mathbf{A} = \mathbf{D} - \mathbf{S}$ and $\mathbf{V} = \mathbf{Y}$. Its solutions are therefore the K eigenvectors corresponding to the smallest non-trivial eigenvalues of the graph Laplacian $\mathbf{D} - \mathbf{S}$,[16] yielding the spectral embedding matrix $\mathbf{Y} \in \mathbb{R}^{N_{\text{trd}}\times K}$. Each row of $\mathbf{Y}$ can be interpreted as the

---

15. See [91] for a detailed proof.
16. The trivial eigenvector $\mathbf{1}$ with eigenvalue 0 is ignored.

low-dimensional coordinate representation of a data-point. Solving Equation 28 ensures that data-points deemed close in the neighbourhood graph $\mathbf{S} \in \{0,1\}^{N_{\text{trd}} \times N_{\text{trd}}}$ are mapped close together in the embedding space, thereby preserving local neighbourhood structure. The computational complexity of solving this eigenproblem is approximately $\mathcal{O}(N_{\text{trd}}^2 K)$.

A key challenge is that the Laplacian eigenvectors computed in this way provide hashcodes only for the N_{trd} training data-points, leaving open the problem of *out-of-sample extension*. In spectral methods, a common strategy is the Nyström extension [6, 92], but without approximation this is prohibitively expensive—$\mathcal{O}(N_{\text{trd}}K)$—comparable to brute-force nearest neighbour search. To address this limitation, [91] propose a simple approximation by assuming that the projected data is sampled from a multidimensional uniform distribution. Under this assumption, they derive an efficient out-of-sample extension in the form of one-dimensional Laplacian eigenfunctions, given by Equation 29.

$$\Psi_{kj}(y_{we}^k) = sin(\frac{\pi}{2} + \frac{f\pi}{b_k - a_k} y_{we}^k) \tag{29}$$

with eigenvalues given by Equation 30:

$$\lambda_{kf} = 1 - e^{-\frac{\gamma^2}{2}|\frac{f\pi}{b_k - a_k}|^2} \tag{30}$$

The eigenfunctions are aligned with the principal directions obtained from PCA, where $f \in \{1, \ldots, K\}$ denotes the frequency, a_k, b_k are parameters of the uniform distribution estimated for projected dimension k, and $y_{we}^k \in \mathbb{R}$ is the projection of data-point $\mathbf{x}_{we}$ onto the kth principal direction. For clarity, we divide the SH algorithm into two stages:

A. **Training step:** Estimate the uniform distribution parameters $\{a_k, b_k\}_{k=1}^K$ and compute the PCA principal directions $\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^K$.

B. **Out-of-sample extension:** Generate hashcodes for novel data-points using the learnt parameters and PCA directions.

(A) Hash function training:

1. Compute the top K principal components $\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^K$ via PCA on the training dataset $\mathbf{X} \in \mathbb{R}^{N_{\text{trd}} \times D}$, and stack them as the columns of $\mathbf{W} \in \mathbb{R}^{D \times K}$.

2. Project $\mathbf{X}$ onto the principal directions: $\mathbf{Y} = \mathbf{XW}$, where $\mathbf{Y} \in \mathbb{R}^{N_{\text{trd}} \times K}$.

3. For each projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{\text{trd}}}$, estimate a uniform distribution by computing $a_k = \min(\mathbf{y}^k)$, $b_k = \max(\mathbf{y}^k)$.

4. For each dimension $\mathbf{y}^k$, compute K analytical eigenfunctions $\{\Psi_{kf}\}_{f=1}^K$ and their associated eigenvalues $\{\lambda_{kf} \in \mathbb{R}\}_{f=1}^K$ using Equations 29–30.

5. Sort all K^2 eigenvalues and select the K eigenfunctions with the smallest values. Denote these as $\{\bar{\Psi}_k\}_{k=1}^K$, their corresponding normal vectors as $\{\bar{\mathbf{w}}_k \in \mathbb{R}^D\}_{k=1}^K$, and their associated uniform distribution parameters as $\{\bar{a}_k, \bar{b}_k\}_{k=1}^K$. Retain all three sets for use in the out-of-sample extension.

(B) Out-of-sample extension:

1. For a query $\mathbf{q}$, compute the K-bit hashcode

$$g(\mathbf{q}) = [h_1(\mathbf{q}), h_2(\mathbf{q}), \ldots, h_K(\mathbf{q})],$$

where each hash function h_k is defined by Equation 31 using the retained sets $\{\bar{\Psi}_k, \bar{\mathbf{w}}_k, \bar{a}_k, \bar{b}_k\}_{k=1}^K$. Here, the sinusoidal partitioning of Equation 29 may be contrasted with the cosine partitioning used in SKLSH (Section 6.2.1).

$$h_k(\mathbf{q}) = \frac{1}{2}(1 + sgn(\bar{\Psi}_k(\bar{\mathbf{w}}_k^\intercal \mathbf{q})) \tag{31}$$

The computational complexity of SH is dominated by the $\mathcal{O}(\min(N_{\mathrm{trd}}^2 D, N_{\mathrm{trd}} D^2))$ operations required to perform PCA on the database. SH favours directions with both a large spread $|b_k - a_k|$ and low spatial frequency f. For low-dimensional data $(D \approx K)$, SH typically selects multiple sinusoidal eigenfunctions of progressively higher frequencies along those eigenvectors capturing the greatest variance. Intuitively, the larger the variance of a projected dimension $\mathbf{y}^k$, the wider its range $|b_k - a_k|$, and the smaller its associated eigenvalue (Equation 30). In such cases, SH has the desirable property of allocating more bits to the most informative hyperplanes, effectively weighting their contribution more heavily in Hamming distance computation. This mitigates the key limitation of PCAH, which is forced to progressively select orthogonal directions of diminishing variance (Section 6.3.1). By front-loading the bit allocation onto informative hyperplanes, SH often achieves higher retrieval effectiveness [52, 59].

In contrast, in high-dimensional settings $(D \gg K)$, where the leading eigenvectors capture similar variance, SH degenerates into PCAH, selecting each PCA hyperplane only once.

Despite the improved retrieval effectiveness of SH compared to LSH, as reported in [91], the method suffers from a significant limitation: its assumption of a uniform data distribution. Anchor Graph Hashing (AGH) [52] addresses this shortcoming by introducing a more realistic approximation that enables efficient application of the Nyström method for out-of-sample extension. We now turn to AGH in Section 6.3.4.

6.3.3 Iterative Quantisation (ITQ)

While Spectral Hashing (SH) implicitly allocates more bits to hyperplanes that capture a greater proportion of the variance in the input space—thereby partially mitigating the imbalanced variance problem—Iterative Quantisation (ITQ) takes a different approach. ITQ explicitly seeks to balance the variance across PCA hyperplanes by learning a rotation of the feature space. Specifically, ITQ introduces an iterative scheme, reminiscent of the k-means algorithm, to learn a rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$ such that the projections onto the principal directions $\mathbf{W} \in \mathbb{R}^{D \times K}$ minimise the quantisation error. This objective can be expressed in matricial form as Equation 32.

$$
\mathrm{argmin}_{\mathbf{B} \in \mathbb{R}^{N_{trd} \times K}, \mathbf{R} \in \mathbb{R}^{K \times K}} \quad \|\mathbf{B} - \mathbf{Y}\mathbf{R}\|_F^2
$$
$$
\text{where } \mathbf{B} \in \{-1, 1\}^{N_{trd} \times K} \tag{32}
$$
$$
\text{subject to } \mathbf{R}^\mathsf{T}\mathbf{R} = K\mathbf{w}\mathbf{e}^{K \times K}
$$

Equation 32 is closely related to the orthogonal Procrustes problem,[17] first introduced by [74]. In the Procrustes setting, the task is to transform one matrix into another using an orthogonal transformation so as to minimise the sum of squared residuals between the transformed and target matrices.

In ITQ, Equation 32 seeks a rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$ that minimises the squared Euclidean distance between the projection vectors $\mathbf{Y} \in \mathbb{R}^{N_{\mathrm{trd}} \times K}$ and their associated binary vectors $\mathbf{B} = \mathrm{sgn}(\mathbf{X}\mathbf{W})$, where PCA hyperplanes are stacked as the columns of $\mathbf{W}$. The optimisation is challenging because both $\mathbf{B}$ and $\mathbf{R}$ are initially unknown: estimating the optimal $\mathbf{R}$ requires knowing $\mathbf{B}$, while estimating $\mathbf{B}$ requires knowing $\mathbf{R}$.

This circular dependency is resolved by an iterative scheme, reminiscent of k-means clustering, that begins with a random initialisation of $\mathbf{R}$ and alternates between two optimisation steps: solving for $\mathbf{B}$ with $\mathbf{R}$ fixed, and solving for $\mathbf{R}$ with $\mathbf{B}$ fixed [27]. The full iterative ITQ procedure is summarised in Algorithm 4.

The key step in ITQ is shown in Line 6 of Algorithm 4. As demonstrated by [32] and [3], when the target matrix $\mathbf{B}$ is fixed, the optimal transformation $\mathbf{R}$ minimising the squared Euclidean distance can be obtained via the singular value decomposition (SVD) of $\mathbf{B}^\mathsf{T}\mathbf{Y}$. Conversely, when $\mathbf{R}$ is fixed, [27] show that the optimal $\mathbf{B}$ minimising Equation 32 is given by applying single-bit quantisation (Section 5.1) (Line 4).

Beyond satisfying properties E_1 and E_2 of an effective hashcode (Section 6.1), ITQ also approximately preserves E_3 and E_4. The balanced partition property (E_3) is encouraged by PCA, which maximises the variance of the projections and has been

---

17. The name derives from a grisly Greek myth. Procrustes offered travelers rest on a "magic" bed that could perfectly accommodate any visitor, regardless of height. In reality, he forcibly stretched or amputated his guests to make them fit the bed. Analogously, the mathematical problem seeks to transform one matrix into another under an orthogonal constraint, often by "forcing" alignment.

Algorithm 4: ITERATIVE QUANTISATION (ITQ) [27]

Input: Data-points $\mathbf{X} \in \mathbb{R}^{N_{\mathrm{trd}} \times D}$, PCA hyperplanes $\mathbf{W} \in \mathbb{R}^{D \times K}$, number of iterations M, randomly initialised rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$

Output: Optimised rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$

1   $\mathbf{Y} \leftarrow \mathbf{XW}$
2   // Project data onto PCA hyperplanes
3   **for** $m \leftarrow 1$ **to** M **do**
4     $\mathbf{B} \leftarrow \mathrm{sgn}(\mathbf{YR})$
5   // Rotate data using R and quantise
6     $\mathbf{U\Sigma V}^{\mathsf{T}} \leftarrow \mathrm{SVD}(\mathbf{B}^{\mathsf{T}}\mathbf{Y})$
7   // Perform SVD on BᵀY
8     $\mathbf{R} \leftarrow \mathbf{VU}^{\mathsf{T}}$
9   // Update R to minimise Eq. 32 for fixed B
10 **end**
11 **return** $\boldsymbol{R}$

---

shown to approximate this property well [87]. Property E_4 (pairwise independence of bits) is approximately satisfied since PCA produces orthogonal hyperplanes, a relaxed form of independence.

The computational bottleneck of ITQ lies in Line 6, where the SVD of a $K \times K$ matrix must be computed at each iteration. This requires $\mathcal{O}(K^3)$ operations, where K is the hashcode length. Once learned, the rotation matrix $\mathbf{R}$ can be applied to construct ITQ hashcodes for unseen queries $\mathbf{q} \in \mathbb{R}^D$ as defined in Equation 33.

$$g_l(\mathbf{q}) = \frac{1}{2}(1 + sgn(\mathbf{RW}^{\mathsf{T}}\mathbf{q})) \tag{33}$$

Here we assume the data has been mean-centered, so that the quantisation threshold is fixed at $t_k = 0$.

We close with two observations on ITQ. First, iterative two-step algorithms such as ITQ are a recurring and effective strategy for addressing otherwise intractable optimisation problems in this field. The need for such schemes stems from the NP-hard nature of directly learning optimal binary hashcodes. A common workaround, first seen in SH (Section 6.3.2), is to relax the binary constraint and learn a continuous embedding $\mathbf{Y} \in \mathbb{R}^{N_{\mathrm{trd}} \times K}$, followed by a separate quantisation step using SBQ or a more advanced binarisation scheme (Section 5). Since this relaxation inevitably produces sub-optimal hashcodes, the central challenge is to minimise the error introduced during the continuous-to-binary conversion by learning hyperplanes whose projections are well aligned with accurate binarisation. ITQ exemplifies this design pattern. Researchers have also begun exploring the harder problem of optimising binary hashcodes directly, without relaxation; see Section 6.4.2 and [50] for further discussion.

Figure 19: Illustration of the effect of ITQ on the feature space. The same data as in Figure 18 is shown after applying the learned rotation $\mathbf{R} \in \mathbb{R}^{K \times K}$ (100 iterations). The variance is more evenly distributed across the two hyperplanes (shown as perpendicular lines), and the quantisation error is reduced: hyperplanes no longer intersect cluster centres. This outcome reflects the optimisation objective of ITQ [27].

Second, we clarify why ITQ is categorised here as a *projection* rather than a *quantisation* method. In Section 5, we defined a quantisation algorithm as one that learns thresholds along a projected dimension and applies them directly to map real-valued projections into binary codes. ITQ does not fit this definition: it does not itself perform thresholding, but rather learns a rotation of the PCA hyperplanes so that subsequent SBQ (Section 5.1) can be applied more effectively. Under this taxonomy, ITQ is best regarded as a projection method, since its primary contribution is to manipulate the feature space to ensure that the resulting projections preserve the locality structure of the input space.

6.3.4 Anchor Graph Hashing (AGH)

We previously described the Hierarchical Quantisation (HQ) algorithm employed by Anchor Graph Hashing (AGH) in Section 5.2. In this section, we focus exclusively on the projection learning component of AGH. AGH optimises the same relaxed objective function as Spectral Hashing (SH), repeated here for convenience (Equation 34):

$$\begin{aligned}
\text{argmin}_{\mathbf{Y} \in \mathbb{R}^{N_{\text{trd}} \times K}} \quad & \text{tr}(\mathbf{Y}^{\mathsf{T}}(\mathbf{D} - \mathbf{S})\mathbf{Y}) \\
\text{subject to} \quad & \mathbf{Y} \in \mathbb{R}^{N_{\text{trd}} \times K}, \\
& \mathbf{Y}^{\mathsf{T}}\mathbf{1} = \mathbf{0}, \\
& \mathbf{Y}^{\mathsf{T}}\mathbf{Y} = N_{\text{trd}}\mathbf{I}^{K \times K}.
\end{aligned} \tag{34}$$

The computational bottlenecks of this formulation are twofold. First, constructing the similarity matrix $\mathbf{S} \in \mathbb{R}^{N_{\text{trd}} \times N_{\text{trd}}}$ requires $\mathcal{O}(N_{\text{trd}}^2 D)$ operations. Second, as with any hashing method, we require K hash functions $\{h_k : \mathbb{R}^D \to \{0,1\}\}_{k=1}^K$ to generate codes for unseen queries. Solving Equation 34 and binarising the resulting eigenvectors provides hashcodes only for the N_{trd} training points used to build $\mathbf{S}$; an efficient out-of-sample extension is therefore essential.

A naive Nyström extension ([6, 92]) would require $\mathcal{O}(N_{\text{trd}}K)$ operations per query, which is impractical for large-scale search. The key contribution of AGH is to replace $\mathbf{S}$ with a sparse, low-rank approximation $\hat{\mathbf{S}}$ constructed from an *anchor graph* $\mathbf{Z} \in \mathbb{R}^{N_{\text{trd}} \times C}$, where $C \ll N_{\text{trd}}$. This avoids explicit computation of $\hat{\mathbf{S}}$, reduces training and test complexity, and circumvents the unrealistic separable uniform distribution assumption made by SH.

The anchor graph $\mathbf{Z}$ stores similarities between each data-point and a small set of $R \ll C$ nearest anchor centroids $\{\mathbf{c}_i\}_{i=1}^C$, obtained via k-means clustering. The entries of $\mathbf{Z}$ are defined as:

$$Z_{ij} = \begin{cases}
\dfrac{\exp(-d^2(\mathbf{x}_j, \mathbf{c}_i)/\gamma)}{\sum\limits_{i' \in \langle j \rangle} \exp(-d^2(\mathbf{x}_j, \mathbf{c}_{i'})/\gamma)} & \text{if } i \in \langle j \rangle, \\
0 & \text{otherwise,}
\end{cases} \tag{35}$$

where γ is the kernel bandwidth, $d(\cdot, \cdot)$ is a distance metric, and $\langle j \rangle$ indexes the R nearest anchors of $\mathbf{x}_j$. Constructing $\mathbf{Z}$ costs $\mathcal{O}(N_{\text{trd}}CD)$, a significant improvement over $\mathcal{O}(N_{\text{trd}}^2 D)$.

From $\mathbf{Z}$, the similarity matrix is approximated as $\hat{\mathbf{S}} = \mathbf{Z}\Sigma^{-1}\mathbf{Z}^{\mathsf{T}}$, with $\Sigma = \text{diag}(\mathbf{Z}^{\mathsf{T}}\mathbf{1})$. The low-rank structure allows the eigenproblem to be solved on a reduced $C \times C$ matrix, $\Sigma^{1/2}\mathbf{Z}^{\mathsf{T}}\mathbf{Z}\Sigma^{-1/2}$, rather than the full $N_{\text{trd}} \times N_{\text{trd}}$ Laplacian.

Given a bit budget K, AGH selects $K' = K/2$ eigenvectors with the largest eigenvalues as hyperplane normals. Stacking these eigenvectors into $\mathbf{V}$ and the eigenvalues into Λ, the graph Laplacian eigenvectors are computed as:

$$\mathbf{Y} = \sqrt{N_{\text{trd}}}\,\mathbf{Z}\Sigma^{-1/2}\mathbf{V}\Lambda^{-1/2} = \mathbf{Z}\mathbf{W}, \tag{36}$$

with training complexity $\mathcal{O}(N_{\text{trd}}CK')$. The columns of $\mathbf{W} \in \mathbb{R}^{C \times K'}$ serve as hyperplane normals in the anchor space.

For an unseen query $\mathbf{q}$, the out-of-sample extension proceeds in two steps. First, $\mathbf{q}$ is non-linearly mapped to $\mathbb{R}^C$ by computing its similarity to the C centroids (Equation 35), producing a sparse vector $\mathbf{z} \in \mathbb{R}^C$ interpretable as a kernelised feature map ([60]). Second, $\mathbf{z}$ is linearly projected onto $\mathbf{w}_k \in \mathbb{R}^C$, yielding the AGH hash function:

$$h_k(\mathbf{q}) = \frac{1}{2}\big(1 + \mathrm{sgn}(\mathbf{w}_k^\mathsf{T}\mathbf{z})\big). \tag{37}$$

This construction can be interpreted as non-linearly embedding the data into a space where linear separation is easier, followed by thresholding. The test-time complexity is $\mathcal{O}(CD+CK')$, a dramatic improvement over $\mathcal{O}(N_{\mathrm{trd}}K)$ for the Nyström method.[18]

Finally, rather than generating one bit per dimension as in Equation 37, the most accurate variant of AGH applies Hierarchical Quantisation (HQ) to the projected dimensions $\mathbf{y}^k = Y_{\bullet k}$, producing two bits per dimension (see Section 5.2).

6.3.5 A Brief Summary

In our first exploration of data-dependent hashing algorithms, we surveyed several of the most widely studied *unsupervised* approaches that position the hashing hyperplanes according to the data distribution. Specifically, we reviewed Principal Components Analysis Hashing (PCAH) (Section 6.3.1), Spectral Hashing (SH) (Section 6.3.2), and Anchor Graph Hashing (AGH) (Section 6.3.4). All three models share a common foundation in dimensionality reduction, leveraging either Principal Components Analysis (PCA) or Laplacian Eigenmaps (LapEig) to derive hyperplanes.

Three out of the four unsupervised methods (PCAH, SH, ITQ) rely directly on PCA, setting the hashing hyperplanes to the right singular vectors obtained from a singular value decomposition (SVD) of the data matrix. Two of these methods (SH and ITQ) explicitly address the issue of *variance imbalance*, where hyperplanes capturing less variance prove unreliable for hashing. As a result, PCAH suffers from degraded retrieval effectiveness as hashcode length increases, since additional bits are drawn from lower-quality hyperplanes. SH mitigates this effect by allocating more bits to high-variance directions, while ITQ applies a learned rotation to redistribute the variance evenly across hyperplanes. Both strategies improve retrieval performance relative to vanilla PCAH, which simply assigns one bit per principal component without adjustment.

By contrast, AGH adopts a LapEig-inspired dimensionality reduction. Here, a neighbourhood graph is first constructed from the input data, and the resulting graph Laplacian eigenvectors are used to define the hashing hyperplanes. Since LapEig is inherently non-projective, the learned eigenvectors yield hashcodes only for the training points used in constructing the graph. A key contribution of AGH is its efficient out-of-sample extension, achieved via a Nyström approximation [92], which enables encoding of unseen query points at practical computational cost.

---

18. Assuming $D \ll N_{\mathrm{trd}}$, as is typical for common image features such as GIST and SIFT.

With the exception of AGH—which makes a deliberate attempt to reduce training complexity—the primary drawback of these methods lies in the heavy computational overhead of matrix factorisation. Solving the SVD or eigenvalue problem typically requires $\mathcal{O}(\min(N^2 D, N D^2))$ operations, rendering these approaches intractable for large-scale, high-dimensional datasets. As we will observe in the following sections, this reliance on matrix factorisation is a recurring theme across both supervised and unsupervised data-dependent hashing models.

6.4 Data-Dependent (Supervised) Projection Methods

In Section 4.3 and Sections 6.2–6.3, we reviewed representative data-independent and unsupervised data-dependent hashing models. Data-independent approaches preserve fixed similarity measures, such as cosine or kernel similarity, that are not data-adaptive and therefore often fail to align with user-defined similarity notions across diverse tasks. Unsupervised data-dependent models, in contrast, assume that discriminative hashcodes can be derived from projections that capture maximal variance in the input space. This assumption is problematic in practice: variance does not always distinguish between semantically different data-points, particularly in real-world image datasets collected "in the wild," where quality, resolution, and topic vary widely. The well-known *semantic gap* in computer vision highlights this mismatch between low-level image features (e.g., GIST, SIFT) and the high-level semantic concepts they are intended to represent [80]. Bridging this gap remains a central challenge in object recognition and image annotation [57], as well as in the broader task of image retrieval.

To address the semantic gap and better capture the complex relationships between data-points (e.g., whether two images depict the same semantic category such as a cat or a person), it is generally advantageous to incorporate even limited supervision. This supervision can take the form of class labels or pairwise constraints indicating whether two data-points should or should not share the same hashcodes. In the visual search domain, [29] identify a range of supervisory sources, from explicit label annotation, to known correspondences between images, to user feedback on search results.

We define a supervised hashing model as one that leverages the same type of information used to establish ground-truth similarity during evaluation (e.g., class labels, metric distances) directly within the hash function learning process. Figure 20 illustrates how supervision can guide hyperplane placement, producing more effective bucketings than purely variance-driven methods.

As in our review of unsupervised methods, we focus here on well-known supervised baselines whose implementations are publicly available, enabling reproducible and fair comparisons to our own methods. Specifically, we review ITQ with Canonical Correlation Analysis (ITQ+CCA) [27], Supervised Hashing with Kernels (KSH) [51], Binary Reconstructive Embedding (BRE) [47], and Self-Taught Hashing (STH) [97]. These methods differ primarily in how they exploit supervisory information

51

(a) **Unsupervised**   (b) **Supervised**

Figure 20: Supervised versus unsupervised projection function learning. Illustration of how pairwise user-provided constraints can yield a more effective bucketing of the space compared to variance-based partitioning. Shapes and colours denote 1-nearest neighbours. In Figure (a), the hyperplane $\mathbf{h}_1 \in \mathbb{R}^D$ is learnt via PCA, with its normal vector $\mathbf{w}_1 \in \mathbb{R}^D$ pointing in the direction of maximum variance. Projecting onto $\mathbf{w}_1$ separates related data-points (same shapes) into different buckets. By contrast, Figure (b) shows the effect of incorporating must-link (dotted lines) and cannot-link (solid lines) constraints, resulting in related data-points being grouped in the same bucket.

to construct an error signal that guides the positioning of the hashing hyperplanes. BRE and KSH, for example, minimise discrepancies between label information and hashcode distances. STH enforces similarity-preserving projections for points with the same label using a LapEig-inspired objective, while ITQ+CCA maximises correlations between data projections and label embeddings. Figure 21 summarises the relationships between these four models.

6.4.1 ITQ + CANONICAL CORRELATION ANALYSIS (CCA)

We reviewed the unsupervised variant of Iterative Quantisation (ITQ) in Section 6.3.3. ITQ learns an orthogonal rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$ that aligns PCA-projected data to the vertices of a binary hypercube, minimising quantisation error. Because ITQ is independent of the procedure used to obtain the orthogonal projection directions $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K]$, it can be adapted to the supervised setting by replacing PCA with a supervised embedding. [27] use Canonical Correlation Analysis (CCA) [33], a multi-view dimensionality reduction method, to learn hyperplanes that align data vectors in the input space $\mathcal{X}$ with label vectors in a second space $\mathcal{Z}$.

Assume N_{trd} training pairs $(\mathbf{x}_i, \mathbf{z}_i)$ arranged as row-centred matrices $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D_x}$ and $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times D_z}$ (typically $D_x \neq D_z$). Each row $\mathbf{z}i \in 0, 1^{D_z}$ is a multi-label indicator for $\mathbf{x}_i$. CCA finds directions $\mathbf{w}k \in \mathbb{R}^{D_x}$ and $\mathbf{u}_k \in \mathbb{R}^{D_z}$ such that the 1D projections

Figure 21: Relationship between the four supervised hashing models reviewed in this section. Arrows indicate transformations needed to convert one model into another. The core distinction lies in how supervisory information is translated into error signals for adjusting hashing hyperplanes.

$\mathbf{Xw}_k$ and $\mathbf{Zu}_k$ are maximally correlated:

$$\operatorname*{argmax}_{\mathbf{w}_k \in \mathbb{R}^{D_x},, \mathbf{u}_k \in \mathbb{R}^{D_z}} \frac{\mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{Z}, \mathbf{u}_k}{\sqrt{\mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X}, \mathbf{w}_k}; \sqrt{\mathbf{u}_k^\intercal \mathbf{Z}^\intercal \mathbf{Z}, \mathbf{u}_k}} \quad \text{s.t.} \quad \mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X}, \mathbf{w}_k = 1, \mathbf{u}_k^\intercal \mathbf{Z}^\intercal \mathbf{Z}, \mathbf{u}_k = 1. \tag{38}$$

This can be solved via the following regularised generalised eigenvalue problem [27]:

$$\mathbf{X}^\intercal \mathbf{Z} \left(\mathbf{Z}^\intercal \mathbf{Z} + \rho \mathbf{I} \right)^{-1} \mathbf{Z}^\intercal \mathbf{X} \mathbf{w}_k = \lambda_k^2 \left(\mathbf{X}^\intercal \mathbf{X} + \rho \mathbf{I} \right) \mathbf{w}_k \tag{39}$$

where λ_k is the k-th canonical correlation and ρ is a small Tikhonov regulariser (set to 0.0001 in [27]). Solving (39) for $k = 1, \ldots, K$ (with descending λ_k and the usual CCA orthogonality constraints) yields the supervised projection matrix $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_K] \in \mathbb{R}^{D_x \times K}$. (An analogous eigenproblem produces the $\mathbf{u}_k$ for $\mathcal{Z}$, which are not required for hashing in $\mathcal{X}$.)

With $\mathbf{W}$ fixed by CCA, ITQ proceeds exactly as in the unsupervised case (Section 6.3.3). Let $\mathbf{V} = \mathbf{XW} \in \mathbb{R}^{N_{trd} \times K}$ denote the supervised embedding of the training data. ITQ then solves

$$\min_{\mathbf{B},, \mathbf{R}}; |\mathbf{B} - \mathbf{VR}|_F^2 \quad \text{s.t.} \quad \mathbf{B} \in {-1, 1}^{Ntrd \times K}, \mathbf{R}^\intercal \mathbf{R} = \mathbf{I}, \tag{40}$$

by alternating updates: (i) $\mathbf{B} \leftarrow \text{sgn}(\mathbf{VR})$; (ii) $\mathbf{R}$ via the orthogonal Procrustes solution. Specifically, let $\mathbf{M} = \mathbf{B}^{\mathsf{T}}\mathbf{V}$ and compute the SVD $\mathbf{M} = \mathbf{U}, \boldsymbol{\Sigma}, \mathbf{V}^{\mathsf{T}}$; then $\mathbf{R} \leftarrow \mathbf{VU}^{\mathsf{T}}$. After a few iterations, quantisation error stabilises and the binary codes are $\mathbf{B} = \text{sgn}(\mathbf{XWR})$.

Complexity. Writing $D = \max(D_x, D_z)$, the overall complexity of ITQ+CCA is $\mathcal{O}(N_{trd}D^2 + D^3)$: $\mathcal{O}(N_{trd}D^2)$ to form the covariance and cross-covariance matrices $(\mathbf{X}^{\mathsf{T}}\mathbf{X}, \mathbf{Z}^{\mathsf{T}}\mathbf{Z}, \mathbf{X}^{\mathsf{T}}\mathbf{Z})$, and $\mathcal{O}(D^3)$ for the matrix inversions/multiplications and the eigen-decomposition [69]. The subsequent ITQ iterations are dominated by multiplications in $\mathbb{R}^{N_{trd} \times K}$ and SVD of a $K \times K$ matrix, which is negligible when $K \ll D$.

Discussion. Replacing PCA with CCA biases the projection $\mathbf{W}$ toward directions that are maximally correlated with the supervisory signal. The subsequent ITQ rotation $\mathbf{R}$ then reduces quantisation error in this supervised subspace. In practice, ITQ+CCA tends to improve semantic neighbourhood preservation relative to unsupervised ITQ, while approximately maintaining the desirable hashing properties E_1–E_4 discussed earlier.

6.4.2 Binary Reconstructive Embedding (BRE)

Binary Reconstructive Embedding (BRE) differs from most hashing methods in that it does *not* rely on spectral relaxation to sidestep the NP-hard problem of learning binary hashcodes directly. Recall from Section 6.3.2 that dropping the sign function and relaxing the binary constraints leads to continuous but ultimately approximate solutions. BRE retains the sign function, tackling the discrete optimisation problem head-on.

Most prior approaches solve for a real-valued projection matrix $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$ and then apply single-bit quantisation (SBQ) to obtain binary codes, a two-stage process with no guarantee that values close to the threshold will yield stable or accurate hashcodes. BRE integrates both stages into a single objective, directly optimising over binary outputs. The supervised objective is given in Equation 41:

$$\text{argmin}_{\mathbf{W} \in \mathbb{R}^{D \times K}} \sum_{ij \in \mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}} \left\{ (1 - S_{ij}) - \frac{1}{K}\|g(\mathbf{x}_{we}) - g(\mathbf{x}_j)\|_2^2 \right\}^2$$
$$\text{subject to } g(\mathbf{x}_{we}) = [h_1(\mathbf{x}_{we}), h_2(\mathbf{x}_{we}), \ldots, h_k(\mathbf{x}_{we})]^{\mathsf{T}} \tag{41}$$
$$\text{where } h_k(\mathbf{x}_{we}) = \frac{1}{2}sgn\left(1 + \sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j, \mathbf{x}_{we})\right)$$

Here, $\mathbf{S} \in \{0, 1\}^{N_{trd} \times N_{trd}}$ is an adjacency matrix, where $S_{ij} = 1$ indicates that $\mathbf{x}_{we}$ and $\mathbf{x}_j$ are related. From the dataset, N_{trd} data-points are sampled to construct $\mathbf{S}$, and C data-points are sampled uniformly at random as anchor points for kernel computation.

$\mathbf{W} \in \mathbb{R}^{C \times K}$ is initialised randomly, and κ is a kernel function $\{\kappa : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}\}$ (linear in the original publication [47]).

The objective in Equation 41 encourages low normalised Hamming distance for similar pairs ($S_{ij} = 1$) and high distance otherwise. Unlike earlier methods, BRE directly retains the sign function in the optimisation, avoiding a disconnect between projection learning and binarisation.

BRE uses a kernelised feature map $\kappa(\cdot, \cdot)$ against C anchor points, projecting data onto hyperplane normals $\{\mathbf{w}_k \in \mathbb{R}^C\}_{k=1}^K$. This formulation resembles Anchor Graph Hashing (AGH, Equation 37), with the key difference that AGH employs k-means centroids and RBF kernels. A more expressive kernel than the linear choice in [47] may improve retrieval effectiveness, though this was not explored in the original work.

Optimising Equation 41 is challenging due to its non-differentiability. [47] propose a coordinate descent algorithm that cycles through the K hash functions. For each, a randomly chosen element W_{jk} of hyperplane $\mathbf{W}_{\bullet k}$ is updated while holding others fixed. A closed-form solution for the optimal W_{jk} can be computed in $\mathcal{O}(N_{trd}^2)$ time. Iterating through all K hash functions requires $\mathcal{O}(KN_{trd}^2 + KN_{trd} \log N_{trd})$ operations.[19]

BRE satisfies properties E_1–E_2 of an effective hashcode (Section 6.1) but does not explicitly enforce E_3 (bit balance) or E_4 (bit independence). In particular, coordinate descent updates introduce correlations between bits since each hash function is sequentially optimised based on previously updated hyperplanes. Nonetheless, BRE's direct treatment of the discrete optimisation problem was an important milestone, and its influence persists in more recent work that similarly avoids spectral relaxation [50, 78].

6.4.3 SUPERVISED HASHING WITH KERNELS (KSH)

Supervised Hashing with Kernels (KSH) [51] adopts a kernelised hash function similar in spirit to AGH (Section 6.3.4) and BRE (Section 6.4.2), but couples it with a distinct, spectrally relaxed optimisation strategy. In practice, KSH often attains state-of-the-art retrieval effectiveness among supervised hashing baselines and is frequently used as a de facto comparison point on standard image retrieval benchmarks. The kernelised hash function is

$$h_k(\mathbf{q}) = \text{sgn}\Big(\sum_{j=1}^C W_{jk}\, \kappa(\mathbf{x}_j, \mathbf{q}) + t_k\Big), \tag{42}$$

where $\kappa : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is a kernel function, $t_k \in \mathbb{R}$ is a scalar threshold, and $\mathbf{W} \in \mathbb{R}^{C \times K}$ collects K hyperplane normals in the anchor space. As in AGH and BRE, a small set of C representative data-points ($C \ll N$) is sampled to serve as

---

19. For clarity, this assumes each of the N_{trd} data-points forms $N_{trd}-1$ supervisory pairs. In practice, BRE samples a smaller set of pairs (e.g., $0.05N_{trd}$) for tractability.

kernel anchors, and N_{trd} data-points ($C < N_{\text{trd}} \ll N$) is used to form a supervised adjacency matrix $\mathbf{S} \in \{-1, 1\}^{N_{\text{trd}} \times N_{\text{trd}}}$ for training.

KSH minimises the discrepancy between supervised similarity labels and inner products of (relaxed) hashcodes. Compared to BRE's objective, KSH removes the sign function during optimisation and measures agreement via inner products rather than Euclidean distances:

$$
\begin{aligned}
\underset{\mathbf{W} \in \mathbb{R}^{C \times K}}{\text{argmin}} \quad & \sum_{ij} \left\{ S_{ij} - \tfrac{1}{K} g(\mathbf{x}_i)^\top g(\mathbf{x}_j) \right\}^2 \\
\text{subject to} \quad & g(\mathbf{x}_i) = \left[h_1(\mathbf{x}_i), \ldots, h_K(\mathbf{x}_i) \right]^\top, \\
& h_k(\mathbf{x}_i) = \text{sgn}\Big(\sum_{j=1}^{C} W_{jk}\, \kappa(\mathbf{x}_j, \mathbf{x}_i) \Big).
\end{aligned}
\tag{43}
$$

To obtain an efficient optimisation, KSH relaxes the sign and optimises over a continuous space, enabling gradient-based updates. It further learns the K hash functions sequentially, each time initialising the current hyperplane by solving an eigenproblem and then refining it via gradient descent. Rewriting the relaxed objective to expose this per-bit optimisation yields:

$$
\begin{aligned}
\underset{\mathbf{W} \in \mathbb{R}^{C \times K}}{\text{argmin}} \quad & \sum_{k=1}^{K} \left\| K\mathbf{S} - \mathbf{y}^k (\mathbf{y}^k)^\top \right\|_F^2 \\
\text{where} \quad & y_i^k = \sum_{j=1}^{C} W_{jk}\, \kappa(\mathbf{x}_j, \mathbf{x}_i),
\end{aligned}
\tag{44}
$$

and we assume mean-centred data so that $t_k = 0$. Here, $\mathbf{y}^k$ denotes the k-th column of the projection matrix $\mathbf{Y} \in \mathbb{R}^{N_{\text{trd}} \times K}$.

Rather than solve the K problems independently, KSH, like BRE, introduces inter-bit dependency via a residue that emphasises pairs misfit by previously learned bits. Specifically, define the residue

$$
\mathbf{R}^{k-1} = K\mathbf{S} - \sum_{\ell=1}^{k-1} \mathbf{y}^\ell (\mathbf{y}^\ell)^\top.
\tag{45}
$$

Pairs with many mismatched signs in earlier bits contribute larger entries to $\mathbf{R}^{k-1}$ and thus have greater influence when learning bit k. With this residue, [51] reduce the per-bit optimisation to:

$$
\begin{aligned}
\underset{\mathbf{w}_k \in \mathbb{R}^C}{\text{argmax}} \quad & \left(\mathbf{K}\mathbf{w}_k \right)^\top \mathbf{R}^{k-1} \left(\mathbf{K}\mathbf{w}_k \right) \\
\text{where} \quad & K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j), \\
\text{subject to} \quad & \left(\mathbf{K}\mathbf{w}_k \right)^\top \left(\mathbf{K}\mathbf{w}_k \right) = L,
\end{aligned}
\tag{46}
$$

with $\mathbf{K} \in \mathbb{R}^{N_{\text{trd}} \times C}$ the kernel matrix between training points and anchors. Comparing Equation (46) to the trace-optimisation template (Equation 24) reveals that the initial solution for $\mathbf{w}_k$ is the leading generalised eigenvector of

$$\mathbf{K}^\top \mathbf{R}^{k-1} \mathbf{K} \mathbf{w}_k = \lambda \mathbf{K}^\top \mathbf{K} \mathbf{w}_k,$$

which is then refined by gradient descent using a smooth (e.g., sigmoid) relaxation of the sign.

Despite being a non-linear model, KSH remains computationally tractable by (i) restricting N_{trd} and C to modest sizes[20], and (ii) operating in the continuous (real-valued) domain during optimisation. The overall time complexity is

$$\mathcal{O}\big(NCK + N_{\text{trd}}^2 CK + N_{\text{trd}} C^2 K + C^3 K\big),$$

accounting for kernel evaluations, residue updates, matrix multiplications, and per-bit eigenproblems. While KSH does not explicitly enforce E_3 (bit balance) or E_4 (bit independence), it satisfies E_1 (neighbourhood preservation) and E_2 (efficient out-of-sample evaluation), producing highly discriminative hashcodes with fast query-time hash computation.

6.4.4 SELF-TAUGHT HASHING (STH)

Self-Taught Hashing (STH) ([97]) employs a two-step procedure for learning hashing hyperplanes. The first step performs a Laplacian Eigenmap dimensionality reduction, while the second learns hyperplanes for out-of-sample extension to unseen queries. STH is therefore reminiscent of the unsupervised data-dependent models Spectral Hashing (SH) (Section 6.3.2) and Anchor Graph Hashing (AGH) (Section 6.3.4). In the first step, K graph Laplacian eigenvectors are extracted from $\mathbf{L} = \mathbf{D} - \mathbf{S}$ by solving the relaxed optimisation in Equation 47:

$$
\begin{aligned}
\text{argmin}_{\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}} \quad & tr(\mathbf{Y}^\top(\mathbf{D} - \mathbf{S})\mathbf{Y}) \\
\text{subject to } & \mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}, \quad \mathbf{Y}^\top \mathbf{D} \mathbf{1} = \mathbf{0}, \quad \mathbf{Y}^\top \mathbf{D} \mathbf{Y} = N_{trd} \mathbf{I}^{K \times K}
\end{aligned}
\tag{47}
$$

where $tr(A) = \sum_i A_{ii}$ is the trace, $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ is a neighbourhood graph formed from class labels ($S_{ij} = 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ share a label, and 0 otherwise), and $\mathbf{D}$ is the diagonal degree matrix with $D_{ii} = \sum_j S_{ij}$. For tractability $N_{trd} \ll N$. Compared with SH (Equation 28), STH introduces $\mathbf{D}$ into the constraints, yielding a *normalised cut* rather than a ratio cut ([1]). This formulation is equivalent to Laplacian Eigenmaps ([5]), with solutions given by the lowest eigenvectors of $L\mathbf{y}^k = \lambda \mathbf{D} \mathbf{y}^k$. The eigenproblem can be solved in $\mathcal{O}(MN_{trd}^2 K)$ using M iterations of the Lanczos algorithm ([26, 97]). As in BRE (Section 6.4.2), STH can also be framed in an unsupervised variant by constructing $\mathbf{S}$ from pairwise distances rather than labels. Solving

---

20. Typical choices are $N_{\text{trd}} \approx 1000$ and $C \approx 300$.

Equation 47 approximately preserves hashcode properties E_3 (balanced partitions) and E_4 (bit independence).

As with SH and AGH, the spectral embedding $\mathbf{Y}$ must be binarised to produce hashcodes, and initially encodes only the N_{trd} training data-points. Rather than use the Nyström method ([6, 92]) as in AGH, or the uniform distribution assumption of SH, STH makes the novel contribution of learning K binary linear Support Vector Machines (SVMs) to directly predict the bits of the binarised spectral embedding with maximum margin. These SVMs provide the required hyperplane normals $\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^K$ for out-of-sample extension. Training K linear SVMs requires $\mathcal{O}(N_{trd}DK)$ time ([38]), while encoding a novel query takes $\mathcal{O}(DK)$. This approach makes STH computationally efficient at test time, while leveraging supervised constraints in the graph construction for higher retrieval effectiveness.

6.4.5 A Brief Summary

We have reviewed four of the most widely studied approaches for incorporating supervision into the learning of hashing hyperplanes for unimodal ANN search. The models considered include ITQ+CCA (Section 6.4.1), Binary Reconstructive Embedding (BRE) (Section 6.4.2), Supervised Hashing with Kernels (KSH) (Section 6.4.3), and Self-Taught Hashing (STH) (Section 6.4.4). Each of these models learns a set of K hyperplanes guided by must-link and cannot-link constraints: must-link pairs should map to the same hashtable bucket, while cannot-link pairs should be separated. For example, two images of cats (must-link) should be collocated, whereas an image of a dog (cannot-link) should be placed in a different bucket.

The key differences between the models arise in how the available supervision is related to the projections or hashcodes to generate an error signal for hyperplane adjustment. BRE and KSH, for instance, both minimise the discrepancy between labels and hashcode similarities, but differ in their formulations: BRE minimises the gap with respect to Hamming distances, while KSH focuses on inner products of the real-valued projections. Their optimisation strategies also diverge significantly. BRE retains the discrete sign function and directly tackles the NP-hard optimisation with a coordinate descent algorithm. KSH instead relaxes the objective into a continuous domain and applies gradient-based optimisation. ITQ+CCA and STH take yet different routes, using CCA embeddings or SVM-based out-of-sample extensions respectively. Together, these approaches illustrate the breadth of strategies available for supervised hyperplane learning.

6.5 Cross-Modality Projection Methods

All models discussed so far—including LSH and SKLSH (Sections 4.3, 6.2.1) and the data-dependent models of Sections 6.3–6.4—address only *unimodal* retrieval, where queries and database entries share the same feature representation. In this case, the learned hyperplanes partition a single modality. This is restrictive, since much real-

world data is inherently multi-modal[21]. For example, an image on Flickr[22] may be described not only by raw pixel values, but also by user-assigned tags and geolocation metadata. Ideally, a retrieval system would allow queries across modalities—for instance, querying with an image to retrieve relevant text tags (Figure 22), or querying with GPS coordinates to retrieve corresponding images.

Figure 22: Cross-modal hashing-based ANN search. In cross-modal hashing, we wish to partition the input space so that semantically similar data-points across modalities fall into the same hashtable buckets. Here, cross-modal hash functions $\mathcal{H}$ assign similar hashcodes to images and documents, enabling fast retrieval of related data across modalities.

Cross-modal hashing models extend unimodal methods by learning two sets of K hyperplanes—$\mathbf{W} \in \mathbb{R}^{D_x \times K}$ for modality $\mathcal{X}$ and $\mathbf{U} \in \mathbb{R}^{D_z \times K}$ for modality $\mathcal{Z}$. Let $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D_x}$ and $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times D_z}$ denote the feature matrices, where $D_x \neq D_z$ in general, and N_{xz} the number of paired data-points across modalities ($N_{xz} \leq N_{trd}$). An adjacency matrix $\mathbf{S} \in \{0,1\}^{N_{xz} \times N_{xz}}$ encodes semantic relationships across modalities ($S_{ij} = 1$ if $(\mathbf{x}_i, \mathbf{z}_j)$ are related). The goal is to ensure that $d_H(g_{\mathcal{X}}(\mathbf{x}_i), g_{\mathcal{Z}}(\mathbf{z}_j)) \approx 0$ when $S_{ij} = 1$, and large otherwise. This naturally extends the unimodal case by enforcing *consistency* across two modalities (Figure 23).

---

21. We use the terms 'modality' and 'feature space' interchangeably.
22. http://www.flickr.com

(a) **Modality** $\mathcal{X}$ (b) **Modality** $\mathcal{Z}$

Figure 23: The essence of cross-modal hashing. In (a) we show feature space $\mathcal{X}$ (e.g. image descriptors) with hyperplanes $\{\mathbf{w}_k\}_{k=1}^{K}$; in (b) we show feature space $\mathcal{Z}$ (e.g. text descriptors) with hyperplanes $\{\mathbf{u}_k\}_{k=1}^{K}$. Similar data-points (same colour/shape) should be assigned the same hashcodes within and across modalities.

Following the same principles as in previous sections, we restrict our review to well-established baselines with widely available implementations. The five representative cross-modal methods we consider are: Cross-View Hashing (CVH) ([49], Section 6.5.1), Co-Regularised Hashing (CRH) ([98], Section 6.5.2), Predictable Dual-View Hashing (PDH) ([70], Section 6.5.4), Inter-Media Hashing (IMH) ([81], Section 6.5.5), and Cross-Modal Semi-Supervised Hashing (CMSSH) ([12], Section 6.5.3).

6.5.1 Cross-View Hashing (CVH)

Cross-View Hashing (CVH)[23] [49] mirrors ITQ+CCA (Section 6.4.1) in using Canonical Correlation Analysis (CCA) to learn two sets of hyperplanes that maximise cross-modal correlation. There are two main differences relative to ITQ+CCA: (i) CVH retains *both* sets of hyperplane normals, $\mathbf{W} \in \mathbb{R}^{D_x \times K}$ and $\mathbf{U} \in \mathbb{R}^{D_z \times K}$, rather than only those of the visual modality; and (ii) CVH does not apply the post-hoc ITQ rotation to balance variance across hyperplanes. The hash functions for the two modalities are

$$
\begin{aligned}
h_k^{\mathcal{X}}(\mathbf{x}_i) &= \tfrac{1}{2}\big(1 + \operatorname{sgn}(\mathbf{w}_k^{\top}\mathbf{x}_i)\big), \\
h_k^{\mathcal{Z}}(\mathbf{z}_i) &= \tfrac{1}{2}\big(1 + \operatorname{sgn}(\mathbf{u}_k^{\top}\mathbf{z}_i)\big).
\end{aligned}
\tag{48}
$$

As with ITQ+CCA, the asymptotic training complexity is $\mathcal{O}(N_{trd}D^2 + D^3)$ with $D = \max(D_x, D_z)$. CVH is one of the earliest cross-modal hashing methods and is widely used as a baseline. The models reviewed next (Sections 6.5.2–6.5.5) introduce

---

23. As is standard in the literature, we adopt the special case of CVH with only cross-modality supervision and one-to-one paired samples across modalities (Section 3.2 in [49]).

Figure 24: Relationship between the five cross-modal hashing models reviewed. Arcs denote transformations necessary to convert one model into another (ignoring intra-modal and out-of-sample-extension terms).

alternative learning schemes for both hyperplane sets that typically achieve higher accuracy on standard image–text benchmarks.

6.5.2 CO-REGULARISED HASHING (CRH)

Co-Regularised Hashing (CRH) [98] learns $2K$ cross-modal hash functions via K sequential max-margin problems. At each step, boosting [23] reweights training pairs so that those not satisfied by previously learned hyperplanes are emphasised in subsequent bits—inducing inter-bit dependency, akin to the unimodal KSH (Section 6.4.3). CRH uses the same linear hash form as CVH (Equation 48). For bit k, the objective over $\mathbf{w}_k \in \mathbb{R}^{D_x}$ and $\mathbf{u}_k \in \mathbb{R}^{D_z}$ contains three components: intra-modal max-margin terms for each modality, a cross-modal consistency term, and ℓ_2 regularisation:

$$
\underset{\mathbf{w}_k, \mathbf{u}_k}{\operatorname{argmin}} \quad \frac{1}{N_x} \sum_{i=1}^{N_x} \left[1 - |\mathbf{w}_k^\top \mathbf{x}_i| \right]_+ \; + \; \frac{1}{N_z} \sum_{j=1}^{N_z} \left[1 - |\mathbf{u}_k^\top \mathbf{z}_j| \right]_+
$$

$$
+ \; \gamma \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} \left(\mathbf{w}_k^\top \mathbf{x}_i - \mathbf{u}_k^\top \mathbf{z}_j \right)^2 \; + \; \frac{\lambda_x}{2} \|\mathbf{w}_k\|^2 \; + \; \frac{\lambda_z}{2} \|\mathbf{u}_k\|^2,
\tag{49}
$$

where $\alpha_{ij} \geq 0$ are AdaBoost weights, $\lambda_x, \lambda_z > 0$ regularise the hyperplanes, $[a]_+ = \max(a, 0)$, and $\gamma > 0$ controls the cross-modal penalty.[24] The intra-modal terms

---

24. In practice CRH also includes a repulsion term for dissimilar pairs; we omit it here for clarity.

promote margins so that points are not projected too close to the decision boundaries, while the cross-modal term penalises discrepancies in projections for related pairs.

Since (49) is non-convex, [98] adopt alternating minimisation: fix $\mathbf{w}_k$ and optimise $\mathbf{u}_k$, and vice versa, each framed as a difference-of-convex program solved via the Concave–Convex Procedure (CCCP) [96]. After learning $(\mathbf{w}_k, \mathbf{u}_k)$, AdaBoost updates the pair weights $\{\alpha_{ij}\}$ based on the disagreement rate of the current cross-modal hash functions, and the process repeats for $k+1$. The overall complexity is $\mathcal{O}(KMND)$, where $D = \max(D_x, D_z)$ and M is the number of Pegasos iterations [77]. CRH satisfies properties E_1 and E_2 (Section 6.1), but does not explicitly enforce E_3 or E_4.

6.5.3 Cross-Modal Similarity-Sensitive Hashing (CMSSH)

Cross-Modal Similarity-Sensitive Hashing (CMSSH) [12] adopts a simpler optimisation than CRH, targeting only cross-modal consistency (no explicit intra-modal terms). It learns $2K$ hash functions sequentially; at step k, the pair of hyperplanes $(\mathbf{w}_k, \mathbf{u}_k)$ is chosen to correct mistakes made by earlier bits via a weighted objective:

$$\operatorname*{argmax}_{\mathbf{w}_k \in \mathbb{R}^{D_x}, \, \mathbf{u}_k \in \mathbb{R}^{D_z}} \quad \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} \, \operatorname{sgn}(\mathbf{w}_k^\top \mathbf{x}_i) \, \operatorname{sgn}(\mathbf{u}_k^\top \mathbf{z}_j), \tag{50}$$

where $\alpha_{ij} \geq 0$ are AdaBoost weights. This is akin to maximising the correlation of signed projections for related cross-modal pairs. Because (50) is non-differentiable, [12] apply the standard spectral relaxation (drop the signs), yielding

$$\operatorname*{argmax}_{\mathbf{w}_k, \mathbf{u}_k} \quad \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} \left(\mathbf{w}_k^\top \mathbf{x}_i\right) \left(\mathbf{u}_k^\top \mathbf{z}_j\right) \; = \; \mathbf{w}_k^\top \left(\sum_{i,j} \alpha_{ij} S_{ij} \, \mathbf{x}_i \mathbf{z}_j^\top\right) \mathbf{u}_k$$
$$= \; \mathbf{w}_k^\top \mathbf{C} \, \mathbf{u}_k, \tag{51}$$

which is solved in closed form via the SVD of $\mathbf{C} \in \mathbb{R}^{D_x \times D_z}$ at cost $\mathcal{O}(D_x^2 D_z + D_x D_z^2 + D_z^3)$.[25] After each bit is learned, the weights $\{\alpha_{ij}\}$ are updated (AdaBoost) to emphasise misclassified pairs, and the process repeats. The learned hyperplanes are then used in the linear hash function (Equation 48). Like CRH, CMSSH preserves E_1 and E_2 but not E_4, due to boosting-induced bit dependencies.

6.5.4 Predictable Dual-View Hashing (PDH)

Like CRH and CMSSH, PDH learns the $2K$ hyperplanes sequentially, solving for $(\mathbf{w}_k, \mathbf{u}_k)$ at step k. PDH employs an SVM-based formulation in which hyperplanes are trained to separate points with opposing bits at maximum margin. Unlike boosting-based methods, PDH explicitly enforces pairwise bit independence (property E_4). Its

---

25. Because $\mathbf{C}$ is generally rectangular, SVD (rather than an eigenproblem) is used.

Algorithm 5: PREDICTABLE DUAL-VIEW HASHING (PDH) [70]

Input: Data $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D_x}$, $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times D_z}$; iterations M
Output: Hyperplanes $\mathbf{W} \in \mathbb{R}^{D_x \times K}$, $\mathbf{U} \in \mathbb{R}^{D_z \times K}$

1 Initialise $\mathbf{W}, \mathbf{U}$ via CCA on $(\mathbf{X}, \mathbf{Z})$
2 for $m \leftarrow 1$ **to** M **do**
3   **for** $k \leftarrow 1$ **to** K **do**
4 $\mathbf{b}_k^x = \mathbf{B}_{\bullet k}^x$,
5 $\mathbf{b}_k^z = \mathbf{B}_{\bullet k}^z$
6   Train SVM_k^x on $\mathbf{X}$ with labels $\mathbf{b}_k^z$
7   Obtain hyperplane $\mathbf{w}_k$ and set $\mathbf{W}_{\bullet k} = \mathbf{w}_k$
8   Update $\mathbf{B}_{\bullet k}^x = \text{sgn}(\mathbf{X}\mathbf{w}_k)$
9   Train SVM_k^z on $\mathbf{Z}$ with labels $\mathbf{b}_k^x$
10   Obtain hyperplane $\mathbf{u}_k$ and set $\mathbf{U}_{\bullet k} = \mathbf{u}_k$
11   Update $\mathbf{B}_{\bullet k}^z = \text{sgn}(\mathbf{Z}\mathbf{u}_k)$
12   **end**
13   Update $\mathbf{B}^x, \mathbf{B}^z$ by solving the eigenproblem in Equation 53
14 end
15 return $\boldsymbol{W}, \boldsymbol{U}$

---

objective is:

$$
\underset{\mathbf{W} \in \mathbb{R}^{D_x \times K}, \mathbf{U} \in \mathbb{R}^{D_z \times K}}{\operatorname{argmin}} \quad \|\mathbf{B}^x (\mathbf{B}^x)^\top - \mathbf{I}\|_2^2 \;+\; \|\mathbf{B}^z (\mathbf{B}^z)^\top - \mathbf{I}\|_2^2 \;+\; \sum_{k=1}^K \|\mathbf{w}_k\|^2 \;+\; \sum_{k=1}^K \|\mathbf{u}_k\|^2
$$

$$
+ \; C_x \sum_{i,k} \xi_{ik}^x \;+\; C_z \sum_{i,k} \xi_{ik}^z
$$

$$
\text{subject to} \quad \mathbf{B}^x = \text{sgn}(\mathbf{X}\mathbf{W}), \quad \mathbf{B}^z = \text{sgn}(\mathbf{Z}\mathbf{U}),
$$

$$
b_{ik}^z (\mathbf{w}_k^\top \mathbf{x}_i) \geq 1 - \xi_{ik}^x, \quad b_{ik}^x (\mathbf{u}_k^\top \mathbf{z}_i) \geq 1 - \xi_{ik}^z,
$$

(52)

where $\xi_{ik}^x, \xi_{ik}^z \in \mathbb{R}$ are slack variables; $C_x, C_z > 0$ trade margin size $(1/\|\mathbf{w}_k\|, 1/\|\mathbf{u}_k\|)$ against violations. The first two terms promote pairwise bit independence by driving off-diagonal correlations toward zero. The last two constraints are max-margin, but note the key cross-supervision: bits from one modality (b_{ik}^z or b_{ik}^x) serve as the training labels for the SVM in the *other* modality—encouraging cross-view consistency over iterations.

PDH alternates the steps in Algorithm 5. First, using current codes $\mathbf{B}^x, \mathbf{B}^z \in \{-1, 1\}^{N_{trd} \times K}$ as labels, train $2K$ SVMs (Lines 6, 9) to update $\mathbf{W}, \mathbf{U}$. Relabel the codes with the resulting hyperplanes (Lines 8, 11), flipping signs for points on the wrong side. Then approximately enforce bit independence by solving the standard

graph Laplacian problem

$$
\underset{\mathbf{Y}_l \in \mathbb{R}^{N_{trd} \times K}}{\operatorname{argmin}} \quad \operatorname{tr}\big(\mathbf{Y}_l^\top (\mathbf{D}_l - \mathbf{S}_l)\mathbf{Y}_l\big)
$$
$$
\text{subject to} \quad \mathbf{Y}_l^\top \mathbf{1} = \mathbf{0}, \quad \mathbf{Y}_l^\top \mathbf{Y}_l = N_{trd}\mathbf{I}, \tag{53}
$$

for $l \in \{x, z\}$, where $\mathbf{S}_l = \mathbf{Y}_l \mathbf{Y}_l^\top$ and $(\mathbf{D}_l)_{ii} = \sum_j (\mathbf{S}_l)_{ij}$. As in Section 6.3, the solution comprises the K smallest-eigenvalue eigenvectors (Line 13), which are then binarised to refresh the codes. Intuitively, this step reduces pairwise correlations between rows of $\mathbf{Y}_l$ while preserving local structure under the inner-product affinity. The loop repeats for M iterations. At convergence, the learned hyperplanes are used in the linear hash function (Equation 48). Training is dominated by solving the eigenproblems across iterations, at $\mathcal{O}(MN_{trd}^2 K)$ with a Lanczos solver [26].

6.5.5 Inter-Media Hashing (IMH)

Inter-Media Hashing (IMH) ([81]) is best understood as a semi-supervised cross-modal hashing model. Unlike earlier methods that rely exclusively on pairwise supervisory information encoded in the cross-modal adjacency matrix $\mathbf{S}^{xz} \in \{0,1\}^{N_{xz} \times N_{xz}}$, IMH also leverages unsupervised structure captured from all N_{trd} data-points *within* each modality, including those absent from $\mathbf{S}^{xz}$. This additional information is represented by a Euclidean k-NN graph constructed separately in each modality. For example, in modality $\mathcal{X}$ the adjacency matrix $\mathbf{S}^x \in \{0,1\}^{N_{trd} \times N_{trd}}$ is given by Equation 54, with an analogous definition for $\mathbf{S}^z$ in modality $\mathcal{Z}$:

$$
S_{ij}^x = \begin{cases} 1 & \text{if } \mathbf{x}_{we} \in NN_k(\mathbf{x}_j) \text{ or } \mathbf{x}_j \in NN_k(\mathbf{x}_{we}) \\ 0 & otherwise \end{cases} \tag{54}
$$

where $NN_k(\mathbf{x}_{we})$ denotes the set of k-nearest neighbours of $\mathbf{x}_{we}$ under a chosen distance metric, typically Euclidean.

This semi-supervised strategy contrasts with CVH, CMSSH, and PDH, which depend solely on $\mathbf{S}^{xz}$. IMH is more closely related to CRH (Section 6.5.2), which also incorporates unsupervised regularisation terms. In the limiting case where learning is restricted to $\mathbf{S}^{xz}$ alone, and $\mathbf{S}^{xz} = \mathbf{we}^{N_{xy} \times N_{xy}}$, IMH reduces to the CCA-based CVH model (Section 6.5.1). The IMH objective is given in Equation 55:

$$
\operatorname{argmin}_{\mathbf{W},\mathbf{U},\mathbf{Y}^x,\mathbf{Y}^z} \quad \lambda \sum_{we,j=1}^{N_{trd}} S_{ij}^x \|\mathbf{y}_{we}^x - \mathbf{y}_j^x\|_2^2 + \lambda \sum_{we,j=1}^{N_{trd}} S_{ij}^z \|\mathbf{y}_{we}^z - \mathbf{y}_j^z\|_2^2 + \sum_{we,j=1}^{N_{xz}} S_{ij}^{xz} \|\mathbf{y}_{we}^x - \mathbf{y}_j^z\|_2^2
$$
$$
+ \sum_{we=1}^{N_{trd}} \|\mathbf{W}^\intercal \mathbf{x}_{we} - \mathbf{y}_{we}^x\|_2^2 + \beta\|\mathbf{W}\|_F^2 + \sum_{j=1}^{N_{trd}} \|\mathbf{U}^\intercal \mathbf{z}_j - \mathbf{y}_j^z\|_2^2 + \beta\|\mathbf{U}\|_F^2
$$
$$
\text{subject to} \ (\mathbf{Y}^x)^\intercal \mathbf{Y}^x = \mathbf{we}^{D_x \times D_x}, \quad (\mathbf{Y}^x)^\intercal \mathbf{1} = \mathbf{0} \tag{55}
$$

where $\mathbf{S}^{xz} = \mathbf{we}^{N_{xz} \times N_{xz}}$, and $\beta, \lambda \in \mathbb{R}$ are user-specified parameters controlling the influence of different terms.

The intuition is straightforward. The first two terms ensure that similar data-points within each modality are mapped to nearby projections, echoing the Laplacian Eigenmap formulation (Equation 28) first introduced in SH (Section 6.3.2) and later extended to the multi-view case in CVH. The third term enforces cross-modal consistency by aligning projections of paired data-points, reminiscent of CRH's inter-modal loss but without per-pair boosting weights. The final regression terms provide out-of-sample extension, yielding hashing hyperplanes $\mathbf{W} \in \mathbb{R}^{D_x \times K}$ and $\mathbf{U} \in \mathbb{R}^{D_z \times K}$ through L_2-regularised linear regression, with the spectral embeddings as targets.

Optimisation proceeds by recasting the problem as a trace minimisation involving $\mathbf{Y}^x$, solved via eigen-decomposition. The K eigenvectors with the smallest eigenvalues form the columns of $\mathbf{Y}^x$, automatically satisfying orthogonality. Once $\mathbf{Y}^x$ is fixed, closed-form solutions are available for $\mathbf{Y}^z$ and the regression matrices $\mathbf{W}, \mathbf{U}$ ([81]).

As with other matrix-factorisation-based models, computational cost is dominated by the eigen-decomposition, requiring $\mathcal{O}(N_{trd}^3)$ operations.[26] IMH preserves properties E_1 and E_2 of an effective hashcode across both modalities, while properties E_3 and E_4 are only guaranteed within $\mathcal{X}$ due to the spectral embedding.

6.5.6 A Brief Summary

We reviewed five influential algorithms for hashing across heterogeneous feature spaces: Cross-View Hashing (CVH), Co-Regularised Hashing (CRH), Cross-Modal Similarity Sensitive Hashing (CMSSH), Predictable Dual-View Hashing (PDH), and Inter-Media Hashing (IMH). Each learns two sets of K hyperplanes — one per modality — such that cross-modal neighbours receive consistent hashcodes. The unifying mechanism is an objective function that incorporates a cross-modal consistency term penalising discrepancies between paired projections. More recent approaches (CRH, PDH, IMH) strengthen this formulation with intra-modal terms, regularising the embeddings by enforcing similarity within each modality. Despite these advances, the primary limitations remain the reliance on computationally expensive matrix factorisations or non-convex optimisation procedures.

7. Conclusion

This chapter has outlined the background needed to understand traditional learning to hash methods. In Section 4.2 we motivated the need for scalable alternatives to brute-force nearest neighbour (NN) search, leading to approximate nearest neighbour (ANN) techniques and, in particular, the seminal framework of Locality Sensitive Hashing (LSH). Section 4.3 described how LSH generates similarity-preserving hash-codes for a range of similarity functions, with a particular focus on the inner product

---

26. For tractability, N_{trd} is typically restricted to $2,000$–$10,000$ in experiments. In real-world applications N_{trd} could be substantially larger.

case. The key property is that similar vectors receive hashcodes with small Hamming distance, enabling constant-time retrieval via hashtables—an exponential improvement over linear scan.

Sections 5–6 then examined the two critical components of hashing—projection (hyperplane learning) and binary quantisation. Retrieval performance depends heavily on how well these components preserve neighbourhood relationships in Hamming space. However, LSH generates both hyperplanes and thresholds randomly, relying on asymptotic guarantees that often fail in practice, especially with limited code length. This can scatter related data-points across buckets, undermining effectiveness. A central line of research in the period therefore sought to relax this data-independence assumption and instead learn projections and thresholds directly from the data.

The survey highlighted data-dependent hashing approaches of that era, including unsupervised (Section 6.3), supervised (Section 6.4), and cross-modal (Section 6.5) models. These formed the comparative baselines commonly adopted in experimental studies at the time.

At the same time, several limitations in the presented methods were evident: 1. Multi-threshold quantisation models (Section 5) were learned entirely in an unsupervised manner, without exploiting label information. 2. Thresholds were typically allocated uniformly across projected dimensions, ignoring variation in discriminative power. 3. Supervised projection functions (Sections 6.4–6.5) often relied on computationally expensive eigendecomposition or kernel methods, constraining scalability. 4. No work combined the learning of projection hypersurfaces with multiple quantisation thresholds in a unified framework.

These deficits defined the boundaries of the traditional (pre-deep learning) field as it was then understood.

The next section turns to the datasets, evaluation paradigms, and metrics that were standard for assessing nearest neighbour search effectiveness, and which provide the historical context for interpreting results across studies of that period.

8. Experimental Methodology

8.1 Introduction

This section outlines the experimental methodology that is *typically* adopted in the nearest-neighbour search literature. It covers the datasets commonly used as testbeds for retrieval studies (Section 8.2), standard definitions of ground truth (Section 8.3), and the evaluation metrics used to quantify retrieval effectiveness (Section 8.6). The intent is to align with established practice in learning-to-hash research while noting areas where methods vary across publications or exhibit design weaknesses. The discussion also highlights remedies that have been proposed in prior work to place evaluation on a more consistent footing.

8.2 Datasets

This survey focuses on learning hash functions for large-scale image retrieval, spanning three common application settings: (1) image-to-image retrieval from a still-image archive; (2) text-to-image retrieval; and (3) image-to-annotation retrieval. Accordingly, both unimodal and cross-modal scenarios are described. "Unimodal" refers to settings where query and database share the same visual representation (e.g., bag-of-visual-words features). "Cross-modal" settings span different modalities, such as text queries over image databases, mirroring contemporary web image search. To maintain comparability with prior art, this section catalogues widely used, publicly available datasets (Sections 8.2.1–8.2.2) and notes the preprocessing conventions most often reported. Public availability facilitates replication; widespread prior use facilitates direct comparison to earlier results.

8.2.1 Unimodal Retrieval Datasets

Four popular image datasets frequently appear in unimodal hashing studies: LabelMe, CIFAR-10, NUS-WIDE, and SIFT1M. They span sizes from 22,019 to 1M images, use diverse descriptors (GIST, SIFT, BoW), and cover varied content (natural scenes, personal photos, logos, drawings), offering a challenging evaluation suite. The specific dataset configurations below mirror those in widely cited work ([42], [78], [51]) and are publicly available.

- **LABELME:** 22,019 images with 512-D GIST descriptors ([83, 72])[27]; features are mean-centered.

- **CIFAR-10:** 60,000 32×32 color images from 80 Million Tiny Images ([45]), each encoded with a 512-D GIST descriptor ([61]) and labeled into 10 classes[28] (6,000 images per class); features are mean-centered.

- **NUS-WIDE:** 269,648 Flickr images with multiple concept tags from an 81-concept vocabulary[29] ([14]); images represented by 500-D BoW over vector-quantized SIFT; features are L_2-normalized and mean-centered. Sample images are shown in Figure 25.

- **SIFT1M:** 1,000,000 128-D SIFT descriptors from Flickr[30] introduced by [37] and now a standard NN benchmark ([42, 34, 87]); features are mean-centered.

---

27. http://www.cs.toronto.edu/~norouzi/research/mlh/
28. http://www.cs.toronto.edu/~kriz/cifar.html
29. http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm
30. http://lear.inrialpes.fr/~jegou/data.php

Figure 25: The NUS-WIDE dataset consists of around 270,000 images randomly sampled from Flickr. Given the diversity of images (from people, animals, landscapes to buildings and drawings) and widely varying resolution, the dataset provides a challenging testbed for image retrieval.

Table 3: Salient statistics of four datasets frequently used in unimodal hashing evaluations. Labels/image and Images/label are means over the full dataset.

Dataset	# images	# labels	Labels/image	Images/label	Descriptor
LABELME	22,019	–	–	–	512-D GIST
CIFAR-10	60,000	10	1	6,000	512-D GIST
NUS-WIDE	269,648	81	1.87	6,220	500-D BoW
SIFT1M	1,000,000	–	–	–	128-D SIFT

8.2.2 Cross-modal Retrieval Datasets

Cross-modal evaluations in the learning-to-hash literature most commonly use the *Wiki* dataset and NUS-WIDE ([49, 98, 81, 70, 12]), both providing paired image–text descriptions—crucial for training and assessment.

Figure 26: Three example Wikipedia article sections (Offa of Mercia, Tasmanian devil, and Chinua Achebe) with their aligned images from the cross-modal Wiki dataset.

- **Wiki:** Derived from 2,866 sections of featured Wikipedia articles[31] ([68]) across the 10 most populated categories; each section pairs a short text (median $\sim$200 words) with an author-assigned image. The commonly used feature set is 128-D SIFT BoW for images and 10-D LDA topic proportions for text ([10]), following [68] and subsequent hashing work ([98]). Examples are shown in Figure 26.

- **NUS-WIDE (cross-modal):** Starting from the raw NUS-WIDE data ([14]), cross-modal studies typically treat user tags as the textual modality. A common practice is to retain image–tag pairs from the 10 most frequent classes, reduce the 5,018-dimension tag space by PCA to 1,000 dimensions (then mean-center), and use the same 500-D visual BoW as in the unimodal setting; L_2 normalization and mean-centering are standard for visuals ([98]).

---

31. http://www.svcl.ucsd.edu/projects/crossmodal/

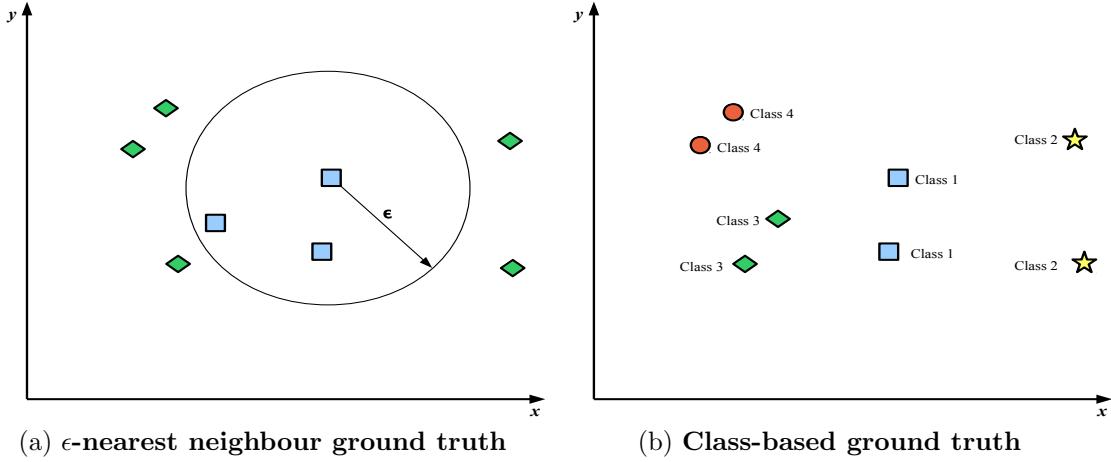(a) ϵ-**nearest neighbour ground truth**   (b) **Class-based ground truth**

Figure 27: Two common ground-truth definitions. (a) ϵ-ball neighbours; (b) class/label overlap.

8.3 Nearest Neighbour Ground-truth Definition

Evaluation requires a notion of *true* nearest neighbours for each query. Two definitions are prevalent: an ϵ-ball in feature space (Section 8.3.1) and class/label-based relevance (Section 8.3.2). To date, there has been little evidence connecting the ϵ-ball definition to user satisfaction.

8.3.1 ϵ-Ball Nearest Neighbours

A common approach is the ϵ-NN definition (Figure 27a)[32]. Following [44, 42, 47, 27], a random sample (e.g., 100 points) from the training set is used to estimate the Euclidean distance at which, on average, a point has R nearest neighbours; ϵ is set to that average distance. The literature commonly uses R=50 ([44, 42, 43, 47, 65, 27]). The ground-truth matrix $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ is obtained by thresholding pairwise distances $\mathbf{D}$ at ϵ (Equation 56).

$$\mathbf{S} = \begin{cases} S_{ij} = 1, & \text{if } D_{ij} \leq \epsilon, \\ S_{ij} = 0, & \text{if } D_{ij} > \epsilon. \end{cases} \tag{56}$$

8.3.2 Class-Based Nearest Neighbours

When modalities differ (e.g., text vs. image) or to align with streams of prior work, class- or tag-based definitions are widely used ([27, 51]). Here, $S_{ij} = 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ share at least one label/annotation; otherwise $S_{ij} = 0$ (Figure 27b).

---

32. An alternative is k-NN graphs; this survey focuses on ϵ-ball ground truth, leaving k-NN based evaluation to future work.

(a) **Hamming ranking evaluation**   (b) **Hash-table bucket evaluation**

Figure 28: Two evaluation paradigms for hashing. (a) Rank by Hamming distance, then score the ranking (e.g., AP, mAP). (b) Retrieve collisions from L hash tables and compute set-based metrics; in the toy example, $TP=3$, $FP=1$, $FN=2$, micro-$F_1=0.78$.

8.4 Evaluation Paradigms

Two paradigms dominate hashing evaluation: *Hamming ranking* (Section 8.4.1) and *hash-table bucket* evaluation (Section 8.4.2); see Figure 28. Hamming ranking is the standard in the learning-to-hash literature; bucket evaluation aligns closely with practical deployments emphasizing fast query time. Both are introduced below, followed by a rationale for the widespread use of Hamming ranking in survey comparisons (Section 8.4.3).

8.4.1 Hamming Ranking Evaluation

In Hamming ranking, binary hash codes are computed for queries and database items; database items are ranked by increasing Hamming distance from each query ([44, 42, 43, 51, 52, 27, 97, 48]). The ranked lists support metrics such as AUPRC (Section 8.6.3) and mAP (Section 8.6.4). This approach abstracts away implementation choices (e.g., specific (K, L) in LSH) while approximating their effect via thresholding on the ranked lists.

8.4.2 Hash-table Bucket-Based Evaluation

Bucket-based evaluation models the operational setting with $\mathcal{O}(1)$ lookup time, aggregating all items that collide with the query across L tables. Precision, recall, and F_β follow from counts of TPs, FPs, and FNs. Although this mirrors deployment priorities, it appears less frequently in published hashing studies, which tend to emphasize ranking-based summaries.

Hash Table Configurations

	Hamming Distance					
		110111 ⌐ Query	$K=3, L=2$	$K=6, L=1$	
	0	110111	$K=2, L=3$	$K=1, L=6$	
		--			t_1
Ranked List	1	110011	$K=3, L=2$	$K=1, L=6$	
			$K=2, L=3$		t_2
		--			
	2	100011	$K=2, L=3$		
		010011	$K=1, L=6$		t_3
		--			
	5	011000	$K=1, L=6$		

Figure 29: Thresholds on a Hamming-ranked list correspond to families of (K, L) that induce the same collisions, making ranking-based metrics a proxy for bucket performance.

8.4.3 Hamming Ranking versus Hash-table Evaluation

Bucket-based effectiveness depends on implementation choices (e.g., (K, L), chaining) and application constraints (e.g., memory budgets). Selecting a single (K, L) binds results to one design point, whereas sweeping many values multiplies result volume. Hamming ranking provides an implementation-agnostic proxy: thresholds on ranked lists correspond to many possible (K, L) settings, as illustrated in Figure 29. Consequently, aggregate ranking metrics such as mAP and AUPRC summarize expected performance across a range of hash-table configurations, motivating their prevalent use in comparisons.

8.5 Constructing Random Dataset Splits

This subsection summarizes split strategies used to form test/validation queries and training/database partitions for learning and evaluating hash functions. Two approaches recur: the *literature-standard* strategy (Section 8.5.1) and an *improved* variant intended to reduce overfitting (Section 8.5.2). Where class-based ground truth is used, balanced sampling across classes within each partition is commonly applied.

8.5.1 Literature-Standard Splits

Many studies adopt repeated random subsampling (ten runs is common) for evaluation ([51, 44, 42, 50, 88]). As illustrated in Figure 30, the dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ is divided into held-out test queries $\mathbf{X}_{teq}$ and a database $\mathbf{X}_{db}$. The database is also used during model development: validation queries $\mathbf{X}_{vaq}$ are run against a validation database $\mathbf{X}_{vad}$ (both sampled from $\mathbf{X}_{db}$) to tune hyperparameters via grid search, while $\mathbf{X}_{trd} \subset \mathbf{X}_{db}$ is used to learn parameters (e.g., hyperplanes, thresholds). The test queries are used once to score against $\mathbf{X}_{db}$.

Figure 30: A common dataset split in prior work. Grey indicates the test portion held out for final scoring.

Figure 31: An evaluation-oriented split that holds out both test queries and a test database (grey), mitigating coupling between training and final scoring.

8.5.2 Improved Splitting Strategy

A known concern with the literature-standard approach is potential overfitting when the same database is used both for training and as the index against which test queries are evaluated. To better assess generalization to unseen databases, an alternative strategy introduces a held-out *test database* $\mathbf{X}_{ted}$ separate from the training/indexing pool. As shown in Figure 31, five splits are used: test queries $\mathbf{X}_{teq}$ and a held-out test database $\mathbf{X}_{ted}$ (used once for final scoring), plus a development pool $\mathbf{X}_{db}$ partitioned into validation queries $\mathbf{X}_{vaq}$, a validation database $\mathbf{X}_{vad}$, and a training split $\mathbf{X}_{trd}$. Repeated random subsampling (e.g., ten runs) remains standard; only the isolation of the test database differs.

8.6 Evaluation Metrics

Consistent with the literature, retrieval effectiveness is summarized using standard IR measures: *precision, recall*, and F_β (Section 8.6.1), *area under the precision–recall curve* (AUPRC; Section 8.6.3), and *mean average precision* (mAP; Section 8.6.4).

8.6.1 PRECISION, RECALL, F_β-MEASURE

Precision, recall, and F_β are set-based metrics applicable to unranked retrieved sets formed either by hash-table collisions or by thresholding a Hamming radius in ranking evaluations. Let $\mathbf{S} \in \{0, 1\}^{N_{trd} \times N_{trd}}$ denote ground truth (Sections 8.3.1–8.3.2), with $S_{ij} = 1$ for true neighbour pairs. For a Hamming-radius threshold D, let $\mathbf{R}$ indicate retrieved items (Equation 57). Then:

$$R_{ij} = \begin{cases} 1, & \text{if } \mathbf{x}_j \text{ is within Hamming radius } D \text{ of } \mathbf{q}_i, \\ 0, & \text{otherwise.} \end{cases} \tag{57}$$

$$TP(\mathbf{q}_i) = \sum_j S_{ij} \cdot R_{ij} \tag{58}$$

$$FN(\mathbf{q}_i) = \sum_j S_{ij} - TP(\mathbf{q}_i) \tag{59}$$

$$FP(\mathbf{q}_i) = \sum_j (1 - S_{ij}) \cdot R_{ij} \tag{60}$$

$$P(\mathbf{q}_i) = \frac{TP(\mathbf{q}_i)}{TP(\mathbf{q}_i) + FP(\mathbf{q}_i)} \tag{61}$$

$$R(\mathbf{q}_i) = \frac{TP(\mathbf{q}_i)}{TP(\mathbf{q}_i) + FN(\mathbf{q}_i)} \tag{62}$$

Micro-averaged precision/recall over Q queries are:

$$P_{micro} = \frac{\sum_{i=1}^{Q} TP(\mathbf{q}_i)}{\sum_{i=1}^{Q} TP(\mathbf{q}_i) + \sum_{i=1}^{Q} FP(\mathbf{q}_i)} \tag{63}$$

$$R_{micro} = \frac{\sum_{i=1}^{Q} TP(\mathbf{q}_i)}{\sum_{i=1}^{Q} TP(\mathbf{q}_i) + \sum_{i=1}^{Q} FN(\mathbf{q}_i)} \tag{64}$$

The F_β-measure ([71]) is:

$$F_\beta = \frac{(1 + \beta^2) P_{micro} R_{micro}}{\beta^2 P_{micro} + R_{micro}} = \frac{(1 + \beta^2) TP_{micro}}{(1 + \beta^2) TP_{micro} + \beta^2 FN_{micro} + FP_{micro}} \tag{65}$$

Figure 32: Example PR curves for CIFAR-10 at 32 bits.

8.6.2 Precision–Recall Curve (PR Curve)

PR curves summarize effectiveness across operating points (e.g., varying Hamming radius D in ranking evaluation or (K, L) in bucket settings). For Hamming ranking, precision/recall are computed for radii $d = 1, \ldots, D_{max}$, tracing the precision–recall frontier (Figure 32).

8.6.3 Area Under the Precision–Recall Curve

AUPRC provides a single-number summary over recall:

$$
\begin{aligned}
AUPRC &= \int_0^1 P(R) \, dR \\
&= \sum_{d=1}^{D_{\max}} P(d) \, \Delta R(d).
\end{aligned}
\tag{66}
$$

where $P(R)$ is micro-precision at recall R, $P(d)$ the precision at Hamming radius d, and $\Delta R(d)$ the recall increment from $d-1$ to d.[33] Higher AUPRC indicates better retrieval.

---

33. The finite sum can be computed via the trapezoidal rule (e.g., `trapz` in MATLAB).

8.6.4 Mean Average Precision (mAP)

mAP summarizes ranking quality across queries. For a query $\mathbf{q}$, let L be the number of relevant items retrieved, $P_{\mathbf{q}}(r)$ the precision at rank r, and $\delta(r)$ an indicator that the r-th item is relevant. Average precision (AP) and mAP are:

$$AP(\mathbf{q}) = \frac{1}{L} \sum_{r=1}^{R} P_{\mathbf{q}}(r)\,\delta(r) \tag{67}$$

$$mAP = \frac{1}{|Q|} \sum_{i=1}^{Q} AP(\mathbf{q}_i) \tag{68}$$

mAP lies in $[0, 1]$ and emphasizes early precision. It is widely reported in several learning-to-hash sub-areas (e.g., supervised/unsupervised data-dependent projection: [52, 51, 27, 97]); to enable direct comparison, mAP is listed alongside AUPRC where appropriate.

8.6.5 Comparing AUPRC and mAP

Different sub-fields prefer different summaries (AUPRC vs. mAP). Since mAP is approximately the average of AUPRC over queries ([84]), both agree when relevant-document counts per query are balanced. Divergence arises under skew: AUPRC (a micro-average) favors performance on queries with many relevant items, whereas mAP (a macro-average) weights queries equally. The choice between them is application-dependent ([75]): system-oriented tasks may emphasize aggregate pair recoveries (AUPRC), while user-oriented tasks often value per-query equity (mAP).

8.7 Summary

This section consolidated evaluation practice for hashing-based ANN retrieval in the historical literature. Commonly used unimodal and cross-modal datasets were catalogued (Section 8.2); two prevailing ground-truth definitions were described (Section 8.3); standard evaluation paradigms—Hamming ranking and hash-table lookup—were contrasted, with motivation for ranking-based summaries in comparative studies (Section 8.4); dataset split strategies were reviewed, including an improved approach that isolates a held-out test database to better assess generalization (Section 8.5); and standard IR metrics (AUPRC, mAP) were defined and compared (Section 8.6).

9. Conclusion

This historical survey has reviewed the methodological landscape for learning to hash in large-scale image and cross-modal retrieval as it stood roughly a decade ago. Our intent has been to consolidate and standardise practices that were widely reported at that time, even though more recent advances in representation learning and approximate nearest neighbour search have since moved the field in new directions. By

examining the choices of datasets, groundtruth definitions, evaluation paradigms, and metrics, we sought to place the early contributions on a consistent foundation and highlight common assumptions and pitfalls.

Several broad lessons emerge from this retrospective:

- **Data-aware binarisation outperformed static rules.** Optimising one or more quantisation thresholds per projected dimension generally yielded more discriminative codes than the once-standard approach of placing a single static threshold at zero.

- **Not all projections were equally informative.** Allocating thresholds non-uniformly—assigning finer quantisation to more informative projections—proved to increase retrieval effectiveness over uniform allocation strategies.

- **Learned projections beat random ones.** Introducing supervision into the placement of hashing hypersurfaces typically improved effectiveness compared to purely random hyperplanes; in some cases, non-linear surfaces further helped when computation permitted.

- **Cross-modal hashing was feasible and valuable.** Extending supervised projection learning across modalities (e.g., text $\leftrightarrow$ image) offered state-of-the-art results on the benchmark datasets of the period.

- **Pipeline coupling mattered.** Early attempts at jointly learning projections and quantisation already showed improvements over treating each step in isolation.

At the same time, we observed recurring *evaluation pitfalls*: (i) inconsistent groundtruth definitions (class labels versus metric ϵ-balls) hindered comparability; (ii) widely adopted split protocols risked overfitting by reusing the same database for training and evaluation; and (iii) reporting practices varied between mAP and AUPRC without always clarifying when each metric was most appropriate.

Recommended practice at the time. To strengthen reproducibility and comparability, the community of that era benefited from: (1) using held-out *test queries and a disjoint test database* (Section 8.5.2); (2) reporting both AUPRC and mAP with clear groundtruth definitions; and (3) releasing code, splits, and preprocessing details whenever possible. These remain good practices today, though the technical context has since shifted.

Open directions: Looking forward from that period, several promising avenues were identified:

- **Human alignment:** verifying whether metric improvements (AUPRC/mAP) correlated with user satisfaction.

- **Online/streaming hashing:** adapting supervised projection learning to non-stationary streams with efficient updates.

- **Cross-lingual retrieval:** extending cross-modal hashing to multilingual document collections without relying on translation.

- **Dependence across bits:** exploring dependent thresholding across dimensions and decorrelation across hypersurfaces.

- **End-to-end objectives:** jointly optimising projection and quantisation in a single training criterion.

In retrospect, this body of work reflects the priorities and assumptions of an earlier research era: compact binary codes, efficiency under constrained hardware, and careful evaluation methodology. While deep neural networks and large-scale vector databases have since transformed the landscape, the historical lessons surveyed here remain instructive. They provide context for understanding how the field evolved and why certain problems—such as evaluation protocols, reproducibility, and the balance between efficiency and accuracy—continue to resonate today.

10. Postscript: Learning2hash.github.io as a Living Literature Review

The literature on Learning to Hash has expanded significantly over the past two decades, with applications now spanning computer vision, natural language processing, bioinformatics, recommendation, and large-scale information retrieval. The evolution from early locality-sensitive hashing approaches to supervised, unsupervised, and self-supervised models, as well as quantization and hybrid methods, reflects the growing sophistication and diversity of techniques in this area. Given the pace of publication across major venues such as CVPR, ICML, NeurIPS, KDD, and SIGIR, the field presents a moving target for systematic synthesis. A consolidated and regularly updated resource is therefore essential to support researchers in identifying methodological advances, application domains, and open challenges.

To address this gap, we introduce *Awesome Learning to Hash* (`https://learning2hash.github.io`), a continuously maintained, community-driven literature review (Figure 33). The website curates and organizes relevant publications into a structured taxonomy, enabling exploration by tags, methods, and application areas. Interactive features—including a two-dimensional map of papers, topic and author explorers, and resource pages—facilitate both high-level overviews and detailed study of specific research themes. In contrast to static surveys, this platform functions as a living review, evolving with contributions from the research community and providing an up-to-date reference point for ongoing developments in hashing-based approximate nearest neighbour search.
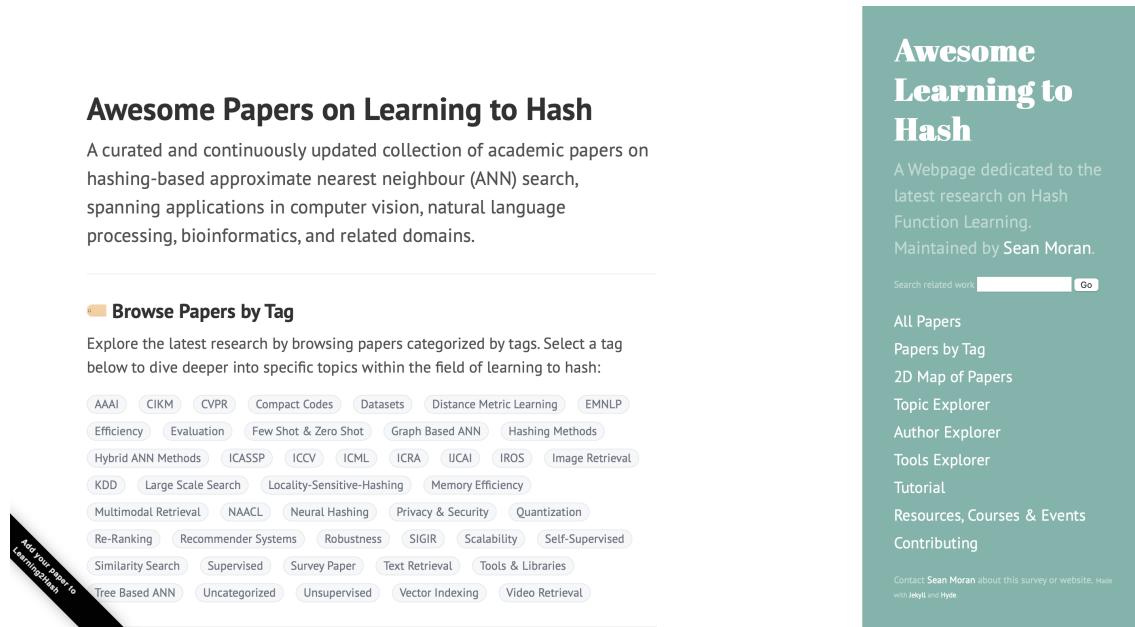
Figure 33: Screenshot of the *Awesome Learning to Hash* website (https://learning2hash.github.io), a continuously maintained literature review of hashing-based approximate nearest neighbour search.

References

[1] Charu C. Aggarwal and Chandan K. Reddy, editors. *Data Clustering: Algorithms and Applications*. CRC Press, 2014.

[2] M-Dyaa Albakour, Craig Macdonald, and Iadh Ounis. Using sensor metadata streams to identify topics of local events in the city. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 711–714, New York, NY, USA, 2015. ACM.

[3] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, May 1987.

[4] Shumeet Baluja and Michele Covell. Learning to Hash: Forgiving Hash Functions and Applications. *Data Min. Knowl. Discov.*, 17(3):402–430, December 2008.

[5] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Comput.*, 15(6):1373–1396, June 2003.

[6] Yoshua Bengio, Jean-Fran**c**ois Paiement, Pascal Vincent, Olivier Delalleau, Nicolas Le Roux, and Marie Ouimet. Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[7] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517, September 1975.

[8] Stefan Berchtold, Christian Böhm, Bernhard Braunmüller, Daniel A. Keim, and Hans-Peter Kriegel. Fast Parallel Similarity Search in Multimedia Databases. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 1–12, New York, NY, USA, 1997. ACM.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[10] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[11] A. Broder. On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–, Washington, DC, USA, 1997. IEEE Computer Society.

[12] Michael M. Bronstein, Alexander M. Bronstein, Fabrice Michel, and Nikos Paragios. Data fusion through cross-modality metric learning using similarity-sensitive hashing. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 0:3594–3601, 2010.

[13] Moses S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[14] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao. Zheng. NUS-WIDE: A Real-World Web Image Database from National University of Singapore. In *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece., 2009.

[15] Ondrej Chum, James Philbin, and Andrew Zisserman. Near Duplicate Image Detection: min-Hash and tf-idf Weighting. In *Proceedings of the British Machine Vision Conference 2008, Leeds, September 2008*, pages 1–10, 2008.

[16] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive Hashing Scheme Based on P-stable Distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 253–262, New York, NY, USA, 2004. ACM.

[17] Thomas Dean, Mark Ruzon, Mark Segal, Jonathon Shlens, Sudheendra Vijaya-narasimhan, and Jay Yagnik. Fast, Accurate Detection of 100,000 Object Classes on a Single Machine. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA, 2013.

[18] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei A. Efros. What Makes Paris Look Like Paris? *ACM Trans. Graph.*, 31(4):101:1–101:9, July 2012.

[19] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 194–202, 1995.

[20] Usama M. Fayyad and Keki B. Irani. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 1022–1029, 1993.

[21] Jinyuan Feng. Mobile Product Search with Bag of Hash Bits and Boundary Reranking. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3005–3012, Washington, DC, USA, 2012. IEEE Computer Society.

[22] Raphael A. Finkel and Jon Louis Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Inf.*, 4:1–9, 1974.

[23] Yoav Freund and Robert E. Schapire. A Decision-theoretic Generalization of Online Learning and an Application to Boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997.

[24] Salvador Garcia, Julian Luengo, Jose A. Saez, Victoria Lopez, and Francisco Herrera. A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning. *IEEE Trans. on Knowl. and Data Eng.*, 25(4):734–750, April 2013.

[25] Michel X. Goemans and David P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. ACM*, 42(6):1115–1145, November 1995.

[26] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.).* Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[27] Yunchao Gong and S. Lazebnik. Iterative Quantization: A Procrustean Approach to Learning Binary Codes. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 817–824, Washington, DC, USA, 2011. IEEE Computer Society.

[28] Kristen Grauman and Trevor Darrell. Fast Contour Matching Using Approximate Earth Mover's Distance. In *CVPR (1)*, pages 220–227, 2004.

[29] Kristen Grauman and Rob Fergus. Learning Binary Hash Codes for Large-Scale Image Search. In Roberto Cipolla, Sebastiano Battiato, and Giovanni Maria Farinella, editors, *Machine Learning for Computer Vision*, volume 411 of *Studies in Computational Intelligence*, pages 49–87. Springer Berlin Heidelberg, 2013.

[30] F. Gray. Pulse code communication, 1953. US Patent 2,632,058.

[31] R. M. Gray and D. L. Neuhoff. Quantization. *IEEE Trans. Inf. Theor.*, 44(6):2325–2383, September 2006.

[32] Richard J. Hanson and Michael J. Norris. Analysis of Measurements Based on the Singular Value Decomposition. *SIAM Journal on Scientific and Statistical Computing*, 2(3):363–373, 1981.

[33] David R. Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis; An overview with application to learning methods. Technical report, Royal Holloway, University of London, May 2003.

[34] Kaiming He, Fang Wen, and Jian Sun. K-Means Hashing: An Affinity-Preserving Quantization Method for Learning Binary Compact Codes. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 2938–2945, Washington, DC, USA, 2013. IEEE Computer Society.

[35] H. Hotelling. Analysis of a complex of statistical variables into principal components. *J. Educ. Psych.*, 24, 1933.

[36] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[37] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, January 2011.

[38] Thorsten Joachims. Training Linear SVMs in Linear Time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 217–226, New York, NY, USA, 2006. ACM.

[39] Norio Katayama and Shin'ichi Satoh. The SR-tree: An Index Structure for High-dimensional Nearest Neighbor Queries. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 369–380, New York, NY, USA, 1997. ACM.

[40] Randy Kerber. ChiMerge: Discretization of Numeric Attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 123–128. AAAI Press, 1992.

[41] Effrosini Kokiopoulou, Jie Chen, and Yousef Saad. Trace optimization and eigen-problems in dimension reduction methods. *Numerical Lin. Alg. with Applic.*, 18(3):565–602, 2011.

[42] Weihao Kong and Wu-Jun Li. Double-Bit Quantization for Hashing. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*, 2012.

[43] Weihao Kong and Wu-Jun Li. Isotropic Hashing. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1655–1663, 2012.

[44] Weihao Kong, Wu-Jun Li, and Minyi Guo. Manhattan Hashing for Large-scale Image Retrieval. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 45–54, New York, NY, USA, 2012. ACM.

[45] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[46] Brian Kulis. Metric Learning: A Survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2013.

[47] Brian Kulis and Trevor Darrell. Learning to Hash with Binary Reconstructive Embeddings. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 1042–1050, 2009.

[48] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pages 2130–2137, 2009.

[49] Shaishav Kumar and Raghavendra Udupa. Learning Hash Functions for Cross-view Similarity Search. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI '11, pages 1360–1365. AAAI Press, 2011.

[50] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. Discrete Graph Hashing. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3419–3427, 2014.

[51] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 2074–2081, 2012.

[52] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with Graphs. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 1–8, 2011.

[53] Xianglong Liu, Junfeng He, and Bo Lang. Reciprocal Hash Tables for Nearest Neighbor Search. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*, 2013.

[54] S. Lloyd. Least Square Quantization in PCM. *IEEE Trans. Inform. Theory*, 28:129–137, 1982.

[55] Sameep Mehta, Srinivasan Parthasarathy, and Hui Yang. Toward Unsupervised Correlation Preserving Discretization. *IEEE Trans. on Knowl. and Data Eng.*, 17(9):1174–1185, September 2005.

[56] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[57] Sean Moran and Victor Lavrenko. Graph Regularised Hashing. In *Advances in Information Retrieval - 37th European Conference on IR Research, ECIR 2015, Vienna, Austria, March 29 - April 2, 2015. Proceedings*, pages 135–146, 2015.

[58] Sean Moran, Victor Lavrenko, and Miles Osborne. Neighbourhood Preserving Quantisation for LSH. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 1009–1012, New York, NY, USA, 2013. ACM.

[59] Sean Moran, Victor Lavrenko, and Miles Osborne. Variable Bit Quantisation for LSH. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 2: Short Papers*, pages 753–758, 2013.

[60] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[61] Aude Oliva and Antonio Torralba. Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. *Int. J. Comput. Vision*, 42(3):145–175, May 2001.

[62] Miles Osborne, Sean Moran, Richard McCreadie, Alexander von Lünen, Martin D. Sykora, Amparo Elizabeth Cano, Neil Ireson, Craig Macdonald, Iadh Ounis, Yulan He, Tom Jackson, Fabio Ciravegna, and Ann O'Brien. Real-Time

Detection, Tracking, and Monitoring of Automatically Discovered Events in Social Media. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pages 37–42, 2014.

[63] Sasa Petrovic. *Real-Time Event Detection in Massive Streams.* PhD thesis, University of Edinburgh, 2012.

[64] Saša Petrović, Miles Osborne, and Victor Lavrenko. Streaming First Story Detection with Application to Twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 181–189, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[65] Maxim Raginsky and Svetlana Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 1509–1517, 2009.

[66] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1177–1184, 2007.

[67] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets.* Cambridge University Press, New York, NY, USA, 2011.

[68] N. Rasiwasia, J. Costa Pereira, E. Coviello, G. Doyle, G.R.G. Lanckriet, R. Levy, and N. Vasconcelos. A New Approach to Cross-Modal Multimedia Retrieval. In *ACM International Conference on Multimedia*, pages 251–260, 2010.

[69] Nikhil Rasiwasia, Dhruv Mahajan, Vijay Mahadevan, and Gaurav Aggarwal. Cluster Canonical Correlation Analysis. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2014.

[70] Mohammad Rastegari, Jonghyun Choi, Shobeir Fakhraei, Hal Daumé III, and Larry S. Davis. Predictable dual-view hashing. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1328–1336, 2013.

[71] C. J. Van Rijsbergen. *Information Retrieval.* Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[72] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. LabelMe: A Database and Web-Based Tool for Image Annotation. *Int. J. Comput. Vision*, 77(1-3):157–173, May 2008.

[73] Yousef Saad, editor. *Numerical Methods for Large Eigenvalue Problems, 2nd revised edition*. SIAM, 2011.

[74] PeterH. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

[75] Fabrizio Sebastiani. Machine Learning in Automated Text Categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002.

[76] Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast Pose Estimation with Parameter-Sensitive Hashing. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 750–, Washington, DC, USA, 2003. IEEE Computer Society.

[77] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 807–814, New York, NY, USA, 2007. ACM.

[78] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised Discrete Hashing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 37–45, June 2015.

[79] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, August 2000.

[80] Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-Based Image Retrieval at the End of the Early Years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12):1349–1380, December 2000.

[81] Jingkuan Song, Yang Yang, Yi Yang, Zi Huang, and Heng Tao Shen. Inter-media Hashing for Large-scale Retrieval from Heterogeneous Data Sources. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 785–796, New York, NY, USA, 2013. ACM.

[82] Kengo Terasawa and Yuzuru Tanaka. Spherical LSH for Approximate Nearest Neighbor Search on Unit Hypersphere. In Frank Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *Algorithms and Data Structures*, volume 4619 of *Lecture Notes in Computer Science*, pages 27–38. Springer Berlin Heidelberg, 2007.

[83] Antonio Torralba, Robert Fergus, and Yair Weiss. Small codes and large image databases for recognition. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*, 2008.

[84] Andrew Turpin and Falk Scholer. User Performance Versus Precision Measures for Simple Search Tasks. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 11–18, New York, NY, USA, 2006. ACM.

[85] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for Similarity Search: A Survey. *CoRR*, abs/1408.2927, 2014.

[86] Jun Wang, Ondrej Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, pages 3424–3431, 2010.

[87] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential Projection Learning for Hashing with Compact Codes. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 1127–1134, 2010.

[88] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-Supervised Hashing for Large-Scale Search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(12):2393–2406, December 2012.

[89] Zhe Wang, Ling-Yu Duan, Jie Lin, Xiaofang Wang, Tiejun Huang, and Wen Gao. Hamming Compatible Quantization for Hashing. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '15, 2015.

[90] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[91] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral Hashing. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 1753–1760, 2008.

[92] Christopher K. I. Williams and Matthias Seeger. Using the Nyström Method to Speed Up Kernel Machines. In T.K. Leen, T.G. Dietterich, and V. Tresp,

editors, *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

[93] Dong Xu, Tat Jen Cham, Shuicheng Yan, Lixin Duan, and Shih-Fu Chang. Near Duplicate Identification With Spatially Aligned Pyramid Matching. *IEEE Transactions on Circuits and Systems for Video Technology*, 20:1068–1079, 2010.

[94] Hao Xu, Jingdong Wang, Zhu Li, Gang Zeng, Shipeng Li, and Nenghai Yu. Complementary Hashing for Approximate Nearest Neighbor Search. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 1631–1638, Washington, DC, USA, 2011. IEEE Computer Society.

[95] Ying Yang and Geoffrey I. Webb. Discretization for naive-Bayes Learning: Managing Discretization Bias and Variance. *Mach. Learn.*, 74(1):39–74, January 2009.

[96] A. L. Yuille and Anand Rangarajan. The Concave-convex Procedure. *Neural Comput.*, 15(4):915–936, April 2003.

[97] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. Self-taught Hashing for Fast Similarity Search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 18–25, New York, NY, USA, 2010. ACM.

[98] Yi Zhen and Dit-Yan Yeung. Co-Regularized Hashing for Multimodal Data. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1385–1393, 2012.